Package java.lang

Interface Summary	
Runnable	The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread.

Class Summary	
Boolean	The Boolean class wraps a value of the primitive type `boolean` in an object.
Byte	The Byte class is the standard wrapper for byte values.
Character	The Character class wraps a value of the primitive type `char` in an object.
Class	Instances of the class `Class` represent classes and interfaces in a running Java application.
Integer	The Integer class wraps a value of the primitive type `int` in an object.
Long	The Long class wraps a value of the primitive type `long` in an object.
Math	The class `Math` contains methods for performing basic numeric operations.
Object	Class `Object` is the root of the class hierarchy.
Runtime	Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running.
Short	The Short class is the standard wrapper for short values.
String	The `String` class represents character strings.
StringBuffer	A string buffer implements a mutable sequence of characters.
System	The `System` class contains several useful class fields and methods.
Thread	A *thread* is a thread of execution in a program.
Throwable	The `Throwable` class is the superclass of all errors and exceptions in the Java language.

Exception Summary	
ArithmeticException	Thrown when an exceptional arithmetic condition has occurred.
ArrayIndexOutOfBoundsException	Thrown to indicate that an array has been accessed with an illegal index.
ArrayStoreException	Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.

ClassCastException	Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
ClassNotFoundException	Thrown when an application tries to load in a class through its string name using: The `forName` method in class `Class`.
Exception	The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.
IllegalAccessException	Thrown when an application tries to load in a class, but the currently executing method does not have access to the definition of the specified class, because the class is not public and in another package.
IllegalArgumentException	Thrown to indicate that a method has been passed an illegal or inappropriate argument.
IllegalMonitorStateException	Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.
IllegalThreadStateException	Thrown to indicate that a thread is not in an appropriate state for the requested operation.
IndexOutOfBoundsException	Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.
InstantiationException	Thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface or is an abstract class.
InterruptedException	Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the `interrupt` method in class `Thread`.
NegativeArraySizeException	Thrown if an application tries to create an array with negative size.
NullPointerException	Thrown when an application attempts to use `null` in a case where an object is required.
NumberFormatException	Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.
RuntimeException	`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.
SecurityException	Thrown by the security manager to indicate a security violation.
StringIndexOutOfBoundsException	Thrown by the `charAt` method in class `String` and by other `String` methods to indicate that an index is either negative or greater than or equal to the size of the string.

Error Summary	
Error	An `Error` is a subclass of `Throwable` that indicates serious problems that a reasonable application should not try to catch.
OutOfMemoryError	Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.
VirtualMachineError	Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.

java.lang
Class ArithmeticException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.ArithmeticException
```

public class **ArithmeticException**
extends RuntimeException

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class.

Since:
JDK1.0

Constructor Summary
`ArithmeticException``()` Constructs an `ArithmeticException` with no detail message.
`ArithmeticException``(String s)` Constructs an `ArithmeticException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ArithmeticException

`public ArithmeticException()`

Constructs an `ArithmeticException` with no detail message.

ArithmeticException

`public ArithmeticException(String s)`

Constructs an `ArithmeticException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class ArrayIndexOutOfBoundsException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IndexOutOfBoundsException
                          |
                          +--java.lang.ArrayIndexOutOfBoundsException
```

---

public class **ArrayIndexOutOfBoundsException**
extends IndexOutOfBoundsException

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Since:

JDK1.0

Constructor Summary

`ArrayIndexOutOfBoundsException()` Constructs an `ArrayIndexOutOfBoundsException` with no detail message.
`ArrayIndexOutOfBoundsException(int index)` Constructs a new `ArrayIndexOutOfBoundsException` class with an argument indicating the illegal index.
`ArrayIndexOutOfBoundsException(String s)` Constructs an `ArrayIndexOutOfBoundsException` class with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ArrayIndexOutOfBoundsException

```
public ArrayIndexOutOfBoundsException()
```

Constructs an `ArrayIndexOutOfBoundsException` with no detail message.

ArrayIndexOutOfBoundsException

```
public ArrayIndexOutOfBoundsException(int index)
```

Constructs a new `ArrayIndexOutOfBoundsException` class with an argument indicating the illegal index.

Parameters:

`index` - the illegal index.

ArrayIndexOutOfBoundsException

```
public ArrayIndexOutOfBoundsException(String s)
```

Constructs an `ArrayIndexOutOfBoundsException` class with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class ArrayStoreException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.ArrayStoreException
```

---

public class **ArrayStoreException**
extends RuntimeException

Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects. For example, the following code generates an `ArrayStoreException`:

```
Object x[] = new String[3];
x[0] = new Integer(0);
```

Since:

JDK1.0

Constructor Summary

`ArrayStoreException()` Constructs an `ArrayStoreException` with no detail message.
`ArrayStoreException(String s)` Constructs an `ArrayStoreException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ArrayStoreException

`public ArrayStoreException()`

Constructs an `ArrayStoreException` with no detail message.

ArrayStoreException

`public ArrayStoreException(String s)`

Constructs an `ArrayStoreException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class Boolean

```
java.lang.Object
  |
  +--java.lang.Boolean
```

---

public final class **Boolean**
extends Object

The Boolean class wraps a value of the primitive type `boolean` in an object. An object of type `Boolean` contains a single field whose type is `boolean`.

Since:

JDK1.0

Constructor Summary

`Boolean``(boolean value)` Allocates a `Boolean` object representing the `value` argument.	

Method Summary

`boolean`	**`booleanValue`**`()` Returns the value of this `Boolean` object as a boolean primitive.
`boolean`	**`equals`**`(Object obj)` Returns `true` if and only if the argument is not `null` and is a `Boolean` object that represents the same `boolean` value as this object.
`int`	**`hashCode`**`()` Returns a hash code for this `Boolean` object.

Methods inherited from class java.lang.Object
`getClass, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Boolean

```
public Boolean(boolean value)
```

Allocates a `Boolean` object representing the `value` argument.

Parameters:

`value` - the value of the `Boolean`.

Method Detail

booleanValue

```
public boolean booleanValue()
```

Returns the value of this `Boolean` object as a boolean primitive.

Returns:

the primitive `boolean` value of this object.

---

hashCode

```
public int hashCode()
```

Returns a hash code for this `Boolean` object.

Overrides:

hashCode in class Object

Returns:

the integer `1231` if this object represents `true`; returns the integer `1237` if this object represents `false`.

---

equals

```
public boolean equals(Object obj)
```

Returns `true` if and only if the argument is not `null` and is a `Boolean` object that represents the same `boolean` value as this object.

Overrides:

equals in class Object

Parameters:

`obj` - the object to compare with.

Returns:

`true` if the Boolean objects represent the same value; `false` otherwise.

---

java.lang
Class Byte

```
java.lang.Object
  |
  +--java.lang.Byte
```

public final class **Byte**
extends Object

The Byte class is the standard wrapper for byte values.

Since:

JDK1.1

Field Summary

static byte	**MAX_VALUE** The maximum value a Byte can have.
static byte	**MIN_VALUE** The minimum value a Byte can have.

Constructor Summary

Byte(byte value) Constructs a Byte object initialized to the specified byte value.	

Method Summary

byte	**byteValue**() Returns the value of this Byte as a byte.
boolean	**equals**(Object obj) Compares this object to the specified object.
int	**hashCode**() Returns a hashcode for this Byte.

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, toString, wait, wait, wait

Field Detail

MIN_VALUE

```
public static final byte MIN_VALUE
```

The minimum value a Byte can have.

MAX_VALUE

```
public static final byte MAX_VALUE
```

The maximum value a Byte can have.

Constructor Detail

Byte

```
public Byte(byte value)
```

Constructs a Byte object initialized to the specified byte value.
Parameters:
`value` - the initial value of the Byte

Method Detail

byteValue

```
public byte byteValue()
```

Returns the value of this Byte as a byte.

---

hashCode

```
public int hashCode()
```

Returns a hashcode for this Byte.
Overrides:
hashCode in class Object
Tags copied from class: Object
Returns:
a hash code value for this object.
See Also:
`Object.equals(java.lang.Object)`, `Hashtable`

---

equals

```
public boolean equals(Object obj)
```

Compares this object to the specified object.
Overrides:
equals in class Object
Parameters:
`obj` - the object to compare with
Returns:
true if the objects are the same; false otherwise.

---

java.lang
Class Character

```
java.lang.Object
  |
  +--java.lang.Character
```

public final class **Character**
extends Object

The Character class wraps a value of the primitive type `char` in an object. An object of type `Character` contains a single field whose type is `char`.

In addition, this class provides several methods for determining the type of a character and converting characters from uppercase to lowercase and vice versa.

Since:
JDK1.0

Field Summary

static int	**MAX_RADIX** The maximum radix available for conversion to and from Strings.
static char	**MAX_VALUE** The constant value of this field is the largest value of type `char`.
static int	**MIN_RADIX** The minimum radix available for conversion to and from Strings.
static char	**MIN_VALUE** The constant value of this field is the smallest value of type `char`.

Constructor Summary

Character(char value) Constructs a `Character` object and initializes it so that it represents the primitive `value` argument.

Method Summary

char	**charValue**() Returns the value of this Character object.
static int	**digit**(char ch, int radix) Returns the numeric value of the character `ch` in the specified radix.
boolean	**equals**(Object obj) Compares this object against the specified object.
int	**hashCode**() Returns a hash code for this Character.
static boolean	**isDigit**(char ch) Determines if the specified character is a digit.
static boolean	**isLowerCase**(char ch) Determines if the specified character is a lowercase character.
static boolean	**isUpperCase**(char ch) Determines if the specified character is an uppercase character.
static char	**toLowerCase**(char ch) The given character is mapped to its lowercase equivalent; if the character has no lowercase equivalent, the character itself is returned.
String	**toString**() Returns a String object representing this character's value.
static char	**toUpperCase**(char ch) Converts the character argument to uppercase; if the character has no lowercase equivalent, the character itself is returned.

Methods inherited from class java.lang.Object
`getClass, notify, notifyAll, wait, wait, wait`

Field Detail

MIN_RADIX

```
public static final int MIN_RADIX
```

The minimum radix available for conversion to and from Strings. The constant value of this field is the smallest value permitted for the radix argument in radix-conversion methods such as the `digit` method, the `forDigit` method, and the `toString` method of class `Integer`.
See Also:
`Integer.toString(int, int)`, `Integer.valueOf(java.lang.String)`

MAX_RADIX

```
public static final int MAX_RADIX
```

The maximum radix available for conversion to and from Strings. The constant value of this field is the largest value permitted for the radix argument in radix-conversion methods such as the `digit` method, the `forDigit` method, and the `toString` method of class `Integer`.

See Also:

`Integer.toString(int, int)`, `Integer.valueOf(java.lang.String)`

MIN_VALUE

```
public static final char MIN_VALUE
```

The constant value of this field is the smallest value of type `char`.

Since:

JDK1.0.2

MAX_VALUE

```
public static final char MAX_VALUE
```

The constant value of this field is the largest value of type `char`.

Since:

JDK1.0.2

Constructor Detail

Character

```
public Character(char value)
```

Constructs a `Character` object and initializes it so that it represents the primitive `value` argument.

Parameters:

`value` - value for the new `Character` object.

Method Detail

charValue

```
public char charValue()
```

Returns the value of this Character object.

Returns:

the primitive `char` value represented by this object.

hashCode

```
public int hashCode()
```

Returns a hash code for this Character.

Overrides:

hashCode in class Object

Returns:

a hash code value for this object.

---

equals

```
public boolean equals(Object obj)
```

Compares this object against the specified object. The result is `true` if and only if the argument is not `null` and is a `Character` object that represents the same `char` value as this object.

Overrides:

equals in class Object

Parameters:

`obj` - the object to compare with.

Returns:

`true` if the objects are the same; `false` otherwise.

---

toString

```
public String toString()
```

Returns a String object representing this character's value. Converts this `Character` object to a string. The result is a string whose length is `1`. The string's sole component is the primitive `char` value represented by this object.

Overrides:

toString in class Object

Returns:

a string representation of this object.

---

isLowerCase

```
public static boolean isLowerCase(char ch)
```

Determines if the specified character is a lowercase character. This is currently only supported for ISO-LATIN-1 characters: "a" through "z".

Parameters:

`ch` - the character to be tested.

Returns:

`true` if the character is lowercase; `false` otherwise.

Since:

JDK1.0

See Also:
`isLowerCase(char)`, `toLowerCase(char)`

---

isUpperCase

```
public static boolean isUpperCase(char ch)
```

Determines if the specified character is an uppercase character. This is currently only supported for ISO-LATIN-1 characters: "A" through "Z".

Parameters:
`ch` - the character to be tested.

Returns:
`true` if the character is uppercase; `false` otherwise.

Since:
1.0

See Also:
`isLowerCase(char)`, `toUpperCase(char)`

---

isDigit

```
public static boolean isDigit(char ch)
```

Determines if the specified character is a digit. This is currently only supported for ISO-LATIN-1 digits: "0" through "9".

Parameters:
`ch` - the character to be tested.

Returns:
`true` if the character is a digit; `false` otherwise.

Since:
JDK1.0

---

toLowerCase

```
public static char toLowerCase(char ch)
```

The given character is mapped to its lowercase equivalent; if the character has no lowercase equivalent, the character itself is returned. This is currently only supported for ISO-LATIN-1 characters.

Parameters:
`ch` - the character to be converted.

Returns:
the lowercase equivalent of the character, if any; otherwise the character itself.

Since:
JDK1.0

See Also:
`isLowerCase(char)`, `isUpperCase(char)`, `toUpperCase(char)`

---

toUpperCase

```
public static char toUpperCase(char ch)
```

Converts the character argument to uppercase; if the character has no lowercase equivalent, the character itself is returned. This is currently only supported for ISO-LATIN-1 characters.

Parameters:
`ch` - the character to be converted.

Returns:
the uppercase equivalent of the character, if any; otherwise the character itself.

Since:
JDK1.0

See Also:
`isLowerCase(char)`, `isUpperCase(char)`, `toLowerCase(char)`

---

digit

```
public static int digit(char ch,
                        int radix)
```

Returns the numeric value of the character `ch` in the specified radix. This is only supported for ISO-LATIN-1 characters.

Parameters:
`ch` - the character to be converted.
`radix` - the radix.

Returns:
the numeric value represented by the character in the specified radix.

Since:
JDK1.0

See Also:
`isDigit(char)`

---

java.lang
Class Class

```
java.lang.Object
  |
  +--java.lang.Class
```

public final class **Class**
extends Object

Instances of the class `Class` represent classes and interfaces in a running Java application. Every array also belongs to a class that is reflected as a `Class` object that is shared by all arrays with the same element type and number of dimensions.

`Class` has no public constructor. Instead `Class` objects are constructed automatically by the Java Virtual Machine as classes are loaded.

The following example uses a `Class` object to print the class name of an object:

```
void printClassName(Object obj) {
    System.out.println("The class of " + obj +
                       " is " + obj.getClass().getName());
}
```

Since:
JDK1.0

Method Summary

`static Class`	**`forName`**`(String className)` Returns the `Class` object associated with the class with the given string name.
`String`	**`getName`**`()` Returns the fully-qualified name of the entity (class, interface, array class, primitive type, or void) represented by this `Class` object, as a `String`.
`InputStream`	**`getResourceAsStream`**`(String name)` Finds a resource with a given name.
`boolean`	**`isArray`**`()` Determines if this `Class` object represents an array class.
`boolean`	**`isAssignableFrom`**`(Class cls)` Determines if the class or interface represented by this `Class` object is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified `Class` parameter.
`boolean`	**`isInstance`**`(Object obj)` Determines if the specified `Object` is assignment-compatible with the object represented by this `Class`.
`boolean`	**`isInterface`**`()` Determines if the specified `Class` object represents an interface type.
`Object`	**`newInstance`**`()` Creates a new instance of a class.
`String`	**`toString`**`()` Converts the object to a string.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Method Detail

toString

```
public String toString()
```

Converts the object to a string. The string representation is the string "class" or "interface", followed by a space, and then by the fully qualified name of the class in the format returned by `getName`. If this `Class` object represents a primitive type, this method returns the name of the primitive type. If this `Class` object represents void this method returns "void".

Overrides:
toString in class Object

Returns:
a string representation of this class object.

---

forName

```
public static Class forName(String className)
                    throws ClassNotFoundException
```

Returns the `Class` object associated with the class with the given string name. Given the fully-qualified name for a class or interface, this method attempts to locate, load and link the class. If it succeeds, returns the Class object representing the class. If it fails, the method throws a ClassNotFoundException.

For example, the following code fragment returns the runtime `Class` descriptor for the class named `java.lang.Thread`:

```
Class t = Class.forName("java.lang.Thread")
```

Parameters:
`className` - the fully qualified name of the desired class.
Returns:
the `Class` descriptor for the class with the specified name.
Throws:
ClassNotFoundException - if the class could not be found.
Since:
JDK1.0

---

newInstance

```
public Object newInstance()
                  throws InstantiationException,
                         IllegalAccessException
```

Creates a new instance of a class.
Returns:
a newly allocated instance of the class represented by this object. This is done exactly as if by a `new` expression with an empty argument list.
Throws:
IllegalAccessException - if the class or initializer is not accessible.
InstantiationException - if an application tries to instantiate an abstract class or an interface, or if the instantiation fails for some other reason.
Since:
JDK1.0

---

isInstance

```
public boolean isInstance(Object obj)
```

Determines if the specified `Object` is assignment-compatible with the object represented by this `Class`. This method is the dynamic equivalent of the Java language `instanceof` operator. The method returns `true` if the specified `Object` argument is non-null and can be cast to the reference type represented by this `Class` object without raising a `ClassCastException.` It

returns `false` otherwise.

Specifically, if this `Class` object represents a declared class, this method returns `true` if the specified `Object` argument is an instance of the represented class (or of any of its subclasses); it returns `false` otherwise. If this `Class` object represents an array class, this method returns `true` if the specified `Object` argument can be converted to an object of the array class by an identity conversion or by a widening reference conversion; it returns `false` otherwise. If this `Class` object represents an interface, this method returns `true` if the class or any superclass of the specified `Object` argument implements this interface; it returns `false` otherwise. If this `Class` object represents a primitive type, this method returns `false`.
Parameters:
`obj` - the object to check
Returns:
true if `obj` is an instance of this class
Since:
JDK1.1

---

isAssignableFrom

```
public boolean isAssignableFrom(Class cls)
```

Determines if the class or interface represented by this `Class` object is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified `Class` parameter. It returns `true` if so; otherwise it returns `false`. If this `Class` object represents a primitive type, this method returns `true` if the specified `Class` parameter is exactly this `Class` object; otherwise it returns `false`.

Specifically, this method tests whether the type represented by the specified `Class` parameter can be converted to the type represented by this `Class` object via an identity conversion or via a widening reference conversion. See *The Java Language Specification*, sections 5.1.1 and 5.1.4 , for details.
Parameters:
`cls` - the `Class` object to be checked
Returns:
the `boolean` value indicating whether objects of the type `cls` can be assigned to objects of this class
Throws:
NullPointerException - if the specified Class parameter is null.
Since:
JDK1.1

---

isInterface

```
public boolean isInterface()
```

Determines if the specified `Class` object represents an interface type.
Returns:
`true` if this object represents an interface; `false` otherwise.

isArray

```
public boolean isArray()
```

Determines if this `Class` object represents an array class.

Returns:

`true` if this object represents an array class; `false` otherwise.

Since:

JDK1.1

---

getName

```
public String getName()
```

Returns the fully-qualified name of the entity (class, interface, array class, primitive type, or void) represented by this `Class` object, as a `String`.

If this `Class` object represents a class of arrays, then the internal form of the name consists of the name of the element type in Java signature format, preceded by one or more "`[`" characters representing the depth of array nesting. Thus:

```
(new Object[3]).getClass().getName()
```

returns "`[Ljava.lang.Object;`" and:

```
(new int[3][4][5][6][7][8][9]).getClass().getName()
```

returns "`[[[[[[[I`". The encoding of element type names is as follows:

```
B            byte
C            char
D            double
F            float
I            int
J            long
Lclassname;  class or interface
S            short
Z            boolean
```

The class or interface name *`classname`* is given in fully qualified form as shown in the example above.

Returns:

the fully qualified name of the class or interface represented by this object.

---

getResourceAsStream

```
public InputStream getResourceAsStream(String name)
```

Finds a resource with a given name. This method returns null if no resource with this name is found. The rules for searching resources associated with a given class are profile specific.

Parameters:

`name` - name of the desired resource

Returns:

a `java.io.InputStream` object.

Since:

JDK1.1

---

java.lang
Class ClassCastException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.ClassCastException
```

public class **ClassCastException**
extends RuntimeException

Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance. For example, the following code generates a `ClassCastException`:

```
Object x = new Integer(0);
System.out.println((String)x);
```

Since:
JDK1.0

Constructor Summary

`ClassCastException()` Constructs a `ClassCastException` with no detail message.
`ClassCastException(String s)` Constructs a `ClassCastException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ClassCastException

```
public ClassCastException()
```

Constructs a `ClassCastException` with no detail message.

ClassCastException

```
public ClassCastException(String s)
```

Constructs a `ClassCastException` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class ClassNotFoundException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.ClassNotFoundException
```

public class **ClassNotFoundException**
extends Exception

Thrown when an application tries to load in a class through its string name using:

- The `forName` method in class `Class`.

but no definition for the class with the specified name could be found.

Since:
JDK1.0
See Also:
`Class.forName(java.lang.String)`

Constructor Summary

`ClassNotFoundException``()` Constructs a `ClassNotFoundException` with no detail message.
`ClassNotFoundException``(String s)` Constructs a `ClassNotFoundException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ClassNotFoundException

`public` **`ClassNotFoundException`**`()`

Constructs a `ClassNotFoundException` with no detail message.

ClassNotFoundException

`public` **`ClassNotFoundException`**`(String s)`

Constructs a `ClassNotFoundException` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class Error

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Error
```

Direct Known Subclasses:
VirtualMachineError

---

public class **Error**
extends Throwable

An `Error` is a subclass of `Throwable` that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.

A method is not required to declare in its `throws` clause any subclasses of `Error` that might be thrown during the execution of the method but not caught, since these errors are abnormal conditions that should never occur.

Since:
JDK1.0

Constructor Summary

`Error()` Constructs an `Error` with no specified detail message.
`Error(String s)` Constructs an Error with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

Error

```
public Error()
```

Constructs an `Error` with no specified detail message.

Error

```
public Error(String s)
```

Constructs an Error with the specified detail message.
Parameters:
`s` - the detail message.

---

java.lang
Class Exception

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
```

Direct Known Subclasses:
ClassNotFoundException, IllegalAccessException, InstantiationException, InterruptedException, IOException, RuntimeException

public class **Exception**
extends Throwable

The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.

Since:
JDK1.0
See Also:
`Error`

Constructor Summary

`Exception()` Constructs an `Exception` with no specified detail message.
`Exception(String s)` Constructs an `Exception` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

Exception

```
public Exception()
```

Constructs an `Exception` with no specified detail message.

Exception

```
public Exception(String s)
```

Constructs an `Exception` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class IllegalAccessException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.IllegalAccessException
```

public class **IllegalAccessException**
extends Exception

Thrown when an application tries to load in a class, but the currently executing method does not have access to the definition of the specified class, because the class is not public and in another package.

An instance of this class can also be thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the current method does not have access to the appropriate zero-argument constructor.

Since:
JDK1.0
See Also:
`Class.forName(java.lang.String)`, `Class.newInstance()`

Constructor Summary

`IllegalAccessException``()` Constructs an `IllegalAccessException` without a detail message.
`IllegalAccessException``(String s)` Constructs an `IllegalAccessException` with a detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

IllegalAccessException

`public` **`IllegalAccessException`**`()`

Constructs an `IllegalAccessException` without a detail message.

IllegalAccessException

`public` **`IllegalAccessException`**`(String s)`

Constructs an `IllegalAccessException` with a detail message.
Parameters:
`s` - the detail message.

java.lang
Class IllegalArgumentException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalArgumentException
```

Direct Known Subclasses:
IllegalThreadStateException, NumberFormatException

public class **IllegalArgumentException**
extends RuntimeException

Thrown to indicate that a method has been passed an illegal or inappropriate argument.

Since:
JDK1.0
See Also:
`Thread.setPriority(int)`

Constructor Summary

`IllegalArgumentException``()` Constructs an `IllegalArgumentException` with no detail message.
`IllegalArgumentException``(String s)` Constructs an `IllegalArgumentException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

IllegalArgumentException

`public` **`IllegalArgumentException`**`()`

Constructs an `IllegalArgumentException` with no detail message.

IllegalArgumentException

`public` **`IllegalArgumentException`**`(String s)`

Constructs an `IllegalArgumentException` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class IllegalMonitorStateException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalMonitorStateException
```

public class **IllegalMonitorStateException**
extends RuntimeException

Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.

Since:
JDK1.0
See Also:
`Object.notify()`, `Object.notifyAll()`, `Object.wait()`, `Object.wait(long)`, `Object.wait(long, int)`

Constructor Summary

`IllegalMonitorStateException()` Constructs an `IllegalMonitorStateException` with no detail message.
`IllegalMonitorStateException(String s)` Constructs an `IllegalMonitorStateException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

IllegalMonitorStateException

`public` **`IllegalMonitorStateException`**`()`

Constructs an `IllegalMonitorStateException` with no detail message.

IllegalMonitorStateException

`public` **`IllegalMonitorStateException`**`(String s)`

Constructs an `IllegalMonitorStateException` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class IllegalThreadStateException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalArgumentException
                          |
                          +--java.lang.IllegalThreadStateException
```

public class **IllegalThreadStateException**
extends IllegalArgumentException

Thrown to indicate that a thread is not in an appropriate state for the requested operation. See, for example, the `suspend` and `resume` methods in class `Thread`.

Since:
JDK1.0

Constructor Summary

`IllegalThreadStateException``()` Constructs an `IllegalThreadStateException` with no detail message.
`IllegalThreadStateException``(String s)` Constructs an `IllegalThreadStateException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

IllegalThreadStateException

`public` **`IllegalThreadStateException`**`()`

Constructs an `IllegalThreadStateException` with no detail message.

IllegalThreadStateException

`public` **`IllegalThreadStateException`**`(String s)`

Constructs an `IllegalThreadStateException` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class IndexOutOfBoundsException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IndexOutOfBoundsException
```

Direct Known Subclasses:
ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

public class **IndexOutOfBoundsException**
extends RuntimeException

Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

Applications can subclass this class to indicate similar exceptions.

Since:
JDK1.0

Constructor Summary

`IndexOutOfBoundsException``()` Constructs an `IndexOutOfBoundsException` with no detail message.
`IndexOutOfBoundsException``(String s)` Constructs an `IndexOutOfBoundsException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

IndexOutOfBoundsException

`public` **`IndexOutOfBoundsException`**`()`

Constructs an `IndexOutOfBoundsException` with no detail message.

IndexOutOfBoundsException

`public` **`IndexOutOfBoundsException`**`(String s)`

Constructs an `IndexOutOfBoundsException` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class InstantiationException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.InstantiationException
```

public class **InstantiationException**
extends Exception

Thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface or is an abstract class.

Since:
JDK1.0
See Also:
`Class.newInstance()`

Constructor Summary

`InstantiationException``()` Constructs an `InstantiationException` with no detail message.
`InstantiationException``(String s)` Constructs an `InstantiationException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

InstantiationException

`public` **`InstantiationException`**`()`

Constructs an `InstantiationException` with no detail message.

InstantiationException

`public` **`InstantiationException`**`(String s)`

Constructs an `InstantiationException` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Class Integer

```
java.lang.Object
  |
  +--java.lang.Integer
```

public final class **Integer**
extends Object

The Integer class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Since:
JDK1.0

Field Summary

static int	**MAX_VALUE** The largest value of type `int`.
static int	**MIN_VALUE** The smallest value of type `int`.

Constructor Summary

Integer(int value) Constructs a newly allocated `Integer` object that represents the primitive `int` argument.	

Method Summary

byte	**byteValue**() Returns the value of this Integer as a byte.
boolean	**equals**(Object obj) Compares this object to the specified object.
int	**hashCode**() Returns a hashcode for this Integer.
int	**intValue**() Returns the value of this Integer as an int.
long	**longValue**() Returns the value of this Integer as a `long`.
static int	**parseInt**(String s) Parses the string argument as a signed decimal integer.
static int	**parseInt**(String s, int radix) Parses the string argument as a signed integer in the radix specified by the second argument.
short	**shortValue**() Returns the value of this Integer as a short.
static String	**toBinaryString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 2.
static String	**toHexString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 16.
static String	**toOctalString**(int i) Creates a string representation of the integer argument as an unsigned integer in base 8.
String	**toString**() Returns a String object representing this Integer's value.
static String	**toString**(int i) Returns a new String object representing the specified integer.
static String	**toString**(int i, int radix) Creates a string representation of the first argument in the radix specified by the second argument.
static Integer	**valueOf**(String s) Returns a new Integer object initialized to the value of the specified String.
static Integer	**valueOf**(String s, int radix) Returns a new Integer object initialized to the value of the specified String.

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Field Detail

MIN_VALUE

```
public static final int MIN_VALUE
```

The smallest value of type `int`. The constant value of this field is `-2147483648`.

MAX_VALUE

```
public static final int MAX_VALUE
```

The largest value of type `int`. The constant value of this field is `2147483647`.

Constructor Detail

Integer

```
public Integer(int value)
```

Constructs a newly allocated `Integer` object that represents the primitive `int` argument.

Parameters:

`value` - the value to be represented by the `Integer`.

Method Detail

toString

```
public static String toString(int i,
                              int radix)
```

Creates a string representation of the first argument in the radix specified by the second argument.

If the radix is smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`, then the radix `10` is used instead.

If the first argument is negative, the first element of the result is the ASCII minus character `'-'` (`'\u002d'`). If the first argument is not negative, no sign character appears in the result.

The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the magnitude will not be the zero character. The following ASCII characters are used as digits:


```
0123456789abcdefghijklmnopqrstuvwxyz
```

These are `'\u0030'` through `'\u0039'` and `'\u0061'` through `'\u007a'`. If the `radix` is *N*, then the first *N* of these characters are used as radix-*N* digits in the order shown. Thus, the digits for hexadecimal (radix 16) are `0123456789abcdef`. If uppercase letters are desired, the `String.toUpperCase()` method may be called on the result:

```
Integer.toString(n, 16).toUpperCase()
```

Parameters:

`i` - an integer.
`radix` - the radix.

Returns:

a string representation of the argument in the specified radix.

See Also:

`Character.MAX_RADIX`, `Character.MIN_RADIX`

---

toHexString

```
public static String toHexString(int i)
```

Creates a string representation of the integer argument as an unsigned integer in base 16.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise, it is equal to the argument. This value is converted to a string of ASCII digits in hexadecimal (base 16) with no extra leading `0`s. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The following characters are used as hexadecimal digits:

```
0123456789abcdef
```

These are the characters `'\u0030'` through `'\u0039'` and `'u\0039'` through `'\u0066'`. If the uppercase letters are desired, the `String.toUpperCase()` method may be called on the result:

```
Long.toHexString(n).toUpperCase()
```

Parameters:

`i` - an integer.

Returns:

the string representation of the unsigned integer value represented by the argument in hexadecimal (base 16).

Since:

JDK1.0.2

---

toOctalString

```
public static String toOctalString(int i)
```

Creates a string representation of the integer argument as an unsigned integer in base 8.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise, it is equal to the argument. This value is converted to a string of ASCII digits in octal (base 8) with no extra leading `0`s.

If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The octal digits are:

```
01234567
```

These are the characters `'\u0030'` through `'\u0037'`.

Parameters:

`i` - an integer

Returns:

the string representation of the unsigned integer value represented by the argument in octal (base 8).

Since:

JDK1.0.2

---

toBinaryString

```
public static String toBinaryString(int i)
```

Creates a string representation of the integer argument as an unsigned integer in base 2.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in binary (base 2) with no extra leading `0`s. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The characters `'0'` (`'\u0030'`) and `'1'` (`'\u0031'`) are used as binary digits.

Parameters:

`i` - an integer.

Returns:

the string representation of the unsigned integer value represented by the argument in binary (base 2).

Since:

JDK1.0.2

---

toString

```
public static String toString(int i)
```

Returns a new String object representing the specified integer. The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and radix `10` were given as arguments to the `toString(int, int)` method.

Parameters:

`i` - an integer to be converted.

Returns:

a string representation of the argument in base 10.

---

parseInt

```
public static int parseInt(String s,
                           int radix)
                    throws NumberFormatException
```

Parses the string argument as a signed integer in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether `Character.digit(char, int)` returns a nonnegative value), except that the first character may be an ASCII minus sign `'-'` (`'\u002d'`) to indicate a negative value. The resulting integer value is returned.

An exception of type `NumberFormatException` is thrown if any of the following situations occurs:

- The first argument is `null` or is a string of length zero.
- The radix is either smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`.
- Any character of the string is not a digit of the specified radix, except that the first character may be a minus sign `'-'` (`'\u002d'`) provided that the string is longer than length 1.
- The integer value represented by the string is not a value of type `int`.

Examples:

```
parseInt("0", 10) returns 0
parseInt("473", 10) returns 473
parseInt("-0", 10) returns 0
parseInt("-FF", 16) returns -255
parseInt("1100110", 2) returns 102
parseInt("2147483647", 10) returns 2147483647
parseInt("-2147483648", 10) returns -2147483648
parseInt("2147483648", 10) throws a NumberFormatException
parseInt("99", 8) throws a NumberFormatException
parseInt("Kona", 10) throws a NumberFormatException
parseInt("Kona", 27) returns 411787
```

Parameters:

`s` - the `String` containing the integer.

`radix` - the radix to be used.

Returns:

the integer represented by the string argument in the specified radix.

Throws:

NumberFormatException - if the string does not contain a parsable integer.

---

parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'\u002d'`) to indicate a negative value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

Parameters:
`s` - a string.

Returns:
the integer represented by the argument in decimal.

Throws:
NumberFormatException - if the string does not contain a parsable integer.

---

valueOf

```
public static Integer valueOf(String s,
                              int radix)
                       throws NumberFormatException
```

Returns a new Integer object initialized to the value of the specified String. The first argument is interpreted as representing a signed integer in the radix specified by the second argument, exactly as if the arguments were given to the `parseInt(java.lang.String, int)` method. The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s, radix))
```

Parameters:
`s` - the string to be parsed.
`radix` - the radix of the integer represented by string `s`

Returns:
a newly constructed `Integer` initialized to the value represented by the string argument in the specified radix.

Throws:
NumberFormatException - if the String cannot be parsed as an `int`.

---

valueOf

```
public static Integer valueOf(String s)
                       throws NumberFormatException
```

Returns a new Integer object initialized to the value of the specified String. The argument is interpreted as representing a signed decimal integer, exactly as if the argument were given to the `parseInt(java.lang.String)` method. The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s))
```


Parameters:
`s` - the string to be parsed.

Returns:
a newly constructed `Integer` initialized to the value represented by the string argument.

Throws:
NumberFormatException - if the string cannot be parsed as an integer.

---

byteValue

```
public byte byteValue()
```

Returns the value of this Integer as a byte.

Since:
JDK1.1

---

shortValue

```
public short shortValue()
```

Returns the value of this Integer as a short.

Since:
JDK1.1

---

intValue

```
public int intValue()
```

Returns the value of this Integer as an int.

Returns:
the `int` value represented by this object.

---

longValue

```
public long longValue()
```

Returns the value of this Integer as a `long`.

Returns:
the `int` value represented by this object that is converted to type `long` and the result of the conversion is returned.

---

toString

```
public String toString()
```

Returns a String object representing this Integer's value. The value is converted to signed decimal representation and returned as a string, exactly as if the integer value were given as an argument to the `toString(int)` method.

Overrides:
toString in class Object
Returns:
a string representation of the value of this object in base 10.

---

hashCode

```
public int hashCode()
```

Returns a hashcode for this Integer.
Overrides:
hashCode in class Object
Returns:
a hash code value for this object, equal to the primitive `int` value represented by this `Integer` object.

---

equals

```
public boolean equals(Object obj)
```

Compares this object to the specified object. The result is `true` if and only if the argument is not `null` and is an `Integer` object that contains the same `int` value as this object.
Overrides:
equals in class Object
Parameters:
`obj` - the object to compare with.
Returns:
`true` if the objects are the same; `false` otherwise.

---

java.lang
Class InterruptedException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.InterruptedException
```

---

public class **InterruptedException**
extends Exception

Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the `interrupt` method in class `Thread`.

Since:
JDK1.0
See Also:
`Object.wait()`, `Object.wait(long)`, `Object.wait(long, int)`, `Thread.sleep(long)`

Constructor Summary

`InterruptedException``()` Constructs an `InterruptedException` with no detail message.
`InterruptedException``(String s)` Constructs an `InterruptedException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

InterruptedException

```
public InterruptedException()
```

Constructs an `InterruptedException` with no detail message.

InterruptedException

```
public InterruptedException(String s)
```

Constructs an `InterruptedException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class Long

```
java.lang.Object
  |
  +--java.lang.Long
```

---

public final class **Long**
extends Object

The Long class wraps a value of the primitive type `long` in an object. An object of type `Long` contains a single field whose type is `long`.

In addition, this class provides several methods for converting a `long` to a `String` and a `String` to a `long`, as well as other constants and methods useful when dealing with a `long`.

Since:

JDK1.0

Field Summary

`static long`	**`MAX_VALUE`** The largest value of type `long`.
`static long`	**`MIN_VALUE`** The smallest value of type `long`.

Constructor Summary

`Long``(long value)` Constructs a newly allocated `Long` object that represents the primitive `long` argument.	

Method Summary

boolean	**equals**(Object obj) Compares this object against the specified object.
int	**hashCode**() Computes a hashcode for this Long.
long	**longValue**() Returns the value of this Long as a long value.
String	**toString**() Returns a String object representing this Long's value.
static String	**toString**(long i) Returns a new String object representing the specified integer.
static String	**toString**(long i, int radix) Creates a string representation of the first argument in the radix specified by the second argument.

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Field Detail

MIN_VALUE

```
public static final long MIN_VALUE
```

The smallest value of type long.

MAX_VALUE

```
public static final long MAX_VALUE
```

The largest value of type long.

Constructor Detail

Long

```
public Long(long value)
```


Constructs a newly allocated Long object that represents the primitive long argument.

Parameters:

value - the value to be represented by the Long object.

Method Detail

toString

```
public static String toString(long i,
                              int radix)
```

Creates a string representation of the first argument in the radix specified by the second argument.

If the radix is smaller than Character.MIN_RADIX or larger than Character.MAX_RADIX, then the radix 10 is used instead.

If the first argument is negative, the first element of the result is the ASCII minus sign '-' ('\u002d'. If the first argument is not negative, no sign character appears in the result.

The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character '0' ('\u0030'); otherwise, the first character of the representation of the magnitude will not be the zero character. The following ASCII characters are used as digits:

```
0123456789abcdefghijklmnopqrstuvwxyz
```

These are '\u0030' through '\u0039' and '\u0061' through '\u007a'. If the radix is *N*, then the first *N* of these characters are used as radix-*N* digits in the order shown. Thus, the digits for hexadecimal (radix 16) are 0123456789abcdef. If uppercase letters are desired, the String.toUpperCase() method may be called on the result:

```
Long.toString(n, 16).toUpperCase()
```

Parameters:

i - a long.

radix - the radix.

Returns:

a string representation of the argument in the specified radix.

See Also:

Character.MAX_RADIX, Character.MIN_RADIX

---

toString

```
public static String toString(long i)
```

Returns a new String object representing the specified integer. The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and the radix 10 were given as arguments to the toString(long, int) method that takes two arguments.

Parameters:

i - a long to be converted.

Returns:
a string representation of the argument in base 10.

---

longValue

```
public long longValue()
```

Returns the value of this Long as a long value.
Returns:
the `long` value represented by this object.

---

toString

```
public String toString()
```

Returns a String object representing this Long's value. The long integer value represented by this Long object is converted to signed decimal representation and returned as a string, exactly as if the long value were given as an argument to the `toString(long)` method that takes one argument.
Overrides:
toString in class Object
Returns:
a string representation of this object in base 10.

---

hashCode

```
public int hashCode()
```

Computes a hashcode for this Long. The result is the exclusive OR of the two halves of the primitive `long` value represented by this `Long` object. That is, the hashcode is the value of the expression:

```
(int)(this.longValue()^(this.longValue()>>>32))
```

Overrides:
hashCode in class Object
Returns:
a hash code value for this object.

---

equals

```
public boolean equals(Object obj)
```

Compares this object against the specified object. The result is `true` if and only if the argument is not `null` and is a `Long` object that contains the same `long` value as this object.
Overrides:
equals in class Object
Parameters:
`obj` - the object to compare with.

Returns:
`true` if the objects are the same; `false` otherwise.

---

java.lang
Class Math

```
java.lang.Object
  |
  +--java.lang.Math
```

public final class **Math**
extends Object

The class `Math` contains methods for performing basic numeric operations.

Since:
1.3

Method Summary

static int	**abs**(int a) Returns the absolute value of an `int` value.
static long	**abs**(long a) Returns the absolute value of a `long` value.
static int	**max**(int a, int b) Returns the greater of two `int` values.
static long	**max**(long a, long b) Returns the greater of two `long` values.
static int	**min**(int a, int b) Returns the smaller of two `int` values.
static long	**min**(long a, long b) Returns the smaller of two `long` values.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

abs

```
public static int abs(int a)
```

Returns the absolute value of an `int` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Integer.MIN_VALUE`, the most negative representable `int` value, the result is that same value, which is negative.

Parameters:
`a` - an `int` value.

Returns:
the absolute value of the argument.

See Also:
`Integer.MIN_VALUE`

---

abs

```
public static long abs(long a)
```

Returns the absolute value of a `long` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Long.MIN_VALUE`, the most negative representable `long` value, the result is that same value, which is negative.

Parameters:
`a` - a `long` value.

Returns:
the absolute value of the argument.

See Also:
`Long.MIN_VALUE`

---

max

```
public static int max(int a,
                      int b)
```

Returns the greater of two `int` values. That is, the result is the argument closer to the value of `Integer.MAX_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:
`a` - an `int` value.
`b` - an `int` value.

Returns:
the larger of `a` and `b`.

See Also:
`Long.MAX_VALUE`

---

max

```
public static long max(long a,
                       long b)
```

Returns the greater of two `long` values. That is, the result is the argument closer to the value of `Long.MAX_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

`a` - a `long` value.

`b` - a `long` value.

Returns:

the larger of `a` and `b`.

See Also:

`Long.MAX_VALUE`

---

min

```
public static int min(int a,
                      int b)
```

Returns the smaller of two `int` values. That is, the result the argument closer to the value of `Integer.MIN_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

`a` - an `int` value.

`b` - an `int` value.

Returns:

the smaller of `a` and `b`.

See Also:

`Long.MIN_VALUE`

---

min

```
public static long min(long a,
                       long b)
```

Returns the smaller of two `long` values. That is, the result is the argument closer to the value of `Long.MIN_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

`a` - a `long` value.

`b` - a `long` value.

Returns:

the smaller of `a` and `b`.

See Also:

`Long.MIN_VALUE`

---

java.lang
Class NegativeArraySizeException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.NegativeArraySizeException
```

---

public class **NegativeArraySizeException**
extends RuntimeException

Thrown if an application tries to create an array with negative size.

Since:

JDK1.0

Constructor Summary

`NegativeArraySizeException``()` Constructs a `NegativeArraySizeException` with no detail message.
`NegativeArraySizeException``(String s)` Constructs a `NegativeArraySizeException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

NegativeArraySizeException

```
public NegativeArraySizeException()
```

Constructs a `NegativeArraySizeException` with no detail message.

NegativeArraySizeException

```
public NegativeArraySizeException(String s)
```

Constructs a `NegativeArraySizeException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class NullPointerException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.NullPointerException
```

---

public class **NullPointerException**
extends RuntimeException

Thrown when an application attempts to use `null` in a case where an object is required. These include:

- Calling the instance method of a `null` object.
- Accessing or modifying the field of a `null` object.
- Taking the length of `null` as if it were an array.
- Accessing or modifying the slots of `null` as if it were an array.
- Throwing `null` as if it were a `Throwable` value.

Applications should throw instances of this class to indicate other illegal uses of the `null` object.

Since:

JDK1.0

Constructor Summary	
`NullPointerException``()` Constructs a `NullPointerException` with no detail message.	
`NullPointerException``(String s)` Constructs a `NullPointerException` with the specified detail message.	

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

NullPointerException

```
public NullPointerException()
```

Constructs a `NullPointerException` with no detail message.

NullPointerException

```
public NullPointerException(String s)
```

Constructs a `NullPointerException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class NumberFormatException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalArgumentException
                          |
                          +--java.lang.NumberFormatException
```

---

public class **NumberFormatException**
extends IllegalArgumentException

Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.

Since:

JDK1.0

See Also:

`Integer.toString()`

Constructor Summary

`NumberFormatException()` Constructs a `NumberFormatException` with no detail message.
`NumberFormatException(String s)` Constructs a `NumberFormatException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

NumberFormatException

`public NumberFormatException()`

Constructs a `NumberFormatException` with no detail message.

NumberFormatException

`public NumberFormatException(String s)`

Constructs a `NumberFormatException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class Object

`java.lang.Object`

---

public class **Object**

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

`Class`

Constructor Summary	
`Object()`	

Method Summary

boolean	**equals**(Object obj) Indicates whether some other object is "equal to" this one.
Class	**getClass**() Returns the runtime class of an object.
int	**hashCode**() Returns a hash code value for the object.
void	**notify**() Wakes up a single thread that is waiting on this object's monitor.
void	**notifyAll**() Wakes up all threads that are waiting on this object's monitor.
String	**toString**() Returns a string representation of the object.
void	**wait**() Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	**wait**(long timeout) Causes current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	**wait**(long timeout, int nanos) Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Constructor Detail

Object

```
public Object()
```

Method Detail

getClass

```
public final Class getClass()
```

Returns the runtime class of an object. That Class object is the object that is locked by static synchronized methods of the represented class.

Returns:

the object of type Class that represents the runtime class of the object.

hashCode

```
public int hashCode()
```

Returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable.

The general contract of hashCode is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)

Returns:

a hash code value for this object.

See Also:

equals(java.lang.Object), Hashtable

equals

```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this one.

The equals method implements an equivalence relation:

- It is *reflexive*: for any reference value x, x.equals(x) should return true.
- It is *symmetric*: for any reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- It is *transitive*: for any reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- It is *consistent*: for any reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the object is modified.
- For any non-null reference value x, x.equals(null) should return false.

The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any reference values x and y, this method returns true if and only if x and y refer to the same object (x==y has the value true).

Parameters:
`obj` - the reference object with which to compare.
Returns:
`true` if this object is the same as the obj argument; `false` otherwise.
See Also:
`Boolean.hashCode()`, `Hashtable`

---

toString

```
public String toString()
```

Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

The `toString` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the at-sign character '`@`', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Returns:
a string representation of the object.

---

notify

```
public final void notify()
```

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. A thread waits on an object's monitor by calling one of the `wait` methods.

The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object. The awakened thread will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened thread enjoys no reliable privilege or disadvantage in being the next thread to lock this object.

This method should only be called by a thread that is the owner of this object's monitor. A thread becomes the owner of the object's monitor in one of three ways:

- By executing a synchronized instance method of that object.
- By executing the body of a `synchronized` statement that synchronizes on the object.
- For objects of type `Class`, by executing a synchronized static method of that class.

Only one thread at a time can own an object's monitor.
Throws:
IllegalMonitorStateException - if the current thread is not the owner of this object's monitor.

See Also:
`notifyAll()`, `wait()`

---

notifyAll

```
public final void notifyAll()
```

Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the `wait` methods.

The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object. The awakened threads will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened threads enjoy no reliable privilege or disadvantage in being the next thread to lock this object.

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.
Throws:
IllegalMonitorStateException - if the current thread is not the owner of this object's monitor.
See Also:
`notify()`, `wait()`

---

wait

```
public final void wait(long timeout)
                throws InterruptedException
```

Causes current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor.

This method causes the current thread (call it *T*) to place itself in the wait set for this object and then to relinquish any and all synchronization claims on this object. Thread *T* becomes disabled for thread scheduling purposes and lies dormant until one of four things happens:

- Some other thread invokes the `notify` method for this object and thread *T* happens to be arbitrarily chosen as the thread to be awakened.
- Some other thread invokes the `notifyAll` method for this object.
- The specified amount of real time has elapsed, more or less. If `timeout` is zero, however, then real time is not taken into consideration and the thread simply waits until notified.

The thread *T* is then removed from the wait set for this object and re-enabled for thread scheduling. It then competes in the usual manner with other threads for the right to synchronize on the object; once it has gained control of the object, all its synchronization claims on the object are restored to the status quo ante - that is, to the situation as of the time that the `wait` method was invoked. Thread *T* then returns from the invocation of the `wait` method. Thus, on return from the `wait` method, the synchronization state of the object and of thread `T` is exactly as it was when the `wait` method was invoked.

Note that the `wait` method, as it places the current thread into the wait set for this object, unlocks only this object; any other objects on which the current thread may be synchronized remain locked while the thread waits.

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.

Parameters:

`timeout` - the maximum time to wait in milliseconds.

Throws:

IllegalArgumentException - if the value of timeout is negative.

IllegalMonitorStateException - if the current thread is not the owner of the object's monitor.

InterruptedException - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also:

`notify()`, `notifyAll()`

---

wait

```
public final void wait(long timeout,
                       int nanos)
                throws InterruptedException
```

Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

This method is similar to the `wait` method of one argument, but it allows finer control over the amount of time to wait for a notification before giving up. The amount of real time, measured in nanoseconds, is given by:

```
1000000*millis+nanos
```

In all other respects, this method does the same thing as the method `wait(long)` of one argument. In particular, `wait(0, 0)` means the same thing as `wait(0)`.

The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until either of the following two conditions has occurred:

- Another thread notifies threads waiting on this object's monitor to wake up either through a call to the `notify` method or the `notifyAll` method.
- The timeout period, specified by `timeout` milliseconds plus `nanos` nanoseconds arguments, has elapsed.

The thread then waits until it can re-obtain ownership of the monitor and resumes execution

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.

Parameters:

`timeout` - the maximum time to wait in milliseconds.

`nanos` - additional time, in nanoseconds range 0-999999.

Throws:

IllegalArgumentException - if the value of timeout is negative or the value of nanos is not in the range 0-999999.

IllegalMonitorStateException - if the current thread is not the owner of this object's monitor.

InterruptedException - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

---

wait

```
public final void wait()
                throws InterruptedException
```

Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object. In other word's this method behaves exactly as if it simply performs the call `wait(0)`.

The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until another thread notifies threads waiting on this object's monitor to wake up either through a call to the `notify` method or the `notifyAll` method. The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.

Throws:

IllegalMonitorStateException - if the current thread is not the owner of the object's monitor.

InterruptedException - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also:

`notify()`, `notifyAll()`

---

java.lang
Class OutOfMemoryError

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Error
              |
              +--java.lang.VirtualMachineError
                    |
                    +--java.lang.OutOfMemoryError
```

public class **OutOfMemoryError**
extends VirtualMachineError

Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

Since:
JDK1.0

Constructor Summary

`OutOfMemoryError()` Constructs an `OutOfMemoryError` with no detail message.
`OutOfMemoryError(String s)` Constructs an `OutOfMemoryError` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

OutOfMemoryError

```
public OutOfMemoryError()
```

Constructs an `OutOfMemoryError` with no detail message.

OutOfMemoryError

```
public OutOfMemoryError(String s)
```

Constructs an `OutOfMemoryError` with the specified detail message.
Parameters:
`s` - the detail message.

java.lang
Interface Runnable

All Known Implementing Classes:
Thread

public interface **Runnable**

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, `Runnable` is implemented by class `Thread`. Being active simply means that a thread has been started and has not yet been stopped.

In addition, `Runnable` provides the means for a class to be active while not subclassing `Thread`. A class that implements `Runnable` can run without subclassing `Thread` by instantiating a `Thread` instance and passing itself in as the target. In most cases, the `Runnable` interface should be used if you are only planning to override the `run()` method and no other `Thread` methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

Since:
JDK1.0
See Also:
`Thread`

Method Summary

`void`	**`run`**`()` When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

Method Detail

run

```
public void run()
```

When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

The general contract of the method `run` is that it may take any action whatsoever.
See Also:
`Thread.run()`

java.lang
Class Runtime

```
java.lang.Object
  |
  +--java.lang.Runtime
```

public class **Runtime**
extends Object

Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the `getRuntime` method.

An application cannot create its own instance of this class.

Since:
JDK1.0
See Also:
`getRuntime()`

Method Summary

`void`	**`exit`**`(int status)` Terminates the currently running Java application.
`long`	**`freeMemory`**`()` Returns the amount of free memory in the system.
`void`	**`gc`**`()` Runs the garbage collector.
`static Runtime`	**`getRuntime`**`()` Returns the runtime object associated with the current Java application.
`long`	**`totalMemory`**`()` Returns the total amount of memory in the Java Virtual Machine.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

getRuntime

```
public static Runtime getRuntime()
```

Returns the runtime object associated with the current Java application. Most of the methods of class `Runtime` are instance methods and must be invoked with respect to the current runtime object.
Returns:
the `Runtime` object associated with the current Java application.

exit

```
public void exit(int status)
```

Terminates the currently running Java application. This method never returns normally.

The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.
Parameters:
`status` - exit status.
Since:
JDK1.0

freeMemory

```
public long freeMemory()
```

Returns the amount of free memory in the system. Calling the `gc` method may result in increasing the value returned by `freeMemory.`
Returns:
an approximation to the total amount of memory currently available for future allocated objects, measured in bytes.

totalMemory

```
public long totalMemory()
```

Returns the total amount of memory in the Java Virtual Machine. The value returned by this method may vary over time, depending on the host environment.

Note that the amount of memory required to hold an object of any given type may be implementation-dependent.
Returns:
the total amount of memory currently available for current and future objects, measured in bytes.

gc

```
public void gc()
```

Runs the garbage collector. Calling this method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made its best effort to recycle all discarded objects.

The name `gc` stands for "garbage collector". The Java Virtual Machine performs this recycling process automatically as needed, in a separate thread, even if the `gc` method is not invoked explicitly.

The method `System.gc()` is hte conventional and convenient means of invoking this method.

java.lang
Class RuntimeException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
```

Direct Known Subclasses:
ArithmeticException, ArrayStoreException, ClassCastException, EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IndexOutOfBoundsException, NegativeArraySizeException, NoSuchElementException, NullPointerException, SecurityException

public class **RuntimeException**
extends Exception

`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

A method is not required to declare in its `throws` clause any subclasses of `RuntimeException` that might be thrown during the execution of the method but not caught.

Since:
JDK1.0

Constructor Summary

`RuntimeException()` Constructs a `RuntimeException` with no detail message.
`RuntimeException(String s)` Constructs a `RuntimeException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

RuntimeException

```
public RuntimeException()
```

Constructs a `RuntimeException` with no detail message.

RuntimeException

```
public RuntimeException(String s)
```

Constructs a `RuntimeException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class SecurityException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.SecurityException
```

---

public class **SecurityException**
extends RuntimeException

Thrown by the security manager to indicate a security violation.

Since:

JDK1.0

Constructor Summary

`SecurityException``()` Constructs a `SecurityException` with no detail message.
`SecurityException``(String s)` Constructs a `SecurityException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

SecurityException

```
public SecurityException()
```

Constructs a `SecurityException` with no detail message.

SecurityException

```
public SecurityException(String s)
```

Constructs a `SecurityException` with the specified detail message.

Parameters:

`s` - the detail message.

---

java.lang
Class Short

```
java.lang.Object
  |
  +--java.lang.Short
```

---

public final class **Short**
extends Object

The Short class is the standard wrapper for short values.

Since:

JDK1.1

Field Summary

static short	**MAX_VALUE** The maximum value a Short can have.
static short	**MIN_VALUE** The minimum value a Short can have.

Constructor Summary

Short(short value) Constructs a Short object initialized to the specified short value.	

Method Summary

boolean	**equals**(Object obj) Compares this object to the specified object.
int	**hashCode**() Returns a hashcode for this Short.
short	**shortValue**() Returns the value of this Short as a short.

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, toString, wait, wait, wait

Field Detail

MIN_VALUE

```
public static final short MIN_VALUE
```

The minimum value a Short can have.

MAX_VALUE

```
public static final short MAX_VALUE
```

The maximum value a Short can have.

Constructor Detail

Short

```
public Short(short value)
```

Constructs a Short object initialized to the specified short value.

Parameters:

`value` - the initial value of the Short

Method Detail

shortValue

```
public short shortValue()
```

Returns the value of this Short as a short.

---

hashCode

```
public int hashCode()
```

Returns a hashcode for this Short.

Overrides:

hashCode in class Object

Tags copied from class: Object

Returns:

a hash code value for this object.

See Also:

`Object.equals(java.lang.Object)`, `Hashtable`

---

equals

```
public boolean equals(Object obj)
```

Compares this object to the specified object.

Overrides:

equals in class Object

Parameters:

`obj` - the object to compare with

Returns:

true if the objects are the same; false otherwise.

---

java.lang
Class String

```
java.lang.Object
  |
  +--java.lang.String
```

public final class **String**
extends Object

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the `StringBuffer` class and its `append` method. String conversions are implemented through the method `toString`, defined by `Object` and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

Since:
JDK1.0

See Also:
`Object.toString()`, `StringBuffer`, `StringBuffer.append(boolean)`, `StringBuffer.append(char)`, `StringBuffer.append(char[])`, `StringBuffer.append(char[], int, int)`, `StringBuffer.append(int)`, `StringBuffer.append(long)`, `StringBuffer.append(java.lang.Object)`, `StringBuffer.append(java.lang.String)`, Character encodings

Constructor Summary

`String``()` Initializes a newly created `String` object so that it represents an empty character sequence.
`String``(byte[] bytes)` Construct a new `String` by converting the specified array of bytes using the platform's default character encoding.
`String``(byte[] bytes, int off, int len)` Construct a new `String` by converting the specified subarray of bytes using the platform's default character encoding.
`String``(byte[] bytes, int off, int len, String enc)` Construct a new `String` by converting the specified subarray of bytes using the specified character encoding.
`String``(byte[] bytes, String enc)` Construct a new `String` by converting the specified array of bytes using the specified character encoding.
`String``(char[] value)` Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument.
`String``(char[] value, int offset, int count)` Allocates a new `String` that contains characters from a subarray of the character array argument.
`String``(String value)` Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
`String``(StringBuffer buffer)` Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

Method Summary

`char`	**`charAt`**`(int index)` Returns the character at the specified index.
`int`	**`compareTo`**`(String anotherString)` Compares two strings lexicographically.
`String`	**`concat`**`(String str)` Concatenates the specified string to the end of this string.
`boolean`	**`endsWith`**`(String suffix)` Tests if this string ends with the specified suffix.
`boolean`	**`equals`**`(Object anObject)` Compares this string to the specified object.

boolean	**equalsIgnoreCase**(String anotherString) Compares this String to another String, ignoring case considerations.
byte[]	**getBytes**() Convert this String into bytes according to the platform's default character encoding, storing the result into a new byte array.
byte[]	**getBytes**(String enc) Convert this String into bytes according to the specified character encoding, storing the result into a new byte array.
void	**getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin) Copies characters from this string into the destination character array.
int	**hashCode**() Returns a hashcode for this string.
int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**regionMatches**(boolean ignoreCase, int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
String	**replace**(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
boolean	**startsWith**(String prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(String prefix, int toffset) Tests if this string starts with the specified prefix beginning a specified index.
String	**substring**(int beginIndex) Returns a new string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.

String	**toLowerCase**() Converts all of the characters in this String to lower case.
String	**toString**() This object (which is already a string!) is itself returned.
String	**toUpperCase**() Converts all of the characters in this String to lower case.
static String	**valueOf**(boolean b) Returns the string representation of the boolean argument.
static String	**valueOf**(char c) Returns the string representation of the char argument.
static String	**valueOf**(char[] data) Returns the string representation of the char array argument.
static String	**valueOf**(char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static String	**valueOf**(int i) Returns the string representation of the int argument.
static String	**valueOf**(long l) Returns the string representation of the long argument.
static String	**valueOf**(Object obj) Returns the string representation of the Object argument.

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

String

```
public String()
```

Initializes a newly created String object so that it represents an empty character sequence.

String

```
public String(String value)
```

Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

Parameters:
`value` - a `String`.

String

```
public String(char[] value)
```

Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.
Parameters:
`value` - the initial value of the string.
Throws:
NullPointerException - if `value` is `null`.

String

```
public String(char[] value,
              int offset,
              int count)
```

Allocates a new `String` that contains characters from a subarray of the character array argument. The `offset` argument is the index of the first character of the subarray and the `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.
Parameters:
`value` - array that is the source of characters.
`offset` - the initial offset.
`count` - the length.
Throws:
IndexOutOfBoundsException - if the `offset` and `count` arguments index characters outside the bounds of the `value` array.
NullPointerException - if `value` is `null`.

String

```
public String(byte[] bytes,
              int off,
              int len,
              String enc)
       throws UnsupportedEncodingException
```

Construct a new `String` by converting the specified subarray of bytes using the specified character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the subarray.
Parameters:
`bytes` - The bytes to be converted into characters
`offset` - Index of the first byte to convert
`length` - Number of bytes to convert
`enc` - The name of a character encoding

Throws:
UnsupportedEncodingException - If the named encoding is not supported
Since:
JDK1.1

String

```
public String(byte[] bytes,
              String enc)
       throws UnsupportedEncodingException
```

Construct a new `String` by converting the specified array of bytes using the specified character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the byte array.
Parameters:
`bytes` - The bytes to be converted into characters
`enc` - The name of a supported character encoding
Throws:
UnsupportedEncodingException - If the named encoding is not supported
Since:
JDK1.1

String

```
public String(byte[] bytes,
              int off,
              int len)
```

Construct a new `String` by converting the specified subarray of bytes using the platform's default character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the subarray.
Parameters:
`bytes` - The bytes to be converted into characters
`offset` - Index of the first byte to convert
`length` - Number of bytes to convert
Since:
JDK1.1

String

```
public String(byte[] bytes)
```

Construct a new `String` by converting the specified array of bytes using the platform's default character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the byte array.
Parameters:
`bytes` - The bytes to be converted into characters
Since:
JDK1.1

String

```
public String(StringBuffer buffer)
```

Allocates a new string that contains the sequence of characters currently contained in the string buffer argument. The contents of the string buffer are copied; subsequent modification of the string buffer does not affect the newly created string.

Parameters:

`buffer` - a `StringBuffer`.

Throws:

NullPointerException - If `buffer` is `null`.

Method Detail

length

```
public int length()
```

Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

Returns:

the length of the sequence of characters represented by this object.

---

charAt

```
public char charAt(int index)
```

Returns the character at the specified index. An index ranges from `0` to `length() - 1`. The first character of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Parameters:

`index` - the index of the character.

Returns:

the character at the specified index of this string. The first character is at index `0`.

Throws:

IndexOutOfBoundsException - if the `index` argument is negative or not less than the length of this string.

---

getChars

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char[] dst,
                     int dstBegin)
```

Copies characters from this string into the destination character array.

The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1` (thus the total number of characters to be copied is `srcEnd-srcBegin`). The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index:


```
dstbegin + (srcEnd-srcBegin) - 1
```

Parameters:

`srcBegin` - index of the first character in the string to copy.
`srcEnd` - index after the last character in the string to copy.
`dst` - the destination array.
`dstBegin` - the start offset in the destination array.

Throws:

IndexOutOfBoundsException - If any of the following is true:

- `srcBegin` is negative.
- `srcBegin` is greater than `srcEnd`
- `srcEnd` is greater than the length of this string
- `dstBegin` is negative
- `dstBegin+(srcEnd-srcBegin)` is larger than `dst.length`

NullPointerException - if `dst` is `null`

---

getBytes

```
public byte[] getBytes(String enc)
              throws UnsupportedEncodingException
```

Convert this `String` into bytes according to the specified character encoding, storing the result into a new byte array.

Parameters:

`enc` - A character-encoding name

Returns:

The resultant byte array

Throws:

UnsupportedEncodingException - If the named encoding is not supported

Since:

JDK1.1

---

getBytes

```
public byte[] getBytes()
```

Convert this `String` into bytes according to the platform's default character encoding, storing the result into a new byte array.

Returns:

the resultant byte array.

Since:

JDK1.1

---

equals

```
public boolean equals(Object anObject)
```

Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

Overrides:
equals in class Object
Parameters:
`anObject` - the object to compare this `String` against.
Returns:
`true` if the `String` are equal; `false` otherwise.
See Also:
`compareTo(java.lang.String)`, `equalsIgnoreCase(java.lang.String)`

---

equalsIgnoreCase

```
public boolean equalsIgnoreCase(String anotherString)
```

Compares this `String` to another `String`, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

Two characters `c1` and `c2` are considered the same, ignoring case if at least one of the following is true:

- The two characters are the same (as compared by the `==` operator).
- Applying the method `Character.toUpperCase(char)` to each character produces the same result.
- Applying the method `Character.toLowerCase(char)` to each character produces the same result.

Parameters:
`anotherString` - the `String` to compare this `String` against.
Returns:
`true` if the argument is not `null` and the `String`s are equal, ignoring case; `false` otherwise.
See Also:
`equals(Object)`, `Character.toLowerCase(char)`, `Character.toUpperCase(char)`

---

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case,

`compareTo` returns the difference of the two character values at position `k` in the two string -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

Parameters:
`anotherString` - the `String` to be compared.
Returns:
the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than `0` if this string is lexicographically greater than the string argument.
Throws:
NullPointerException - if `anotherString` is `null`.

---

regionMatches

```
public boolean regionMatches(boolean ignoreCase,
                             int toffset,
                             String other,
                             int ooffset,
                             int len)
```

Tests if two string regions are equal.

A substring of this `String` object is compared to a substring of the argument `other`. The result is `true` if these substrings represent character sequences that are the same, ignoring case if and only if `ignoreCase` is true. The substring of this `String` object to be compared begins at index `toffset` and has length `len`. The substring of `other` to be compared begins at index `ooffset` and has length `len`. The result is `false` if and only if at least one of the following is true:

- `toffset` is negative.
- `ooffset` is negative.
- `toffset+len` is greater than the length of this `String` object.
- `ooffset+len` is greater than the length of the other argument.
- There is some nonnegative integer *k* less than `len` such that:

```
this.charAt(toffset+k) != other.charAt(ooffset+k)
```

- `ignoreCase` is `true` and there is some nonnegative integer *k* less than `len` such that:

```
Character.toLowerCase(this.charAt(toffset+k)) !=
Character.toLowerCase(other.charAt(ooffset+k))
```

and:

```
Character.toUpperCase(this.charAt(toffset+k)) !=
        Character.toUpperCase(other.charAt(ooffset+k))
```

Parameters:

`ignoreCase` - if `true`, ignore case when comparing characters.
`toffset` - the starting offset of the subregion in this string.
`other` - the string argument.
`ooffset` - the starting offset of the subregion in the string argument.
`len` - the number of characters to compare.

Returns:

`true` if the specified subregion of this string matches the specified subregion of the string argument; `false` otherwise. Whether the matching is exact or case insensitive depends on the `ignoreCase` argument.

---

startsWith

```
public boolean startsWith(String prefix,
                          int toffset)
```

Tests if this string starts with the specified prefix beginning a specified index.

Parameters:

`prefix` - the prefix.
`toffset` - where to begin looking in the string.

Returns:

`true` if the character sequence represented by the argument is a prefix of the substring of this object starting at index `toffset`; `false` otherwise. The result is `false` if `toffset` is negative or greater than the length of this `String` object; otherwise the result is the same as the result of the expression

```
this.subString(toffset).startsWith(prefix)
```

Throws:

NullPointerException - if `prefix` is `null`.

---

startsWith

```
public boolean startsWith(String prefix)
```

Tests if this string starts with the specified prefix.

Parameters:

`prefix` - the prefix.

Returns:

`true` if the character sequence represented by the argument is a prefix of the character sequence represented by this string; `false` otherwise. Note also that `true` will be returned if the argument is an empty string or is equal to this `String` object as determined by the `equals(Object)` method.

Throws:

NullPointerException - if `prefix` is `null`.

Since:

JDK1. 0

---

endsWith

```
public boolean endsWith(String suffix)
```

Tests if this string ends with the specified suffix.

Parameters:

`suffix` - the suffix.

Returns:

`true` if the character sequence represented by the argument is a suffix of the character sequence represented by this object; `false` otherwise. Note that the result will be `true` if the argument is the empty string or is equal to this `String` object as determined by the `equals(Object)` method.

Throws:

NullPointerException - if `suffix` is `null`.

---

hashCode

```
public int hashCode()
```

Returns a hashcode for this string. The hashcode for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

Overrides:

hashCode in class Object

Returns:

a hash code value for this object.

---

indexOf

```
public int indexOf(int ch)
```

Returns the index within this string of the first occurrence of the specified character. If a character with value `ch` occurs in the character sequence represented by this `String` object, then the index of the first such occurrence is returned -- that is, the smallest value *k* such that:

```
this.charAt(k) == ch
```

is `true`. If no such character occurs in this string, then `-1` is returned.

Parameters:

`ch` - a character.

Returns:

the index of the first occurrence of the character in the character sequence represented by this object, or `-1` if the character does not occur.

---

indexOf

```
public int indexOf(int ch,
                   int fromIndex)
```

Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

If a character with value `ch` occurs in the character sequence represented by this `String` object at an index no smaller than `fromIndex`, then the index of the first such occurrence is returned--that is, the smallest value *k* such that:

```
(this.charAt(k) == ch) && (k >= fromIndex)
```

is true. If no such character occurs in this string at or after position `fromIndex`, then `-1` is returned.

There is no restriction on the value of `fromIndex`. If it is negative, it has the same effect as if it were zero: this entire string may be searched. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: `-1` is returned.

Parameters:

`ch` - a character.

`fromIndex` - the index to start the search from.

Returns:

the index of the first occurrence of the character in the character sequence represented by this object that is greater than or equal to `fromIndex`, or `-1` if the character does not occur.

---

lastIndexOf

```
public int lastIndexOf(int ch)
```

Returns the index within this string of the last occurrence of the specified character. That is, the index returned is the largest value *k* such that:

```
this.charAt(k) == ch
```

is true. The String is searched backwards starting at the last character.

Parameters:

`ch` - a character.

Returns:

the index of the last occurrence of the character in the character sequence represented by this object, or `-1` if the character does not occur.

---

lastIndexOf

```
public int lastIndexOf(int ch,
                       int fromIndex)
```

Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index. That is, the index returned is the largest value *k* such that:


```
this.charAt(k) == ch) && (k <= fromIndex)
```

is true.

Parameters:

`ch` - a character.

`fromIndex` - the index to start the search from. There is no restriction on the value of `fromIndex`. If it is greater than or equal to the length of this string, it has the same effect as if it were equal to one less than the length of this string: this entire string may be searched. If it is negative, it has the same effect as if it were -1: `-1` is returned.

Returns:

the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to `fromIndex`, or `-1` if the character does not occur before that point.

---

substring

```
public String substring(int beginIndex)
```

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

Parameters:

`beginIndex` - the beginning index, inclusive.

Returns:

the specified substring.

Throws:

IndexOutOfBoundsException - if `beginIndex` is negative or larger than the length of this `String` object.

---

substring

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:

`beginIndex` - the beginning index, inclusive.

`endIndex` - the ending index, exclusive.

Returns:
the specified substring.
Throws:
IndexOutOfBoundsException - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

---

concat

```
public String concat(String str)
```

Concatenates the specified string to the end of this string.

If the length of the argument string is `0`, then this `String` object is returned. Otherwise, a new `String` object is created, representing a character sequence that is the concatenation of the character sequence represented by this `String` object and the character sequence represented by the argument string.

Examples:

```
"cares".concat("s") returns "caress"
"to".concat("get").concat("her") returns "together"
```

Parameters:
`str` - the `String` that is concatenated to the end of this `String`.
Returns:
a string that represents the concatenation of this object's characters followed by the string argument's characters.
Throws:
NullPointerException - if `str` is `null`.

---

replace

```
public String replace(char oldChar,
                      char newChar)
```

Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

If the character `oldChar` does not occur in the character sequence represented by this `String` object, then a reference to this `String` object is returned. Otherwise, a new `String` object is created that represents a character sequence identical to the character sequence represented by this `String` object, except that every occurrence of `oldChar` is replaced by an occurrence of `newChar`.

Examples:

```
"mesquite in your cellar".replace('e', 'o')
        returns "mosquito in your collar"
"the war of baronets".replace('r', 'y')
        returns "the way of bayonets"
"sparring with a purple porpoise".replace('p', 't')
        returns "starring with a turtle tortoise"
"JonL".replace('q', 'x') returns "JonL" (no change)
```


Parameters:
`oldChar` - the old character.
`newChar` - the new character.
Returns:
a string derived from this string by replacing every occurrence of `oldChar` with `newChar`.

---

toLowerCase

```
public String toLowerCase()
```

Converts all of the characters in this String to lower case.

Note - This only works for ISO-Latin-1
Returns:
the String, converted to lowercase.
See Also:
`Character.toLowerCase(char)`, `toUpperCase()`

---

toUpperCase

```
public String toUpperCase()
```

Converts all of the characters in this String to lower case.

Note - This only works for ISO-Latin-1
Returns:
the String, converted to lowercase.
See Also:
`Character.toLowerCase(char)`, `toUpperCase()`

---

toString

```
public String toString()
```

This object (which is already a string!) is itself returned.
Overrides:
toString in class Object
Returns:
the string itself.

---

toCharArray

```
public char[] toCharArray()
```

Converts this string to a new character array.
Returns:
a newly allocated character array whose length is the length of this string and whose contents are initialized to contain the character sequence represented by this string.

valueOf

```
public static String valueOf(Object obj)
```

Returns the string representation of the `Object` argument.
Parameters:
`obj` - an `Object`.
Returns:
if the argument is `null`, then a string equal to `"null"`; otherwise, the value of `obj.toString()` is returned.
See Also:
`Object.toString()`

valueOf

```
public static String valueOf(char[] data)
```

Returns the string representation of the `char` array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.
Parameters:
`data` - a `char` array.
Returns:
a newly allocated string representing the same sequence of characters contained in the character array argument.

valueOf

```
public static String valueOf(char[] data,
                             int offset,
                             int count)
```

Returns the string representation of a specific subarray of the `char` array argument.

The `offset` argument is the index of the first character of the subarray. The `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.
Parameters:
`data` - the character array.
`offset` - the initial offset into the value of the `String`.
`count` - the length of the value of the `String`.
Returns:
a newly allocated string representing the sequence of characters contained in the subarray of the character array argument.
Throws:
NullPointerException - if `data` is `null`.
IndexOutOfBoundsException - if `offset` is negative, or `count` is negative, or `offset+count` is larger than `data.length`.

valueOf

```
public static String valueOf(boolean b)
```

Returns the string representation of the `boolean` argument.
Parameters:
`b` - a `boolean`.
Returns:
if the argument is `true`, a string equal to `"true"` is returned; otherwise, a string equal to `"false"` is returned.

valueOf

```
public static String valueOf(char c)
```

Returns the string representation of the `char` argument.
Parameters:
`c` - a `char`.
Returns:
a newly allocated string of length `1` containing as its single character the argument `c`.

valueOf

```
public static String valueOf(int i)
```

Returns the string representation of the `int` argument.

The representation is exactly the one returned by the `Integer.toString` method of one argument.
Parameters:
`i` - an `int`.
Returns:
a newly allocated string containing a string representation of the `int` argument.
See Also:
`Integer.toString(int, int)`

valueOf

```
public static String valueOf(long l)
```

Returns the string representation of the `long` argument.

The representation is exactly the one returned by the `Long.toString` method of one argument.
Parameters:
`l` - a `long`.
Returns:
a newly allocated string containing a string representation of the `long` argument.
See Also:
`Long.toString(long)`

java.lang
Class StringBuffer

```
java.lang.Object
  |
  +--java.lang.StringBuffer
```

public final class **StringBuffer**
extends Object

A string buffer implements a mutable sequence of characters. A string buffer is like a `String`, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

String buffers are safe for use by multiple threads. The methods are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.

String buffers are used by the compiler to implement the binary string concatenation operator +. For example, the code:

```
x = "a" + 4 + "c"
```

is compiled to the equivalent of:

```
x = new StringBuffer().append("a").append(4).append("c")
                      .toString()
```

which creates a new string buffer (initially empty), appends the string representation of each operand to the string buffer in turn, and then converts the contents of the string buffer to a string. Overall, this avoids creating many temporary strings.

The principal operations on a `StringBuffer` are the `append` and `insert` methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string buffer. The `append` method always adds these characters at the end of the buffer; the `insert` method adds the characters at a specified point.

For example, if `z` refers to a string buffer object whose current contents are "`start`", then the method call `z.append("le")` would cause the string buffer to contain "`startle`", whereas `z.insert(4, "le")` would alter the string buffer to contain "`starlet`".

In general, if sb refers to an instance of a `StringBuffer`, then `sb.append(x)` has the same effect as `sb.insert(sb.length(), x)`.

Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger.

Since:
JDK1.0

See Also:
`ByteArrayOutputStream`, `String`

Constructor Summary

`StringBuffer``()` Constructs a string buffer with no characters in it and an initial capacity of 16 characters.
`StringBuffer``(int length)` Constructs a string buffer with no characters in it and an initial capacity specified by the `length` argument.
`StringBuffer``(String str)` Constructs a string buffer so that it represents the same sequence of characters as the string argument; in other words, the initial contents of the string buffer is a copy of the argument string.

Method Summary

`StringBuffer`	**`append`**`(boolean b)` Appends the string representation of the `boolean` argument to the string buffer.
`StringBuffer`	**`append`**`(char c)` Appends the string representation of the `char` argument to this string buffer.
`StringBuffer`	**`append`**`(char[] str)` Appends the string representation of the `char` array argument to this string buffer.
`StringBuffer`	**`append`**`(char[] str, int offset, int len)` Appends the string representation of a subarray of the `char` array argument to this string buffer.
`StringBuffer`	**`append`**`(int i)` Appends the string representation of the `int` argument to this string buffer.
`StringBuffer`	**`append`**`(long l)` Appends the string representation of the `long` argument to this string buffer.
`StringBuffer`	**`append`**`(Object obj)` Appends the string representation of the `Object` argument to this string buffer.
`StringBuffer`	**`append`**`(String str)` Appends the string to this string buffer.
`int`	**`capacity`**`()` Returns the current capacity of the String buffer.
`char`	**`charAt`**`(int index)` The specified character of the sequence currently represented by the string buffer, as indicated by the `index` argument, is returned.

`StringBuffer`	**`delete`**`(int start, int end)` Removes the characters in a substring of this `StringBuffer`.
`StringBuffer`	**`deleteCharAt`**`(int index)` Removes the character at the specified position in this `StringBuffer` (shortening the `StringBuffer` by one character).
`void`	**`ensureCapacity`**`(int minimumCapacity)` Ensures that the capacity of the buffer is at least equal to the specified minimum.
`void`	**`getChars`**`(int srcBegin, int srcEnd, char[] dst, int dstBegin)` Characters are copied from this string buffer into the destination character array `dst`.
`StringBuffer`	**`insert`**`(int offset, boolean b)` Inserts the string representation of the `boolean` argument into this string buffer.
`StringBuffer`	**`insert`**`(int offset, char c)` Inserts the string representation of the `char` argument into this string buffer.
`StringBuffer`	**`insert`**`(int offset, char[] str)` Inserts the string representation of the `char` array argument into this string buffer.
`StringBuffer`	**`insert`**`(int offset, int i)` Inserts the string representation of the second `int` argument into this string buffer.
`StringBuffer`	**`insert`**`(int offset, long l)` Inserts the string representation of the `long` argument into this string buffer.
`StringBuffer`	**`insert`**`(int offset, Object obj)` Inserts the string representation of the `Object` argument into this string buffer.
`StringBuffer`	**`insert`**`(int offset, String str)` Inserts the string into this string buffer.
`int`	**`length`**`()` Returns the length (character count) of this string buffer.
`StringBuffer`	**`reverse`**`()` The character sequence contained in this string buffer is replaced by the reverse of the sequence.
`void`	**`setCharAt`**`(int index, char ch)` The character at the specified index of this string buffer is set to `ch`.
`void`	**`setLength`**`(int newLength)` Sets the length of this String buffer.
`String`	**`toString`**`()` Converts to a string representing the data in this string buffer.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

StringBuffer

```
public StringBuffer()
```

Constructs a string buffer with no characters in it and an initial capacity of 16 characters.

StringBuffer

```
public StringBuffer(int length)
```

Constructs a string buffer with no characters in it and an initial capacity specified by the `length` argument.

Parameters:

`length` - the initial capacity.

Throws:

NegativeArraySizeException - if the `length` argument is less than `0`.

StringBuffer

```
public StringBuffer(String str)
```

Constructs a string buffer so that it represents the same sequence of characters as the string argument; in other words, the initial contents of the string buffer is a copy of the argument string. The initial capacity of the string buffer is `16` plus the length of the string argument.

Parameters:

`str` - the initial contents of the buffer.

Method Detail

length

```
public int length()
```

Returns the length (character count) of this string buffer.

Returns:

the length of the sequence of characters currently represented by this string buffer.

---

capacity

```
public int capacity()
```

Returns the current capacity of the String buffer. The capacity is the amount of storage available for newly inserted characters; beyond which an allocation will occur.

Returns:

the current capacity of this string buffer.

---

ensureCapacity

```
public void ensureCapacity(int minimumCapacity)
```

Ensures that the capacity of the buffer is at least equal to the specified minimum. If the current capacity of this string buffer is less than the argument, then a new internal buffer is allocated with greater capacity. The new capacity is the larger of:

- The `minimumCapacity` argument.
- Twice the old capacity, plus `2`.

If the `minimumCapacity` argument is nonpositive, this method takes no action and simply returns.

Parameters:

`minimumCapacity` - the minimum desired capacity.

---

setLength

```
public void setLength(int newLength)
```

Sets the length of this String buffer. This string buffer is altered to represent a new character sequence whose length is specified by the argument. For every nonnegative index *k* less than `newLength`, the character at index *k* in the new character sequence is the same as the character at index *k* in the old sequence if *k* is less than the length of the old character sequence; otherwise, it is the null character `'\u0000'`. In other words, if the `newLength` argument is less than the current length of the string buffer, the string buffer is truncated to contain exactly the number of characters given by the `newLength` argument.

If the `newLength` argument is greater than or equal to the current length, sufficient null characters (`'\u0000'`) are appended to the string buffer so that length becomes the `newLength` argument.

The `newLength` argument must be greater than or equal to `0`.

Parameters:

`newLength` - the new length of the buffer.

Throws:

IndexOutOfBoundsException - if the `newLength` argument is negative.

See Also:

`length()`

---

charAt

```
public char charAt(int index)
```

The specified character of the sequence currently represented by the string buffer, as indicated by the `index` argument, is returned. The first character of a string buffer is at index `0`, the next at index `1`, and so on, for array indexing.

The index argument must be greater than or equal to `0`, and less than the length of this string buffer.

Parameters:

`index` - the index of the desired character.

Returns:

the character at the specified index of this string buffer.

Throws:

IndexOutOfBoundsException - if `index` is negative or greater than or equal to `length()`.

See Also:

`length()`

---

getChars

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char[] dst,
                     int dstBegin)
```

Characters are copied from this string buffer into the destination character array `dst`. The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1`. The total number of characters to be copied is `srcEnd-srcBegin`. The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index:

```
dstbegin + (srcEnd-srcBegin) - 1
```

Parameters:

`srcBegin` - start copying at this offset in the string buffer.
`srcEnd` - stop copying at this offset in the string buffer.
`dst` - the array to copy the data into.
`dstBegin` - offset into `dst`.

Throws:

NullPointerException - if `dst` is `null`.
IndexOutOfBoundsException - if any of the following is true:

- `srcBegin` is negative
- `dstBegin` is negative
- the `srcBegin` argument is greater than the `srcEnd` argument.
- `srcEnd` is greater than `this.length()`, the current length of this string buffer.
- `dstBegin+srcEnd-srcBegin` is greater than `dst.length`

---

setCharAt

```
public void setCharAt(int index,
                      char ch)
```

The character at the specified index of this string buffer is set to `ch`. The string buffer is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character `ch` at position `index`.

The offset argument must be greater than or equal to `0`, and less than the length of this string buffer.

Parameters:

`index` - the index of the character to modify.
`ch` - the new character.

Throws:

IndexOutOfBoundsException - if `index` is negative or greater than or equal to `length()`.

See Also:

`length()`

---

append

```
public StringBuffer append(Object obj)
```

Appends the string representation of the `Object` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:

`obj` - an `Object`.

Returns:

a reference to this `StringBuffer` object.

See Also:

`String.valueOf(java.lang.Object)`, `append(java.lang.String)`

---

append

```
public StringBuffer append(String str)
```

Appends the string to this string buffer.

The characters of the `String` argument are appended, in order, to the contents of this string buffer, increasing the length of this string buffer by the length of the argument. If `str` is `null`, then the four characters `"null"` are appended to this string buffer.

Let *n* be the length of the old character sequence, the one contained in the string buffer just prior to execution of the `append` method. Then the character at index *k* in the new character sequence is equal to the character at index *k* in the old character sequence, if *k* is less than *n*; otherwise, it is equal to the character at index *k*-*n* in the argument `str`.

Parameters:

`str` - a string.

Returns:
a reference to this `StringBuffer`.

---

append

```
public StringBuffer append(char[] str)
```

Appends the string representation of the `char` array argument to this string buffer.

The characters of the array argument are appended, in order, to the contents of this string buffer. The length of this string buffer increases by the length of the argument.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char[])` and the characters of that string were then `appended` to this `StringBuffer` object.

Parameters:
`str` - the characters to be appended.

Returns:
a reference to this `StringBuffer` object.

---

append

```
public StringBuffer append(char[] str,
                           int offset,
                           int len)
```

Appends the string representation of a subarray of the `char` array argument to this string buffer.

Characters of the character array `str`, starting at index `offset`, are appended, in order, to the contents of this string buffer. The length of this string buffer increases by the value of `len`.

The overall effect is exactly as if the arguments were converted to a string by the method `String.valueOf(char[],int,int)` and the characters of that string were then `appended` to this `StringBuffer` object.

Parameters:
`str` - the characters to be appended.
`offset` - the index of the first character to append.
`len` - the number of characters to append.

Returns:
a reference to this `StringBuffer` object.

---

append

```
public StringBuffer append(boolean b)
```

Appends the string representation of the `boolean` argument to the string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`b` - a `boolean`.

Returns:
a reference to this `StringBuffer`.

See Also:
`String.valueOf(boolean)`, `append(java.lang.String)`

---

append

```
public StringBuffer append(char c)
```

Appends the string representation of the `char` argument to this string buffer.

The argument is appended to the contents of this string buffer. The length of this string buffer increases by `1`.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char)` and the character in that string were then `appended` to this `StringBuffer` object.

Parameters:
`c` - a `char`.

Returns:
a reference to this `StringBuffer` object.

---

append

```
public StringBuffer append(int i)
```

Appends the string representation of the `int` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`i` - an `int`.

Returns:
a reference to this `StringBuffer` object.

See Also:
`String.valueOf(int)`, `append(java.lang.String)`

---

append

```
public StringBuffer append(long l)
```

Appends the string representation of the `long` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`l` - a `long`.

Returns:
a reference to this `StringBuffer` object.
See Also:
`String.valueOf(long)`, `append(java.lang.String)`

---

delete

```
public StringBuffer delete(int start,
                           int end)
```

Removes the characters in a substring of this `StringBuffer`. The substring begins at the specified `start` and extends to the character at index `end - 1` or to the end of the `StringBuffer` if no such character exists. If `start` is equal to `end`, no changes are made.
Parameters:
`start` - The beginning index, inclusive.
`end` - The ending index, exclusive.
Returns:
This string buffer.
Throws:
StringIndexOutOfBoundsException - if `start` is negative, greater than `length()`, or greater than `end`.
Since:
1.2

---

deleteCharAt

```
public StringBuffer deleteCharAt(int index)
```

Removes the character at the specified position in this `StringBuffer` (shortening the `StringBuffer` by one character).
Parameters:
`index` - Index of character to remove
Returns:
This string buffer.
Throws:
StringIndexOutOfBoundsException - if the `index` is negative or greater than or equal to `length()`.
Since:
1.2

---

insert

```
public StringBuffer insert(int offset,
                           Object obj)
```

Inserts the string representation of the `Object` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.
Parameters:
`offset` - the offset.
`obj` - an `Object`.
Returns:
a reference to this `StringBuffer` object.
Throws:
StringIndexOutOfBoundsException - if the offset is invalid.
See Also:
`String.valueOf(java.lang.Object)`, `insert(int, java.lang.String)`, `length()`

---

insert

```
public StringBuffer insert(int offset,
                           String str)
```

Inserts the string into this string buffer.

The characters of the `String` argument are inserted, in order, into this string buffer at the indicated offset, moving up any characters originally above that position and increasing the length of this string buffer by the length of the argument. If `str` is `null`, then the four characters `"null"` are inserted into this string buffer.

The character at index *k* in the new character sequence is equal to:

- the character at index *k* in the old character sequence, if *k* is less than `offset`
- the character at index *k*`-offset` in the argument `str`, if *k* is not less than `offset` but is less than `offset+str.length()`
- the character at index *k*`-str.length()` in the old character sequence, if *k* is not less than `offset+str.length()`

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.
Parameters:
`offset` - the offset.
`str` - a string.
Returns:
a reference to this `StringBuffer` object.
Throws:
StringIndexOutOfBoundsException - if the offset is invalid.
See Also:
`length()`

---

insert

```
public StringBuffer insert(int offset,
                           char[] str)
```

Inserts the string representation of the `char` array argument into this string buffer.

The characters of the array argument are inserted into the contents of this string buffer at the position indicated by `offset`. The length of this string buffer increases by the length of the argument.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char[])` and the characters of that string were then `inserted` into this `StringBuffer` object at the position indicated by `offset`.

Parameters:

`offset` - the offset.

`str` - a character array.

Returns:

a reference to this `StringBuffer` object.

Throws:

StringIndexOutOfBoundsException - if the offset is invalid.

---

insert

```
public StringBuffer insert(int offset,
                           boolean b)
```

Inserts the string representation of the `boolean` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:

`offset` - the offset.

`b` - a `boolean`.

Returns:

a reference to this `StringBuffer` object.

Throws:

StringIndexOutOfBoundsException - if the offset is invalid.

See Also:

`String.valueOf(boolean)`, `insert(int, java.lang.String)`, `length()`

---

insert

```
public StringBuffer insert(int offset,
                           char c)
```

Inserts the string representation of the `char` argument into this string buffer.

The second argument is inserted into the contents of this string buffer at the position indicated by `offset`. The length of this string buffer increases by one.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char)` and the character in that string were then `inserted` into this `StringBuffer` object at the position indicated by `offset`.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:

`offset` - the offset.

`c` - a `char`.

Returns:

a reference to this `StringBuffer` object.

Throws:

IndexOutOfBoundsException - if the offset is invalid.

See Also:

`length()`

---

insert

```
public StringBuffer insert(int offset,
                           int i)
```

Inserts the string representation of the second `int` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:

`offset` - the offset.

`i` - an `int`.

Returns:

a reference to this `StringBuffer` object.

Throws:

StringIndexOutOfBoundsException - if the offset is invalid.

See Also:

`String.valueOf(int)`, `insert(int, java.lang.String)`, `length()`

---

insert

```
public StringBuffer insert(int offset,
                           long l)
```

Inserts the string representation of the `long` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the position indicated by `offset`.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:

`offset` - the offset.

`l` - a `long`.

Returns:

a reference to this `StringBuffer` object.

Throws:

StringIndexOutOfBoundsException - if the offset is invalid.

See Also:

`String.valueOf(long)`, `insert(int, java.lang.String)`, `length()`

---

reverse

```
public StringBuffer reverse()
```

The character sequence contained in this string buffer is replaced by the reverse of the sequence.

Let *n* be the length of the old character sequence, the one contained in the string buffer just prior to execution of the `reverse` method. Then the character at index *k* in the new character sequence is equal to the character at index *n-k-1* in the old character sequence.

Returns:

a reference to this

---

toString

```
public String toString()
```

Converts to a string representing the data in this string buffer. A new `String` object is allocated and initialized to contain the character sequence currently represented by this string buffer. This `String` is then returned. Subsequent changes to the string buffer do not affect the contents of the `String`.

Implementation advice: This method can be coded so as to create a new `String` object without allocating new memory to hold a copy of the character sequence. Instead, the string can share the memory used by the string buffer. Any subsequent operation that alters the content or capacity of the string buffer must then make a copy of the internal buffer at that time. This strategy is effective for reducing the amount of memory allocated by a string concatenation operation when it is implemented using a string buffer.

Overrides:

toString in class Object

Returns:

a string representation of the string buffer.

---

java.lang
Class StringIndexOutOfBoundsException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IndexOutOfBoundsException
                          |
                          +--java.lang.StringIndexOutOfBoundsException
```

---

public class **StringIndexOutOfBoundsException**
extends IndexOutOfBoundsException

Thrown by the `charAt` method in class `String` and by other `String` methods to indicate that an index is either negative or greater than or equal to the size of the string.

Since:

JDK1.0

See Also:

`String.charAt(int)`

Constructor Summary

`StringIndexOutOfBoundsException``()` Constructs a `StringIndexOutOfBoundsException` with no detail message.
`StringIndexOutOfBoundsException``(int index)` Constructs a new `StringIndexOutOfBoundsException` class with an argument indicating the illegal index.
`StringIndexOutOfBoundsException``(String s)` Constructs a `StringIndexOutOfBoundsException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

StringIndexOutOfBoundsException

```
public StringIndexOutOfBoundsException()
```

Constructs a `StringIndexOutOfBoundsException` with no detail message.
Since:
JDK1.0.

StringIndexOutOfBoundsException

```
public StringIndexOutOfBoundsException(String s)
```

Constructs a `StringIndexOutOfBoundsException` with the specified detail message.
Parameters:
`s` - the detail message.

StringIndexOutOfBoundsException

```
public StringIndexOutOfBoundsException(int index)
```

Constructs a new `StringIndexOutOfBoundsException` class with an argument indicating the illegal index.
Parameters:
`index` - the illegal index.

---

java.lang
Class System

```
java.lang.Object
  |
  +--java.lang.System
```

---

public final class **System**
extends Object

The `System` class contains several useful class fields and methods. It cannot be instantiated.

Since:
JDK1.0

Field Summary

`static PrintStream`	**`err`** The "standard" error output stream.
`static PrintStream`	**`out`** The "standard" output stream.

Method Summary

`static void`	**`arraycopy`**`(Object src, int src_position, Object dst, int dst_position, int length)` Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.
`static long`	**`currentTimeMillis`**`()` Returns the current time in milliseconds.
`static void`	**`exit`**`(int status)` Terminates the currently running Java application.
`static void`	**`gc`**`()` Runs the garbage collector.
`static String`	**`getProperty`**`(String key)` Gets the system property indicated by the specified key.
`static int`	**`identityHashCode`**`(Object x)` Returns the same hashcode for the given object as would be returned by the default method hashCode(), whether or not the given object's class overrides hashCode().

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

out

```
public static final PrintStream out
```

The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

For simple stand-alone Java applications, a typical way to write a line of output data is:

```
System.out.println(data)
```

See the `println` methods in class `PrintStream`.

See Also:

`PrintStream.println()`, `PrintStream.println(boolean)`, `PrintStream.println(char)`, `PrintStream.println(char[])`, `PrintStream.println(int)`, `PrintStream.println(long)`, `PrintStream.println(java.lang.Object)`, `PrintStream.println(java.lang.String)`

---

err

```
public static final PrintStream err
```

The "standard" error output stream. This stream is already open and ready to accept output data.

Typically this stream corresponds to display output or another output destination specified by the host environment or user. By convention, this output stream is used to display error messages or other information that should come to the immediate attention of a user even if the principal output stream, the value of the variable `out`, has been redirected to a file or other destination that is typically not continuously monitored.

Method Detail

currentTimeMillis

```
public static long currentTimeMillis()
```

Returns the current time in milliseconds.

Returns:

the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

---

arraycopy

```
public static void arraycopy(Object src,
                             int src_position,
                             Object dst,
                             int dst_position,
                             int length)
```

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. A subsequence of array components are copied from the source array referenced by `src` to the destination array referenced by `dst`. The number of components copied is equal to the `length` argument. The components at positions `srcOffset` through `srcOffset+length-1` in the source array are copied into positions `dstOffset` through `dstOffset+length-1`, respectively, of the destination array.

If the `src` and `dst` arguments refer to the same array object, then the copying is performed as if the components at positions `srcOffset` through `srcOffset+length-1` were first copied to a temporary array with `length` components and then the contents of the temporary array were copied into positions `dstOffset` through `dstOffset+length-1` of the destination array.

If `dst` is `null`, then a `NullPointerException` is thrown.

If `src` is `null`, then a `NullPointerException` is thrown and the destination array is not modified.

Otherwise, if any of the following is true, an `ArrayStoreException` is thrown and the destination is not modified:

- The `src` argument refers to an object that is not an array.
- The `dst` argument refers to an object that is not an array.
- The `src` argument and `dst` argument refer to arrays whose component types are different primitive types.
- The `src` argument refers to an array with a primitive component type and the `dst` argument refers to an array with a reference component type.
- The `src` argument refers to an array with a reference component type and the `dst` argument refers to an array with a primitive component type.

Otherwise, if any of the following is true, an `IndexOutOfBoundsException` is thrown and the destination is not modified:

- The `srcOffset` argument is negative.
- The `dstOffset` argument is negative.
- The `length` argument is negative.
- `srcOffset+length` is greater than `src.length`, the length of the source array.
- `dstOffset+length` is greater than `dst.length`, the length of the destination array.

Otherwise, if any actual component of the source array from position `srcOffset` through `srcOffset+length-1` cannot be converted to the component type of the destination array by assignment conversion, an `ArrayStoreException` is thrown. In this case, let ***k*** be the smallest nonnegative integer less than length such that `src[srcOffset+`*k*`]` cannot be

converted to the component type of the destination array; when the exception is thrown, source array components from positions `srcOffset` through `srcOffset+`*k*`-1` will already have been copied to destination array positions `dstOffset` through `dstOffset+`*k*`-1` and no other positions of the destination array will have been modified. (Because of the restrictions already itemized, this paragraph effectively applies only to the situation where both arrays have component types that are reference types.)

Parameters:

`src` - the source array.
`src_position` - start position in the source array.
`dst` - the destination array.
`dst_position` - pos start position in the destination data.
`length` - the number of array elements to be copied.

Throws:

IndexOutOfBoundsException - if copying would cause access of data outside array bounds.
ArrayStoreException - if an element in the `src` array could not be stored into the `dest` array because of a type mismatch.
NullPointerException - if either `src` or `dst` is `null`.

---

identityHashCode

```
public static int identityHashCode(Object x)
```

Returns the same hashcode for the given object as would be returned by the default method hashCode(), whether or not the given object's class overrides hashCode(). The hashcode for the null reference is zero.

Parameters:

`x` - object for which the hashCode is to be calculated

Returns:

the hashCode

Since:

JDK1.1

---

getProperty

```
public static String getProperty(String key)
```

Gets the system property indicated by the specified key.

Parameters:

`key` - the name of the system property.

Returns:

the string value of the system property, or `null` if there is no property with that key.

Throws:

NullPointerException - if `key` is `null`.
IllegalArgumentException - if `key` is empty.

---

exit

```
public static void exit(int status)
```

Terminates the currently running Java application. The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.

This method calls the `exit` method in class `Runtime`. This method never returns normally.

The call `System.exit(n)` is effectively equivalent to the call:

```
Runtime.getRuntime().exit(n)
```

Parameters:

`status` - exit status.

See Also:

`Runtime.exit(int)`

---

gc

```
public static void gc()
```

Runs the garbage collector.

Calling the `gc` method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to reclaim space from all discarded objects.

The call `System.gc()` is effectively equivalent to the call:

```
Runtime.getRuntime().gc()
```

See Also:

`Runtime.gc()`

---

java.lang
Class Thread

```
java.lang.Object
  |
  +--java.lang.Thread
```

public class **Thread**
extends Object
implements Runnable

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

There are two ways to create a new thread of execution. One is to declare a class to be a subclass of `Thread`. This subclass should override the `run` method of class `Thread`. An instance of the subclass can then be allocated and started. For example, a thread that computes primes larger than a stated value could be written as follows:

```
class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeThread p = new PrimeThread(143);
p.start();
```

The other way to create a thread is to declare a class that implements the `Runnable` interface. That class then implements the `run` method. An instance of the class can then be allocated, passed as an argument when creating `Thread`, and started. The same example in this other style looks like the following:


```
class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
```

Since:
JDK1.0
See Also:
`Runnable`, `Runtime.exit(int)`, `run()`

Field Summary

static int	**MAX_PRIORITY** The maximum priority that a thread can have.
static int	**MIN_PRIORITY** The minimum priority that a thread can have.
static int	**NORM_PRIORITY** The default priority that is assigned to a thread.

Constructor Summary

Thread() Allocates a new `Thread` object.	
Thread(Runnable target) Allocates a new `Thread` object.	

Method Summary

static int	**activeCount**() Returns the current number of active threads in the VM.
static Thread	**currentThread**() Returns a reference to the currently executing thread object.
int	**getPriority**() Returns this thread's priority.
boolean	**isAlive**() Tests if this thread is alive.
void	**join**() Waits for this thread to die.
void	**run**() If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called; otherwise, this method does nothing and returns.
void	**setPriority**(int newPriority) Changes the priority of this thread.
static void	**sleep**(long millis) Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
void	**start**() Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.
String	**toString**() Returns a string representation of this thread, including a unique number that identifies the thread and the thread's priority.
static void	**yield**() Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

MIN_PRIORITY

```
public static final int MIN_PRIORITY
```

The minimum priority that a thread can have.

NORM_PRIORITY

```
public static final int NORM_PRIORITY
```

The default priority that is assigned to a thread.

MAX_PRIORITY

```
public static final int MAX_PRIORITY
```

The maximum priority that a thread can have.

Constructor Detail

Thread

```
public Thread()
```

Allocates a new Thread object. This constructor has the same effect as Thread(null, null, *gname*), where ***gname*** is a newly generated name. Automatically generated names are of the form "Thread-"+*n*, where *n* is an integer.

Threads created this way must have overridden their run() method to actually do anything. An example illustrating this method being used follows:

```
import java.lang.*;

class plain01 implements Runnable {
    String name;
    plain01() {
        name = null;
    }
    plain01(String s) {
        name = s;
    }
    public void run() {
        if (name == null)
            System.out.println("A new thread created");
        else
            System.out.println("A new thread with name " + name +
                               " created");
    }
}
class threadtest01 {
    public static void main(String args[] ) {
        int failed = 0 ;

        Thread t1 = new Thread();
```

```
            if (t1 != null)
                System.out.println("new Thread() succeed");
            else {
                System.out.println("new Thread() failed");
                failed++;
            }
        }
    }
```

See Also:
`Runnable`

Thread

```
public Thread(Runnable target)
```

Allocates a new `Thread` object. This constructor has the same effect as `Thread(null, target, gname)`, where *gname* is a newly generated name. Automatically generated names are of the form `"Thread-"`+*n*, where *n* is an integer.
Parameters:
`target` - the object whose `run` method is called.

Method Detail

currentThread

```
public static Thread currentThread()
```

Returns a reference to the currently executing thread object.
Returns:
the currently executing thread.

---

yield

```
public static void yield()
```

Causes the currently executing thread object to temporarily pause and allow other threads to execute.

---

sleep

```
public static void sleep(long millis)
                  throws InterruptedException
```

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors.
Parameters:
`millis` - the length of time to sleep in milliseconds.
Throws:
InterruptedException - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also:
`Object.notify()`

---

start

```
public void start()
```

Causes this thread to begin execution; the Java Virtual Machine calls the `run` method of this thread.

The result is that two threads are running concurrently: the current thread (which returns from the call to the `start` method) and the other thread (which executes its `run` method).
Throws:
IllegalThreadStateException - if the thread was already started.
See Also:
`run()`

---

run

```
public void run()
```

If this thread was constructed using a separate `Runnable` run object, then that `Runnable` object's `run` method is called; otherwise, this method does nothing and returns.

Subclasses of `Thread` should override this method.
Specified by:
run in interface Runnable
See Also:
`start()`, `Runnable.run()`

---

isAlive

```
public final boolean isAlive()
```

Tests if this thread is alive. A thread is alive if it has been started and has not yet died.
Returns:
`true` if this thread is alive; `false` otherwise.

---

setPriority

```
public final void setPriority(int newPriority)
```

Changes the priority of this thread.
Parameters:
`newPriority` - priority to set this thread to
Throws:
IllegalArgumentException - If the priority is not in the range `MIN_PRIORITY` to `MAX_PRIORITY`.

See Also:
`getPriority()`, `getPriority()`, `MAX_PRIORITY`, `MIN_PRIORITY`

---

getPriority

```
public final int getPriority()
```

Returns this thread's priority.
Returns:
this thread's name.
See Also:
`setPriority(int)`, `setPriority(int)`

---

activeCount

```
public static int activeCount()
```

Returns the current number of active threads in the VM.
Returns:
the current number of threads in this thread's thread group.

---

join

```
public final void join()
                throws InterruptedException
```

Waits for this thread to die.
Throws:
InterruptedException - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

---

toString

```
public String toString()
```

Returns a string representation of this thread, including a unique number that identifies the thread and the thread's priority.
Overrides:
toString in class Object
Returns:
a string representation of this thread.

---

java.lang
Class Throwable

```
java.lang.Object
  |
  +--java.lang.Throwable
```

Direct Known Subclasses:
Error, Exception

---

public class **Throwable**
extends Object

The `Throwable` class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or of one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java `throw` statement. Similarly, only this class or one of its subclasses can be the argument type in a `catch` clause.

Instances of two subclasses, `Error` and `Exception`, are conventionally used to indicate that exceptional situations have occurred. Typically, these instances are freshly created in the context of the exceptional situation so as to include relevant information (such as stack trace data).

By convention, class `Throwable` and its subclasses have two constructors, one that takes no arguments and one that takes a `String` argument that can be used to produce an error message.

A `Throwable` class contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error.

Here is one example of catching an exception:

```
try {
    int a[] = new int[2];
    a[4];
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("exception: " + e.getMessage());
    e.printStackTrace();
}
```

Since:
JDK1.0

Constructor Summary

`Throwable()` Constructs a new `Throwable` with `null` as its error message string.
`Throwable(String message)` Constructs a new `Throwable` with the specified error message.

Method Summary

`String`	**`getMessage()`** Returns the error message string of this throwable object.
`void`	**`printStackTrace()`**
`String`	**`toString()`** Returns a short description of this throwable object.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

Throwable

```
public Throwable()
```

Constructs a new `Throwable` with `null` as its error message string. Also, the method

---

Throwable

```
public Throwable(String message)
```

Constructs a new `Throwable` with the specified error message.

Parameters:

`message` - the error message. The error message is saved for later retrieval by the `getMessage()` method.

Method Detail

getMessage

```
public String getMessage()
```

Returns the error message string of this throwable object.

Returns:

the error message string of this `Throwable` object if it was `created` with an error message string; or `null` if it was `created` with no error message.

---

toString

```
public String toString()
```

Returns a short description of this throwable object. If this `Throwable` object was `created` with an error message string, then the result is the concatenation of three strings:

- The name of the actual class of this object
- ": " (a colon and a space)
- The result of the `getMessage()` method for this object

If this `Throwable` object was `created` with no error message string, then the name of the actual class of this object is returned.

Overrides:

toString in class Object

Returns:

a string representation of this `Throwable`.

---

printStackTrace

```
public void printStackTrace()
```

---

java.lang
Class VirtualMachineError

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Error
              |
              +--java.lang.VirtualMachineError
```

Direct Known Subclasses:
OutOfMemoryError

---

public abstract class **VirtualMachineError**
extends Error

Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.

Since:
JDK1.0

Constructor Summary

`VirtualMachineError()` Constructs a `VirtualMachineError` with no detail message.
`VirtualMachineError(String s)` Constructs a `VirtualMachineError` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

VirtualMachineError

`public VirtualMachineError()`

Constructs a `VirtualMachineError` with no detail message.

VirtualMachineError

`public VirtualMachineError(String s)`

Constructs a `VirtualMachineError` with the specified detail message.
Parameters:
`s` - the detail message.

Package java.io

Interface Summary	
DataInput	The `DataInput` interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types.
DataOutput	The `DataOutput` interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.

Class Summary	
ByteArrayInputStream	A `ByteArrayInputStream` contains an internal buffer that contains bytes that may be read from the stream.
ByteArrayOutputStream	This class implements an output stream in which the data is written into a byte array.
DataInputStream	A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.
DataOutputStream	A data input stream lets an application write primitive Java data types to an output stream in a portable way.
InputStream	This abstract class is the superclass of all classes representing an input stream of bytes.
InputStreamReader	An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and translates them into characters according to a specified character encoding.
OutputStream	This abstract class is the superclass of all classes representing an output stream of bytes.
OutputStreamWriter	An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are translated into bytes according to a specified character encoding.
PrintStream	A `PrintStream` adds functionality to another output stream, namely the ability to print representations of various data values conveniently.
Reader	Abstract class for reading character streams.
Writer	Abstract class for writing to character streams.

Exception Summary	
EOFException	Signals that an end of file or end of stream has been reached unexpectedly during input.
InterruptedIOException	Signals that an I/O operation has been interrupted.
IOException	Signals that an I/O exception of some sort has occurred.
UnsupportedEncodingException	The Character Encoding is not supported.
UTFDataFormatException	Signals that a malformed UTF-8 string has been read in a data input stream or by any class that implements the data input interface.

java.io
Class ByteArrayInputStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--java.io.ByteArrayInputStream
```

public class **ByteArrayInputStream**
extends InputStream

A `ByteArrayInputStream` contains an internal buffer that contains bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the `read` method.

Since:
JDK1.0

Field Summary

protected byte[]	**buf** An array of bytes that was provided by the creator of the stream.
protected int	**count** The index one greater than the last valid character in the input stream buffer.
protected int	**mark** The currently marked position in the stream.
protected int	**pos** The index of the next character to read from the input stream buffer.

Constructor Summary

ByteArrayInputStream(byte[] buf) Creates a `ByteArrayInputStream` so that it uses `buf` as its buffer array.
ByteArrayInputStream(byte[] buf, int offset, int length) Creates `ByteArrayInputStream` that uses `buf` as its buffer array.

Method Summary

int	**available**() Returns the number of bytes that can be read from this input stream without blocking.
void	**close**() Closes this input stream and releases any system resources associated with the stream.
void	**mark**(int readAheadLimit) Set the current marked position in the stream.
boolean	**markSupported**() Tests if ByteArrayInputStream supports mark/reset.
int	**read**() Reads the next byte of data from this input stream.
int	**read**(byte[] b, int off, int len) Reads up to `len` bytes of data into an array of bytes from this input stream.
void	**reset**() Resets the buffer to the marked position.
long	**skip**(long n) Skips `n` bytes of input from this input stream.

Methods inherited from class java.io.InputStream

`read`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

buf

`protected byte[] buf`

An array of bytes that was provided by the creator of the stream. Elements `buf[0]` through `buf[count-1]` are the only bytes that can ever be read from the stream; element `buf[pos]` is the next byte to be read.

pos

```
protected int pos
```

The index of the next character to read from the input stream buffer. This value should always be nonnegative and not larger than the value of `count`. The next byte to be read from the input stream buffer will be `buf[pos]`.

mark

```
protected int mark
```

The currently marked position in the stream. ByteArrayInputStream objects are marked at position zero by default when constructed. They may be marked at another position within the buffer by the `mark()` method. The current buffer position is set to this point by the `reset()` method.

Since:

JDK1.1

count

```
protected int count
```

The index one greater than the last valid character in the input stream buffer. This value should always be nonnegative and not larger than the length of `buf`. It is one greater than the position of the last byte within `buf` that can ever be read from the input stream buffer.

Constructor Detail

ByteArrayInputStream

```
public ByteArrayInputStream(byte[] buf)
```

Creates a `ByteArrayInputStream` so that it uses `buf` as its buffer array. The buffer array is not copied. The initial value of `pos` is `0` and the initial value of `count` is the length of `buf`.

Parameters:

`buf` - the input buffer.

ByteArrayInputStream

```
public ByteArrayInputStream(byte[] buf,
                            int offset,
                            int length)
```

Creates `ByteArrayInputStream` that uses `buf` as its buffer array. The initial value of `pos` is `offset` and the initial value of `count` is `offset+len`. The buffer array is not copied.

Note that if bytes are simply read from the resulting input stream, elements `buf[pos]` through `buf[pos+len-1]` will be read; however, if a `reset` operation is performed, then bytes `buf[0]` through b`uf[pos-1]` will then become available for input.

Parameters:

`buf` - the input buffer.
`offset` - the offset in the buffer of the first byte to read.
`length` - the maximum number of bytes to read from the buffer.

Method Detail

read

```
public int read()
```

Reads the next byte of data from this input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned.

This `read` method cannot block.

Overrides:

read in class InputStream

Returns:

the next byte of data, or `-1` if the end of the stream has been reached.

read

```
public int read(byte[] b,
                int off,
                int len)
```

Reads up to `len` bytes of data into an array of bytes from this input stream. If `pos` equals `count`, then `-1` is returned to indicate end of file. Otherwise, the number `k` of bytes read is equal to the smaller of `len` and `count-pos`. If `k` is positive, then bytes `buf[pos]` through `buf[pos+k-1]` are copied into `b[off]` through `b[off+k-1]` in the manner performed by `System.arraycopy`. The value `k` is added into `pos` and `k` is returned.

This `read` method cannot block.

Overrides:

read in class InputStream

Parameters:

`b` - the buffer into which the data is read.
`off` - the start offset of the data.
`len` - the maximum number of bytes read.

Returns:

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

skip

```
public long skip(long n)
```

Skips n bytes of input from this input stream. Fewer bytes might be skipped if the end of the input stream is reached. The actual number k of bytes to be skipped is equal to the smaller of n and count-pos. The value k is added into pos and k is returned.

Overrides:
skip in class InputStream

Parameters:
n - the number of bytes to be skipped.

Returns:
the actual number of bytes skipped.

---

available

```
public int available()
```

Returns the number of bytes that can be read from this input stream without blocking. The value returned is count - pos, which is the number of bytes remaining to be read from the input buffer.

Overrides:
available in class InputStream

Returns:
the number of bytes that can be read from the input stream without blocking.

---

markSupported

```
public boolean markSupported()
```

Tests if ByteArrayInputStream supports mark/reset.

Overrides:
markSupported in class InputStream

Since:
JDK1.1

---

mark

```
public void mark(int readAheadLimit)
```

Set the current marked position in the stream. ByteArrayInputStream objects are marked at position zero by default when constructed. They may be marked at another position within the buffer by this method.

Overrides:
mark in class InputStream

Since:
JDK1.1

---

reset

```
public void reset()
```

Resets the buffer to the marked position. The marked position is the beginning unless another position was marked. The value of pos is set to 0.

Overrides:
reset in class InputStream

Tags copied from class: InputStream

Throws:
IOException - if this stream has not been marked or if the mark has been invalidated.

See Also:
`InputStream.mark(int)`, `IOException`

---

close

```
public void close()
           throws IOException
```

Closes this input stream and releases any system resources associated with the stream.

Overrides:
close in class InputStream

Tags copied from class: InputStream

Throws:
IOException - if an I/O error occurs.

---

java.io
Class ByteArrayOutputStream

```
java.lang.Object
  |
  +--java.io.OutputStream
        |
        +--java.io.ByteArrayOutputStream
```

public class **ByteArrayOutputStream**
extends OutputStream

This class implements an output stream in which the data is written into a byte array. The buffer automatically grows as data is written to it. The data can be retrieved using `toByteArray()` and `toString()`.

Since:
JDK1.0

Field Summary

protected byte[]	**buf** The buffer where data is stored.
protected int	**count** The number of valid bytes in the buffer.

Constructor Summary

ByteArrayOutputStream() Creates a new byte array output stream.
ByteArrayOutputStream(int size) Creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

Method Summary

void	**close**() Closes this output stream and releases any system resources associated with this stream.
void	**reset**() Resets the `count` field of this byte array output stream to zero, so that all currently accumulated output in the output stream is discarded.
int	**size**() Returns the current size of the buffer.
byte[]	**toByteArray**() Creates a newly allocated byte array.
void	**write**(byte[] b, int off, int len) Writes `len` bytes from the specified byte array starting at offset `off` to this byte array output stream.
void	**write**(int b) Writes the specified byte to this byte array output stream.

Methods inherited from class java.io.OutputStream
flush, write

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

buf

```
protected byte[] buf
```

The buffer where data is stored.

count

```
protected int count
```

The number of valid bytes in the buffer.

Constructor Detail

ByteArrayOutputStream

```
public ByteArrayOutputStream()
```

Creates a new byte array output stream. The buffer capacity is initially 32 bytes, though its size increases if necessary.

ByteArrayOutputStream

```
public ByteArrayOutputStream(int size)
```

Creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

Parameters:

`size` - the initial size.

Throws:

IllegalArgumentException - if size is negative.

Method Detail

write

```
public void write(int b)
```

Writes the specified byte to this byte array output stream.

Overrides:

write in class OutputStream

Parameters:

`b` - the byte to be written.

---

write

```
public void write(byte[] b,
                  int off,
                  int len)
```

Writes `len` bytes from the specified byte array starting at offset `off` to this byte array output stream.

Overrides:

write in class OutputStream

Parameters:

`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

---

reset

```
public void reset()
```

Resets the `count` field of this byte array output stream to zero, so that all currently accumulated output in the output stream is discarded. The output stream can be used again, reusing the already allocated buffer space.

See Also:

`ByteArrayInputStream.count`

---

toByteArray

```
public byte[] toByteArray()
```

Creates a newly allocated byte array. Its size is the current size of this output stream and the valid contents of the buffer have been copied into it.

Returns:

the current contents of this output stream, as a byte array.

See Also:

`size()`

---

size

```
public int size()
```

Returns the current size of the buffer.

Returns:

the value of the `count` field, which is the number of valid bytes in this output stream.

See Also:

`count`

---

close

```
public void close()
            throws IOException
```

Closes this output stream and releases any system resources associated with this stream. A closed stream cannot perform output operations and cannot be reopened.

Overrides:

close in class OutputStream

Tags copied from class: OutputStream

Throws:

IOException - if an I/O error occurs.

---

java.io
Interface DataInput

All Known Subinterfaces:
Datagram

All Known Implementing Classes:
DataInputStream

public interface **DataInput**

The `DataInput` interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types. There is also a facility for reconstructing a `String` from data in Java modified UTF-8 format.

It is generally true of all the reading routines in this interface that if end of file is reached before the desired number of bytes has been read, an `EOFException` (which is a kind of `IOException`) is thrown. If any byte cannot be read for any reason other than end of file, an `IOException` other than `EOFException` is thrown. In particular, an `IOException` may be thrown if the input stream has been closed.

Since:
JDK1.0
See Also:
`DataOutput`

Method Summary

`boolean`	**`readBoolean`**`()` Reads one input byte and returns `true` if that byte is nonzero, `false` if that byte is zero.
`byte`	**`readByte`**`()` Reads and returns one input byte.
`char`	**`readChar`**`()` Reads an input `char` and returns the `char` value.
`void`	**`readFully`**`(byte[] b)` Reads some bytes from an input stream and stores them into the buffer array `b`.
`void`	**`readFully`**`(byte[] b, int off, int len)` Reads `len` bytes from an input stream.
`int`	**`readInt`**`()` Reads four input bytes and returns an `int` value.
`long`	**`readLong`**`()` Reads eight input bytes and returns a `long` value.
`short`	**`readShort`**`()` Reads two input bytes and returns a `short` value.
`int`	**`readUnsignedByte`**`()` Reads one input byte, zero-extends it to type `int`, and returns the result, which is therefore in the range `0` through `255`.
`int`	**`readUnsignedShort`**`()` Reads two input bytes and returns an `int` value in the range `0` through `65535`.
`String`	**`readUTF`**`()` Reads in a string that has been encoded using a modified UTF-8 format.
`int`	**`skipBytes`**`(int n)` Makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes.

Method Detail

readFully

```
public void readFully(byte[] b)
              throws IOException
```

Reads some bytes from an input stream and stores them into the buffer array `b`. The number of bytes read is equal to the length of `b`.

This method blocks until one of the following conditions occurs:

- `b.length` bytes of input data are available, in which case a normal return is made.
- End of file is detected, in which case an `EOFException` is thrown.
- An I/O error occurs, in which case an `IOException` other than `EOFException` is thrown.

If `b` is `null`, a `NullPointerException` is thrown. If `b.length` is zero, then no bytes are read. Otherwise, the first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on. If an exception is thrown from this method, then it may be that some but not all bytes of `b` have been updated with data from the input stream.

Parameters:

`b` - the buffer into which the data is read.

Throws:

EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readFully

```
public void readFully(byte[] b,
                      int off,
                      int len)
               throws IOException
```

Reads `len` bytes from an input stream.

This method blocks until one of the following conditions occurs:

- `len` bytes of input data are available, in which case a normal return is made.
- End of file is detected, in which case an `EOFException` is thrown.
- An I/O error occurs, in which case an `IOException` other than `EOFException` is thrown.

If `b` is `null`, a `NullPointerException` is thrown. If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown. If `len` is zero, then no bytes are read. Otherwise, the first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`.

Parameters:

`b` - the buffer into which the data is read.
`off` - an int specifying the offset into the data.
`len` - an int specifying the number of bytes to read.

Throws:

EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

skipBytes

```
public int skipBytes(int n)
             throws IOException
```


Makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes. However, it may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. This method never throws an `EOFException`. The actual number of bytes skipped is returned.

Parameters:

`n` - the number of bytes to be skipped.

Returns:

the number of bytes skipped, which is always `n`.

Throws:

EOFException - if this stream reaches the end before skipping all the bytes.
IOException - if an I/O error occurs.

---

readBoolean

```
public boolean readBoolean()
                    throws IOException
```

Reads one input byte and returns `true` if that byte is nonzero, `false` if that byte is zero. This method is suitable for reading the byte written by the `writeBoolean` method of interface `DataOutput`.

Returns:

the `boolean` value read.

Throws:

EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readByte

```
public byte readByte()
            throws IOException
```

Reads and returns one input byte. The byte is treated as a signed value in the range `-128` through `127`, inclusive. This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput`.

Returns:

the 8-bit value read.

Throws:

EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readUnsignedByte

```
public int readUnsignedByte()
                     throws IOException
```

Reads one input byte, zero-extends it to type `int`, and returns the result, which is therefore in the range `0` through `255`. This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput` if the argument to `writeByte` was intended to be a value in the range `0` through `255`.

Returns:
the unsigned 8-bit value read.
Throws:
EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readShort

```
public short readShort()
               throws IOException
```

Reads two input bytes and returns a `short` value. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(short)((a << 8) * | (b & 0xff))
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput`.
Returns:
the 16-bit value read.
Throws:
EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readUnsignedShort

```
public int readUnsignedShort()
                    throws IOException
```

Reads two input bytes and returns an `int` value in the range `0` through `65535`. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(((a & 0xff) << 8) | (b & 0xff))
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput` if the argument to `writeShort` was intended to be a value in the range `0` through `65535`.
Returns:
the unsigned 16-bit value read.
Throws:
EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readChar

```
public char readChar()
             throws IOException
```


Reads an input `char` and returns the `char` value. A Unicode `char` is made up of two bytes. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(char)((a << 8) | (b & 0xff))
```

This method is suitable for reading bytes written by the `writeChar` method of interface `DataOutput`.
Returns:
the Unicode `char` read.
Throws:
EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readInt

```
public int readInt()
           throws IOException
```

Reads four input bytes and returns an `int` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, and `d` be the fourth byte. The value returned is:

```
(((a & 0xff) << 24) | ((b & 0xff) << 16) |
((c & 0xff) << 8) | (d & 0xff))
```

This method is suitable for reading bytes written by the `writeInt` method of interface `DataOutput`.
Returns:
the `int` value read.
Throws:
EOFException - if this stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

readLong

```
public long readLong()
            throws IOException
```

Reads eight input bytes and returns a `long` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, `d` be the fourth byte, `e` be the fifth byte, `f` be the sixth byte, `g` be the seventh byte, and `h` be the eighth byte. The value returned is:

```
(((long)(a & 0xff) << 56) |
((long)(b & 0xff) << 48) |
((long)(c & 0xff) << 40) |
((long)(d & 0xff) << 32) |
((long)(e & 0xff) << 24) |
((long)(f & 0xff) << 16) |
((long)(g & 0xff) <<  8) |
((long)(h & 0xff)))
```

This method is suitable for reading bytes written by the `writeLong` method of interface `DataOutput`.

Returns:

the `long` value read.

Throws:

EOFException - if this stream reaches the end before reading all the bytes.

IOException - if an I/O error occurs.

---

readUTF

```
public String readUTF()
              throws IOException
```

Reads in a string that has been encoded using a modified UTF-8 format. The general contract of `readUTF` is that it reads a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`.

First, two bytes are read and used to construct an unsigned 16-bit integer in exactly the manner of the `readUnsignedShort` method . This integer value is called the *UTF length* and specifies the number of additional bytes to be read. These bytes are then converted to characters by considering them in groups. The length of each group is computed from the value of the first byte of the group. The byte following a group, if any, is the first byte of the next group.

If the first byte of a group matches the bit pattern `0xxxxxxx` (where `x` means "may be `0` or `1`"), then the group consists of just that byte. The byte is zero-extended to form a character.

If the first byte of a group matches the bit pattern `110xxxxx`, then the group consists of that byte `a` and a second byte `b`. If there is no byte `b` (because byte `a` was the last of the bytes to be read), or if byte `b` does not match the bit pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown. Otherwise, the group is converted to the character:

```
(char)(((a& 0x1F) << 6) | (b & 0x3F))
```

If the first byte of a group matches the bit pattern `1110xxxx`, then the group consists of that byte `a` and two more bytes `b` and `c`. If there is no byte `c` (because byte `a` was one of the last two of the bytes to be read), or either byte `b` or byte `c` does not match the bit pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown. Otherwise, the group is converted to the character:

```
 (char)(((a & 0x0F) << 12) | ((b & 0x3F) << 6) | (c & 0x3F))
```

If the first byte of a group matches the pattern `1111xxxx` or the pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown.

If end of file is encountered at any time during this entire process, then an `EOFException` is thrown.

After every group has been converted to a character by this process, the characters are gathered, in the same order in which their corresponding groups were read from the input stream, to form a `String`, which is returned.

The `writeUTF` method of interface `DataOutput` may be used to write data that is suitable for reading by this method.

Returns:

a Unicode string.

Throws:

EOFException - if this stream reaches the end before reading all the bytes.

IOException - if an I/O error occurs.

UTFDataFormatException - if the bytes do not represent a valid UTF-8 encoding of a string.

---

java.io
Class DataInputStream

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--java.io.DataInputStream
```

public class **DataInputStream**
extends InputStream
implements DataInput

A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way. An application uses a data output stream to write data that can later be read by a data input stream.

Field Summary

protected InputStream	**in** The input stream

Constructor Summary

DataInputStream(InputStream in) Creates a DataInputStream and saves its argument, the input stream in, for later use.

Method Summary

int	**available**() Returns the number of bytes that can be read from this input stream without blocking.
void	**close**() Closes this input stream and releases any system resources associated with the stream.
void	**mark**(int readlimit) Marks the current position in this input stream.
boolean	**markSupported**() Tests if this input stream supports the mark and reset methods.
int	**read**() Reads the next byte of data from this input stream.

int	**read**(byte[] b, int off, int len) Reads up to len bytes of data from this input stream into an array of bytes.
boolean	**readBoolean**() See the general contract of the readBoolean method of DataInput.
byte	**readByte**() See the general contract of the readByte method of DataInput.
char	**readChar**() See the general contract of the readChar method of DataInput.
void	**readFully**(byte[] b) See the general contract of the readFully method of DataInput.
void	**readFully**(byte[] b, int off, int len) See the general contract of the readFully method of DataInput.
int	**readInt**() See the general contract of the readInt method of DataInput.
long	**readLong**() See the general contract of the readLong method of DataInput.
short	**readShort**() See the general contract of the readShort method of DataInput.
int	**readUnsignedByte**() See the general contract of the readUnsignedByte method of DataInput.
int	**readUnsignedShort**() See the general contract of the readUnsignedShort method of DataInput.
String	**readUTF**() See the general contract of the readUTF method of DataInput.
static String	**readUTF**(DataInput in) Reads from the stream in a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a String.
void	**reset**() Repositions this stream to the position at the time the mark method was last called on this input stream.
long	**skip**(long n) Skips over and discards n bytes of data from the input stream.
int	**skipBytes**(int n) See the general contract of the skipBytes method of DataInput.

Methods inherited from class java.io.InputStream
read

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

in

```
protected InputStream in
```

The input stream

Constructor Detail

DataInputStream

```
public DataInputStream(InputStream in)
```

Creates a `DataInputStream` and saves its argument, the input stream `in`, for later use.

Parameters:

`in` - the input stream.

Method Detail

read

```
public int read()
         throws IOException
```

Reads the next byte of data from this input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

This method simply performs `in.read()` and returns the result.

Overrides:

read in class InputStream

Returns:

the next byte of data, or `-1` if the end of the stream is reached.

Throws:

IOException - if an I/O error occurs.

read

```
public int read(byte[] b,
                int off,
                int len)
         throws IOException
```

Reads up to `len` bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

This method simply performs `in.read(b, off, len)` and returns the result.

Overrides:

read in class InputStream

Parameters:

`b` - the buffer into which the data is read.

`off` - the start offset of the data.

`len` - the maximum number of bytes read.

Returns:

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:

IOException - if an I/O error occurs.

readFully

```
public void readFully(byte[] b)
              throws IOException
```

See the general contract of the `readFully` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified by:

readFully in interface DataInput

Parameters:

`b` - the buffer into which the data is read.

Throws:

EOFException - if this input stream reaches the end before reading all the bytes.

IOException - if an I/O error occurs.

readFully

```
public void readFully(byte[] b,
                      int off,
                      int len)
               throws IOException
```

See the general contract of the `readFully` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified by:
readFully in interface DataInput
Parameters:
`b` - the buffer into which the data is read.
`off` - the start offset of the data.
`len` - the number of bytes to read.
Throws:
EOFException - if this input stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

---

skipBytes

```
public int skipBytes(int n)
             throws IOException
```

See the general contract of the `skipBytes` method of `DataInput`.

Bytes for this operation are read from the contained input stream.
Specified by:
skipBytes in interface DataInput
Parameters:
`n` - the number of bytes to be skipped.
Returns:
the actual number of bytes skipped.
Throws:
IOException - if an I/O error occurs.

---

readBoolean

```
public boolean readBoolean()
                  throws IOException
```

See the general contract of the `readBoolean` method of `DataInput`.

Bytes for this operation are read from the contained input stream.
Specified by:
readBoolean in interface DataInput
Returns:
the `boolean` value read.
Throws:
EOFException - if this input stream has reached the end.
IOException - if an I/O error occurs.

---

readByte

```
public byte readByte()
             throws IOException
```


See the general contract of the `readByte` method of `DataInput`.

Bytes for this operation are read from the contained input stream.
Specified by:
readByte in interface DataInput
Returns:
the next byte of this input stream as a signed 8-bit `byte`.
Throws:
EOFException - if this input stream has reached the end.
IOException - if an I/O error occurs.

---

readUnsignedByte

```
public int readUnsignedByte()
                   throws IOException
```

See the general contract of the `readUnsignedByte` method of `DataInput`.

Bytes for this operation are read from the contained input stream.
Specified by:
readUnsignedByte in interface DataInput
Returns:
the next byte of this input stream, interpreted as an unsigned 8-bit number.
Throws:
EOFException - if this input stream has reached the end.
IOException - if an I/O error occurs.

---

readShort

```
public short readShort()
               throws IOException
```

See the general contract of the `readShort` method of `DataInput`.

Bytes for this operation are read from the contained input stream.
Specified by:
readShort in interface DataInput
Returns:
the next two bytes of this input stream, interpreted as a signed 16-bit number.
Throws:
EOFException - if this input stream reaches the end before reading two bytes.
IOException - if an I/O error occurs.

---

readUnsignedShort

```
public int readUnsignedShort()
                    throws IOException
```

See the general contract of the `readUnsignedShort` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified by:
readUnsignedShort in interface DataInput

Returns:
the next two bytes of this input stream, interpreted as an unsigned 16-bit integer.

Throws:
EOFException - if this input stream reaches the end before reading two bytes.
IOException - if an I/O error occurs.

---

readChar

```
public char readChar()
             throws IOException
```

See the general contract of the `readChar` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified by:
readChar in interface DataInput

Returns:
the next two bytes of this input stream as a Unicode character.

Throws:
EOFException - if this input stream reaches the end before reading two bytes.
IOException - if an I/O error occurs.

---

readInt

```
public int readInt()
           throws IOException
```

See the general contract of the `readInt` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified by:
readInt in interface DataInput

Returns:
the next four bytes of this input stream, interpreted as an `int`.

Throws:
EOFException - if this input stream reaches the end before reading four bytes.
IOException - if an I/O error occurs.

---

readLong

```
public long readLong()
             throws IOException
```


See the general contract of the `readLong` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified by:
readLong in interface DataInput

Returns:
the next eight bytes of this input stream, interpreted as a `long`.

Throws:
EOFException - if this input stream reaches the end before reading eight bytes.
IOException - if an I/O error occurs.

---

readUTF

```
public String readUTF()
              throws IOException
```

See the general contract of the `readUTF` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified by:
readUTF in interface DataInput

Returns:
a Unicode string.

Throws:
EOFException - if this input stream reaches the end before reading all the bytes.
IOException - if an I/O error occurs.

See Also:
`readUTF(java.io.DataInput)`

---

readUTF

```
public static String readUTF(DataInput in)
                    throws IOException
```

Reads from the stream `in` a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`. The details of the modified UTF-8 representation are exactly the same as for the `readUTF` method of `DataInput`.

Parameters:
`in` - a data input stream.

Returns:
a Unicode string.

Throws:
EOFException - if the input stream reaches the end before all the bytes.
IOException - if an I/O error occurs.
UTFDataFormatException - if the bytes do not represent a valid UTF-8 encoding of a Unicode string.

See Also:
`readUnsignedShort()`

skip

```
public long skip(long n)
          throws IOException
```

Skips over and discards `n` bytes of data from the input stream. The `skip` method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly `0`. The actual number of bytes skipped is returned.

This method simply performs `in.skip(n)`.

Overrides:
skip in class InputStream

Parameters:
`n` - the number of bytes to be skipped.

Returns:
the actual number of bytes skipped.

Throws:
IOException - if an I/O error occurs.

available

```
public int available()
             throws IOException
```

Returns the number of bytes that can be read from this input stream without blocking.

This method simply performs `in.available(n)` and returns the result.

Overrides:
available in class InputStream

Returns:
the number of bytes that can be read from the input stream without blocking.

Throws:
IOException - if an I/O error occurs.

close

```
public void close()
           throws IOException
```

Closes this input stream and releases any system resources associated with the stream. This method simply performs `in.close()`.

Overrides:
close in class InputStream

Throws:
IOException - if an I/O error occurs.

mark

```
public void mark(int readlimit)
```

Marks the current position in this input stream. A subsequent call to the `reset` method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The `readlimit` argument tells this input stream to allow that many bytes to be read before the mark position gets invalidated.

This method simply performs `in.mark(readlimit)`.

Overrides:
mark in class InputStream

Parameters:
`readlimit` - the maximum limit of bytes that can be read before the mark position becomes invalid.

reset

```
public void reset()
           throws IOException
```

Repositions this stream to the position at the time the `mark` method was last called on this input stream.

This method simply performs `in.reset()`.

Stream marks are intended to be used in situations where you need to read ahead a little to see what's in the stream. Often this is most easily done by invoking some general parser. If the stream is of the type handled by the parse, it just chugs along happily. If the stream is not of that type, the parser should toss an exception when it fails. If this happens within readlimit bytes, it allows the outer code to reset the stream and try another parser.

Overrides:
reset in class InputStream

Throws:
IOException - if the stream has not been marked or if the mark has been invalidated.

markSupported

```
public boolean markSupported()
```

Tests if this input stream supports the `mark` and `reset` methods. This method simply performs `in.markSupported()`.

Overrides:
markSupported in class InputStream

Returns:
`true` if this stream type supports the `mark` and `reset` method; `false` otherwise.

java.io
Interface DataOutput

All Known Subinterfaces:
Datagram

All Known Implementing Classes:
DataOutputStream

public interface **DataOutput**

The `DataOutput` interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream. There is also a facility for converting a `String` into Java modified UTF-8 format and writing the resulting series of bytes.

For all the methods in this interface that write bytes, it is generally true that if a byte cannot be written for any reason, an `IOException` is thrown.

Since:
JDK1.0
See Also:
`DataInput`

Method Summary

void	**`write`**`(byte[] b)` Writes to the output stream all the bytes in array `b`.
void	**`write`**`(byte[] b, int off, int len)` Writes `len` bytes from array `b`, in order, to the output stream.
void	**`write`**`(int b)` Writes to the output stream the eight low-order bits of the argument `b`.
void	**`writeBoolean`**`(boolean v)` Writes a `boolean` value to this output stream.
void	**`writeByte`**`(int v)` Writes to the output stream the eight low- order bits of the argument `v`.
void	**`writeChar`**`(int v)` Writes a `char` value, which is comprised of two bytes, to the output stream.
void	**`writeChars`**`(String s)` Writes every character in the string `s`, to the output stream, in order, two bytes per character.
void	**`writeInt`**`(int v)` Writes an `int` value, which is comprised of four bytes, to the output stream.
void	**`writeLong`**`(long v)` Writes an `long` value, which is comprised of four bytes, to the output stream.
void	**`writeShort`**`(int v)` Writes two bytes to the output stream to represent the value of the argument.
void	**`writeUTF`**`(String str)` Writes two bytes of length information to the output stream, followed by the Java modified UTF representation of every character in the string `s`.

Method Detail

write

```
public void write(int b)
          throws IOException
```

Writes to the output stream the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.
Parameters:
`b` - the byte to be written.
Throws:
IOException - if an I/O error occurs.

write

```
public void write(byte[] b)
           throws IOException
```

Writes to the output stream all the bytes in array b. If b is null, a NullPointerException is thrown. If b.length is zero, then no bytes are written. Otherwise, the byte b[0] is written first, then b[1], and so on; the last byte written is b[b.length-1].

Parameters:

b - the data.

Throws:

IOException - if an I/O error occurs.

write

```
public void write(byte[] b,
                  int off,
                  int len)
           throws IOException
```

Writes len bytes from array b, in order, to the output stream. If b is null, a NullPointerException is thrown. If off is negative, or len is negative, or off+len is greater than the length of the array b, then an IndexOutOfBoundsException is thrown. If len is zero, then no bytes are written. Otherwise, the byte b[off] is written first, then b[off+1], and so on; the last byte written is b[off+len-1].

Parameters:

b - the data.

off - the start offset in the data.

len - the number of bytes to write.

Throws:

IOException - if an I/O error occurs.

writeBoolean

```
public void writeBoolean(boolean v)
                 throws IOException
```

Writes a boolean value to this output stream. If the argument v is true, the value (byte)1 is written; if v is false, the value (byte)0 is written. The byte written by this method may be read by the readBoolean method of interface DataInput, which will then return a boolean equal to v.

Parameters:

v - the boolean to be written.

Throws:

IOException - if an I/O error occurs.

writeByte

```
public void writeByte(int v)
              throws IOException
```

Writes to the output stream the eight low- order bits of the argument v. The 24 high-order bits of v are ignored. (This means that writeByte does exactly the same thing as write for an integer argument.) The byte written by this method may be read by the readByte method of interface DataInput, which will then return a byte equal to (byte)v.

Parameters:

v - the byte value to be written.

Throws:

IOException - if an I/O error occurs.

writeShort

```
public void writeShort(int v)
               throws IOException
```

Writes two bytes to the output stream to represent the value of the argument. The byte values to be written, in the order shown, are:

```
(byte)(0xff & (v >> 8))
(byte)(0xff & v)
```

The bytes written by this method may be read by the readShort method of interface DataInput , which will then return a short equal to (short)v.

Parameters:

v - the short value to be written.

Throws:

IOException - if an I/O error occurs.

writeChar

```
public void writeChar(int v)
              throws IOException
```

Writes a char value, which is comprised of two bytes, to the output stream. The byte values to be written, in the order shown, are:

```
(byte)(0xff & (v >> 8))
(byte)(0xff & v)
```

The bytes written by this method may be read by the readChar method of interface DataInput , which will then return a char equal to (char)v.

Parameters:

v - the char value to be written.

Throws:

IOException - if an I/O error occurs.

writeInt

```
public void writeInt(int v)
             throws IOException
```

Writes an `int` value, which is comprised of four bytes, to the output stream. The byte values to be written, in the order shown, are:

```
 (byte)(0xff & (v >> 24))
 (byte)(0xff & (v >> 16))
 (byte)(0xff & (v >>    8))
 (byte)(0xff & v)
```

The bytes written by this method may be read by the `readInt` method of interface `DataInput` , which will then return an `int` equal to `v`.

Parameters:

`v` - the `int` value to be written.

Throws:

IOException - if an I/O error occurs.

writeLong

```
public void writeLong(long v)
              throws IOException
```

Writes an `long` value, which is comprised of four bytes, to the output stream. The byte values to be written, in the order shown, are:

```
 (byte)(0xff & (v >> 48))
 (byte)(0xff & (v >> 40))
 (byte)(0xff & (v >> 32))
 (byte)(0xff & (v >> 24))
 (byte)(0xff & (v >> 16))
 (byte)(0xff & (v >>  8))
 (byte)(0xff & v)
```

The bytes written by this method may be read by the `readLong` method of interface `DataInput` , which will then return a `long` equal to `v`.

Parameters:

`v` - the `long` value to be written.

Throws:

IOException - if an I/O error occurs.

writeChars

```
public void writeChars(String s)
               throws IOException
```

Writes every character in the string `s`, to the output stream, in order, two bytes per character. If `s` is `null`, a `NullPointerException` is thrown. If `s.length` is zero, then no characters are written. Otherwise, the character `s[0]` is written first, then `s[1]`, and so on; the last character

written is `s[s.length-1]`. For each character, two bytes are actually written, high-order byte first, in exactly the manner of the `writeChar` method.

Parameters:

`s` - the string value to be written.

Throws:

IOException - if an I/O error occurs.

writeUTF

```
public void writeUTF(String str)
             throws IOException
```

Writes two bytes of length information to the output stream, followed by the Java modified UTF representation of every character in the string `s`. If `s` is `null`, a `NullPointerException` is thrown. Each character in the string `s` is converted to a group of one, two, or three bytes, depending on the value of the character.

If a character `c` is in the range `\u0001` through `\u007f`, it is represented by one byte:

```
(byte)c
```

If a character `c` is `\u0000` or is in the range `\u0080` through `\u07ff`, then it is represented by two bytes, to be written in the order shown:

```
 (byte)(0xc0 | (0x1f & (c >> 6)))
 (byte)(0x80 | (0x3f & c))
```

If a character `c` is in the range `\u0800` through `uffff`, then it is represented by three bytes, to be written in the order shown:

```
 (byte)(0xe0 | (0x0f & (c >> 12)))
 (byte)(0x80 | (0x3f & (c >>  6)))
 (byte)(0x80 | (0x3f & c))
```

First, the total number of bytes needed to represent all the characters of `s` is calculated. If this number is larger than `65535`, then a `UTFDataFormatError` is thrown. Otherwise, this length is written to the output stream in exactly the manner of the `writeShort` method; after this, the one-, two-, or three-byte representation of each character in the string `s` is written.

The bytes written by this method may be read by the `readUTF` method of interface `DataInput` , which will then return a `String` equal to `s`.

Parameters:

`str` - the string value to be written.

Throws:

IOException - if an I/O error occurs.

java.io
Class DataOutputStream

```
java.lang.Object
  |
  +--java.io.OutputStream
        |
        +--java.io.DataOutputStream
```

public class **DataOutputStream**
extends OutputStream
implements DataOutput

A data input stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in.

Since:
JDK1.0
See Also:
`DataInputStream`

Field Summary

protected OutputStream	**out** The output stream

Constructor Summary

DataOutputStream(OutputStream out) Creates a new data output stream to write data to the specified underlying output stream.

Method Summary

void	**close**() Closes this output stream and releases any system resources associated with the stream.
void	**flush**() Flushes this data output stream.
void	**write**(byte[] b, int off, int len) Writes `len` bytes from the specified byte array starting at offset `off` to the underlying output stream.
void	**write**(int b) Writes the specified byte (the low eight bits of the argument `b`) to the underlying output stream.
void	**writeBoolean**(boolean v) Writes a `boolean` to the underlying output stream as a 1-byte value.
void	**writeByte**(int v) Writes out a `byte` to the underlying output stream as a 1-byte value.
void	**writeChar**(int v) Writes a `char` to the underlying output stream as a 2-byte value, high byte first.
void	**writeChars**(String s) Writes a string to the underlying output stream as a sequence of characters.
void	**writeInt**(int v) Writes an `int` to the underlying output stream as four bytes, high byte first.
void	**writeLong**(long v) Writes a `long` to the underlying output stream as eight bytes, high byte first.
void	**writeShort**(int v) Writes a `short` to the underlying output stream as two bytes, high byte first.
void	**writeUTF**(String str) Writes a string to the underlying output stream using UTF-8 encoding in a machine-independent manner.

Methods inherited from class java.io.OutputStream
`write`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

out

```
protected OutputStream out
```

The output stream

Constructor Detail

DataOutputStream

```
public DataOutputStream(OutputStream out)
```

Creates a new data output stream to write data to the specified underlying output stream. The counter `written` is set to zero.

Parameters:

`out` - the underlying output stream, to be saved for later use.

Method Detail

write

```
public void write(int b)
           throws IOException
```

Writes the specified byte (the low eight bits of the argument `b`) to the underlying output stream. If no exception is thrown, the counter `written` is incremented by `1`.

Implements the `write` method of `OutputStream`.

Specified by:

write in interface DataOutput

Overrides:

write in class OutputStream

Parameters:

`b` - the `byte` to be written.

Throws:

IOException - if an I/O error occurs.

---

write

```
public void write(byte[] b,
                  int off,
                  int len)
           throws IOException
```

Writes `len` bytes from the specified byte array starting at offset `off` to the underlying output stream. If no exception is thrown, the counter `written` is incremented by `len`.

Specified by:

write in interface DataOutput

Overrides:

write in class OutputStream

Parameters:

`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

Throws:

IOException - if an I/O error occurs.

---

flush

```
public void flush()
           throws IOException
```

Flushes this data output stream. This forces any buffered output bytes to be written out to the stream.

The `flush` method of `DataOutputStream` calls the `flush` method of its underlying output stream.

Overrides:

flush in class OutputStream

Throws:

IOException - if an I/O error occurs.

---

close

```
public void close()
           throws IOException
```

Closes this output stream and releases any system resources associated with the stream.

The `close` method calls its `flush` method, and then calls the `close` method of its underlying output stream.

Overrides:

close in class OutputStream

Throws:

IOException - if an I/O error occurs.

---

writeBoolean

```
public void writeBoolean(boolean v)
                  throws IOException
```

Writes a `boolean` to the underlying output stream as a 1-byte value. The value `true` is written out as the value `(byte)1`; the value `false` is written out as the value `(byte)0`. If no exception is thrown, the counter `written` is incremented by `1`.

Specified by:

writeBoolean in interface DataOutput

Parameters:
v - a `boolean` value to be written.
Throws:
IOException - if an I/O error occurs.

---

writeByte

```
public void writeByte(int v)
               throws IOException
```

Writes out a `byte` to the underlying output stream as a 1-byte value. If no exception is thrown, the counter `written` is incremented by `1`.
Specified by:
writeByte in interface DataOutput
Parameters:
v - a `byte` value to be written.
Throws:
IOException - if an I/O error occurs.

---

writeShort

```
public void writeShort(int v)
                throws IOException
```

Writes a `short` to the underlying output stream as two bytes, high byte first. If no exception is thrown, the counter `written` is incremented by `2`.
Specified by:
writeShort in interface DataOutput
Parameters:
v - a `short` to be written.
Throws:
IOException - if an I/O error occurs.

---

writeChar

```
public void writeChar(int v)
               throws IOException
```

Writes a `char` to the underlying output stream as a 2-byte value, high byte first. If no exception is thrown, the counter `written` is incremented by `2`.
Specified by:
writeChar in interface DataOutput
Parameters:
v - a `char` value to be written.
Throws:
IOException - if an I/O error occurs.

---

writeInt

```
public void writeInt(int v)
             throws IOException
```

Writes an `int` to the underlying output stream as four bytes, high byte first. If no exception is thrown, the counter `written` is incremented by `4`.
Specified by:
writeInt in interface DataOutput
Parameters:
v - an `int` to be written.
Throws:
IOException - if an I/O error occurs.

---

writeLong

```
public void writeLong(long v)
               throws IOException
```

Writes a `long` to the underlying output stream as eight bytes, high byte first. In no exception is thrown, the counter `written` is incremented by `8`.
Specified by:
writeLong in interface DataOutput
Parameters:
v - a `long` to be written.
Throws:
IOException - if an I/O error occurs.

---

writeChars

```
public void writeChars(String s)
                throws IOException
```

Writes a string to the underlying output stream as a sequence of characters. Each character is written to the data output stream as if by the `writeChar` method. If no exception is thrown, the counter `written` is incremented by twice the length of `s`.
Specified by:
writeChars in interface DataOutput
Parameters:
s - a `String` value to be written.
Throws:
IOException - if an I/O error occurs.
See Also:
`writeChar(int)`

---

writeUTF

```
public void writeUTF(String str)
            throws IOException
```

Writes a string to the underlying output stream using UTF-8 encoding in a machine-independent manner.

First, two bytes are written to the output stream as if by the `writeShort` method giving the number of bytes to follow. This value is the number of bytes actually written out, not the length of the string. Following the length, each character of the string is output, in sequence, using the UTF-8 encoding for the character. If no exception is thrown, the counter `written` is incremented by the total number of bytes written to the output stream. This will be at least two plus the length of `str`, and at most two plus thrice the length of `str`.

Specified by:
writeUTF in interface DataOutput

Parameters:
`str` - a string to be written.

Throws:
IOException - if an I/O error occurs.

java.io
Class EOFException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.EOFException
```

public class **EOFException**
extends IOException

Signals that an end of file or end of stream has been reached unexpectedly during input.

This exception is mainly used by data input streams, which generally expect a binary file in a specific format, and for which an end of stream is an unusual condition. Most other input streams return a special value on end of stream.

Note that some input operations react to end-of-file by returning a distinguished value (such as `-1`) rather than by throwing an exception.

Since:
JDK1.0

See Also:
`DataInputStream`, `IOException`

Constructor Summary

`EOFException()` Constructs an `EOFException` with `null` as its error detail message.
`EOFException(String s)` Constructs an `EOFException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

EOFException

```
public EOFException()
```

Constructs an `EOFException` with `null` as its error detail message.

EOFException

```
public EOFException(String s)
```

Constructs an `EOFException` with the specified detail message. The string `s` may later be retrieved by the `Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:

`s` - the detail message.

---

java.io
Class IOException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
```

Direct Known Subclasses:

ConnectionNotFoundException, EOFException, InterruptedIOException, UnsupportedEncodingException, UTFDataFormatException

---

public class **IOException**
extends Exception

Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

Since:

JDK1.0

See Also:

`InputStream`, `OutputStream`

Constructor Summary

`IOException``()` Constructs an `IOException` with `null` as its error detail message.
`IOException``(String s)` Constructs an `IOException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

IOException

```
public IOException()
```

Constructs an `IOException` with `null` as its error detail message.

IOException

```
public IOException(String s)
```

Constructs an `IOException` with the specified detail message. The error message string `s` can later be retrieved by the `Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:

`s` - the detail message.

---

java.io
Class InputStream

```
java.lang.Object
  |
  +--java.io.InputStream
```

Direct Known Subclasses:

ByteArrayInputStream, DataInputStream

---

public abstract class **InputStream**
extends Object

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:

JDK1.0

See Also:

`read()`, `OutputStream`

Constructor Summary	
`InputStream()`	

Method Summary

int	**available**() Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
void	**close**() Closes this input stream and releases any system resources associated with the stream.
void	**mark**(int readlimit) Marks the current position in this input stream.
boolean	**markSupported**() Tests if this input stream supports the mark and reset methods.
abstract int	**read**() Reads the next byte of data from the input stream.
int	**read**(byte[] b) Reads some number of bytes from the input stream and stores them into the buffer array b.
int	**read**(byte[] b, int off, int len) Reads up to len bytes of data from the input stream into an array of bytes.
void	**reset**() Repositions this stream to the position at the time the mark method was last called on this input stream.
long	**skip**(long n) Skips over and discards n bytes of data from this input stream.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

InputStream

```
public InputStream()
```

Method Detail

read

```
public abstract int read()
                  throws IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

A subclass must provide an implementation of this method.

Returns:
the next byte of data, or `-1` if the end of the stream is reached.

Throws:
IOException - if an I/O error occurs.

---

read

```
public int read(byte[] b)
         throws IOException
```

Reads some number of bytes from the input stream and stores them into the buffer array `b`. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown. If the length of `b` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on. The number of bytes read is, at most, equal to the length of `b`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[0]` through `b[`*k*`-1]`, leaving elements `b[`*k*`]` through `b[b.length-1]` unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

The `read(b)` method for class `InputStream` has the same effect as:

```
read(b, 0, b.length)
```

Parameters:
`b` - the buffer into which the data is read.

Returns:
the total number of bytes read into the buffer, or `-1` is there is no more data because the end of the stream has been reached.

Throws:
IOException - if an I/O error occurs.

See Also:
`read(byte[], int, int)`

read

```
public int read(byte[] b,
                int off,
                int len)
         throws IOException
```

Reads up to `len` bytes of data from the input stream into an array of bytes. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[off]` through `b[off+`*k*`-1]`, leaving elements `b[off+`*k*`]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

The `read(b, off, len)` method for class `InputStream` simply calls the method `read()` repeatedly. If the first such call results in an `IOException`, that exception is returned from the call to the `read(b, off, len)` method. If any subsequent call to `read()` results in a `IOException`, the exception is caught and treated as if it were end of file; the bytes read up to that point are stored into `b` and the number of bytes read before the exception occurred is returned. Subclasses are encouraged to provide a more efficient implementation of this method.

Parameters:

`b` - the buffer into which the data is read.

`off` - the start offset in array `b` at which the data is written.

`len` - the maximum number of bytes to read.

Returns:

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:

IOException - if an I/O error occurs.

See Also:

`read()`

skip

```
public long skip(long n)
          throws IOException
```

Skips over and discards `n` bytes of data from this input stream. The `skip` method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly `0`. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. The actual number of bytes skipped is returned. If `n` is negative, no bytes are skipped.

The `skip` method of `InputStream` creates a byte array and then repeatedly reads into it until `n` bytes have been read or the end of the stream has been reached. Subclasses are encouraged to provide a more efficient implementation of this method.

Parameters:

`n` - the number of bytes to be skipped.

Returns:

the actual number of bytes skipped.

Throws:

IOException - if an I/O error occurs.

available

```
public int available()
           throws IOException
```

Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream. The next caller might be the same thread or or another thread.

The `available` method for class `InputStream` always returns `0`.

This method should be overridden by subclasses.

Returns:

the number of bytes that can be read from this input stream without blocking.

Throws:

IOException - if an I/O error occurs.

close

```
public void close()
          throws IOException
```

Closes this input stream and releases any system resources associated with the stream.

The `close` method of `InputStream` does nothing.

Throws:

IOException - if an I/O error occurs.

mark

```
public void mark(int readlimit)
```

Marks the current position in this input stream. A subsequent call to the `reset` method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The `readlimit` arguments tells this input stream to allow that many bytes to be read before the mark position gets invalidated.

The general contract of `mark` is that, if the method `markSupported` returns `true`, the stream somehow remembers all the bytes read after the call to `mark` and stands ready to supply those same bytes again if and whenever the method `reset` is called. However, the stream is not required to remember any data at all if more than `readlimit` bytes are read from the stream before `reset` is called.

The `mark` method of `InputStream` does nothing.

Parameters:

`readlimit` - the maximum limit of bytes that can be read before the mark position becomes invalid.

See Also:

`reset()`

---

reset

```
public void reset()
          throws IOException
```

Repositions this stream to the position at the time the `mark` method was last called on this input stream.

The general contract of `reset` is:

- If the method `markSupported` returns `true`, then:
  - If the method `mark` has not been called since the stream was created, or the number of bytes read from the stream since `mark` was last called is larger than the argument to `mark` at that last call, then an `IOException` might be thrown.
  - If such an `IOException` is not thrown, then the stream is reset to a state such that all the bytes read since the most recent call to `mark` (or since the start of the file, if `mark` has not been called) will be resupplied to subsequent callers of the `read` method, followed by any bytes that otherwise would have been the next input data as of the time of the call to `reset`.
- If the method `markSupported` returns `false`, then:
  - The call to `reset` may throw an `IOException`.
  - If an `IOException` is not thrown, then the stream is reset to a fixed state that depends on the particular type of the input stream and how it was created. The bytes that will be supplied to subsequent callers of the `read` method depend on the particular type of the input stream.

The method `reset` for class `InputStream` does nothing and always throws an `IOException`.

Throws:

IOException - if this stream has not been marked or if the mark has been invalidated.

See Also:

`mark(int)`, `IOException`

---

markSupported

```
public boolean markSupported()
```

Tests if this input stream supports the `mark` and `reset` methods. The `markSupported` method of `InputStream` returns `false`.

Returns:

`true` if this true type supports the mark and reset method; `false` otherwise.

See Also:

`mark(int)`, `reset()`

---

java.io
Class InputStreamReader

```
java.lang.Object
  |
  +--java.io.Reader
        |
        +--java.io.InputStreamReader
```

public class **InputStreamReader**
extends Reader

An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and translates them into characters according to a specified character encoding. The encoding that it uses may be specified by name, or the platform's default encoding may be accepted.

Each invocation of one of an InputStreamReader's read() methods may cause one or more bytes to be read from the underlying byte-input stream. To enable the efficient conversion of bytes to characters, more bytes may be read ahead from the underlying stream than are necessary to satisfy the current read operation.

For top efficiency, consider wrapping an InputStreamReader within a BufferedReader. For example:

```
BufferedReader in
  = new BufferedReader(new InputStreamReader(System.in));
```

Field Summary

protected Reader	**in** The underlying character-input stream.

Fields inherited from class java.io.Reader
lock

Constructor Summary

InputStreamReader(InputStream is) Create an InputStreamReader that uses the default character encoding.
InputStreamReader(InputStream is, String enc) Create an InputStreamReader that uses the named character encoding.

Method Summary

void	**close**() Close the stream.
void	**mark**(int readAheadLimit) Mark the present position in the stream.
boolean	**markSupported**() Tell whether this stream supports the mark() operation.
int	**read**() Read a single character.
int	**read**(char[] cbuf, int off, int len) Read characters into a portion of an array.
boolean	**ready**() Tell whether this stream is ready to be read.
void	**reset**() Reset the stream.
long	**skip**(long n) Skip characters.

Methods inherited from class java.io.Reader
read

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

in

```
protected Reader in
```

The underlying character-input stream.

Constructor Detail

InputStreamReader

```
public InputStreamReader(InputStream is)
```

Create an InputStreamReader that uses the default character encoding.
Parameters:
in - An InputStream

InputStreamReader

```
public InputStreamReader(InputStream is,
                         String enc)
                  throws UnsupportedEncodingException
```

Create an InputStreamReader that uses the named character encoding.
Parameters:
in - An InputStream
enc - The name of a supported
Throws:
UnsupportedEncodingException - If the named encoding is not supported

Method Detail

read

```
public int read()
         throws IOException
```

Read a single character.
Overrides:
read in class Reader
Throws:
IOException - If an I/O error occurs

---

read

```
public int read(char[] cbuf,
                int off,
                int len)
         throws IOException
```

Read characters into a portion of an array.
Overrides:
read in class Reader
Throws:
IOException - If an I/O error occurs

---

skip

```
public long skip(long n)
          throws IOException
```

Skip characters.
Overrides:
skip in class Reader
Throws:
IOException - If an I/O error occurs

---

ready

```
public boolean ready()
              throws IOException
```

Tell whether this stream is ready to be read.
Overrides:
ready in class Reader
Throws:
IOException - If an I/O error occurs

---

markSupported

```
public boolean markSupported()
```

Tell whether this stream supports the mark() operation.
Overrides:
markSupported in class Reader
Tags copied from class: Reader
Returns:
true if and only if this stream supports the mark operation.

---

mark

```
public void mark(int readAheadLimit)
          throws IOException
```

Mark the present position in the stream.
Overrides:
mark in class Reader
Throws:
IOException - If an I/O error occurs

---

reset

```
public void reset()
           throws IOException
```

Reset the stream.
Overrides:
reset in class Reader
Throws:
IOException - If an I/O error occurs

---

close

```
public void close()
            throws IOException
```

Close the stream.
Overrides:
close in class Reader
Throws:
IOException - If an I/O error occurs

---

java.io
Class InterruptedIOException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.InterruptedIOException
```

---

public class **InterruptedIOException**
extends IOException

Signals that an I/O operation has been interrupted. An `InterruptedIOException` is thrown to indicate that an input or output transfer has been terminated because the thread performing it was terminated. The field `bytesTransferred` indicates how many bytes were successfully transferred before the interruption occurred.

Since:
JDK1.0
See Also:
`InputStream`, `OutputStream`

Field Summary

`int`	**`bytesTransferred`** Reports how many bytes had been transferred as part of the I/O operation before it was interrupted.

Constructor Summary

`InterruptedIOException``()` Constructs an `InterruptedIOException` with `null` as its error detail message.
`InterruptedIOException``(String s)` Constructs an `InterruptedIOException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

bytesTransferred

```
public int bytesTransferred
```

Reports how many bytes had been transferred as part of the I/O operation before it was interrupted.

Constructor Detail

InterruptedIOException

```
public InterruptedIOException()
```

Constructs an `InterruptedIOException` with `null` as its error detail message.

InterruptedIOException

```
public InterruptedIOException(String s)
```

Constructs an `InterruptedIOException` with the specified detail message. The string `s` can be retrieved later by the `Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:

`s` - the detail message.

---

java.io
Class OutputStream

```
java.lang.Object
  |
  +--java.io.OutputStream
```

Direct Known Subclasses:

ByteArrayOutputStream, DataOutputStream, PrintStream

---

public abstract class **OutputStream**
extends Object

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:

JDK1.0

See Also:

`InputStream`, `write(int)`

Constructor Summary

`OutputStream()`	

Method Summary

`void`	**`close()`** Closes this output stream and releases any system resources associated with this stream.
`void`	**`flush()`** Flushes this output stream and forces any buffered output bytes to be written out.
`void`	**`write(byte[] b)`** Writes `b.length` bytes from the specified byte array to this output stream.
`void`	**`write(byte[] b, int off, int len)`** Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.
`abstract void`	**`write(int b)`** Writes the specified byte to this output stream.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

OutputStream

```
public OutputStream()
```

Method Detail

write

```
public abstract void write(int b)
                   throws IOException
```

Writes the specified byte to this output stream. The general contract for `write` is that one byte is written to the output stream. The byte to be written is the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.

Subclasses of `OutputStream` must provide an implementation for this method.

Parameters:

`b` - the `byte`.

Throws:

IOException - if an I/O error occurs. In particular, an `IOException` may be thrown if the output stream has been closed.

---

write

```
public void write(byte[] b)
          throws IOException
```

Writes `b.length` bytes from the specified byte array to this output stream. The general contract for `write(b)` is that it should have exactly the same effect as the call `write(b, 0, b.length)`.

Parameters:

`b` - the data.

Throws:

IOException - if an I/O error occurs.

See Also:

`write(byte[], int, int)`

---

write

```
public void write(byte[] b,
                  int off,
                  int len)
           throws IOException
```

Writes `len` bytes from the specified byte array starting at offset `off` to this output stream. The general contract for `write(b, off, len)` is that some of the bytes in the array `b` are written to the output stream in order; element `b[off]` is the first byte written and `b[off+len-1]` is the last byte written by this operation.

The `write` method of `OutputStream` calls the write method of one argument on each of the bytes to be written out. Subclasses are encouraged to override this method and provide a more efficient implementation.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

Parameters:

`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

Throws:

IOException - if an I/O error occurs. In particular, an `IOException` is thrown if the output stream is closed.

---

flush

```
public void flush()
          throws IOException
```

Flushes this output stream and forces any buffered output bytes to be written out. The general contract of `flush` is that calling it is an indication that, if any bytes previously written have been buffered by the implementation of the output stream, such bytes should immediately be written to their intended destination.

The `flush` method of `OutputStream` does nothing.

Throws:

IOException - if an I/O error occurs.

---

close

```
public void close()
          throws IOException
```

Closes this output stream and releases any system resources associated with this stream. The general contract of `close` is that it closes the output stream. A closed stream cannot perform output operations and cannot be reopened.

The `close` method of `OutputStream` does nothing.

Throws:

IOException - if an I/O error occurs.

java.io
Class OutputStreamWriter

```
java.lang.Object
  |
  +--java.io.Writer
        |
        +--java.io.OutputStreamWriter
```

public class **OutputStreamWriter**
extends Writer

An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are translated into bytes according to a specified character encoding. The encoding that it uses may be specified by name, or the platform's default encoding may be accepted.

Each invocation of a write() method causes the encoding converter to be invoked on the given character(s). The resulting bytes are accumulated in a buffer before being written to the underlying output stream. The size of this buffer may be specified, but by default it is large enough for most purposes. Note that the characters passed to the write() methods are not buffered.

For top efficiency, consider wrapping an OutputStreamWriter within a BufferedWriter so as to avoid frequent converter invocations. For example:

```
Writer out
  = new BufferedWriter(new OutputStreamWriter(System.out));
```

Field Summary

protected Writer	**out** The underlying character-output stream.

Fields inherited from class java.io.Writer

`lock`

Constructor Summary

OutputStreamWriter(OutputStream os) Create an OutputStreamWriter that uses the default character encoding.
OutputStreamWriter(OutputStream os, String enc) Create an OutputStreamWriter that uses the named character encoding.

Method Summary

void	**close**() Close the stream.
void	**flush**() Flush the stream.
void	**write**(char[] cbuf, int off, int len) Write a portion of an array of characters.
void	**write**(int c) Write a single character.
void	**write**(String str, int off, int len) Write a portion of a string.

Methods inherited from class java.io.Writer

write, write

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

out

```
protected Writer out
```

The underlying character-output stream.

Constructor Detail

OutputStreamWriter

```
public OutputStreamWriter(OutputStream os)
```

Create an OutputStreamWriter that uses the default character encoding.

Parameters:

out - An OutputStream

OutputStreamWriter

```
public OutputStreamWriter(OutputStream os,
                          String enc)
                   throws UnsupportedEncodingException
```

Create an OutputStreamWriter that uses the named character encoding.

Parameters:

out - An OutputStream

enc - The name of a supported

Throws:

UnsupportedEncodingException - If the named encoding is not supported

Method Detail

write

```
public void write(int c)
           throws IOException
```

Write a single character.

Overrides:

write in class Writer

Throws:

IOException - If an I/O error occurs

write

```
public void write(char[] cbuf,
                  int off,
                  int len)
           throws IOException
```

Write a portion of an array of characters.

Overrides:

write in class Writer

Parameters:

cbuf - Buffer of characters to be written

off - Offset from which to start reading characters

len - Number of characters to be written

Throws:

IOException - If an I/O error occurs

write

```
public void write(String str,
                  int off,
                  int len)
           throws IOException
```

Write a portion of a string.

Overrides:

write in class Writer

Parameters:

`str` - String to be written

`off` - Offset from which to start reading characters

`len` - Number of characters to be written

Throws:

IOException - If an I/O error occurs

---

flush

```
public void flush()
            throws IOException
```

Flush the stream.

Overrides:

flush in class Writer

Throws:

IOException - If an I/O error occurs

---

close

```
public void close()
            throws IOException
```

Close the stream.

Overrides:

close in class Writer

Throws:

IOException - If an I/O error occurs

---

java.io
Class PrintStream

```
java.lang.Object
  |
  +--java.io.OutputStream
        |
        +--java.io.PrintStream
```

---

public class **PrintStream**
extends OutputStream

A `PrintStream` adds functionality to another output stream, namely the ability to print representations of various data values conveniently. Two other features are provided as well. Unlike other output streams, a `PrintStream` never throws an `IOException`; instead, exceptional situations merely set an internal flag that can be tested via the `checkError` method. Optionally, a `PrintStream` can be created so as to flush automatically; this means that the `flush` method is automatically invoked after a byte array is written, one of the `println` methods is invoked, or a newline character or byte (`'\n'`) is written.

All characters printed by a `PrintStream` are converted into bytes using the platform's default character encoding.

Since:

JDK1.0

Constructor Summary

`PrintStream``(OutputStream out)` Create a new print stream.	

Method Summary

`boolean`	**`checkError`**`()` Flush the stream and check its error state.
`void`	**`close`**`()` Close the stream.
`void`	**`flush`**`()` Flush the stream.
`void`	**`print`**`(boolean b)` Print a boolean value.
`void`	**`print`**`(char c)` Print a character.

void	**print**(char[] s) Print an array of characters.
void	**print**(int i) Print an integer.
void	**print**(long l) Print a long integer.
void	**print**(Object obj) Print an object.
void	**print**(String s) Print a string.
void	**println**() Terminate the current line by writing the line separator string.
void	**println**(boolean x) Print a boolean and then terminate the line.
void	**println**(char x) Print a character and then terminate the line.
void	**println**(char[] x) Print an array of characters and then terminate the line.
void	**println**(int x) Print an integer and then terminate the line.
void	**println**(long x) Print a long and then terminate the line.
void	**println**(Object x) Print an Object and then terminate the line.
void	**println**(String x) Print a String and then terminate the line.
protected void	**setError**() Set the error state of the stream to true.
void	**write**(byte[] buf, int off, int len) Write len bytes from the specified byte array starting at offset off to this stream.
void	**write**(int b) Write the specified byte to this stream.

Methods inherited from class java.io.OutputStream
write

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PrintStream

```
public PrintStream(OutputStream out)
```

Create a new print stream. This stream will not flush automatically.

Parameters:

out - The output stream to which values and objects will be printed

Method Detail

flush

```
public void flush()
```

Flush the stream. This is done by writing any buffered output bytes to the underlying output stream and then flushing that stream.

Overrides:

flush in class OutputStream

See Also:

OutputStream.flush()

---

close

```
public void close()
```

Close the stream. This is done by flushing the stream and then closing the underlying output stream.

Overrides:

close in class OutputStream

See Also:

OutputStream.close()

---

checkError

```
public boolean checkError()
```

Flush the stream and check its error state. The internal error state is set to true when the underlying output stream throws an IOException, and when the setError method is invoked.

Returns:
True if and only if this stream has encountered an `IOException`, or the `setError` method has been invoked

---

setError

```
protected void setError()
```

Set the error state of the stream to `true`.
Since:
JDK1.1

---

write

```
public void write(int b)
```

Write the specified byte to this stream. If the byte is a newline and automatic flushing is enabled then the `flush` method will be invoked.

Note that the byte is written as given; to write a character that will be translated according to the platform's default character encoding, use the `print(char)` or `println(char)` methods.
Overrides:
write in class OutputStream
Parameters:
`b` - The byte to be written
See Also:
`print(char)`, `println(char)`

---

write

```
public void write(byte[] buf,
                  int off,
                  int len)
```

Write `len` bytes from the specified byte array starting at offset `off` to this stream. If automatic flushing is enabled then the `flush` method will be invoked.

Note that the bytes will be written as given; to write characters that will be translated according to the platform's default character encoding, use the `print(char)` or `println(char)` methods.
Overrides:
write in class OutputStream
Parameters:
`buf` - A byte array
`off` - Offset from which to start taking bytes
`len` - Number of bytes to write

---

print

```
public void print(boolean b)
```

Print a boolean value. The string produced by `String.valueOf(boolean)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.
Parameters:
`b` - The `boolean` to be printed

---

print

```
public void print(char c)
```

Print a character. The character is translated into one or more bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.
Parameters:
`c` - The `char` to be printed

---

print

```
public void print(int i)
```

Print an integer. The string produced by `String.valueOf(int)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.
Parameters:
`i` - The `int` to be printed
See Also:
`Integer.toString(int)`

---

print

```
public void print(long l)
```

Print a long integer. The string produced by `String.valueOf(long)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.
Parameters:
`l` - The `long` to be printed
See Also:
`Long.toString(long)`

---

print

```
public void print(char[] s)
```

Print an array of characters. The characters are converted into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`s` - The array of chars to be printed

Throws:

NullPointerException - If `s` is `null`

---

print

```
public void print(String s)
```

Print a string. If the argument is `null` then the string `"null"` is printed. Otherwise, the string's characters are converted into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`s` - The `String` to be printed

---

print

```
public void print(Object obj)
```

Print an object. The string produced by the `String.valueOf(Object)` method is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`obj` - The `Object` to be printed

See Also:

`Object.toString()`

---

println

```
public void println()
```

Terminate the current line by writing the line separator string. The line separator string is defined by the system property `line.separator`, and is not necessarily a single newline character (`'\n'`).

---

println

```
public void println(boolean x)
```

Print a boolean and then terminate the line. This method behaves as though it invokes `print(boolean)` and then `println()`.

Parameters:

`x` - The `boolean` to be printed

---

println

```
public void println(char x)
```

Print a character and then terminate the line. This method behaves as though it invokes `print(char)` and then `println()`.

Parameters:

`x` - The `char` to be printed.

---

println

```
public void println(int x)
```

Print an integer and then terminate the line. This method behaves as though it invokes `print(int)` and then `println()`.

Parameters:

`x` - The `int` to be printed.

---

println

```
public void println(long x)
```

Print a long and then terminate the line. This method behaves as though it invokes `print(long)` and then `println()`.

Parameters:

`x` - a The `long` to be printed.

---

println

```
public void println(char[] x)
```

Print an array of characters and then terminate the line. This method behaves as though it invokes `print(char[])` and then `println()`.

Parameters:

`x` - an array of chars to print.

---

println

```
public void println(String x)
```

Print a String and then terminate the line. This method behaves as though it invokes `print(String)` and then `println()`.

Parameters:

`x` - The `String` to be printed.

println

```
public void println(Object x)
```

Print an Object and then terminate the line. This method behaves as though it invokes `print(Object)` and then `println()`.

Parameters:

`x` - The `Object` to be printed.

java.io
Class Reader

```
java.lang.Object
  |
  +--java.io.Reader
```

Direct Known Subclasses:

InputStreamReader

public abstract class **Reader**
extends Object

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

See Also:

`InputStreamReader`, `Writer`

Field Summary

`protected Object`	**`lock`** The object used to synchronize operations on this stream.

Constructor Summary

`protected`	**`Reader()`** Create a new character-stream reader whose critical sections will synchronize on the reader itself.
`protected`	**`Reader(Object lock)`** Create a new character-stream reader whose critical sections will synchronize on the given object.

Method Summary

abstract void	**close**() Close the stream.
void	**mark**(int readAheadLimit) Mark the present position in the stream.
boolean	**markSupported**() Tell whether this stream supports the mark() operation.
int	**read**() Read a single character.
int	**read**(char[] cbuf) Read characters into an array.
abstract int	**read**(char[] cbuf, int off, int len) Read characters into a portion of an array.
boolean	**ready**() Tell whether this stream is ready to be read.
void	**reset**() Reset the stream.
long	**skip**(long n) Skip characters.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

lock

```
protected Object lock
```

The object used to synchronize operations on this stream. For efficiency, a character-stream object may use an object other than itself to protect critical sections. A subclass should therefore use the object in this field rather than `this` or a synchronized method.

Constructor Detail

Reader

```
protected Reader()
```

Create a new character-stream reader whose critical sections will synchronize on the reader itself.

Reader

```
protected Reader(Object lock)
```

Create a new character-stream reader whose critical sections will synchronize on the given object.

Parameters:

`lock` - The Object to synchronize on.

Method Detail

read

```
public int read()
         throws IOException
```

Read a single character. This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.

Subclasses that intend to support efficient single-character input should override this method.

Returns:

The character read, as an integer in the range 0 to 65535 (`0x00-0xffff`), or -1 if the end of the stream has been reached

Throws:

IOException - If an I/O error occurs

---

read

```
public int read(char[] cbuf)
         throws IOException
```

Read characters into an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

Parameters:

`cbuf` - Destination buffer

Returns:

The number of bytes read, or -1 if the end of the stream has been reached

Throws:

IOException - If an I/O error occurs

---

read

```
public abstract int read(char[] cbuf,
                         int off,
                         int len)
                  throws IOException
```

Read characters into a portion of an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

Parameters:

`cbuf` - Destination buffer

`off` - Offset at which to start storing characters

`len` - Maximum number of characters to read

Returns:

The number of characters read, or -1 if the end of the stream has been reached

Throws:

IOException - If an I/O error occurs

---

skip

```
public long skip(long n)
          throws IOException
```

Skip characters. This method will block until some characters are available, an I/O error occurs, or the end of the stream is reached.

Parameters:

`n` - The number of characters to skip

Returns:

The number of characters actually skipped

Throws:

IllegalArgumentException - If `n` is negative.

IOException - If an I/O error occurs

---

ready

```
public boolean ready()
              throws IOException
```

Tell whether this stream is ready to be read.

Returns:

True if the next read() is guaranteed not to block for input, false otherwise. Note that returning false does not guarantee that the next read will block.

Throws:

IOException - If an I/O error occurs

---

markSupported

```
public boolean markSupported()
```


Tell whether this stream supports the mark() operation. The default implementation always returns false. Subclasses should override this method.

Returns:

true if and only if this stream supports the mark operation.

---

mark

```
public void mark(int readAheadLimit)
          throws IOException
```

Mark the present position in the stream. Subsequent calls to reset() will attempt to reposition the stream to this point. Not all character-input streams support the mark() operation.

Parameters:

`readAheadLimit` - Limit on the number of characters that may be read while still preserving the mark. After reading this many characters, attempting to reset the stream may fail.

Throws:

IOException - If the stream does not support mark(), or if some other I/O error occurs

---

reset

```
public void reset()
           throws IOException
```

Reset the stream. If the stream has been marked, then attempt to reposition it at the mark. If the stream has not been marked, then attempt to reset it in some way appropriate to the particular stream, for example by repositioning it to its starting point. Not all character-input streams support the reset() operation, and some support reset() without supporting mark().

Throws:

IOException - If the stream has not been marked, or if the mark has been invalidated, or if the stream does not support reset(), or if some other I/O error occurs

---

close

```
public abstract void close()
                  throws IOException
```

Close the stream. Once a stream has been closed, further read(), ready(), mark(), or reset() invocations will throw an IOException. Closing a previously-closed stream, however, has no effect.

Throws:

IOException - If an I/O error occurs

---

java.io
Class UTFDataFormatException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.UTFDataFormatException
```

public class **UTFDataFormatException**
extends IOException

Signals that a malformed UTF-8 string has been read in a data input stream or by any class that implements the data input interface. See the `writeUTF` method for the format in which UTF-8 strings are read and written.

Since:
JDK1.0
See Also:
`DataInput`, `DataInputStream.readUTF(java.io.DataInput)`, `IOException`

Constructor Summary

`UTFDataFormatException``()` Constructs a `UTFDataFormatException` with `null` as its error detail message.
`UTFDataFormatException``(String s)` Constructs a `UTFDataFormatException` with the specified detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

UTFDataFormatException

`public` **`UTFDataFormatException`**`()`

Constructs a `UTFDataFormatException` with `null` as its error detail message.

UTFDataFormatException

`public` **`UTFDataFormatException`**`(String s)`

Constructs a `UTFDataFormatException` with the specified detail message. The string `s` can be retrieved later by the `Throwable.getMessage()` method of class `java.lang.Throwable`.
Parameters:
`s` - the detail message.

java.io
Class UnsupportedEncodingException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.UnsupportedEncodingException
```

public class **UnsupportedEncodingException**
extends IOException

The Character Encoding is not supported.

Since:
JDK1.1

Constructor Summary

`UnsupportedEncodingException()` Constructs an UnsupportedEncodingException without a detail message.
`UnsupportedEncodingException(String s)` Constructs an UnsupportedEncodingException with a detail message.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

UnsupportedEncodingException

`public` **`UnsupportedEncodingException`**`()`

Constructs an UnsupportedEncodingException without a detail message.

UnsupportedEncodingException

`public` **`UnsupportedEncodingException`**`(String s)`

Constructs an UnsupportedEncodingException with a detail message.
Parameters:
`s` - Describes the reason for the exception.

java.io
Class Writer

```
java.lang.Object
  |
  +--java.io.Writer
```

Direct Known Subclasses:
OutputStreamWriter

public abstract class **Writer**
extends Object

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:
JDK1.1
See Also:
`Writer`, `OutputStreamWriter`, `Reader`

Field Summary

`protected Object`	**`lock`** The object used to synchronize operations on this stream.

Constructor Summary

`protected`	**`Writer()`** Create a new character-stream writer whose critical sections will synchronize on the writer itself.
`protected`	**`Writer(Object lock)`** Create a new character-stream writer whose critical sections will synchronize on the given object.

Method Summary

`abstract void`	**`close()`** Close the stream, flushing it first.
`abstract void`	**`flush()`** Flush the stream.
`void`	**`write(char[] cbuf)`** Write an array of characters.
`abstract void`	**`write(char[] cbuf, int off, int len)`** Write a portion of an array of characters.
`void`	**`write(int c)`** Write a single character.
`void`	**`write(String str)`** Write a string.
`void`	**`write(String str, int off, int len)`** Write a portion of a string.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

lock

`protected Object` **`lock`**

The object used to synchronize operations on this stream. For efficiency, a character-stream object may use an object other than itself to protect critical sections. A subclass should therefore use the object in this field rather than `this` or a synchronized method.

Constructor Detail

Writer

`protected` **`Writer()`**

Create a new character-stream writer whose critical sections will synchronize on the writer itself.

Writer

```
protected Writer(Object lock)
```

Create a new character-stream writer whose critical sections will synchronize on the given object.
Parameters:
`lock` - Object to synchronize on.

Method Detail

write

```
public void write(int c)
           throws IOException
```

Write a single character. The character to be written is contained in the 16 low-order bits of the given integer value; the 16 high-order bits are ignored.

Subclasses that intend to support efficient single-character output should override this method.
Parameters:
`c` - int specifying a character to be written.
Throws:
IOException - If an I/O error occurs

---

write

```
public void write(char[] cbuf)
           throws IOException
```

Write an array of characters.
Parameters:
`cbuf` - Array of characters to be written
Throws:
IOException - If an I/O error occurs

---

write

```
public abstract void write(char[] cbuf,
                           int off,
                           int len)
                    throws IOException
```

Write a portion of an array of characters.
Parameters:
`cbuf` - Array of characters
`off` - Offset from which to start writing characters
`len` - Number of characters to write
Throws:
IOException - If an I/O error occurs

write

```
public void write(String str)
           throws IOException
```

Write a string.
Parameters:
`str` - String to be written
Throws:
IOException - If an I/O error occurs

---

write

```
public void write(String str,
                  int off,
                  int len)
           throws IOException
```

Write a portion of a string.
Parameters:
`str` - A String
`off` - Offset from which to start writing characters
`len` - Number of characters to write
Throws:
IOException - If an I/O error occurs

---

flush

```
public abstract void flush()
                    throws IOException
```

Flush the stream. If the stream has saved any characters from the various write() methods in a buffer, write them immediately to their intended destination. Then, if that destination is another character or byte stream, flush it. Thus one flush() invocation will flush all the buffers in a chain of Writers and OutputStreams.
Throws:
IOException - If an I/O error occurs

---

close

```
public abstract void close()
                    throws IOException
```

Close the stream, flushing it first. Once a stream has been closed, further write() or flush() invocations will cause an IOException to be thrown. Closing a previously-closed stream, however, has no effect.
Throws:
IOException - If an I/O error occurs

Package java.util

Interface Summary	
Enumeration	An object that implements the Enumeration interface generates a series of elements, one at a time.

Class Summary	
Calendar	This is a very much slimmed down version of the J2SE Calendar class for J2ME.
Hashtable	This class implements a hashtable, which maps keys to values.
Random	An instance of this class is used to generate a stream of pseudorandom numbers.
Stack	The `Stack` class represents a last-in-first-out (LIFO) stack of objects.
Vector	The `Vector` class implements a growable array of objects.

Exception Summary	
EmptyStackException	Thrown by methods in the `Stack` class to indicate that the stack is empty.
NoSuchElementException	Thrown by the `nextElement` method of an `Enumeration` to indicate that there are no more elements in the enumeration.

java.util
Class Calendar

```
java.lang.Object
  |
  +--java.util.Calendar
```

public class **Calendar**
extends Object

This is a very much slimmed down version of the J2SE Calendar class for J2ME. It is a pure subset of the original class and all the field offset and returned values are the same so programs that use this class should also be able to run unchanged on J2SE.

Field Summary

static int	**APRIL**
static int	**AUGUST**
static int	**DAY_OF_MONTH**
static int	**DECEMBER**
static int	**FEBRUARY**
static int	**FRIDAY**
static int	**HOUR**
static int	**JANUARY**
static int	**JULY**
static int	**JUNE**
static int	**MARCH**
static int	**MAY**
static int	**MILLISECOND**

static int	**MINUTE**
static int	**MONDAY**
static int	**MONTH**
static int	**NOVEMBER**
static int	**OCTOBER**
static int	**SATURDAY**
static int	**SECOND**
static int	**SEPTEMBER**
static int	**SUNDAY**
static int	**THURSDAY**
static int	**TUESDAY**
static int	**WEDNESDAY**
static int	**YEAR**

Constructor Summary

protected	**Calendar**() This private array is used to communicate with the native code

Method Summary

int	**get**(int field) Gets the value for a given time field.
static Calendar	**getInstance**() Gets a calendar using the default time zone and locale.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

YEAR

public static final int **YEAR**

MONTH

public static final int **MONTH**

DAY_OF_MONTH

public static final int **DAY_OF_MONTH**

HOUR

public static final int **HOUR**

MINUTE

public static final int **MINUTE**

SECOND

public static final int **SECOND**

MILLISECOND

public static final int **MILLISECOND**

SUNDAY

public static final int **SUNDAY**

MONDAY

public static final int **MONDAY**

TUESDAY

public static final int **TUESDAY**

WEDNESDAY

public static final int **WEDNESDAY**

THURSDAY

public static final int **THURSDAY**

FRIDAY

public static final int **FRIDAY**

SATURDAY

public static final int **SATURDAY**

JANUARY

public static final int **JANUARY**

FEBRUARY

public static final int **FEBRUARY**

MARCH

public static final int **MARCH**

APRIL

public static final int **APRIL**

MAY

```
public static final int MAY
```

JUNE

```
public static final int JUNE
```

JULY

```
public static final int JULY
```

AUGUST

```
public static final int AUGUST
```

SEPTEMBER

```
public static final int SEPTEMBER
```

OCTOBER

```
public static final int OCTOBER
```

NOVEMBER

```
public static final int NOVEMBER
```

DECEMBER

```
public static final int DECEMBER
```

Constructor Detail

Calendar

```
protected Calendar()
```

This private array is used to communicate with the native code

Method Detail

getInstance

```
public static Calendar getInstance()
```

Gets a calendar using the default time zone and locale.
Returns:
a Calendar.

---

get

```
public final int get(int field)
```

Gets the value for a given time field.
Parameters:
`field` - the given time field.
Returns:
the value for the given time field.

---

java.util
Class EmptyStackException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.util.EmptyStackException
```

public class **EmptyStackException**
extends RuntimeException

Thrown by methods in the `Stack` class to indicate that the stack is empty.

Since:
JDK1.0
See Also:
`Stack`

Constructor Summary

`EmptyStackException()` Constructs a new `EmptyStackException` with `null` as its error message string.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

EmptyStackException

`public EmptyStackException()`

Constructs a new `EmptyStackException` with `null` as its error message string.

java.util
Interface Enumeration

public interface **Enumeration**

An object that implements the Enumeration interface generates a series of elements, one at a time. Successive calls to the `nextElement` method return successive elements of the series.

For example, to print all elements of a vector *v*:

```
for (Enumeration e = v.elements() ; e.hasMoreElements() ;) {
    System.out.println(e.nextElement());     }
```

Methods are provided to enumerate through the elements of a vector, the keys of a hashtable, and the values in a hashtable.

Since:
JDK1.0
See Also:
`nextElement()`, `Hashtable`, `Hashtable.elements()`, `Hashtable.keys()`, `Vector`, `Vector.elements()`

Method Summary

`boolean`	**`hasMoreElements`**`()` Tests if this enumeration contains more elements.
`Object`	**`nextElement`**`()` Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Method Detail

hasMoreElements

```
public boolean hasMoreElements()
```

Tests if this enumeration contains more elements.
Returns:
`true` if and only if this enumeration object contains at least one more element to provide; `false` otherwise.

nextElement

```
public Object nextElement()
```

Returns the next element of this enumeration if this enumeration object has at least one more element to provide.
Returns:
the next element of this enumeration.
Throws:
NoSuchElementException - if no more elements exist.

java.util
Class Hashtable

```
java.lang.Object
  |
  +--java.util.Hashtable
```

public class **Hashtable**
extends Object

This class implements a hashtable, which maps keys to values. Any non-`null` object can be used as a key or as a value.

To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the `hashCode` method and the `equals` method.

An instance of `Hashtable` has two parameters that affect its efficiency: its *capacity* and its *load factor*. The load factor should be between 0.0 and 1.0. When the number of entries in the hashtable exceeds the product of the load factor and the current capacity, the capacity is increased by calling the `rehash` method. Larger load factors use memory more efficiently, at the expense of larger expected time per lookup.

If many entries are to be made into a `Hashtable`, creating it with a sufficiently large capacity may allow the entries to be inserted more efficiently than letting it perform automatic rehashing as needed to grow the table.

This example creates a hashtable of numbers. It uses the names of the numbers as keys:

```
Hashtable numbers = new Hashtable();
numbers.put("one", new Integer(1));
numbers.put("two", new Integer(2));
numbers.put("three", new Integer(3));
```

To retrieve a number, use the following code:

```
Integer n = (Integer)numbers.get("two");
if (n != null) {
    System.out.println("two = " + n);
}
```

Since:
JDK1.0
See Also:
`Object.equals(java.lang.Object)`, `Object.hashCode()`, `rehash()`

Constructor Summary

`Hashtable``()`	Constructs a new, empty hashtable with a default capacity and load factor.
`Hashtable``(int initialCapacity)`	Constructs a new, empty hashtable with the specified initial capacity and the specified load factor.

Method Summary

`void`	**`clear`**`()` Clears this hashtable so that it contains no keys.
`boolean`	**`contains`**`(Object value)` Tests if some key maps into the specified value in this hashtable.
`boolean`	**`containsKey`**`(Object key)` Tests if the specified object is a key in this hashtable.
`Enumeration`	**`elements`**`()` Returns an enumeration of the values in this hashtable.
`Object`	**`get`**`(Object key)` Returns the value to which the specified key is mapped in this hashtable.
`boolean`	**`isEmpty`**`()` Tests if this hashtable maps no keys to values.
`Enumeration`	**`keys`**`()` Returns an enumeration of the keys in this hashtable.
`Object`	**`put`**`(Object key, Object value)` Maps the specified `key` to the specified `value` in this hashtable.
`protected void`	**`rehash`**`()` Rehashes the contents of the hashtable into a hashtable with a larger capacity.
`Object`	**`remove`**`(Object key)` Removes the key (and its corresponding value) from this hashtable.
`int`	**`size`**`()` Returns the number of keys in this hashtable.
`String`	**`toString`**`()` Returns a rather long string representation of this hashtable.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

Hashtable

```
public Hashtable(int initialCapacity)
```

Constructs a new, empty hashtable with the specified initial capacity and the specified load factor.
Parameters:
`initialCapacity` - the initial capacity of the hashtable.
Throws:
IllegalArgumentException - if the initial capacity is less than zero
Since:
JDK1.0

Hashtable

```
public Hashtable()
```

Constructs a new, empty hashtable with a default capacity and load factor.
Since:
JDK1.0

Method Detail

size

```
public int size()
```

Returns the number of keys in this hashtable.
Returns:
the number of keys in this hashtable.
Since:
JDK1.0

---

isEmpty

```
public boolean isEmpty()
```

Tests if this hashtable maps no keys to values.
Returns:
`true` if this hashtable maps no keys to values; `false` otherwise.
Since:
JDK1.0

---

keys

```
public Enumeration keys()
```

Returns an enumeration of the keys in this hashtable.
Returns:
an enumeration of the keys in this hashtable.
Since:
JDK1.0
See Also:
`Enumeration`, `elements()`

---

elements

```
public Enumeration elements()
```

Returns an enumeration of the values in this hashtable. Use the Enumeration methods on the returned object to fetch the elements sequentially.
Returns:
an enumeration of the values in this hashtable.
Since:
JDK1.0
See Also:
`Enumeration`, `keys()`

---

contains

```
public boolean contains(Object value)
```

Tests if some key maps into the specified value in this hashtable. This operation is more expensive than the `containsKey` method.
Parameters:
`value` - a value to search for.
Returns:
`true` if some key maps to the `value` argument in this hashtable; `false` otherwise.
Throws:
NullPointerException - if the value is `null`.
Since:
JDK1.0
See Also:
`containsKey(java.lang.Object)`

---

containsKey

```
public boolean containsKey(Object key)
```

Tests if the specified object is a key in this hashtable.
Parameters:
`key` - possible key.

Returns:
`true` if the specified object is a key in this hashtable; `false` otherwise.
Since:
JDK1.0
See Also:
`contains(java.lang.Object)`

---

get

```
public Object get(Object key)
```

Returns the value to which the specified key is mapped in this hashtable.
Parameters:
`key` - a key in the hashtable.
Returns:
the value to which the key is mapped in this hashtable; `null` if the key is not mapped to any value in this hashtable.
Since:
JDK1.0
See Also:
`put(java.lang.Object, java.lang.Object)`

---

rehash

```
protected void rehash()
```

Rehashes the contents of the hashtable into a hashtable with a larger capacity. This method is called automatically when the number of keys in the hashtable exceeds this hashtable's capacity and load factor.
Since:
JDK1.0

---

put

```
public Object put(Object key,
                  Object value)
```

Maps the specified `key` to the specified `value` in this hashtable. Neither the key nor the value can be `null`.

The value can be retrieved by calling the `get` method with a key that is equal to the original key.
Parameters:
`key` - the hashtable key.
`value` - the value.
Returns:
the previous value of the specified key in this hashtable, or `null` if it did not have one.
Throws:
NullPointerException - if the key or value is `null`.

Since:
JDK1.0
See Also:
`Object.equals(java.lang.Object)`, `get(java.lang.Object)`

---

remove

```
public Object remove(Object key)
```

Removes the key (and its corresponding value) from this hashtable. This method does nothing if the key is not in the hashtable.
Parameters:
`key` - the key that needs to be removed.
Returns:
the value to which the key had been mapped in this hashtable, or `null` if the key did not have a mapping.
Since:
JDK1.0

---

clear

```
public void clear()
```

Clears this hashtable so that it contains no keys.
Since:
JDK1.0

---

toString

```
public String toString()
```

Returns a rather long string representation of this hashtable.
Overrides:
toString in class Object
Returns:
a string representation of this hashtable.
Since:
JDK1.0

---

java.util
Class NoSuchElementException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.util.NoSuchElementException
```

public class **NoSuchElementException**
extends RuntimeException

Thrown by the `nextElement` method of an `Enumeration` to indicate that there are no more elements in the enumeration.

Since:
JDK1.0
See Also:
`Enumeration`, `Enumeration.nextElement()`

Constructor Summary

`NoSuchElementException``()` Constructs a `NoSuchElementException` with `null` as its error message string.
`NoSuchElementException``(String s)` Constructs a `NoSuchElementException`, saving a reference to the error message string `s` for later retrieval by the `getMessage` method.

Methods inherited from class java.lang.Throwable
`getMessage, printStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

NoSuchElementException

`public` **`NoSuchElementException`**`()`

Constructs a `NoSuchElementException` with `null` as its error message string.

NoSuchElementException

`public` **`NoSuchElementException`**`(String s)`

Constructs a `NoSuchElementException`, saving a reference to the error message string `s` for later retrieval by the `getMessage` method.
Parameters:
`s` - the detail message.

java.util
Class Random

```
java.lang.Object
  |
  +--java.util.Random
```

public class **Random**
extends Object

An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. (See Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1.)

If two instances of `Random` are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers. In order to guarantee this property, particular algorithms are specified for the class `Random`. Java implementations must use all the algorithms shown here for the class `Random`, for the sake of absolute portability of Java code. However, subclasses of class `Random` are permitted to use other algorithms, so long as they adhere to the general contracts for all the methods.

The algorithms implemented by class `Random` use a `protected` utility method that on each invocation can supply up to 32 pseudorandomly generated bits.

Many applications will find the `random` method in class `Math` simpler to use.

Since:
JDK1.0

Constructor Summary

`Random()` Creates a new random number generator.	
`Random(long seed)` Creates a new random number generator using a single `long` seed:	

Method Summary

`protected int`	**`next(int bits)`** Generates the next pseudorandom number.
`int`	**`nextInt()`** Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.
`long`	**`nextLong()`** Returns the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence.
`void`	**`setSeed(long seed)`** Sets the seed of this random number generator using a single `long` seed.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Random

`public Random()`

Creates a new random number generator. Its seed is initialized to a value based on the current time:

```
public Random() { this(System.currentTimeMillis()); }
```

See Also:
`System.currentTimeMillis()`

Random

`public Random(long seed)`

Creates a new random number generator using a single `long` seed:

```
public Random(long seed) { setSeed(seed); }
```

Used by method `next` to hold the state of the pseudorandom number generator.
Parameters:
`seed` - the initial seed.

See Also:
`setSeed(long)`

Method Detail

setSeed

```
public void setSeed(long seed)
```

Sets the seed of this random number generator using a single `long` seed. The general contract of `setSeed` is that it alters the state of this random number generator object so as to be in exactly the same state as if it had just been created with the argument `seed` as a seed. The method `setSeed` is implemented by class Random as follows:

```
synchronized public void setSeed(long seed) {
      this.seed = (seed ^ 0x5DEECE66DL) & ((1L << 48) - 1);
      haveNextNextGaussian = false;
}
```

The implementation of `setSeed` by class `Random` happens to use only 48 bits of the given seed. In general, however, an overriding method may use all 64 bits of the long argument as a seed value.

Parameters:
`seed` - the initial seed.

---

next

```
protected int next(int bits)
```

Generates the next pseudorandom number. Subclass should override this, as this is used by all other methods.

The general contract of `next` is that it returns an `int` value and if the argument bits is between `1` and `32` (inclusive), then that many low-order bits of the returned value will be (approximately) independently chosen bit values, each of which is (approximately) equally likely to be `0` or `1`. The method `next` is implemented by class `Random` as follows:

```
synchronized protected int next(int bits) {
      seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
      return (int)(seed >>> (48 - bits));
}
```

This is a linear congruential pseudorandom number generator, as defined by D. H. Lehmer and described by Donald E. Knuth in *The Art of Computer Programming,* Volume 2: *Seminumerical Algorithms*, section 3.2.1.

Parameters:
`bits` - random bits

Returns:
the next pseudorandom value from this random number generator's sequence.

Since:
JDK1.1

---

nextInt

```
public int nextInt()
```

Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence. The general contract of `nextInt` is that one `int` value is pseudorandomly generated and returned. All 2^{32} possible `int` values are produced with (approximately) equal probability. The method `nextInt` is implemented by class `Random` as follows:

```
public int nextInt() {  return next(32); }
```

Returns:
the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.

---

nextLong

```
public long nextLong()
```

Returns the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence. The general contract of `nextLong` is that one long value is pseudorandomly generated and returned. All 2^{64} possible `long` values are produced with (approximately) equal probability. The method `nextLong` is implemented by class `Random` as follows:

```
public long nextLong() {
      return ((long)next(32) << 32) + next(32);
}
```

Returns:
the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence.

---

java.util
Class Stack

```
java.lang.Object
  |
  +--java.util.Vector
        |
        +--java.util.Stack
```

public class **Stack**
extends Vector

The `Stack` class represents a last-in-first-out (LIFO) stack of objects. It extends class `Vector` with five operations that allow a vector to be treated as a stack. The usual `push` and `pop` operations are provided, as well as a method to `peek` at the top item on the stack, a method to test for whether the stack is `empty`, and a method to `search` the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

Since:
JDK1.0

Fields inherited from class java.util.Vector
`capacityIncrement, elementCount, elementData`

Constructor Summary

`Stack()` Creates an empty Stack.	

Method Summary

`boolean`	**`empty()`** Tests if this stack is empty.
`Object`	**`peek()`** Looks at the object at the top of this stack without removing it from the stack.
`Object`	**`pop()`** Removes the object at the top of this stack and returns that object as the value of this function.
`Object`	**`push(Object item)`** Pushes an item onto the top of this stack.
`int`	**`search(Object o)`** Returns the 1-based position where an object is on this stack.

Methods inherited from class java.util.Vector
`addElement, capacity, contains, copyInto, elementAt, elements, ensureCapacity, firstElement, indexOf, indexOf, insertElementAt, isEmpty, lastElement, lastIndexOf, lastIndexOf, removeAllElements, removeElement, removeElementAt, setElementAt, setSize, size, toString, trimToSize`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

Stack

```
public Stack()
```

Creates an empty Stack.

Method Detail

push

```
public Object push(Object item)
```

Pushes an item onto the top of this stack. This has exactly the same effect as:

```
addElement(item)
```

Parameters:
`item` - the item to be pushed onto this stack.
Returns:
the `item` argument.
See Also:
`Vector.addElement(java.lang.Object)`

---

pop

```
public Object pop()
```

Removes the object at the top of this stack and returns that object as the value of this function.
Returns:
The object at the top of this stack (the last item of the `Vector` object).
Throws:
EmptyStackException - if this stack is empty.

---

peek

```
public Object peek()
```

Looks at the object at the top of this stack without removing it from the stack.
Returns:
the object at the top of this stack (the last item of the `Vector` object).
Throws:
EmptyStackException - if this stack is empty.

---

empty

```
public boolean empty()
```

Tests if this stack is empty.
Returns:
`true` if and only if this stack contains no items; `false` otherwise.

---

search

```
public int search(Object o)
```

Returns the 1-based position where an object is on this stack. If the object `o` occurs as an item in this stack, this method returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance `1`. The `equals` method is used to compare `o` to the items in this stack.
Parameters:
`o` - the desired object.

Returns:
the 1-based position from the top of the stack where the object is located; the return value `-1` indicates that the object is not on the stack.

---

java.util
Class Vector

```
java.lang.Object
  |
  +--java.util.Vector
```

Direct Known Subclasses:
Stack

public class **Vector**
extends Object

The `Vector` class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a `Vector` can grow or shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Each vector tries to optimize storage management by maintaining a `capacity` and a `capacityIncrement`. The `capacity` is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of `capacityIncrement`. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

Since:
JDK1.0

Field Summary

`protected int`	**`capacityIncrement`** The amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity.
`protected int`	**`elementCount`** The number of valid components in the vector.
`protected Object[]`	**`elementData`** The array buffer into which the components of the vector are stored.

Constructor Summary

`Vector()` Constructs an empty vector.
`Vector(int initialCapacity)` Constructs an empty vector with the specified initial capacity.
`Vector(int initialCapacity, int capacityIncrement)` Constructs an empty vector with the specified initial capacity and capacity increment.

Method Summary

`void`	**`addElement(Object obj)`** Adds the specified component to the end of this vector, increasing its size by one.
`int`	**`capacity()`** Returns the current capacity of this vector.
`boolean`	**`contains(Object elem)`** Tests if the specified object is a component in this vector.
`void`	**`copyInto(Object[] anArray)`** Copies the components of this vector into the specified array.
`Object`	**`elementAt(int index)`** Returns the component at the specified index.
`Enumeration`	**`elements()`** Returns an enumeration of the components of this vector.
`void`	**`ensureCapacity(int minCapacity)`** Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.
`Object`	**`firstElement()`** Returns the first component of this vector.
`int`	**`indexOf(Object elem)`** Searches for the first occurence of the given argument, testing for equality using the `equals` method.
`int`	**`indexOf(Object elem, int index)`** Searches for the first occurence of the given argument, beginning the search at `index`, and testing for equality using the `equals` method.
`void`	**`insertElementAt(Object obj, int index)`** Inserts the specified object as a component in this vector at the specified `index`.
`boolean`	**`isEmpty()`** Tests if this vector has no components.
`Object`	**`lastElement()`** Returns the last component of the vector.
`int`	**`lastIndexOf(Object elem)`** Returns the index of the last occurrence of the specified object in this vector.
`int`	**`lastIndexOf(Object elem, int index)`** Searches backwards for the specified object, starting from the specified index, and returns an index to it.
`void`	**`removeAllElements()`** Removes all components from this vector and sets its size to zero.

boolean	**removeElement**(Object obj) Removes the first occurrence of the argument from this vector.
void	**removeElementAt**(int index) Deletes the component at the specified index.
void	**setElementAt**(Object obj, int index) Sets the component at the specified index of this vector to be the specified object.
void	**setSize**(int newSize) Sets the size of this vector.
int	**size**() Returns the number of components in this vector.
String	**toString**() Returns a string representation of this vector.
void	**trimToSize**() Trims the capacity of this vector to be the vector's current size.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

elementData

```
protected Object[] elementData
```

The array buffer into which the components of the vector are stored. The capacity of the vector is the length of this array buffer.

Since:

JDK1.0

---

elementCount

```
protected int elementCount
```

The number of valid components in the vector.

Since:

JDK1.0

---

capacityIncrement

```
protected int capacityIncrement
```

The amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity. If the capacity increment is `0`, the capacity of the vector is doubled each time it needs to grow.

Since:

JDK1.0

Constructor Detail

Vector

```
public Vector(int initialCapacity,
              int capacityIncrement)
```

Constructs an empty vector with the specified initial capacity and capacity increment.

Parameters:

`initialCapacity` - the initial capacity of the vector.

`capacityIncrement` - the amount by which the capacity is increased when the vector overflows.

Throws:

IllegalArgumentException - if the specified initial capacity is negative

---

Vector

```
public Vector(int initialCapacity)
```

Constructs an empty vector with the specified initial capacity.

Parameters:

`initialCapacity` - the initial capacity of the vector.

Since:

JDK1.0

Vector

```
public Vector()
```

Constructs an empty vector.

Since:

JDK1.0

Method Detail

copyInto

```
public final void copyInto(Object[] anArray)
```

Copies the components of this vector into the specified array. The array must be big enough to hold all the objects in this vector.

Parameters:

`anArray` - the array into which the components get copied.

Since:

JDK1.0

---

trimToSize

```
public final void trimToSize()
```

Trims the capacity of this vector to be the vector's current size. An application can use this operation to minimize the storage of a vector.

Since:

JDK1.0

---

ensureCapacity

```
public final void ensureCapacity(int minCapacity)
```

Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

Parameters:

`minCapacity` - the desired minimum capacity.

Since:

JDK1.0

---

setSize

```
public final void setSize(int newSize)
```

Sets the size of this vector. If the new size is greater than the current size, new `null` items are added to the end of the vector. If the new size is less than the current size, all components at index `newSize` and greater are discarded.

Parameters:

`newSize` - the new size of this vector.

Since:

JDK1.0

---

capacity

```
public final int capacity()
```


Returns the current capacity of this vector.

Returns:

the current capacity of this vector.

Since:

JDK1.0

---

size

```
public final int size()
```

Returns the number of components in this vector.

Returns:

the number of components in this vector.

Since:

JDK1.0

---

isEmpty

```
public final boolean isEmpty()
```

Tests if this vector has no components.

Returns:

`true` if this vector has no components; `false` otherwise.

Since:

JDK1.0

---

elements

```
public final Enumeration elements()
```

Returns an enumeration of the components of this vector.

Returns:

an enumeration of the components of this vector.

Since:

JDK1.0

See Also:

`Enumeration`

---

contains

```
public final boolean contains(Object elem)
```

Tests if the specified object is a component in this vector.

Parameters:

`elem` - an object.

Returns:

`true` if the specified object is a component in this vector; `false` otherwise.

Since:
JDK1.0

---

indexOf

```
public final int indexOf(Object elem)
```

Searches for the first occurence of the given argument, testing for equality using the `equals` method.
Parameters:
`elem` - an object.
Returns:
the index of the first occurrence of the argument in this vector; returns `-1` if the object is not found.
Since:
JDK1.0
See Also:
`Object.equals(java.lang.Object)`

---

indexOf

```
public final int indexOf(Object elem,
                         int index)
```

Searches for the first occurence of the given argument, beginning the search at `index`, and testing for equality using the `equals` method.
Parameters:
`elem` - an object.
`index` - the index to start searching from.
Returns:
the index of the first occurrence of the object argument in this vector at position `index` or later in the vector; returns `-1` if the object is not found.
Since:
JDK1.0
See Also:
`Object.equals(java.lang.Object)`

---

lastIndexOf

```
public final int lastIndexOf(Object elem)
```

Returns the index of the last occurrence of the specified object in this vector.
Parameters:
`elem` - the desired component.
Returns:
the index of the last occurrence of the specified object in this vector; returns `-1` if the object is not found.
Since:
JDK1.0

---

lastIndexOf

```
public final int lastIndexOf(Object elem,
                             int index)
```

Searches backwards for the specified object, starting from the specified index, and returns an index to it.
Parameters:
`elem` - the desired component.
`index` - the index to start searching from.
Returns:
the index of the last occurrence of the specified object in this vector at position less than `index` in the vector; `-1` if the object is not found.
Since:
JDK1.0

---

elementAt

```
public final Object elementAt(int index)
```

Returns the component at the specified index.
Parameters:
`index` - an index into this vector.
Returns:
the component at the specified index.
Throws:
ArrayIndexOutOfBoundsException - if an invalid index was given.
Since:
JDK1.0

---

firstElement

```
public final Object firstElement()
```

Returns the first component of this vector.
Returns:
the first component of this vector.
Throws:
NoSuchElementException - if this vector has no components.
Since:
JDK1.0

---

lastElement

```
public final Object lastElement()
```

Returns the last component of the vector.

Returns:
the last component of the vector, i.e., the component at index `size() - 1`.
Throws:
NoSuchElementException - if this vector is empty.
Since:
JDK1.0

---

setElementAt

```
public final void setElementAt(Object obj,
                               int index)
```

Sets the component at the specified `index` of this vector to be the specified object. The previous component at that position is discarded.

The index must be a value greater than or equal to `0` and less than the current size of the vector.
Parameters:
`obj` - what the component is to be set to.
`index` - the specified index.
Throws:
ArrayIndexOutOfBoundsException - if the index was invalid.
Since:
JDK1.0
See Also:
`size()`

---

removeElementAt

```
public final void removeElementAt(int index)
```

Deletes the component at the specified index. Each component in this vector with an index greater or equal to the specified `index` is shifted downward to have an index one smaller than the value it had previously.

The index must be a value greater than or equal to `0` and less than the current size of the vector.
Parameters:
`index` - the index of the object to remove.
Throws:
ArrayIndexOutOfBoundsException - if the index was invalid.
Since:
JDK1.0
See Also:
`size()`

---

insertElementAt

```
public final void insertElementAt(Object obj,
                                  int index)
```


Inserts the specified object as a component in this vector at the specified `index`. Each component in this vector with an index greater or equal to the specified `index` is shifted upward to have an index one greater than the value it had previously.

The index must be a value greater than or equal to `0` and less than or equal to the current size of the vector.
Parameters:
`obj` - the component to insert.
`index` - where to insert the new component.
Throws:
ArrayIndexOutOfBoundsException - if the index was invalid.
Since:
JDK1.0
See Also:
`size()`

---

addElement

```
public final void addElement(Object obj)
```

Adds the specified component to the end of this vector, increasing its size by one. The capacity of this vector is increased if its size becomes greater than its capacity.
Parameters:
`obj` - the component to be added.
Since:
JDK1.0

---

removeElement

```
public final boolean removeElement(Object obj)
```

Removes the first occurrence of the argument from this vector. If the object is found in this vector, each component in the vector with an index greater or equal to the object's index is shifted downward to have an index one smaller than the value it had previously.
Parameters:
`obj` - the component to be removed.
Returns:
`true` if the argument was a component of this vector; `false` otherwise.
Since:
JDK1.0

---

removeAllElements

```
public final void removeAllElements()
```

Removes all components from this vector and sets its size to zero.
Since:
JDK1.0

toString

```
public final String toString()
```

Returns a string representation of this vector.

Overrides:

toString in class Object

Returns:

a string representation of this vector.

Since:

JDK1.0

Package javax.microedition.io

Interface Summary	
Connection	This is the most basic type of generic connection.
ContentConnection	This interface defines the stream connection over which content is passed.
Datagram	This is the generic datagram interface.
DatagramConnection	This interface defines the capabilities that a datagram connection must have.
InputConnection	This interface defines the capabilities that an input stream connection must have.
OutputConnection	This interface defines the capabilities that an output stream connection must have.
StreamConnection	This interface defines the capabilities that a stream connection must have.
StreamConnectionNotifier	This interface defines the capabilities that a connection notifier must have.

Class Summary	
Connector	This class is a placeholder for the static methods used to create all the connection objects.

Exception Summary	
ConnectionNotFoundException	This class is used to signal that a connection target cannot be found

javax.microedition.io
Interface Connection

All Known Subinterfaces:
ContentConnection, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier

public interface **Connection**

This is the most basic type of generic connection. Only the close method is defined. The open method defined here because opening is always done by the Connector.open() methods.

Method Summary

void	**close**() Close the connection.

Method Detail

close

```
public void close()
            throws IOException
```

Close the connection.

When the connection has been closed access to all methods except this one will cause an an IOException to be thrown. Closing an already closed connection has no effect. Streams derived from a connection may remain open after this method is called. This may cause the connection to remain open (but access to its methods are rejected) until any derived streams are closed themselves.

Throws:
IOException - If an I/O error occurs

javax.microedition.io
Class ConnectionNotFoundException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--javax.microedition.io.ConnectionNotFoundException
```

public class **ConnectionNotFoundException**
extends IOException

This class is used to signal that a connection target cannot be found

Constructor Summary

ConnectionNotFoundException() Constructs a ConnectionNotFoundException with no detail message.
ConnectionNotFoundException(String s) Constructs a ConnectionNotFoundException with the specified detail message.

Methods inherited from class java.lang.Throwable
getMessage, printStackTrace, toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

ConnectionNotFoundException

```
public ConnectionNotFoundException()
```

Constructs a ConnectionNotFoundException with no detail message. A detail message is a String that describes this particular exception.

ConnectionNotFoundException

```
public ConnectionNotFoundException(String s)
```

Constructs a ConnectionNotFoundException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:

`s` - the detail message

javax.microedition.io
Class Connector

```
java.lang.Object
  |
  +--javax.microedition.io.Connector
```

public class **Connector**
extends Object

This class is a placeholder for the static methods used to create all the connection objects.

This is done by dynamically looking up a class the name of which is formed from the platform name and the protocol of the requested connection. The parameter string describing the target conforms to the URL format as described in RFC 1808. This takes the general form:

`{scheme}:[{target}][{parms}]`

Where `{scheme}` is the name of a protocol such as *http*}.

The `{target}` is normally some kind of network address, but protocols may regard this as a fairly flexible field when the connection is not network oriented.

Any `{parms}` are formed as a series of equates on the form ";x=y" such as ;type=a.

An option second parameter may be specified to the open function. The is a mode flag that indicated to the protocol handler the intentions of the calling code. The options here are to specify if the connection is going to be read (READ), written (WRITE), or both (READ_WRITE). The validity of these flag settings is protocol dependent. For instance a connection for a printer would not allow read access, and would throw an IllegalArgumentException if this was attempted. Omitting this parameter results in READ_WRITE being used by default.

An optional third parameter is a boolean flag to indicate if the calling code has been written in such a way as to handle timeout exceptions. If this is selected the protocol may throw an InterruptedIOException when it detects a timeout condition. This flag is only a hint to the protocol handler and it is no guarantee that such exceptions will be throws. Omitting this parameter results in no exceptions being thrown. The timeout period is not specified in the open call because this is protocol specific. Protocol implementors can either hardwire an appropriate value or read them from an external source such as the system properties.

Because of the common occurrence of opening connections just to gain access to an input or output stream four functions are provided for this purpose. See also: `DatagramConnection` for information relating to datagram addressing

Field Summary	
static int	**READ** Access mode
static int	**READ_WRITE** Access mode
static int	**WRITE** Access mode

Method Summary	
static Connection	**open**(String name) Create and open a Connection
static Connection	**open**(String name, int mode) Create and open a Connection
static Connection	**open**(String name, int mode, boolean timeouts) Create and open a Connection
static DataInputStream	**openDataInputStream**(String name) Create and open a connection input stream
static DataOutputStream	**openDataOutputStream**(String name) Create and open a connection output stream
static InputStream	**openInputStream**(String name) Create and open a connection input stream
static OutputStream	**openOutputStream**(String name) Create and open a connection output stream

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

READ

```
public static final int READ
```

Access mode

WRITE

```
public static final int WRITE
```

Access mode

READ_WRITE

```
public static final int READ_WRITE
```

Access mode

Method Detail

open

```
public static Connection open(String name)
                       throws IOException
```

Create and open a Connection

Parameters:

`string` - The URL for the connection.

Returns:

A new Connection object.

Throws:

IllegalArgumentException - If a parameter is invalid.
ConnectionNotFoundException - If the connection cannot be found.
IOException - If some other kind of I/O error occurs.

open

```
public static Connection open(String name,
                              int mode)
                       throws IOException
```

Create and open a Connection

Parameters:

`string` - The URL for the connection.
`mode` - The access mode.

Returns:

A new Connection object.

Throws:

IllegalArgumentException - If a parameter is invalid.
ConnectionNotFoundException - If the connection cannot be found.
IOException - If some other kind of I/O error occurs.

open

```
public static Connection open(String name,
                              int mode,
                              boolean timeouts)
                       throws IOException
```

Create and open a Connection

Parameters:

`string` - The URL for the connection

`mode` - The access mode

`timeouts` - A flag to indicate that the called wants timeout exceptions

Returns:

A new Connection object

Throws:

IllegalArgumentException - If a parameter is invalid.

ConnectionNotFoundException - If the connection cannot be found.

IOException - If some other kind of I/O error occurs.

---

openDataInputStream

```
public static DataInputStream openDataInputStream(String name)
                                          throws IOException
```

Create and open a connection input stream

Parameters:

`string` - The URL for the connection.

Returns:

A DataInputStream.

Throws:

IllegalArgumentException - If a parameter is invalid.

ConnectionNotFoundException - If the connection cannot be found.

IOException - If some other kind of I/O error occurs.

---

openDataOutputStream

```
public static DataOutputStream openDataOutputStream(String name)
                                           throws IOException
```

Create and open a connection output stream

Parameters:

`string` - The URL for the connection.

Returns:

A DataOutputStream.

Throws:

IllegalArgumentException - If a parameter is invalid.

ConnectionNotFoundException - If the connection cannot be found.

IOException - If some other kind of I/O error occurs.

---

openInputStream

```
public static InputStream openInputStream(String name)
                                   throws IOException
```

Create and open a connection input stream

Parameters:

`string` - The URL for the connection.

Returns:

An InputStream.

Throws:

IllegalArgumentException - If a parameter is invalid.

ConnectionNotFoundException - If the connection cannot be found.

IOException - If some other kind of I/O error occurs.

---

openOutputStream

```
public static OutputStream openOutputStream(String name)
                                    throws IOException
```

Create and open a connection output stream

Parameters:

`string` - The URL for the connection.

Returns:

An OutputStream.

Throws:

IllegalArgumentException - If a parameter is invalid.

ConnectionNotFoundException - If the connection cannot be found.

IOException - If some other kind of I/O error occurs.

---

javax.microedition.io
Interface ContentConnection

public interface **ContentConnection**
extends StreamConnection

This interface defines the stream connection over which content is passed.

Method Summary

`String`	**`getEncoding`**`()` Returns a string describing the encoding of the content which the resource connected to is providing.
`long`	**`getLength`**`()` Returns the length of the content which is being provided.
`String`	**`getType`**`()` Returns the type of content that the resource connected to is providing.

Methods inherited from interface javax.microedition.io.InputConnection
`openDataInputStream, openInputStream`

Methods inherited from interface javax.microedition.io.OutputConnection
`openDataOutputStream, openOutputStream`

Method Detail

getType

`public String` **`getType`**`()`

Returns the type of content that the resource connected to is providing. E.g. if the connection is via HTTP, then the value of the `content-type` header field is returned.

Returns:

the content type of the resource that the URL references, or `null` if not known.

getEncoding

`public String` **`getEncoding`**`()`

Returns a string describing the encoding of the content which the resource connected to is providing. E.g. if the connection is via HTTP, the value of the `content-encoding` header field is returned.

Returns:

the content encoding of the resource that the URL references, or `null` if not known.

getLength

`public long` **`getLength`**`()`

Returns the length of the content which is being provided. E.g. if the connection is via HTTP, then the value of the `content-length` header field is returned.

Returns:

the content length of the resource that this connection's URL references, or `-1` if the content length is not known.

javax.microedition.io
Interface Datagram

public interface **Datagram**
extends DataInput, DataOutput

This is the generic datagram interface. It represents an object that will act as the holder of data to be send or received from a datagram connection. The DataInput and DataOutput interfaces are extended by this interface to provide a simple way to read and write binary data in and out of the datagram buffer. A special function reset() may be called to reset the read/write point to the beginning of the buffer.

Method Summary

`String`	**`getAddress`**`()` Get the address in the datagram
`byte[]`	**`getData`**`()` Get the buffer
`int`	**`getLength`**`()` Get the length
`int`	**`getOffset`**`()` Get the offset
`void`	**`reset`**`()` Reset the read/write pointer and zeros the offset and length parameters.
`void`	**`setAddress`**`(Datagram reference)` Set datagram address, copying the address from another datagram.
`void`	**`setAddress`**`(String addr)` Set datagram address.
`void`	**`setData`**`(byte[] buffer, int offset, int len)` Set the buffer, offset and length
`void`	**`setLength`**`(int len)` Set the length

Methods inherited from interface java.io.DataInput
`readBoolean, readByte, readChar, readFully, readFully, readInt, readLong, readShort, readUnsignedByte, readUnsignedShort, readUTF, skipBytes`

Methods inherited from interface java.io.DataOutput
`write, write, write, writeBoolean, writeByte, writeChar, writeChars, writeInt, writeLong, writeShort, writeUTF`

Method Detail

getAddress

```
public String getAddress()
```

Get the address in the datagram
Returns:
the address in string form, or null if no address was set

getData

```
public byte[] getData()
```

Get the buffer
Returns:
the data buffer

getLength

```
public int getLength()
```

Get the length
Returns:
the length of the data

getOffset

```
public int getOffset()
```

Get the offset
Returns:
the offset into the data buffer

setAddress

```
public void setAddress(String addr)
                throws IOException
```

Set datagram address. The parameter string describing the target of the datagram takes the form:

```
{protocol}:{target}
```

E.g. The "target" can be "//{host}:{port}" (but is not necessarily limited to this.)
So in this example a datagram connection for sending to a server could be addressed as so:

```
datagram://123.456.789.12:1234
```

Note that if the address of a datagram is not specified, then it defaults to that of the connection.

Parameters:

`addr` - the new target address as a URL

Throws:

IllegalArgumentException - if the address is not valid

---

setAddress

```
public void setAddress(Datagram reference)
```

Set datagram address, copying the address from another datagram.

Parameters:

`reference` - the datagram who's address will be copied as the new target address for this datagram.

Throws:

IllegalArgumentException - if the address is not valid

---

setLength

```
public void setLength(int len)
```

Set the length

Parameters:

`len` - the new length of the data

Throws:

IllegalArgumentException - if the length is negative or larger than the buffer

---

setData

```
public void setData(byte[] buffer,
                    int offset,
                    int len)
```

Set the buffer, offset and length

Parameters:

`addr` - the data buffer

`offset` - the offset into the data buffer

`len` - the length of the data in the buffer

Throws:

IllegalArgumentException - if the length or offset fall outside the buffer

---

reset

```
public void reset()
```

Reset the read/write pointer and zeros the offset and length parameters.

---

javax.microedition.io
Interface DatagramConnection

public interface **DatagramConnection**
extends Connection

This interface defines the capabilities that a datagram connection must have.

The parameter string describing the target of the connection takes the form:

```
{protocol}:[//{host}]:{port}
```

A datagram connection can be opened in a "client" mode or a "server" mode. If the "//{host}" is missing then it is opened as a "server" (by "server", this means that a client application initiates communication). When the "//{host}" is specified the connection is opened as a client.

Examples:

A datagram connection for accepting datagrams
datagram://:1234

A datagram connection for sending to a server:
datagram://123.456.789.12:1234

Note that the port number in "server mode" (unspecified host name) is that of the receiving port. The port number in "client mode" (host name specified) is that of the target port. The reply to port in both cases is never unspecified. In "server mode", the same port number is used for both receiving and sending. In "client mode", the reply-to port is always dynamically allocated.

Method Summary

int	**getMaximumLength**() Get the maximum length a datagram can be.
int	**getNominalLength**() Get the nominal length of a datagram.
Datagram	**newDatagram**(byte[] buf, int size) Make a new datagram object
Datagram	**newDatagram**(byte[] buf, int size, String addr) Make a new datagram object
Datagram	**newDatagram**(int size) Make a new datagram object automatically allocating a buffer
Datagram	**newDatagram**(int size, String addr) Make a new datagram object
void	**receive**(Datagram dgram) Receive a datagram
void	**send**(Datagram dgram) Send a datagram

Methods inherited from interface javax.microedition.io.Connection
close

Method Detail

getMaximumLength

```
public int getMaximumLength()
                    throws IOException
```

Get the maximum length a datagram can be.
Returns:
address The length.

getNominalLength

```
public int getNominalLength()
                    throws IOException
```

Get the nominal length of a datagram.
Returns:
address The length.

send

```
public void send(Datagram dgram)
          throws IOException
```

Send a datagram
Parameters:
`dgram` - A datagram.
Throws:
IOException - If an I/O error occurs.
InterruptedIOException - Timeout or upon closing the connection with outstanding I/O.

receive

```
public void receive(Datagram dgram)
            throws IOException
```

Receive a datagram
Parameters:
`dgram` - A datagram.
Throws:
IOException - If an I/O error occurs.
InterruptedIOException - Timeout or upon closing the connection with outstanding I/O.

newDatagram

```
public Datagram newDatagram(int size)
                    throws IOException
```

Make a new datagram object automatically allocating a buffer
Parameters:
`size` - The length of the buffer to be allocated for the datagram
Returns:
A new datagram

newDatagram

```
public Datagram newDatagram(int size,
                            String addr)
                    throws IOException
```

Make a new datagram object
Parameters:
`size` - The length of the buffer to be used
`addr` - The address to which the datagram must go
Returns:
A new datagram

newDatagram

```
public Datagram newDatagram(byte[] buf,
                            int size)
                    throws IOException
```

Make a new datagram object
Parameters:
`buf` - The buffer to be used in the datagram
`size` - The length of the buffer to be allocated for the datagram
Returns:
A new datagram
Throws:
IllegalArgumentException - if the length is negative or larger than the buffer

newDatagram

```
public Datagram newDatagram(byte[] buf,
                            int size,
                            String addr)
                    throws IOException
```

Make a new datagram object
Parameters:
`buf` - The buffer to be used in the datagram
`size` - The length of the buffer to be used
`addr` - The address to which the datagram must go
Returns:
A new datagram
Throws:
IllegalArgumentException - if the length is negative or larger than the buffer

javax.microedition.io
Interface InputConnection

All Known Subinterfaces:
ContentConnection, StreamConnection

public interface **InputConnection**
extends Connection

This interface defines the capabilities that an input stream connection must have.

Method Summary

DataInputStream	**openDataInputStream()** Open and return a data input stream for a connection.
InputStream	**openInputStream()** Open and return an input stream for a connection.

Methods inherited from interface javax.microedition.io.Connection
close

Method Detail

openInputStream

```
public InputStream openInputStream()
                           throws IOException
```

Open and return an input stream for a connection.
Returns:
An input stream
Throws:
IOException - If an I/O error occurs

openDataInputStream

```
public DataInputStream openDataInputStream()
                                 throws IOException
```


Open and return a data input stream for a connection.
Returns:
An input stream
Throws:
IOException - If an I/O error occurs

javax.microedition.io
Interface OutputConnection

All Known Subinterfaces:
ContentConnection, StreamConnection

public interface **OutputConnection**
extends Connection

This interface defines the capabilities that an output stream connection must have.

Method Summary

DataOutputStream	**openDataOutputStream**() Open and return a data output stream for a connection.
OutputStream	**openOutputStream**() Open and return an output stream for a connection.

Methods inherited from interface javax.microedition.io.Connection

close

Method Detail

openOutputStream

```
public OutputStream openOutputStream()
                            throws IOException
```

Open and return an output stream for a connection.
Returns:
An output stream
Throws:
IOException - If an I/O error occurs

openDataOutputStream

```
public DataOutputStream openDataOutputStream()
                                  throws IOException
```


Open and return a data output stream for a connection.
Returns:
An output stream
Throws:
IOException - If an I/O error occurs

javax.microedition.io
Interface StreamConnection

All Known Subinterfaces:
ContentConnection

public interface **StreamConnection**
extends InputConnection, OutputConnection

This interface defines the capabilities that a stream connection must have.

Methods inherited from interface javax.microedition.io.InputConnection
`openDataInputStream, openInputStream`

Methods inherited from interface javax.microedition.io.OutputConnection
`openDataOutputStream, openOutputStream`

javax.microedition.io
Interface StreamConnectionNotifier

public interface **StreamConnectionNotifier**
extends Connection

This interface defines the capabilities that a connection notifier must have.

Method Summary

`StreamConnection`	**`acceptAndOpen()`** Returns a `StreamConnection` that represents a server side socket connection

Methods inherited from interface javax.microedition.io.Connection
`close`

Method Detail

acceptAndOpen

```
public StreamConnection acceptAndOpen()
                              throws IOException
```

Returns a `StreamConnection` that represents a server side socket connection
Returns:
A socket to communicate with a client.
Throws:
IOException - If an I/O error occurs.

A

B

C

D

E

F

G

H

I

InputStream() - Constructor for class java.io.InputStream

InputStreamReader - class java.io.InputStreamReader.
An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and translates them into characters according to a specified character encoding.
InputStreamReader(InputStream) - Constructor for class java.io.InputStreamReader
Create an InputStreamReader that uses the default character encoding.
InputStreamReader(InputStream, String) - Constructor for class java.io.InputStreamReader
Create an InputStreamReader that uses the named character encoding.
insert(int, boolean) - Method in class java.lang.StringBuffer
Inserts the string representation of the `boolean` argument into this string buffer.
insert(int, char) - Method in class java.lang.StringBuffer
Inserts the string representation of the `char` argument into this string buffer.
insert(int, char[]) - Method in class java.lang.StringBuffer
Inserts the string representation of the `char` array argument into this string buffer.
insert(int, int) - Method in class java.lang.StringBuffer
Inserts the string representation of the second `int` argument into this string buffer.
insert(int, long) - Method in class java.lang.StringBuffer
Inserts the string representation of the `long` argument into this string buffer.
insert(int, Object) - Method in class java.lang.StringBuffer
Inserts the string representation of the `Object` argument into this string buffer.
insert(int, String) - Method in class java.lang.StringBuffer
Inserts the string into this string buffer.
insertElementAt(Object, int) - Method in class java.util.Vector
Inserts the specified object as a component in this vector at the specified `index`.
InstantiationException - exception java.lang.InstantiationException.
Thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface or is an abstract class.
InstantiationException() - Constructor for class java.lang.InstantiationException
Constructs an `InstantiationException` with no detail message.
InstantiationException(String) - Constructor for class java.lang.InstantiationException
Constructs an `InstantiationException` with the specified detail message.
Integer - class java.lang.Integer.
The Integer class wraps a value of the primitive type `int` in an object.
Integer(int) - Constructor for class java.lang.Integer
Constructs a newly allocated `Integer` object that represents the primitive `int` argument.
InterruptedException - exception java.lang.InterruptedException.
Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the `interrupt` method in class `Thread`.
InterruptedException() - Constructor for class java.lang.InterruptedException
Constructs an `InterruptedException` with no detail message.
InterruptedException(String) - Constructor for class java.lang.InterruptedException
Constructs an `InterruptedException` with the specified detail message.
InterruptedIOException - exception java.io.InterruptedIOException.
Signals that an I/O operation has been interrupted.
InterruptedIOException() - Constructor for class java.io.InterruptedIOException
Constructs an `InterruptedIOException` with `null` as its error detail message.
InterruptedIOException(String) - Constructor for class java.io.InterruptedIOException
Constructs an `InterruptedIOException` with the specified detail message.

intValue() - Method in class java.lang.Integer
Returns the value of this Integer as an int.
IOException - exception java.io.IOException.
Signals that an I/O exception of some sort has occurred.
IOException() - Constructor for class java.io.IOException
Constructs an `IOException` with `null` as its error detail message.
IOException(String) - Constructor for class java.io.IOException
Constructs an `IOException` with the specified detail message.
isAlive() - Method in class java.lang.Thread
Tests if this thread is alive.
isArray() - Method in class java.lang.Class
Determines if this `Class` object represents an array class.
isAssignableFrom(Class) - Method in class java.lang.Class
Determines if the class or interface represented by this `Class` object is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified `Class` parameter.
isDigit(char) - Static method in class java.lang.Character
Determines if the specified character is a digit.
isEmpty() - Method in class java.util.Vector
Tests if this vector has no components.
isEmpty() - Method in class java.util.Hashtable
Tests if this hashtable maps no keys to values.
isInstance(Object) - Method in class java.lang.Class
Determines if the specified `Object` is assignment-compatible with the object represented by this `Class`.
isInterface() - Method in class java.lang.Class
Determines if the specified `Class` object represents an interface type.
isLowerCase(char) - Static method in class java.lang.Character
Determines if the specified character is a lowercase character.
isUpperCase(char) - Static method in class java.lang.Character
Determines if the specified character is an uppercase character.

J

JANUARY - Static variable in class java.util.Calendar

java.io - package java.io

java.lang - package java.lang

java.util - package java.util

javax.microedition.io - package javax.microedition.io

join() - Method in class java.lang.Thread
Waits for this thread to die.
JULY - Static variable in class java.util.Calendar

JUNE - Static variable in class java.util.Calendar

K

keys() - Method in class java.util.Hashtable
Returns an enumeration of the keys in this hashtable.

L

lastElement() - Method in class java.util.Vector
Returns the last component of the vector.
lastIndexOf(int) - Method in class java.lang.String
Returns the index within this string of the last occurrence of the specified character.
lastIndexOf(int, int) - Method in class java.lang.String
Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
lastIndexOf(Object) - Method in class java.util.Vector
Returns the index of the last occurrence of the specified object in this vector.
lastIndexOf(Object, int) - Method in class java.util.Vector
Searches backwards for the specified object, starting from the specified index, and returns an index to it.
length() - Method in class java.lang.StringBuffer
Returns the length (character count) of this string buffer.
length() - Method in class java.lang.String
Returns the length of this string.
lock - Variable in class java.io.Reader
The object used to synchronize operations on this stream.
lock - Variable in class java.io.Writer
The object used to synchronize operations on this stream.
Long - class java.lang.Long.
The Long class wraps a value of the primitive type `long` in an object.
Long(long) - Constructor for class java.lang.Long
Constructs a newly allocated `Long` object that represents the primitive `long` argument.
longValue() - Method in class java.lang.Long
Returns the value of this Long as a long value.
longValue() - Method in class java.lang.Integer
Returns the value of this Integer as a `long`.

M

MARCH - Static variable in class java.util.Calendar

mark - Variable in class java.io.ByteArrayInputStream
The currently marked position in the stream.

mark(int) - Method in class java.io.Reader
Mark the present position in the stream.
mark(int) - Method in class java.io.InputStreamReader
Mark the present position in the stream.
mark(int) - Method in class java.io.InputStream
Marks the current position in this input stream.
mark(int) - Method in class java.io.ByteArrayInputStream
Set the current marked position in the stream.
mark(int) - Method in class java.io.DataInputStream
Marks the current position in this input stream.
markSupported() - Method in class java.io.Reader
Tell whether this stream supports the mark() operation.
markSupported() - Method in class java.io.InputStreamReader
Tell whether this stream supports the mark() operation.
markSupported() - Method in class java.io.InputStream
Tests if this input stream supports the `mark` and `reset` methods.
markSupported() - Method in class java.io.ByteArrayInputStream
Tests if ByteArrayInputStream supports mark/reset.
markSupported() - Method in class java.io.DataInputStream
Tests if this input stream supports the `mark` and `reset` methods.
Math - class java.lang.Math.
The class `Math` contains methods for performing basic numeric operations.
MAX_PRIORITY - Static variable in class java.lang.Thread
The maximum priority that a thread can have.
MAX_RADIX - Static variable in class java.lang.Character
The maximum radix available for conversion to and from Strings.
MAX_VALUE - Static variable in class java.lang.Long
The largest value of type `long`.
MAX_VALUE - Static variable in class java.lang.Character
The constant value of this field is the largest value of type `char`.
MAX_VALUE - Static variable in class java.lang.Byte
The maximum value a Byte can have.
MAX_VALUE - Static variable in class java.lang.Integer
The largest value of type `int`.
MAX_VALUE - Static variable in class java.lang.Short
The maximum value a Short can have.
max(int, int) - Static method in class java.lang.Math
Returns the greater of two `int` values.
max(long, long) - Static method in class java.lang.Math
Returns the greater of two `long` values.
MAY - Static variable in class java.util.Calendar

MILLISECOND - Static variable in class java.util.Calendar

MIN_PRIORITY - Static variable in class java.lang.Thread
The minimum priority that a thread can have.
MIN_RADIX - Static variable in class java.lang.Character
The minimum radix available for conversion to and from Strings.
MIN_VALUE - Static variable in class java.lang.Long
The smallest value of type `long`.

MIN_VALUE - Static variable in class java.lang.Character
The constant value of this field is the smallest value of type `char`.
MIN_VALUE - Static variable in class java.lang.Byte
The minimum value a Byte can have.
MIN_VALUE - Static variable in class java.lang.Integer
The smallest value of type `int`.
MIN_VALUE - Static variable in class java.lang.Short
The minimum value a Short can have.
min(int, int) - Static method in class java.lang.Math
Returns the smaller of two `int` values.
min(long, long) - Static method in class java.lang.Math
Returns the smaller of two `long` values.
MINUTE - Static variable in class java.util.Calendar

MONDAY - Static variable in class java.util.Calendar

MONTH - Static variable in class java.util.Calendar

N

NegativeArraySizeException - exception java.lang.NegativeArraySizeException.
Thrown if an application tries to create an array with negative size.
NegativeArraySizeException() - Constructor for class java.lang.NegativeArraySizeException
Constructs a `NegativeArraySizeException` with no detail message.
NegativeArraySizeException(String) - Constructor for class java.lang.NegativeArraySizeException
Constructs a `NegativeArraySizeException` with the specified detail message.
newDatagram(byte[], int) - Method in interface javax.microedition.io.DatagramConnection
Make a new datagram object
newDatagram(byte[], int, String) - Method in interface javax.microedition.io.DatagramConnection
Make a new datagram object
newDatagram(int) - Method in interface javax.microedition.io.DatagramConnection
Make a new datagram object automatically allocating a buffer
newDatagram(int, String) - Method in interface javax.microedition.io.DatagramConnection
Make a new datagram object
newInstance() - Method in class java.lang.Class
Creates a new instance of a class.
next(int) - Method in class java.util.Random
Generates the next pseudorandom number.
nextElement() - Method in interface java.util.Enumeration
Returns the next element of this enumeration if this enumeration object has at least one more element to provide.
nextInt() - Method in class java.util.Random
Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.
nextLong() - Method in class java.util.Random
Returns the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence.

NORM_PRIORITY - Static variable in class java.lang.Thread
The default priority that is assigned to a thread.
NoSuchElementException - exception java.util.NoSuchElementException.
Thrown by the `nextElement` method of an `Enumeration` to indicate that there are no more elements in the enumeration.
NoSuchElementException() - Constructor for class java.util.NoSuchElementException
Constructs a `NoSuchElementException` with `null` as its error message string.
NoSuchElementException(String) - Constructor for class java.util.NoSuchElementException
Constructs a `NoSuchElementException`, saving a reference to the error message string `s` for later retrieval by the `getMessage` method.
notify() - Method in class java.lang.Object
Wakes up a single thread that is waiting on this object's monitor.
notifyAll() - Method in class java.lang.Object
Wakes up all threads that are waiting on this object's monitor.
NOVEMBER - Static variable in class java.util.Calendar

NullPointerException - exception java.lang.NullPointerException.
Thrown when an application attempts to use `null` in a case where an object is required.
NullPointerException() - Constructor for class java.lang.NullPointerException
Constructs a `NullPointerException` with no detail message.
NullPointerException(String) - Constructor for class java.lang.NullPointerException
Constructs a `NullPointerException` with the specified detail message.
NumberFormatException - exception java.lang.NumberFormatException.
Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.
NumberFormatException() - Constructor for class java.lang.NumberFormatException
Constructs a `NumberFormatException` with no detail message.
NumberFormatException(String) - Constructor for class java.lang.NumberFormatException
Constructs a `NumberFormatException` with the specified detail message.

O

Object - class java.lang.Object.
Class `Object` is the root of the class hierarchy.
Object() - Constructor for class java.lang.Object

OCTOBER - Static variable in class java.util.Calendar

open(String) - Static method in class javax.microedition.io.Connector
Create and open a Connection
open(String, int) - Static method in class javax.microedition.io.Connector
Create and open a Connection
open(String, int, boolean) - Static method in class javax.microedition.io.Connector
Create and open a Connection
openDataInputStream() - Method in interface javax.microedition.io.InputConnection
Open and return a data input stream for a connection.
openDataInputStream(String) - Static method in class javax.microedition.io.Connector
Create and open a connection input stream

P

R

Random() - Constructor for class java.util.Random
Creates a new random number generator.
Random(long) - Constructor for class java.util.Random
Creates a new random number generator using a single `long` seed:
READ - Static variable in class javax.microedition.io.Connector
Access mode
READ_WRITE - Static variable in class javax.microedition.io.Connector
Access mode
read() - Method in class java.io.Reader
Read a single character.
read() - Method in class java.io.InputStreamReader
Read a single character.
read() - Method in class java.io.InputStream
Reads the next byte of data from the input stream.
read() - Method in class java.io.ByteArrayInputStream
Reads the next byte of data from this input stream.
read() - Method in class java.io.DataInputStream
Reads the next byte of data from this input stream.
read(byte[]) - Method in class java.io.InputStream
Reads some number of bytes from the input stream and stores them into the buffer array `b`.
read(byte[], int, int) - Method in class java.io.InputStream
Reads up to `len` bytes of data from the input stream into an array of bytes.
read(byte[], int, int) - Method in class java.io.ByteArrayInputStream
Reads up to `len` bytes of data into an array of bytes from this input stream.
read(byte[], int, int) - Method in class java.io.DataInputStream
Reads up to `len` bytes of data from this input stream into an array of bytes.
read(char[]) - Method in class java.io.Reader
Read characters into an array.
read(char[], int, int) - Method in class java.io.Reader
Read characters into a portion of an array.
read(char[], int, int) - Method in class java.io.InputStreamReader
Read characters into a portion of an array.
readBoolean() - Method in class java.io.DataInputStream
See the general contract of the `readBoolean` method of `DataInput`.
readBoolean() - Method in interface java.io.DataInput
Reads one input byte and returns `true` if that byte is nonzero, `false` if that byte is zero.
readByte() - Method in class java.io.DataInputStream
See the general contract of the `readByte` method of `DataInput`.
readByte() - Method in interface java.io.DataInput
Reads and returns one input byte.
readChar() - Method in class java.io.DataInputStream
See the general contract of the `readChar` method of `DataInput`.
readChar() - Method in interface java.io.DataInput
Reads an input `char` and returns the `char` value.
Reader - class java.io.Reader.
Abstract class for reading character streams.
Reader() - Constructor for class java.io.Reader
Create a new character-stream reader whose critical sections will synchronize on the reader itself.
Reader(Object) - Constructor for class java.io.Reader
Create a new character-stream reader whose critical sections will synchronize on the given object.

readFully(byte[]) - Method in class java.io.DataInputStream
See the general contract of the `readFully` method of `DataInput`.
readFully(byte[]) - Method in interface java.io.DataInput
Reads some bytes from an input stream and stores them into the buffer array `b`.
readFully(byte[], int, int) - Method in class java.io.DataInputStream
See the general contract of the `readFully` method of `DataInput`.
readFully(byte[], int, int) - Method in interface java.io.DataInput
Reads `len` bytes from an input stream.
readInt() - Method in class java.io.DataInputStream
See the general contract of the `readInt` method of `DataInput`.
readInt() - Method in interface java.io.DataInput
Reads four input bytes and returns an `int` value.
readLong() - Method in class java.io.DataInputStream
See the general contract of the `readLong` method of `DataInput`.
readLong() - Method in interface java.io.DataInput
Reads eight input bytes and returns a `long` value.
readShort() - Method in class java.io.DataInputStream
See the general contract of the `readShort` method of `DataInput`.
readShort() - Method in interface java.io.DataInput
Reads two input bytes and returns a `short` value.
readUnsignedByte() - Method in class java.io.DataInputStream
See the general contract of the `readUnsignedByte` method of `DataInput`.
readUnsignedByte() - Method in interface java.io.DataInput
Reads one input byte, zero-extends it to type `int`, and returns the result, which is therefore in the range `0` through `255`.
readUnsignedShort() - Method in class java.io.DataInputStream
See the general contract of the `readUnsignedShort` method of `DataInput`.
readUnsignedShort() - Method in interface java.io.DataInput
Reads two input bytes and returns an `int` value in the range `0` through `65535`.
readUTF() - Method in class java.io.DataInputStream
See the general contract of the `readUTF` method of `DataInput`.
readUTF() - Method in interface java.io.DataInput
Reads in a string that has been encoded using a modified UTF-8 format.
readUTF(DataInput) - Static method in class java.io.DataInputStream
Reads from the stream `in` a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`.
ready() - Method in class java.io.Reader
Tell whether this stream is ready to be read.
ready() - Method in class java.io.InputStreamReader
Tell whether this stream is ready to be read.
receive(Datagram) - Method in interface javax.microedition.io.DatagramConnection
Receive a datagram
regionMatches(boolean, int, String, int, int) - Method in class java.lang.String
Tests if two string regions are equal.
rehash() - Method in class java.util.Hashtable
Rehashes the contents of the hashtable into a hashtable with a larger capacity.
remove(Object) - Method in class java.util.Hashtable
Removes the key (and its corresponding value) from this hashtable.
removeAllElements() - Method in class java.util.Vector
Removes all components from this vector and sets its size to zero.

S

size() - Method in class java.util.Vector
Returns the number of components in this vector.
size() - Method in class java.util.Hashtable
Returns the number of keys in this hashtable.
skip(long) - Method in class java.io.Reader
Skip characters.
skip(long) - Method in class java.io.InputStreamReader
Skip characters.
skip(long) - Method in class java.io.InputStream
Skips over and discards `n` bytes of data from this input stream.
skip(long) - Method in class java.io.ByteArrayInputStream
Skips `n` bytes of input from this input stream.
skip(long) - Method in class java.io.DataInputStream
Skips over and discards `n` bytes of data from the input stream.
skipBytes(int) - Method in class java.io.DataInputStream
See the general contract of the `skipBytes` method of `DataInput`.
skipBytes(int) - Method in interface java.io.DataInput
Makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes.
sleep(long) - Static method in class java.lang.Thread
Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
Stack - class java.util.Stack.
The `Stack` class represents a last-in-first-out (LIFO) stack of objects.
Stack() - Constructor for class java.util.Stack
Creates an empty Stack.
start() - Method in class java.lang.Thread
Causes this thread to begin execution; the Java Virtual Machine calls the `run` method of this thread.
startsWith(String) - Method in class java.lang.String
Tests if this string starts with the specified prefix.
startsWith(String, int) - Method in class java.lang.String
Tests if this string starts with the specified prefix beginning a specified index.
StreamConnection - interface javax.microedition.io.StreamConnection.
This interface defines the capabilities that a stream connection must have.
StreamConnectionNotifier - interface javax.microedition.io.StreamConnectionNotifier.
This interface defines the capabilities that a connection notifier must have.
String - class java.lang.String.
The `String` class represents character strings.
String() - Constructor for class java.lang.String
Initializes a newly created `String` object so that it represents an empty character sequence.
String(byte[]) - Constructor for class java.lang.String
Construct a new `String` by converting the specified array of bytes using the platform's default character encoding.
String(byte[], int, int) - Constructor for class java.lang.String
Construct a new `String` by converting the specified subarray of bytes using the platform's default character encoding.
String(byte[], int, int, String) - Constructor for class java.lang.String
Construct a new `String` by converting the specified subarray of bytes using the specified character encoding.

String(byte[], String) - Constructor for class java.lang.String
Construct a new `String` by converting the specified array of bytes using the specified character encoding.
String(char[]) - Constructor for class java.lang.String
Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument.
String(char[], int, int) - Constructor for class java.lang.String
Allocates a new `String` that contains characters from a subarray of the character array argument.
String(String) - Constructor for class java.lang.String
Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
String(StringBuffer) - Constructor for class java.lang.String
Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.
StringBuffer - class java.lang.StringBuffer.
A string buffer implements a mutable sequence of characters.
StringBuffer() - Constructor for class java.lang.StringBuffer
Constructs a string buffer with no characters in it and an initial capacity of 16 characters.
StringBuffer(int) - Constructor for class java.lang.StringBuffer
Constructs a string buffer with no characters in it and an initial capacity specified by the `length` argument.
StringBuffer(String) - Constructor for class java.lang.StringBuffer
Constructs a string buffer so that it represents the same sequence of characters as the string argument; in other words, the initial contents of the string buffer is a copy of the argument string.
StringIndexOutOfBoundsException - exception java.lang.StringIndexOutOfBoundsException.
Thrown by the `charAt` method in class `String` and by other `String` methods to indicate that an index is either negative or greater than or equal to the size of the string.
StringIndexOutOfBoundsException() - Constructor for class java.lang.StringIndexOutOfBoundsException
Constructs a `StringIndexOutOfBoundsException` with no detail message.
StringIndexOutOfBoundsException(int) - Constructor for class java.lang.StringIndexOutOfBoundsException
Constructs a new `StringIndexOutOfBoundsException` class with an argument indicating the illegal index.
StringIndexOutOfBoundsException(String) - Constructor for class java.lang.StringIndexOutOfBoundsException
Constructs a `StringIndexOutOfBoundsException` with the specified detail message.
substring(int) - Method in class java.lang.String
Returns a new string that is a substring of this string.
substring(int, int) - Method in class java.lang.String
Returns a new string that is a substring of this string.
SUNDAY - Static variable in class java.util.Calendar

System - class java.lang.System.
The `System` class contains several useful class fields and methods.

T

Thread - class java.lang.Thread.
A *thread* is a thread of execution in a program.
Thread() - Constructor for class java.lang.Thread
Allocates a new `Thread` object.
Thread(Runnable) - Constructor for class java.lang.Thread
Allocates a new `Thread` object.
Throwable - class java.lang.Throwable.
The `Throwable` class is the superclass of all errors and exceptions in the Java language.
Throwable() - Constructor for class java.lang.Throwable
Constructs a new `Throwable` with `null` as its error message string.
Throwable(String) - Constructor for class java.lang.Throwable
Constructs a new `Throwable` with the specified error message.
THURSDAY - Static variable in class java.util.Calendar

toBinaryString(int) - Static method in class java.lang.Integer
Creates a string representation of the integer argument as an unsigned integer in base 2.
toByteArray() - Method in class java.io.ByteArrayOutputStream
Creates a newly allocated byte array.
toCharArray() - Method in class java.lang.String
Converts this string to a new character array.
toHexString(int) - Static method in class java.lang.Integer
Creates a string representation of the integer argument as an unsigned integer in base 16.
toLowerCase() - Method in class java.lang.String
Converts all of the characters in this String to lower case.
toLowerCase(char) - Static method in class java.lang.Character
The given character is mapped to its lowercase equivalent; if the character has no lowercase equivalent, the character itself is returned.
toOctalString(int) - Static method in class java.lang.Integer
Creates a string representation of the integer argument as an unsigned integer in base 8.
toString() - Method in class java.lang.Object
Returns a string representation of the object.
toString() - Method in class java.lang.Long
Returns a String object representing this Long's value.
toString() - Method in class java.lang.Character
Returns a String object representing this character's value.
toString() - Method in class java.lang.Throwable
Returns a short description of this throwable object.
toString() - Method in class java.lang.Class
Converts the object to a string.
toString() - Method in class java.lang.Integer
Returns a String object representing this Integer's value.
toString() - Method in class java.lang.StringBuffer
Converts to a string representing the data in this string buffer.
toString() - Method in class java.lang.String
This object (which is already a string!) is itself returned.
toString() - Method in class java.lang.Thread
Returns a string representation of this thread, including a unique number that identifies the thread and the thread's priority.

toString() - Method in class java.util.Vector
Returns a string representation of this vector.
toString() - Method in class java.util.Hashtable
Returns a rather long string representation of this hashtable.
toString(int) - Static method in class java.lang.Integer
Returns a new String object representing the specified integer.
toString(int, int) - Static method in class java.lang.Integer
Creates a string representation of the first argument in the radix specified by the second argument.
toString(long) - Static method in class java.lang.Long
Returns a new String object representing the specified integer.
toString(long, int) - Static method in class java.lang.Long
Creates a string representation of the first argument in the radix specified by the second argument.
totalMemory() - Method in class java.lang.Runtime
Returns the total amount of memory in the Java Virtual Machine.
toUpperCase() - Method in class java.lang.String
Converts all of the characters in this String to lower case.
toUpperCase(char) - Static method in class java.lang.Character
Converts the character argument to uppercase; if the character has no lowercase equivalent, the character itself is returned.
trimToSize() - Method in class java.util.Vector
Trims the capacity of this vector to be the vector's current size.
TUESDAY - Static variable in class java.util.Calendar

U

UnsupportedEncodingException - exception java.io.UnsupportedEncodingException.
The Character Encoding is not supported.
UnsupportedEncodingException() - Constructor for class java.io.UnsupportedEncodingException
Constructs an UnsupportedEncodingException without a detail message.
UnsupportedEncodingException(String) - Constructor for class java.io.UnsupportedEncodingException
Constructs an UnsupportedEncodingException with a detail message.
UTFDataFormatException - exception java.io.UTFDataFormatException.
Signals that a malformed UTF-8 string has been read in a data input stream or by any class that implements the data input interface.
UTFDataFormatException() - Constructor for class java.io.UTFDataFormatException
Constructs a `UTFDataFormatException` with `null` as its error detail message.
UTFDataFormatException(String) - Constructor for class java.io.UTFDataFormatException
Constructs a `UTFDataFormatException` with the specified detail message.

V

valueOf(boolean) - Static method in class java.lang.String
Returns the string representation of the `boolean` argument.
valueOf(char) - Static method in class java.lang.String
Returns the string representation of the `char` argument.

valueOf(char[]) - Static method in class java.lang.String
Returns the string representation of the `char` array argument.
valueOf(char[], int, int) - Static method in class java.lang.String
Returns the string representation of a specific subarray of the `char` array argument.
valueOf(int) - Static method in class java.lang.String
Returns the string representation of the `int` argument.
valueOf(long) - Static method in class java.lang.String
Returns the string representation of the `long` argument.
valueOf(Object) - Static method in class java.lang.String
Returns the string representation of the `Object` argument.
valueOf(String) - Static method in class java.lang.Integer
Returns a new Integer object initialized to the value of the specified String.
valueOf(String, int) - Static method in class java.lang.Integer
Returns a new Integer object initialized to the value of the specified String.
Vector - class java.util.Vector.
The `Vector` class implements a growable array of objects.
Vector() - Constructor for class java.util.Vector
Constructs an empty vector.
Vector(int) - Constructor for class java.util.Vector
Constructs an empty vector with the specified initial capacity.
Vector(int, int) - Constructor for class java.util.Vector
Constructs an empty vector with the specified initial capacity and capacity increment.
VirtualMachineError - error java.lang.VirtualMachineError.
Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.
VirtualMachineError() - Constructor for class java.lang.VirtualMachineError
Constructs a `VirtualMachineError` with no detail message.
VirtualMachineError(String) - Constructor for class java.lang.VirtualMachineError
Constructs a `VirtualMachineError` with the specified detail message.

W

wait() - Method in class java.lang.Object
Causes current thread to wait until another thread invokes the `Object.notify()` method or the `Object.notifyAll()` method for this object.
wait(long) - Method in class java.lang.Object
Causes current thread to wait until either another thread invokes the `Object.notify()` method or the `Object.notifyAll()` method for this object, or a specified amount of time has elapsed.
wait(long, int) - Method in class java.lang.Object
Causes current thread to wait until another thread invokes the `Object.notify()` method or the `Object.notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.
WEDNESDAY - Static variable in class java.util.Calendar

WRITE - Static variable in class javax.microedition.io.Connector
Access mode
write(byte[]) - Method in class java.io.OutputStream
Writes `b.length` bytes from the specified byte array to this output stream.

write(byte[]) - Method in interface java.io.DataOutput
Writes to the output stream all the bytes in array `b`.
write(byte[], int, int) - Method in class java.io.OutputStream
Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.
write(byte[], int, int) - Method in class java.io.DataOutputStream
Writes `len` bytes from the specified byte array starting at offset `off` to the underlying output stream.
write(byte[], int, int) - Method in class java.io.ByteArrayOutputStream
Writes `len` bytes from the specified byte array starting at offset `off` to this byte array output stream.
write(byte[], int, int) - Method in class java.io.PrintStream
Write `len` bytes from the specified byte array starting at offset `off` to this stream.
write(byte[], int, int) - Method in interface java.io.DataOutput
Writes `len` bytes from array `b`, in order, to the output stream.
write(char[]) - Method in class java.io.Writer
Write an array of characters.
write(char[], int, int) - Method in class java.io.Writer
Write a portion of an array of characters.
write(char[], int, int) - Method in class java.io.OutputStreamWriter
Write a portion of an array of characters.
write(int) - Method in class java.io.OutputStream
Writes the specified byte to this output stream.
write(int) - Method in class java.io.DataOutputStream
Writes the specified byte (the low eight bits of the argument `b`) to the underlying output stream.
write(int) - Method in class java.io.Writer
Write a single character.
write(int) - Method in class java.io.OutputStreamWriter
Write a single character.
write(int) - Method in class java.io.ByteArrayOutputStream
Writes the specified byte to this byte array output stream.
write(int) - Method in class java.io.PrintStream
Write the specified byte to this stream.
write(int) - Method in interface java.io.DataOutput
Writes to the output stream the eight low-order bits of the argument `b`.
write(String) - Method in class java.io.Writer
Write a string.
write(String, int, int) - Method in class java.io.Writer
Write a portion of a string.
write(String, int, int) - Method in class java.io.OutputStreamWriter
Write a portion of a string.
writeBoolean(boolean) - Method in class java.io.DataOutputStream
Writes a `boolean` to the underlying output stream as a 1-byte value.
writeBoolean(boolean) - Method in interface java.io.DataOutput
Writes a `boolean` value to this output stream.
writeByte(int) - Method in class java.io.DataOutputStream
Writes out a `byte` to the underlying output stream as a 1-byte value.
writeByte(int) - Method in interface java.io.DataOutput
Writes to the output stream the eight low- order bits of the argument `v`.
writeChar(int) - Method in class java.io.DataOutputStream
Writes a `char` to the underlying output stream as a 2-byte value, high byte first.

Y

A B C D E F G H I J K L M N O P R S T U V W Y
