Goldman Sachs

GS Collections and Java 8

JCP Executive Committee F2F Meeting – London, UK

May 13-14, 2014

What is GS Collections?

Goldman Sachs

- Open source Java collections framework developed in Goldman Sachs
  - In development since 2004
  - Hosted on GitHub w/ Apache 2.0 License
    - github.com/goldmansachs/gs-collections
- GS Collections Kata
  - Internal training developed in 2007
  - Taught to > 1,500 GS Java developers
  - Hosted on GitHub w/ Apache 2.0 License
    - github.com/goldmansachs/gs-collections-kata

Why did we build GS Collections?

Goldman Sachs

1. Extensive use of collections across all of our Java code
2. Fast response times needed so collections in memory
   - Optimizations for both space and speed
3. Constantly changing environment
   - Less boilerplate code to maintain is more desirable so we can focus on business opportunities
4. Usage of patterns improves code review speed and quality
   - Great support in modern IDE's for pattern based operations
5. Needed features from all of the major collection libraries
   - We get to have that without having to manage the vastness of that diversification

Why do we use GS Collections?

Goldman Sachs

1. Object-Oriented Design + Functional API
2. Rich Lambda-Ready Iteration API
   - Fluent API that works great with Java 8
3. Memory Efficient Containers
4. Completely compatible with JDK interfaces
5. Additional Object Containers not in JDK
6. Primitive Containers
7. Immutable Containers
8. Parallel Eager and Lazy iteration
9. Similar to Smalltalk (api) and Scala (hierarchy)

Questions for this group

Goldman Sachs

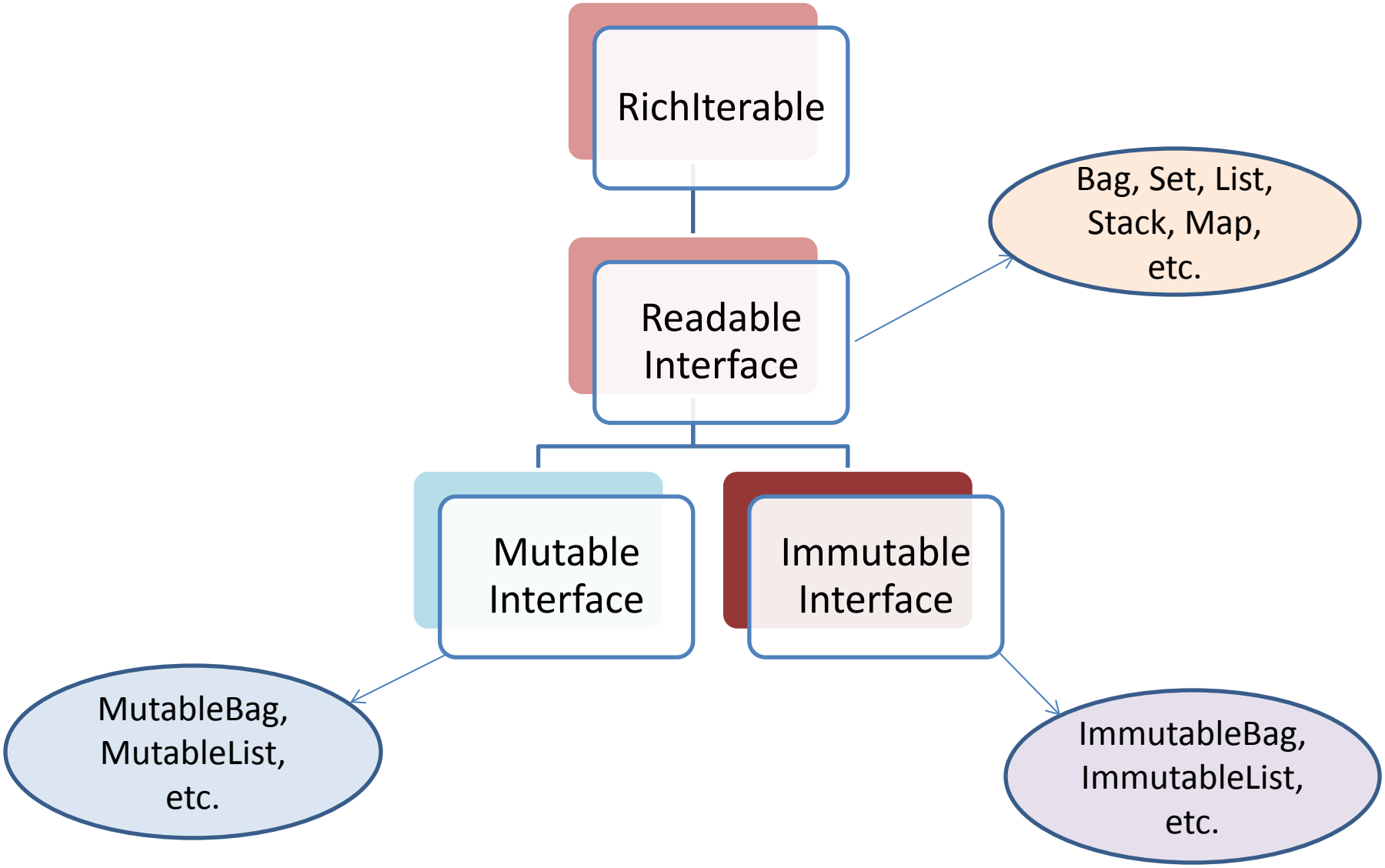
- Do you use GS Collections today?
- Is there anything that keeps you from using GS Collections in your code bases?
- What are our plans for Java 9/10 for the Java Collections framework?
  - Will anyone sponsor a JSR for Collections 2?
  - Should GS sponsor a JSR for Collections 2?
- Should a Java Collections 2 implementation “Rock Hard or Go Home”?
  - Full primitive support, immutable containers, multimaps, bags, bimaps, eager/lazy, serial/parallel, etc.

Framework Comparisons

Goldman Sachs

Features	GSC 5.0	Java 8	Guava	Trove	Scala
Rich API	✔	✔	✔		✔
Interfaces	Readable, Mutable, Immutable, FixedSize, Lazy	Mutable, Stream	Mutable, Fluent	Mutable	Readable, Mutable, Immutable, Lazy
Optimized Set & Map	✔ (+Bag)			✔	
Immutable Collections	✔		✔		✔
Primitive Collections	✔ (+Bag, +Immutable)			✔	
Multimaps	✔ (+Bag, +SortedBag)		✔ (+Linked)		(Multimap trait)
Bags (Multisets)	✔		✔		
BiMaps	✔		✔		
Iteration Styles	Eager/Lazy, Serial/Parallel	Lazy, Serial/Parallel	Lazy, Serial	Eager, Serial	Eager/Lazy, Serial/Parallel (Lazy Only)

GS Collections: Design Concepts



GS Collections: RichIterable API

Goldman Sachs

RichIterable interface: 50+ Unique Methods, 75+ Total Methods (w/overloads)		
λ aggregateBy	λ flatCollect	notEmpty
λ aggregateInPlaceBy	λ forEach	λ partition
λ allSatisfy	λ forEachWith	λ reject
λ anySatisfy	λ forEachWithIndex	λ rejectWith
appendString	getFirst	λ select
asLazy	getLast	λ selectWith
chunk	λ groupBy	size
λ collect	λ groupByEach	toArray
λ collectIf	λ injectInto	toBag
λ collectWith	isEmpty	toList
contains	Iterator	λ toMap
containsAll	makeString	toSet
containsAllArguments	λ max	λ toSortedList / toSortedListBy
containsAllIterable	λ maxBy	λ toSortedMap
λ count	λ min	λ toSortedSet / toSortedSetBy
λ detect	λ minBy	zip
λ detectIfNone	λ noneSatisfy	zipWithIndex

Eager versus Lazy Iteration

Goldman Sachs

Iteration Style	GSC Example	JDK 8 Example
Serial Eager (collect)	`List<Address> addresses =` `  people.collect(Person::getAddress);`	

Serial Lazy (collect / map)	`LazyIterable<Address> addresses =`
	`  people.asLazy()`
	`  .collect(Person::getAddress);`
		`Stream<Address> addresses =`
		`  people.stream()`
		`  .map(Person::getAddress);`

Serial Lazy (collect / map, toList)	`List<Address> addresses =`
	`  people.asLazy()`
	`  .collect(Person::getAddress)`
	`  .toList();`
		`List<Address> addresses =`
		`  people.stream()`
		`  .map(Person::getAddress)`
		`  .collect(Collectors.toList());`

Parallel Eager	`Collection<Address> addresses =`
	`  ParallelIterate.collect(`
	`  people, Person::getAddress);`	

Parallel Lazy	`ParallelListIterable<Address> addresses =`
	`  people.asParallel(executor, batchSize)`
	`  .collect(Person::getAddress);`
		`Stream<Address> addresses =`
		`  people.parallelStream()`
		`  .map(Person::getAddress);`

GS Collections in terms of lines of code

- 295,036 lines of Java code
- 9,959 lines of Scala code
- 76,240 lines of StringTemplate templates

- Total Java code after code generation:
  - **1,438,477** lines of Java code your developers don't have to write and can use for free

Reducing code using lambdas

Goldman Sachs

- Converted nearly all of our anonymous inner classes in tests to lambdas and method references.
- 9% reduction of code in our unit test module
  - Dropped from 105,206 LOC to 95,775
- Many examples of code savings from Java 8 can be found in our unit test suite module in GitHub.
  - Possibly one of the first large scale usages of Java 8 lambdas and method references in a code base

Memory Optimization

Goldman Sachs

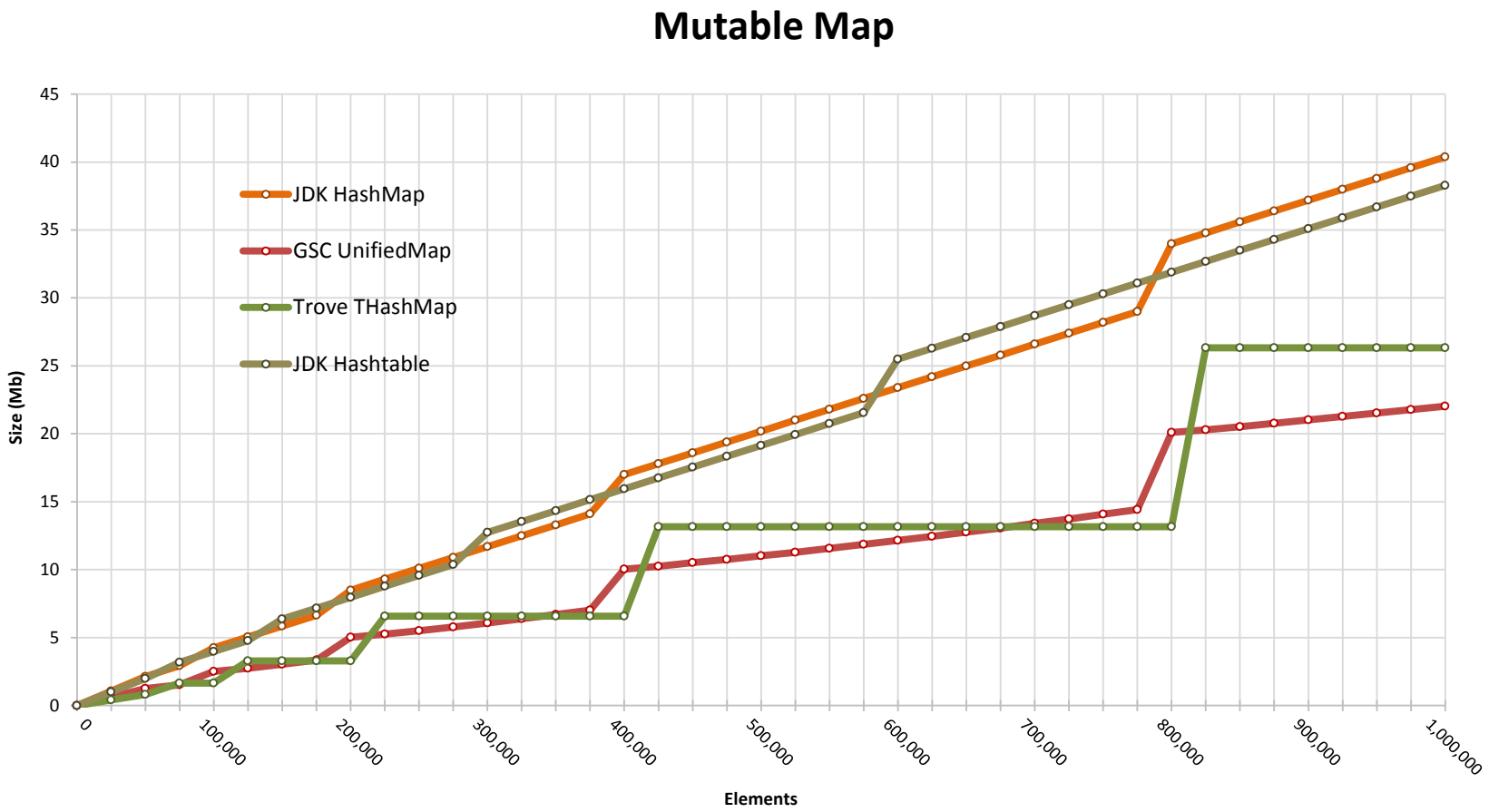
- UnifiedMap – built without using Entry objects.
- UnifiedSet – not built using a map.
- Empty should be empty.
- Primitive Collections.
- Memory efficient containers for small-sized collections (>1 and < 11).

UnifiedMap

Goldman Sachs

- For every put, HashMap creates an Entry object.
- UnifiedMap stores keys and values in alternate slots on a single array.
- Consecutive memory locations are faster to access.

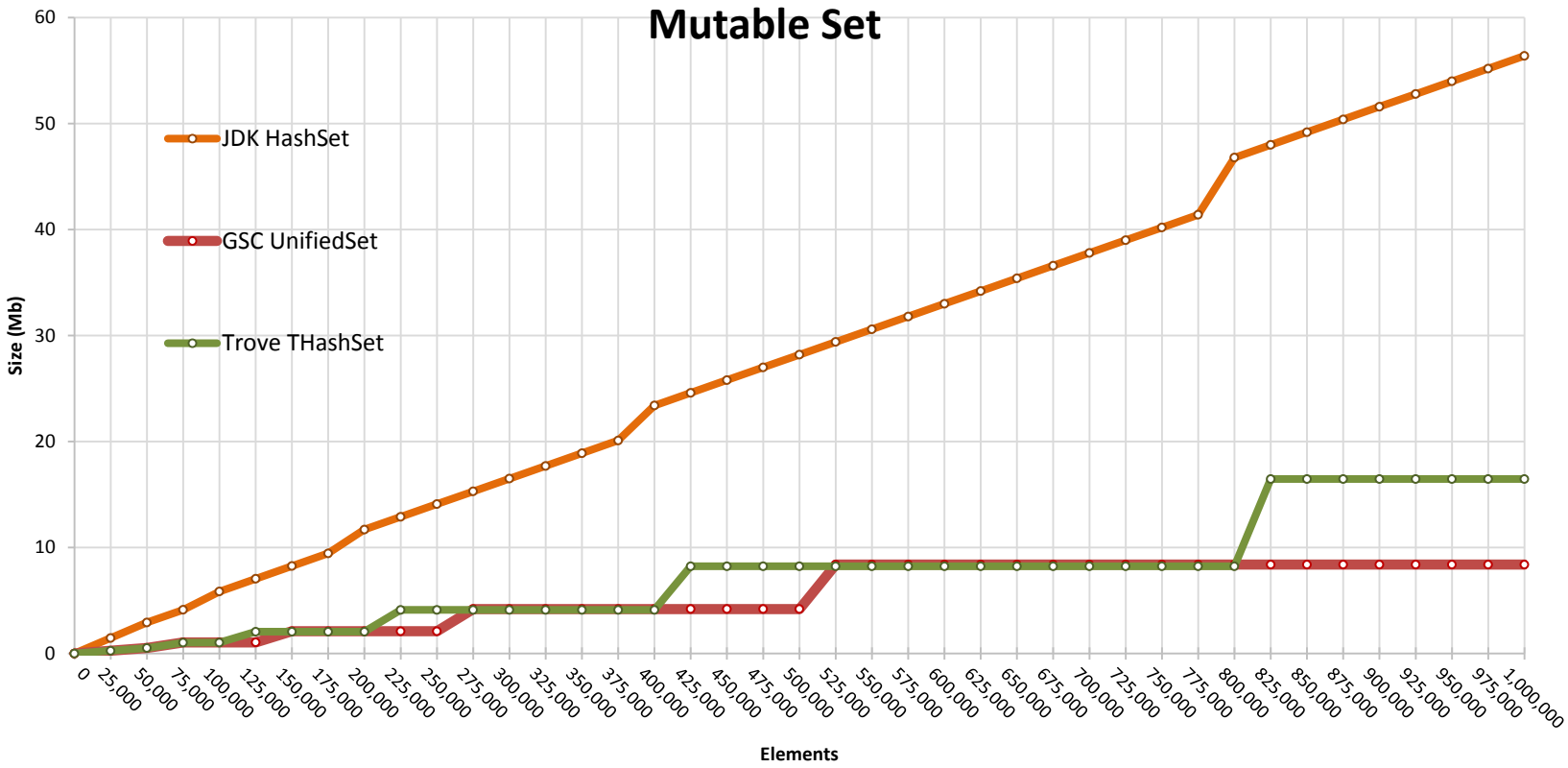
Save 50% Memory with the GSC UnifiedMap



UnifiedSet

- HashSet uses HashMap as its backing collection.
- For every add, an Entry object is created with the element as key and null value.
- UnifiedSet uses an array as its backing collection.

Save 400% Memory with the GSC UnifiedSet

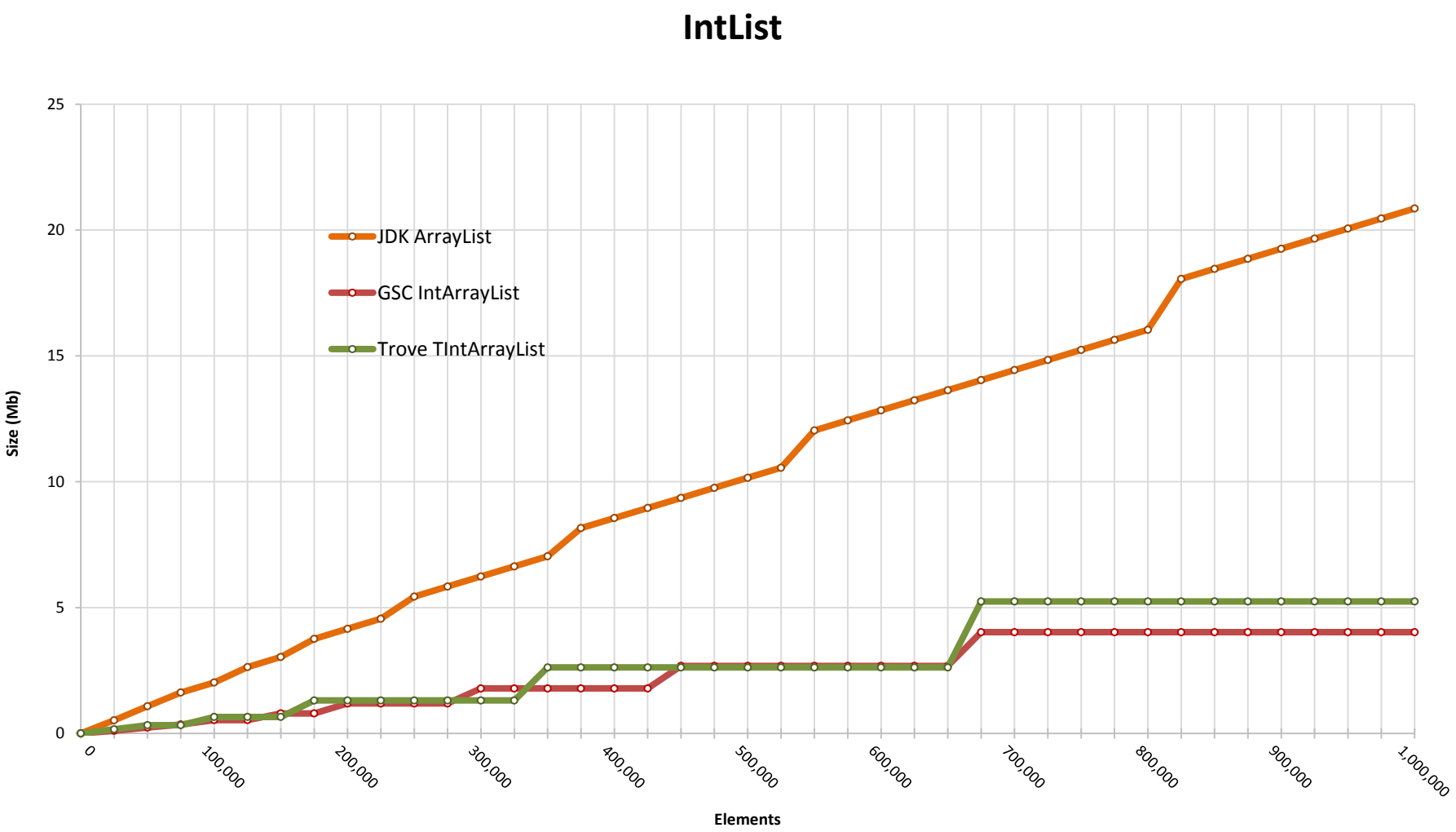


Primitive Collections

Goldman Sachs

- **What are primitives?**
  - Not everything in java is an Object.
  - Primitive types are *automatic* variables that are not references.
  - The variables hold the value and its place on the stack, so it's much more efficient.
- **Why primitive collections?**
  - Reduced memory usage
  - Improved performance
  - Eliminates the need to depend on multiple libraries – PCJ, Trove etc.
- **What primitive collections are available in GSC?**
  - List, Set, Map, Stack, Bag
- **For what all primitive types?**
  - All eight: boolean, byte, char, double, float, int, long, short

Save Memory with Primitive Collections: IntList



Questions for this group

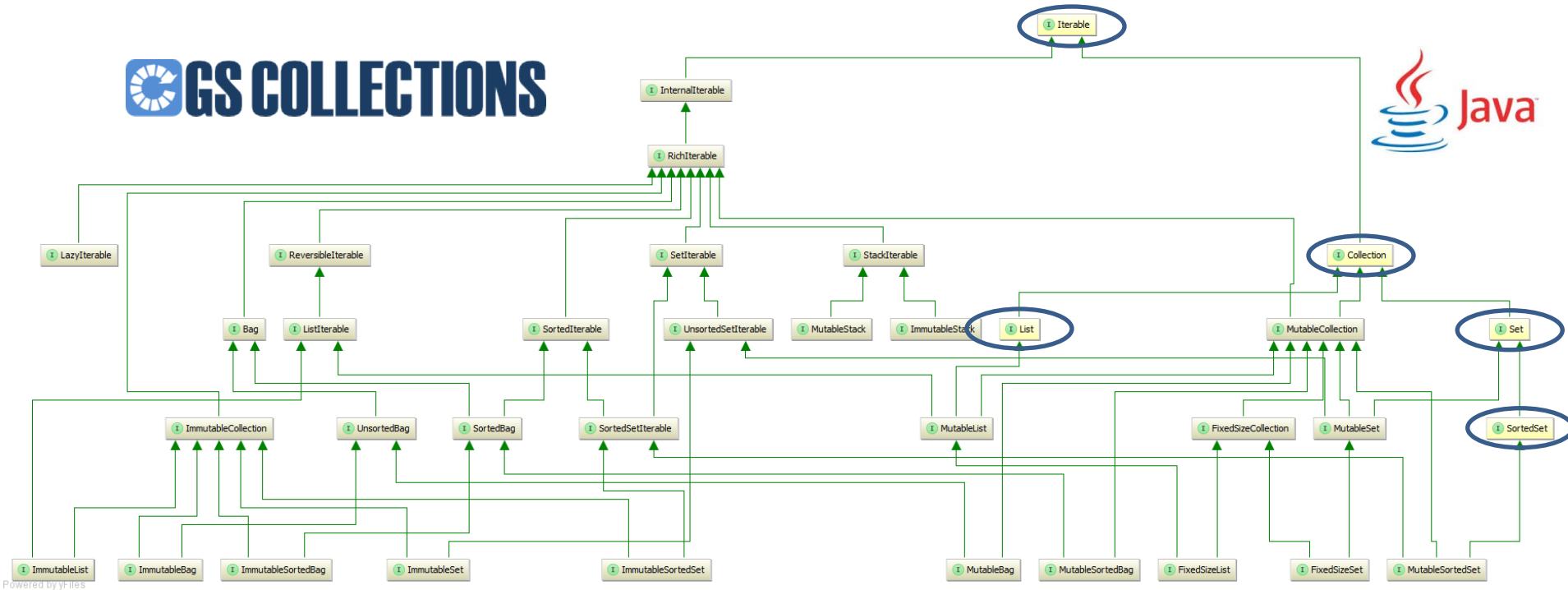
Goldman Sachs

- Do you use GS Collections today?
- Is there anything that keeps you from using GS Collections in your code bases?
- What are our plans for Java 9/10 for the Java Collections framework?
  - Will anyone sponsor a JSR for Collections 2?
  - Should GS sponsor a JSR for Collections 2?
- Should a Java Collections 2 implementation “Rock Hard or Go Home”?
  - Full primitive support, immutable containers, multimaps, bags, bimaps, eager/lazy, serial/parallel, etc.

Appendix

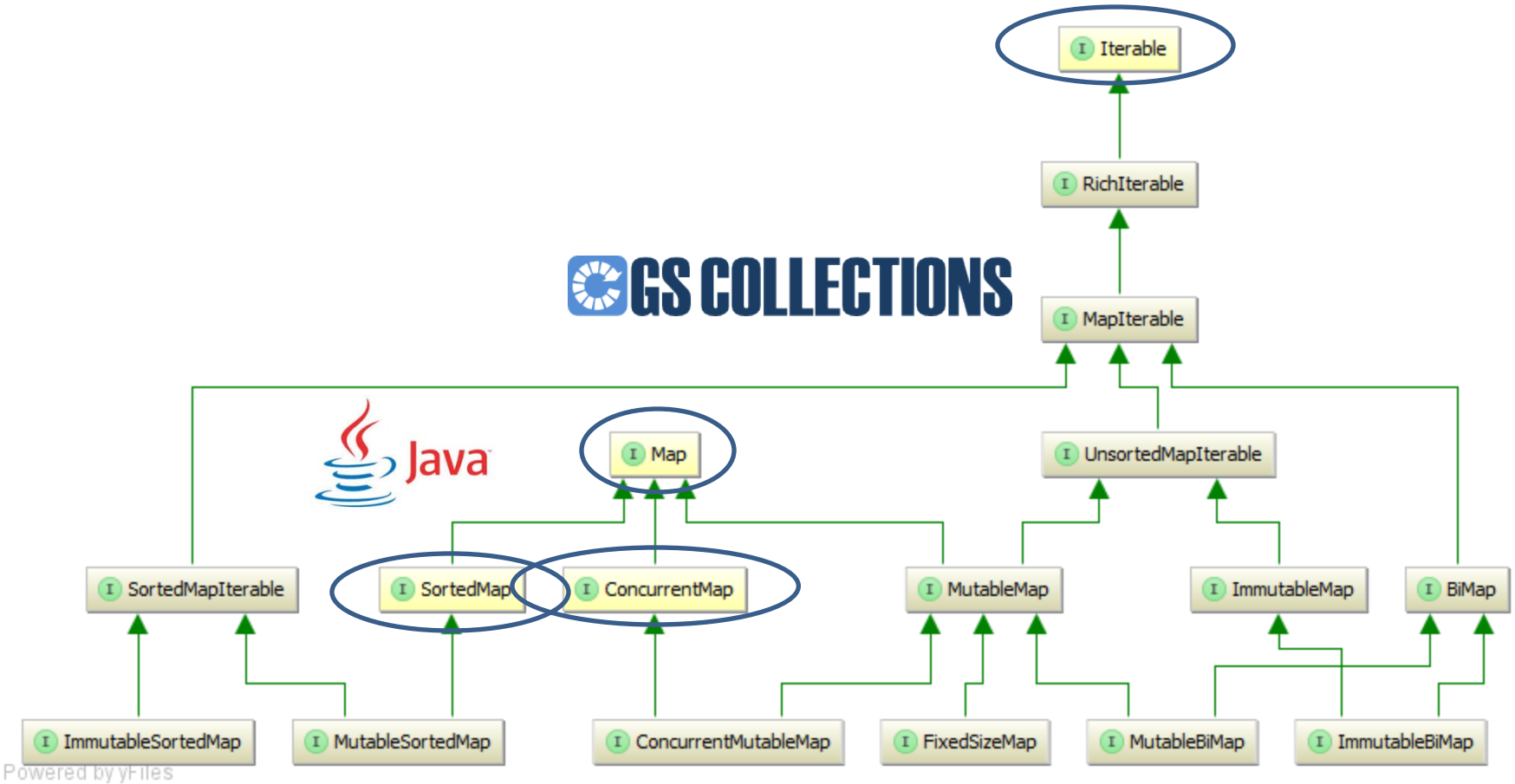
- GS Collections Class Hierarchy
- GS Collections Java 8 Code examples

GS Collections: Collection Hierarchy



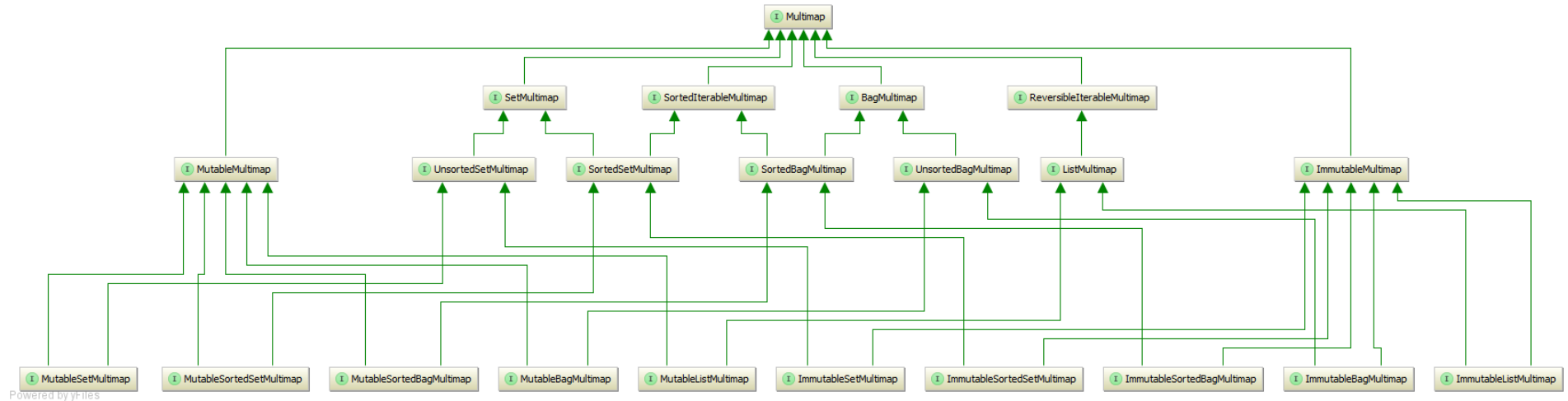
GS Collections adds 30 interfaces to enhance the 5 basic JDK Collections interfaces

GS Collections: Map Hierarchy



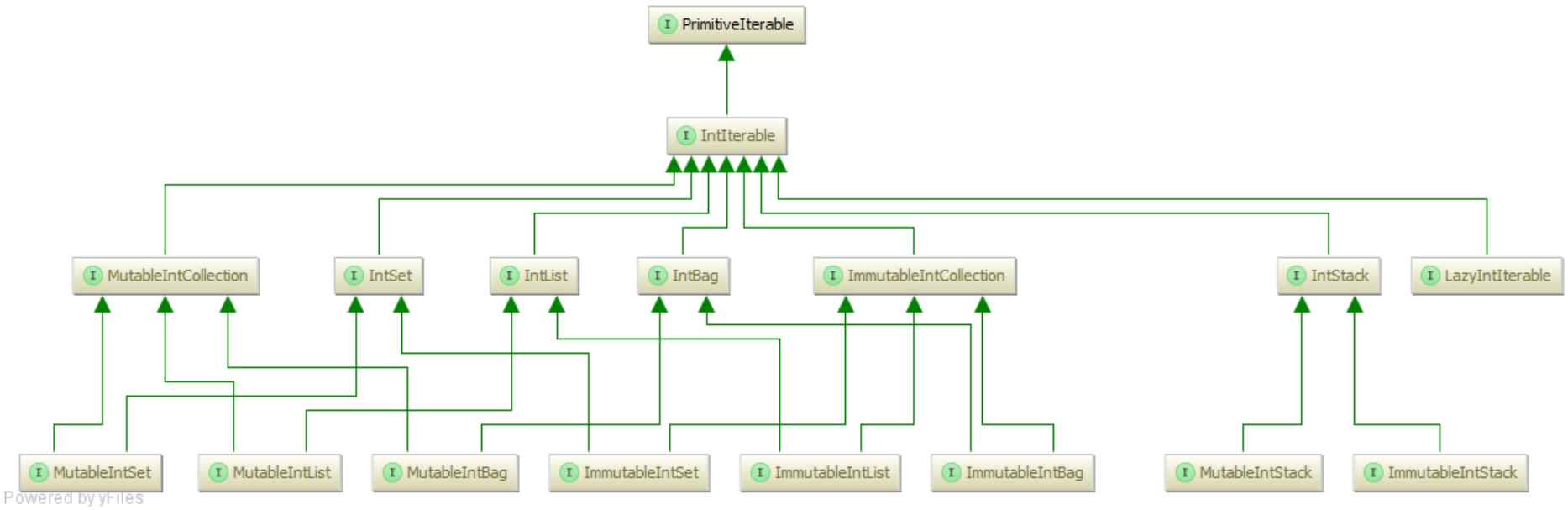
GS Collections adds 12 interfaces to enhance the 3 basic JDK Map interfaces
Note: GS Collections Maps are RichIterable on their values

GS Collections: Multimap Hierarchy



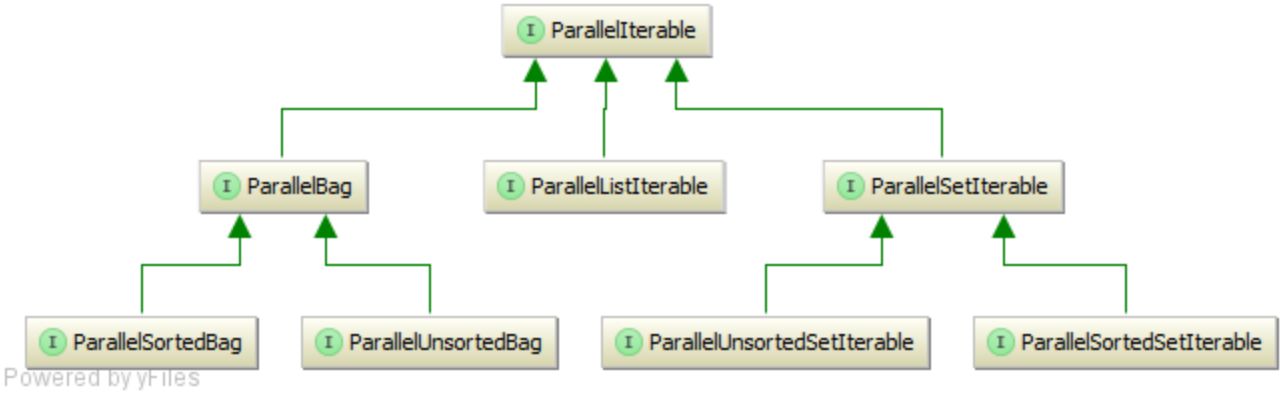
GS Collections adds 22 interfaces to support different forms of Multimaps

GS Collections: PrimitiveIterable Hierarchy

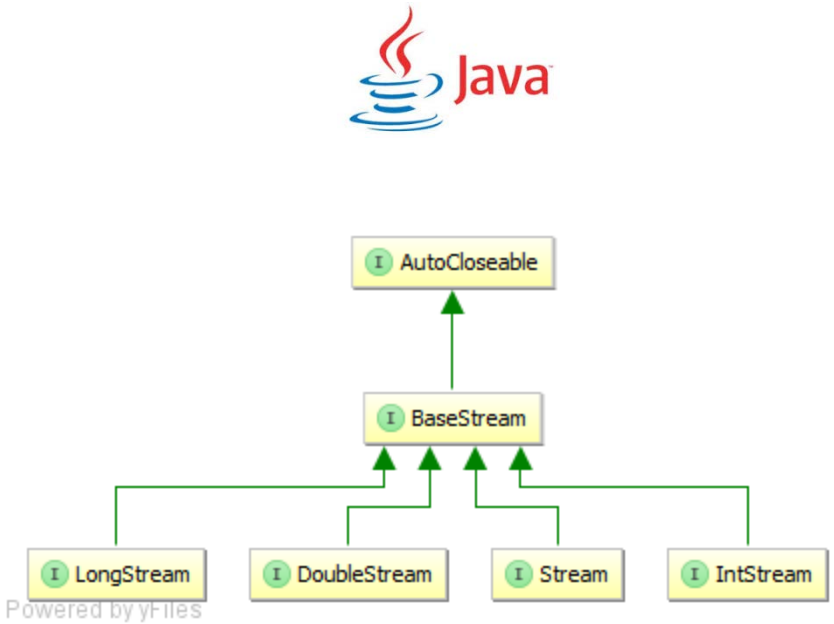


GS Collections adds 16 interfaces x 8 primitive types
(boolean, byte, char, double, float, int, long, short)

GS Collections: ParallelIterable Hierarchy



Java 8 Streams



GSC Kata Example#1 w/ JDK 5-7

Goldman Sachs

```
@Test
public void getCustomerNames()
{
    Function<Customer, String> fn = new Function<Customer, String>()
    {
        @Override
        public String valueOf(Customer customer)
        {
            return customer.getName();
        }
    };

    /**
     * Get the name of each of the company's customers.
     */
    MutableList<Customer> customers = this.company.getCustomers();
    MutableList<String> names = customers.collect(fn);
    MutableList<String> expectedNames =
        FastList.newListWith("Fred", "Mary", "Bill");
    Assert.assertEquals(expectedNames, names);
}
```

GSC Kata Example#1 w/ JDK 8

Goldman Sachs

```
@Test
public void getCustomerNames()
{
    /**
     * Get the name of each of the company's customers.
     */
    MutableList<Customer> cs = this.company.getCustomers();
    MutableList<String> names = cs.collect(Customer::getName);

    MutableList<String> expected =
        FastList.newListWith("Fred", "Mary", "Bill");
    Assert.assertEquals(expected, names);
}
```

* Saved 8 lines of code with method reference *

GSC Kata Example#2 w/ JDK 5-7

Goldman Sachs

```
@Test
public void filterOrderValues()
{
    MutableList<Order> orders = this.company.getMostRecentCustomer().getOrders();
    DoubleList filtered = this.company.getMostRecentCustomer()
        .getOrders()
        .asLazy()
        .collectDouble(Order.TO_VALUE)
        .select(DoublePredicates.greaterThan(1.5))
        .toSortedList();
    Assert.assertEquals(DoubleArrayList.newListWith(1.75, 372.5), filtered);
}
```

```
public class Order
{
    public static final DoubleFunction<Order> TO_VALUE = new DoubleFunction<Order>()
    {
        public double doubleValueOf(Order order)
        {
            return order.getValue();
        }
    };
    ...
```

GSC Kata Example#2 w/ JDK 8

Goldman Sachs

```
@Test
public void filterOrderValues()
{
    DoubleList filtered = this.company.getMostRecentCustomer()
        .getOrders()
        .asLazy()
        .collectDouble(Order::getValue)
        .select(value -> value > 1.5 )
        .toSortedList();

    Assert.assertEquals(
        DoubleArrayList.newListWith(1.75, 372.5),
        filtered);
}
```

* Saved 8 lines of code with method reference *

References

- Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.