BNY MELLON

JSR for Java Collections 2.0

Enhancing the Java Core Libraries for the next 25 years

Donald Raab, Chandra Guntur, Nikhil Nanivadekar

JCP Executive Committee Meeting

April 23, 2020

Information Classification: Public

Agenda

Some background on the past decade in the Java collections space

Proposal: Create a JSR for Java Collections 2.0

The Great News

A Glimpse of the Future

Recommendations

Appendix

– Serial / Eager Algorithms

BNY MELLON

Some background on the past decade in the Java collections space

- A Java collections framework design – JVM Language Summit – July 2012
- GS Collections and Java 8 – **JCP Executive Committee F2F Meeting** – London – May 2014
- Large HashMap overview: JDK, FastUtil, Goldman Sachs, HPPC, Koloboke, Trove – Jan. 2015
- GS Collections moves to the Eclipse Foundation – InfoQ – Jan. 2016
- Java Collections Cheat Sheet – JRebel – April 2016
- Energy Profiles of Java Collections Classes – May 2016
- Collections.Compare(() -> {JDK; Apache; Eclipse; Guava…}); – Devoxx US (March) & JavaOne 2017
- Vavr, Collections, and Java Stream API Collectors – Aug. 2017
- New Collection Types Explained – Nov. 2017
- Optimization Strategies with Eclipse Collections – April 2018
- Java Collections are Evolving – June 2018
- Recommending Energy-Efficient Java Collections – ACM – May 2019
- The Best of Java Collections [Tutorials] – Dec. 2019
- Java Streams are great but it's time for better Java Collections – Feb. 2020

BNY MELLON

Proposal: Create a JSR for Java Collections 2.0

Who will support a JSR for Java Collections 2.0?

- What are our plans for adding a new collections framework in Java 17 (September 2021)?

Determining the scope of Collections 2.0. This is some of what is missing today…

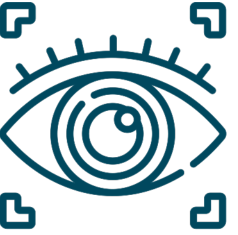
- Functional and optimized eager APIs directly on the collections
- Missing types: Multimaps, Bags, BiMaps, Tree/Trie, Table
- Memory Efficient Containers
- Separate Mutable and Immutable Hierarchies and Factories
- Primitive Collections
- 64-bit collections (size is long, arrays are 64-bit)
- Persistent Collections
- Off-heap Collections
- Lazy *Iterable* APIs for object and primitive collections
- A *distinct* Parallel *Iterable* Hierarchy

Some features may be out of scope. Some features could be delivered in follow on minor releases.

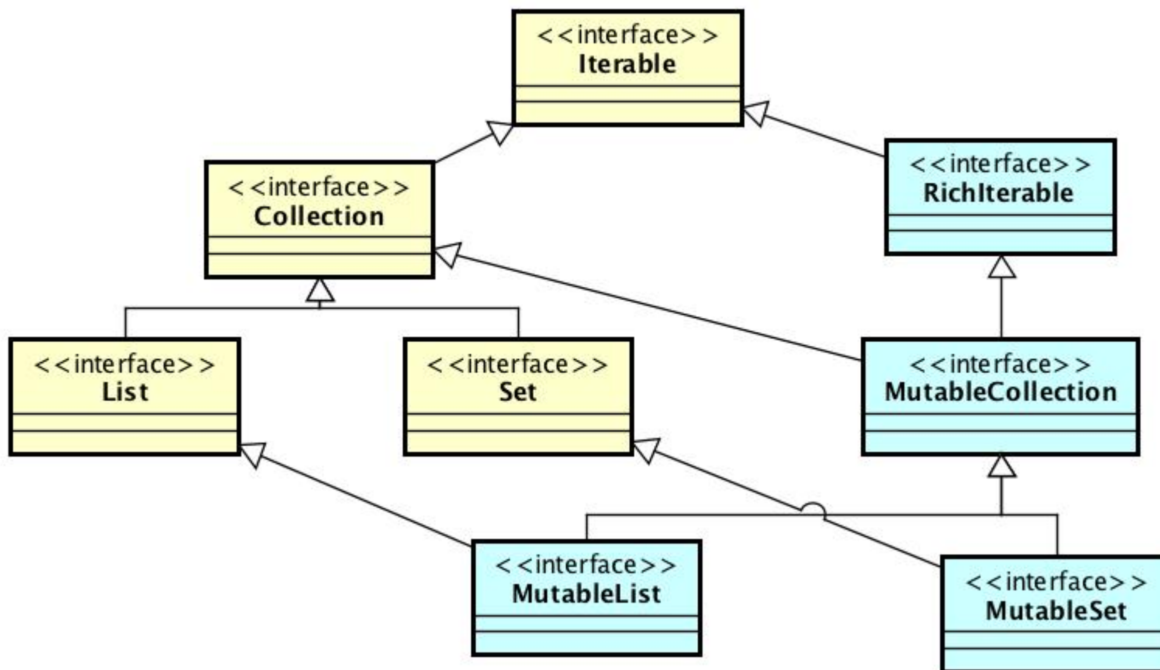
The Great News

- Java 8 with Lambdas, Methods References, Default Methods and Streams is an amazing success!
- There is an enormous body of work to learn and borrow from for a JSR for Java Collections 2.0
  - OSS Java Collections Frameworks
    - Apache Commons Collections
    - Eclipse Collections
    - FastUtil
    - Functional Java
    - Google Guava
    - Vavr
  - Languages
    - Lisp, Smalltalk, Python, Ruby
    - Groovy, Kotlin, Scala, Clojure
    - Haskell, F#, C#
    - Others

A Glimpse of the Future

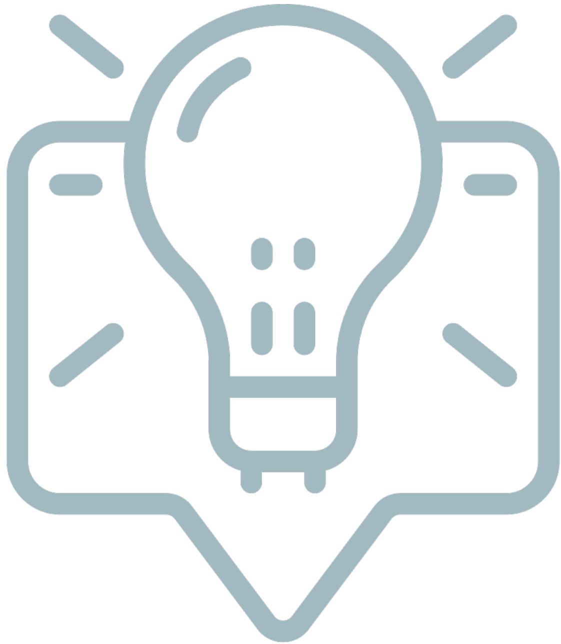


- Demo – Deck of Cards Kata from https://github.com/BNYMellon/CodeKatas
  - Kata for comparing Java Collections Frameworks
  - New Custom Collections Framework
    - Implements sampling of methods available on Java Streams



- **Eager Iteration Protocols**
  - `filter / filterNot`
  - `map / flatMap`
  - `reduce / collect`
  - `anyMatch / allMatch / noneMatch`
  - `groupBy / countBy`
  - `findFirst`
  - `count`
- **Converter Methods**
  - `toCollection, toList, toSet`
  - `asUnmodifiable`
- **Factory Methods**
  - `empty / of`
  - `fromIterable / fromStream`

Recommendations



- Java needs a new collections framework to help motivate developers to upgrade to newer releases.
  – A new framework should provide adaptation/transition from the existing framework
- Implement serial/eager iteration protocols directly on collections to make Java easier to teach and learn.
- Identify a JSR Spec Lead and form an Expert Group for a Java Collections 2.0 JSR
  – Leverage talent in OSS Collections talent for the Expert Group and Reference Implementation.

BNY MELLON

Appendix

Information Classification: Public

Serial / Eager Algorithms

- filter
- map
- collect
- groupBy (using Map)
- groupBy (using Multimap)
- countBy (using Map)
- countBy (using Bag)
- anyMatch
- allMatch
- noneMatch
- count
- findFirst

BNY MELLON

Make Java Easier to Learn – Serial / Eager – Filter

MutableList.filter(Predicate)

```
default MutableList<T> filter(Predicate<? super T> predicate) {
    var mutableList = MutableList.<T>empty();
    for (T each : this) {
        if (predicate.test(each)) {
            mutableList.add(each);
        }
    }
    return mutableList;
}
```


BNY MELLON

Make Java Easier to Learn – Serial / Eager – Map

MutableList.map(Function)

```
default <V> MutableList<V> map(Function<? super T, ? extends V> function) {
    var mutableList = MutableList.<V>empty();
    for (T each : this) {
        mutableList.add(function.apply(each));
    }
    return mutableList;
}
```


BNY MELLON

Make Java Easier to Learn – Serial / Eager – Collect

RichIterable.collect(Collector)

```
default <R, A> R collect(Collector<? super T, A, R> collector) {
    A mutableResult = collector.supplier().get();
    // BiConsumer<A, ? super T> accumulator = collector.accumulator();
    var accumulator = collector.accumulator();
    for (T each : this) {
        accumulator.accept(mutableResult, each);
    }
    return collector.finisher().apply(mutableResult);
}
```


BNY MELLON

Make Java Easier to Learn – Serial / Eager – GroupBy (using Map)

MutableList.groupBy(Function)

```
default <K, V> MutableMap<K, MutableList<T>> groupBy(Function<? super T, ? extends K> function) {
    var mutableMap = MutableMap.<K, MutableList<T>>empty();
    for (T each : this) {
        K key = function.apply(each);
        mutableMap.getIfAbsentPut(key, MutableList::empty)
                .add(each);
    }
    return mutableMap;
}
```


BNY MELLON

Make Java Easier to Learn – Serial / Eager – GroupBy (using Multimap)

MutableList.groupBy(Function)

```java
default <K> MutableListMultimap<K, T> groupBy(Function<? super T, ? extends K> function) {
    var multimap = MutableListMultimap.<K, T>empty();
    for (T each : this) {
        K key = function.apply(each);
        multimap.put(key, each);
    }
    return multimap;
}
```


Information Classification: Public
BNY MELLON

Make Java Easier to Learn – Serial / Eager – CountBy (using Map)

RichIterable.countBy(Function)

```
default <K> MutableMap<K, Long> countBy(Function<? super T, ? extends K> function) {
    MutableMap<K, Long> counts = MutableMap.empty();
    for (T each : this) {
        K key = function.apply(each);
        Long value = counts.get(key);
        if (value == null) {
            value = 0L;
        }
        counts.put(key, value + 1L);
    }
    return counts;
}
```


BNY MELLON

Make Java Easier to Learn – Serial / Eager – CountBy (using Bag)

RichIterable.countBy(Function)

```java
default <K> MutableBag<K> countBy(Function<? super T, ? extends K> function) {
    MutableBag<K> counts = MutableBag.empty();
    for (T each : this) {
        K key = function.apply(each);
        counts.add(key);
    }
    return counts;
}
```


Information Classification: Public

Make Java Easier to Learn – Serial / Eager – AnyMatch

RichIterable.anyMatch(Predicate)

```java
default boolean anyMatch(Predicate<? super T> predicate) {
    for (T each : this) {
        if (predicate.test(each)) {
            return true;
        }
    }
    return false;
}
```


BNY MELLON

Make Java Easier to Learn – Serial / Eager –AllMatch

RichIterable.allMatch(Predicate)

```java
default boolean allMatch(Predicate<? super T> predicate) {
    for (T each : this) {
        if (!predicate.test(each)) {
            return false;
        }
    }
    return true;
}
```


BNY MELLON

Make Java Easier to Learn – Serial / Eager – NoneMatch

RichIterable.noneMatch(Predicate)

```
default boolean noneMatch(Predicate<? super T> predicate) {
    for (T each : this) {
        if (predicate.test(each)) {
            return false;
        }
    }
    return true;
}
```


Information Classification: Public

BNY MELLON

Make Java Easier to Learn – Serial / Eager – Count

RichIterable.count(Predicate)

```
default int count(Predicate<? super T> predicate) {
    int count = 0;
    for (T each : this) {
        if (predicate.test(each)) {
            count++;
        }
    }
    return count;
}
```


Information Classification: Public

BNY MELLON

Make Java Easier to Learn – Serial / Eager – FindFirst

RichIterable.findFirst(Predicate)

```
default Optional<T> findFirst(Predicate<? super T> predicate) {
    for (T each : this) {
        if (predicate.test(each)) {
            return Optional.of(each);
        }
    }
    return Optional.empty();
}
```


BNY MELLON

Disclosures & Disclaimers

BNY Mellon is the corporate brand of The Bank of New York Mellon Corporation and may be used as a generic term to reference the corporation as a whole and/or its various subsidiaries generally. Products and services may be provided under various brand names in various countries by duly authorized and regulated subsidiaries, affiliates, and joint ventures of The Bank of New York Mellon Corporation. Not all products and services are offered in all countries.

BNY Mellon will not be responsible for updating any information contained within this material and opinions and information contained herein are subject to change without notice.

BNY Mellon assumes no direct or consequential liability for any errors in or reliance upon this material. This material may not be reproduced or disseminated in any form without the express prior written permission of BNY Mellon.

©2020 The Bank of New York Mellon Corporation. All rights reserved.

BNY MELLON