

FINALLY

Java 8

Functional, Fluent And Fun!

New features of the language and how GS
Collections makes them even better, faster,
stronger

May 19, 2014

We Did It!

“Two years, seven months, and eighteen days after the release of JDK 7, production-ready builds of JDK 8 are now available for download!”

(from a blog post by Mark Reinhold, the Chief Architect of the Java Platform Group at Oracle)

Java 8 Is A Major Feature Release

- **Lambda Expressions** – treat functionality or code as data
- **Method references** – easy-to-read lambda expressions for methods that already have a name
- **Default methods** – add new functionality to the interfaces while being binary compatible with legacy code
- **Type Annotations** – apply an annotation anywhere a type is used, not just on a declaration
- **Improved type inference** – get better typing with less typing
- **Stream API** – functional-style operations on streams of elements, integrated into the Collections API
- **Date-Time** – packages that provide a comprehensive date-time model
- **Nashorn** – a JavaScript runtime in Java
- ...and a lot more

Lambda Expressions

- Aka “closures” or “anonymous methods”
- Can be assigned to variables, passed as arguments to a method invocation or returned as the result of a method or another lambda expression
- Address drawbacks of anonymous inner classes
 - Lightweight syntax
 - Inferred final
- Fit into the existing Java type system
 - Functional interfaces (one abstract method)
 - Compatible with the existing frameworks
 - Target typing – the type of a lambda expression is inferred from the context

Lambdas and Method References

```
Runnable run = () -> System.out.println("Hello");  
Comparator<String> cL =  
    (s1, s2) -> s1.compareTo(s2);
```

HELLO
my name is
Expression
Lambda

```
Comparator<String> cMR = String::compareTo;  
Callable<ArrayList> callable = ArrayList::new;  
Procedure<String> procedure = System.out::println;  
Function<Integer, String> function = String::valueOf;  
Predicate<String> isItFred = "Fred"::equals;
```

HELLO
my name is
Method
Reference

```
TransactionManager.execute(() -> {  
    Person person = new Person();  
    person.setName("Bob");  
    person.setAge(55);  
});
```


HELLO
my name is
Statement
Lambda

Improved Type Inference


- Improve readability of code by reducing explicit type-arguments in generic method calls
- For each potentially applicable method, the compiler determines whether the lambda expression is compatible with the corresponding target type, and also infers any type arguments
- If the choice of a best declaration is ambiguous, casts can provide a workaround

```
private List<Person> people =  
    Arrays.asList(new Person("Dave", 23), new Person("Joe", 32), new Person("Bob", 17));
```

```
this.people.sort(new Comparator<Person>()  
{  
    public int compare(Person p1, Person p2)  
    {  
        return p1.getName().compareTo(p2.getName());  
    }  
});
```



```
Comparator<Person> c = (Person p1, Person p2) -> p1.getName().compareTo(p2.getName());  
this.people.sort(c);
```



```
this.people.sort((p1, p2) -> p1.getName().compareTo(p2.getName()));
```

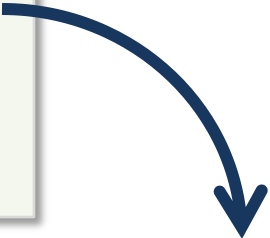
Effectively Final

```
public void testOldStyle()
{
    final String name = "Bob";

    Runnable r = new Runnable()
    {
        public void run()
        {
            System.out.println("hello, " + name);
        }
    };

    new Thread(r).start();
}
```

A local variable is effectively final if its initial value is never changed — in other words, declaring it final would not cause a compilation failure



```
public void testEightStyle()
{
    String name = "Bob";
    Runnable r = () -> System.out.println("hello, " + name);
    new Thread(r).start();
}
```

Streams

- Stream is a way to extend Collections with bulk operations (filter(), map(), etc.)
- Why Streams and not just new methods on Collection?
 - Reducing conflict surface area – new methods may conflict with the methods others have added to collections (Hello, GS Collections!)
 - Reducing user model confusion – in-place mutation vs. producing new streams (mutative and functional methods)
 - A type for intermediate results of bulk operations
 - Stream != Collection
 - Stream != Iterable
 - Stream is like an Iterator – the values flow by, and when they're consumed, they're gone

Stream Examples

```
List<Person> people = Arrays.asList(  
    new Person("Bob", 17),  
    new Person("Dave", 23),  
    new Person("Joe", 32));
```

```
Stream<String> nameStream = people.stream()  
    .filter(person -> person.getAge() > 21)  
    .map(Person::getName);
```

```
nameStream.forEach(System.out::println);
```

```
// When stream values are gone, they are gone. Let's try again:  
nameStream.forEach(System.out::println);
```

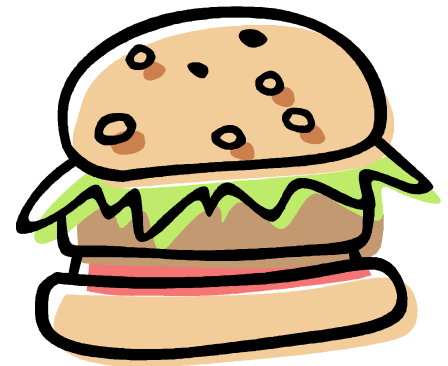
```
java.lang.IllegalStateException: stream has already been operated upon or closed  
    at java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:229)  
    at java.util.stream.ReferencePipeline.forEach(ReferencePipeline.java:418)  
    ...
```

Streams: “Like A Burger”

```
List<Person> people = Arrays.asList(new Person("Bob", 17),  
    new Person("Dave", 23), new Person("Joe", 32));
```

```
List<String> names = people.stream() ← Bun  
    .filter(person -> person.getAge() > 21)  
    .map(Person::getName) ← Meat  
    .collect(Collectors.<String>toList()); ← Bun
```

```
names.forEach(System.out::println);
```



Default Methods

- Default methods are a mechanism to extend interfaces in a backward compatible way
- Until Java 8 adding new methods to an interface has been impossible without forcing modification to its existing subtypes

“Public defender methods” – if you can’t afford an implementation, one will be provided for you at no charge

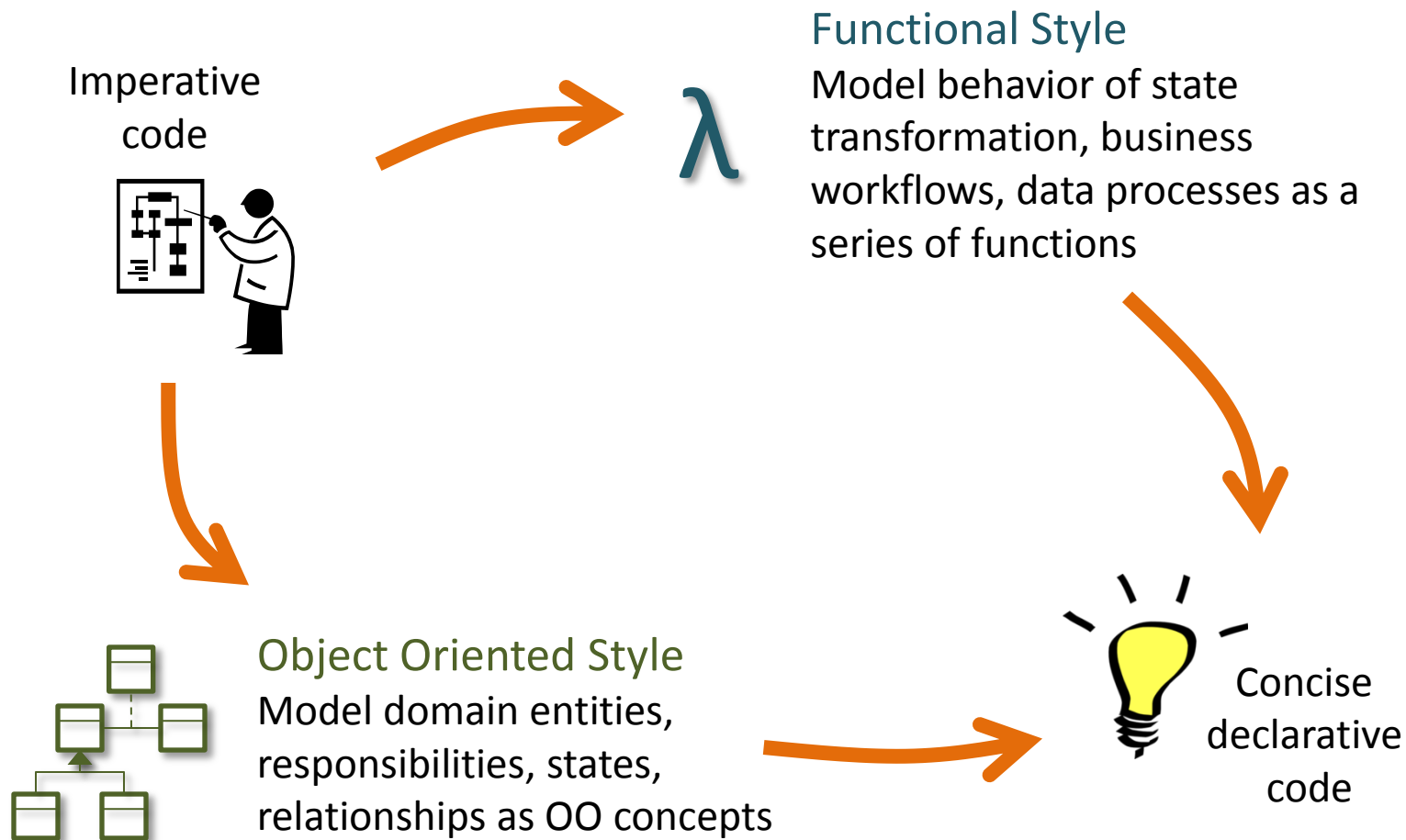
- When you extend an interface that contains a default method, you can do the following:
 - Not mention the default method at all, which lets your extended interface inherit the default method
 - Redecclare the default method, which makes it abstract
 - Redefine the default method, which overrides it
- Interfaces are stateless
 - Java does not have multiple implementation inheritance
 - Java always had multiple interface inheritance

Default Method Examples - JDK

```
public interface Iterable<T> {  
    ...  
    default Spliterator<T> spliterator() {  
        return Spliterators.spliteratorUnknownSize(iterator(), 0);  
    }  
    ...  
    default void forEach(Consumer<? super T> action) {  
        Objects.requireNonNull(action);  
        for (T t : this) {  
            action.accept(t);  
        }  
    }  
    ...  
}
```

```
public interface Collection<E> extends Iterable<E> {  
    ...  
    @Override  
    default Spliterator<E> spliterator() {  
        return Spliterators.spliterator(this, 0);  
    }  
    ...  
    default Stream<E> stream() {  
        return StreamSupport.stream(spliterator(), false);  
    }  
    ...  
}
```

Java: Object Oriented & Functional



What is GS Collections?

- 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

GSC Lines of Code



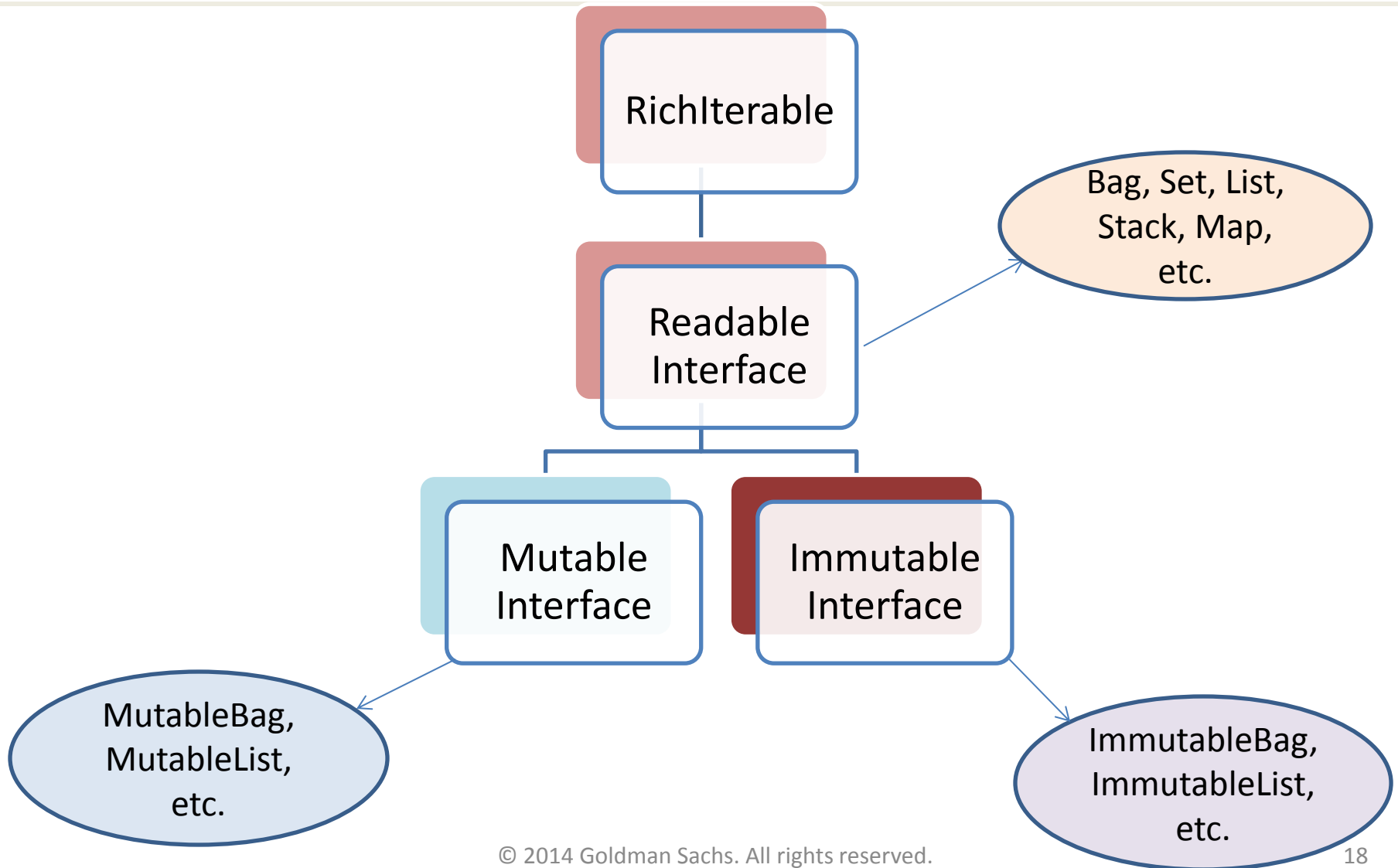
- 295,036 lines of Java code
- 9,959 lines of Scala code
- 76,240 lines of StringTemplate templates
- Total Java code after code generation:
 - **1,438,477** lines of Java code your developers don't have to write and can use for free

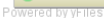
Framework Comparisons

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)

[illegible]

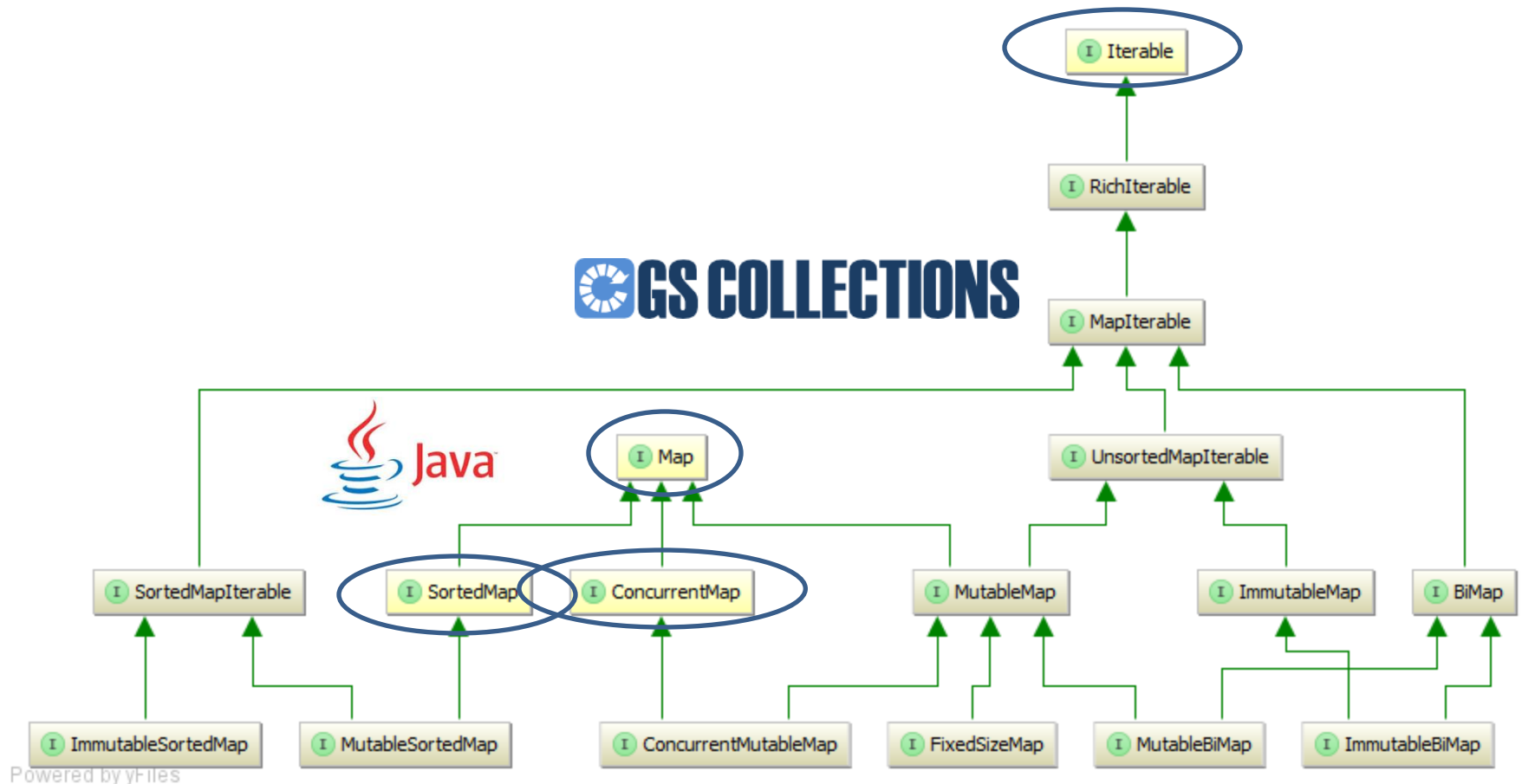
Design Concepts





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

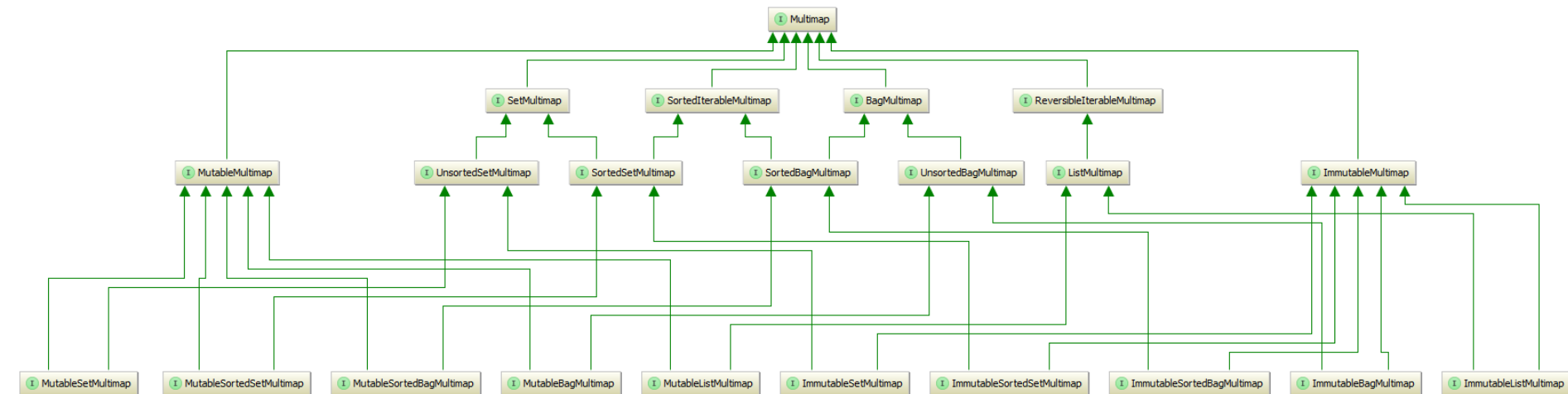
Map Hierarchy



GS Collections adds 12 interfaces to enhance the 3 basic JDK Map interfaces

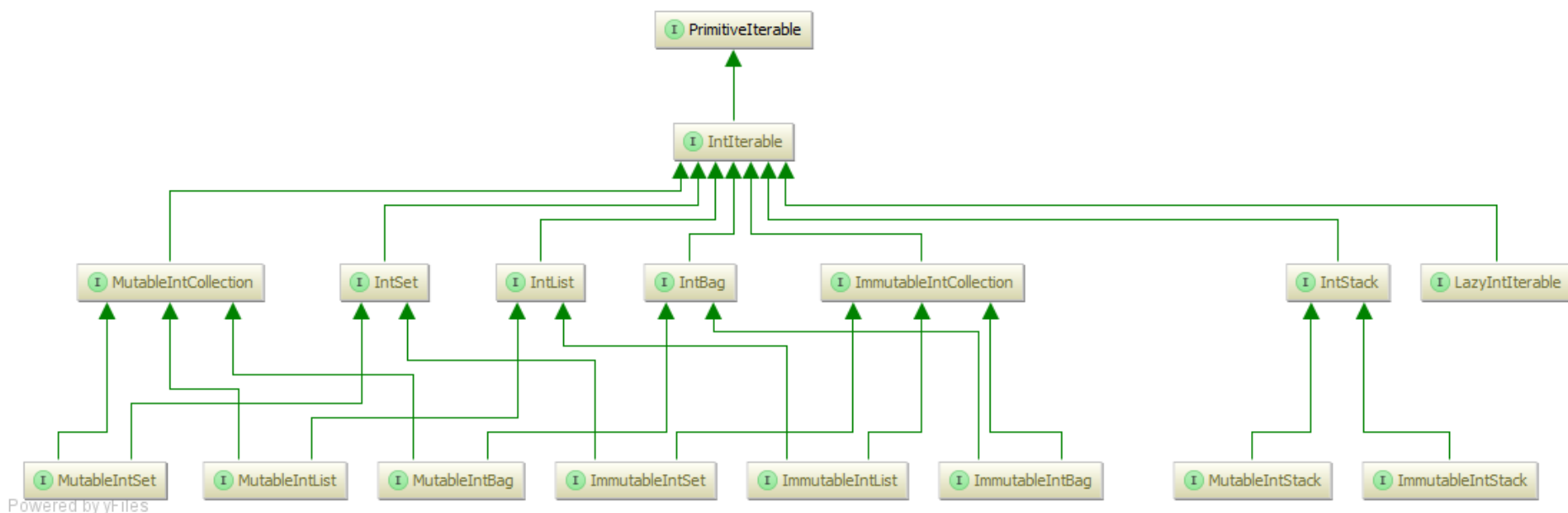
Note: GS Collections Maps are RichIterable on their values

Multimap Hierarchy



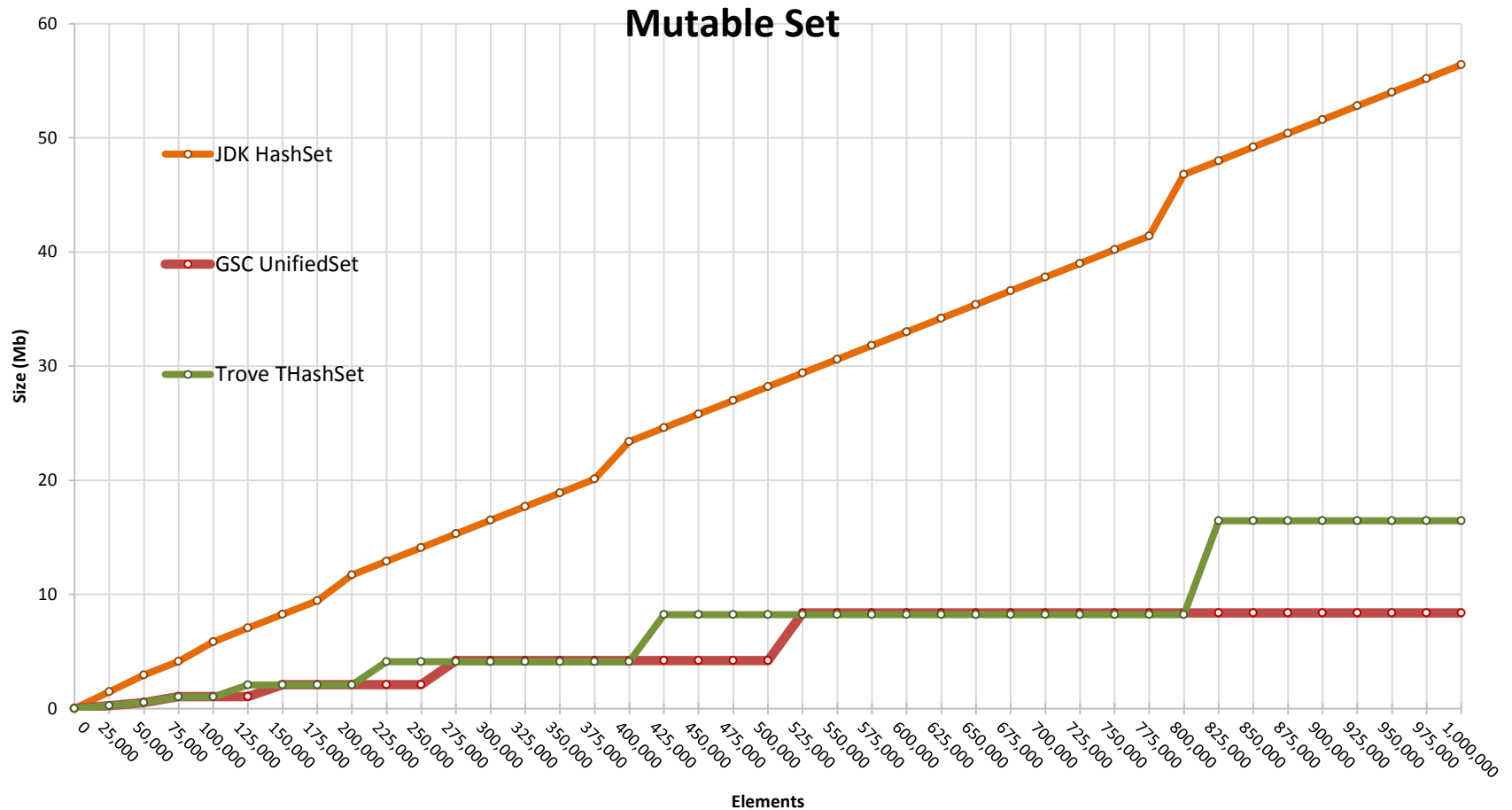
GS Collections adds 22 interfaces to support different forms of Multimaps

Primitive Type Collection Hierarchy



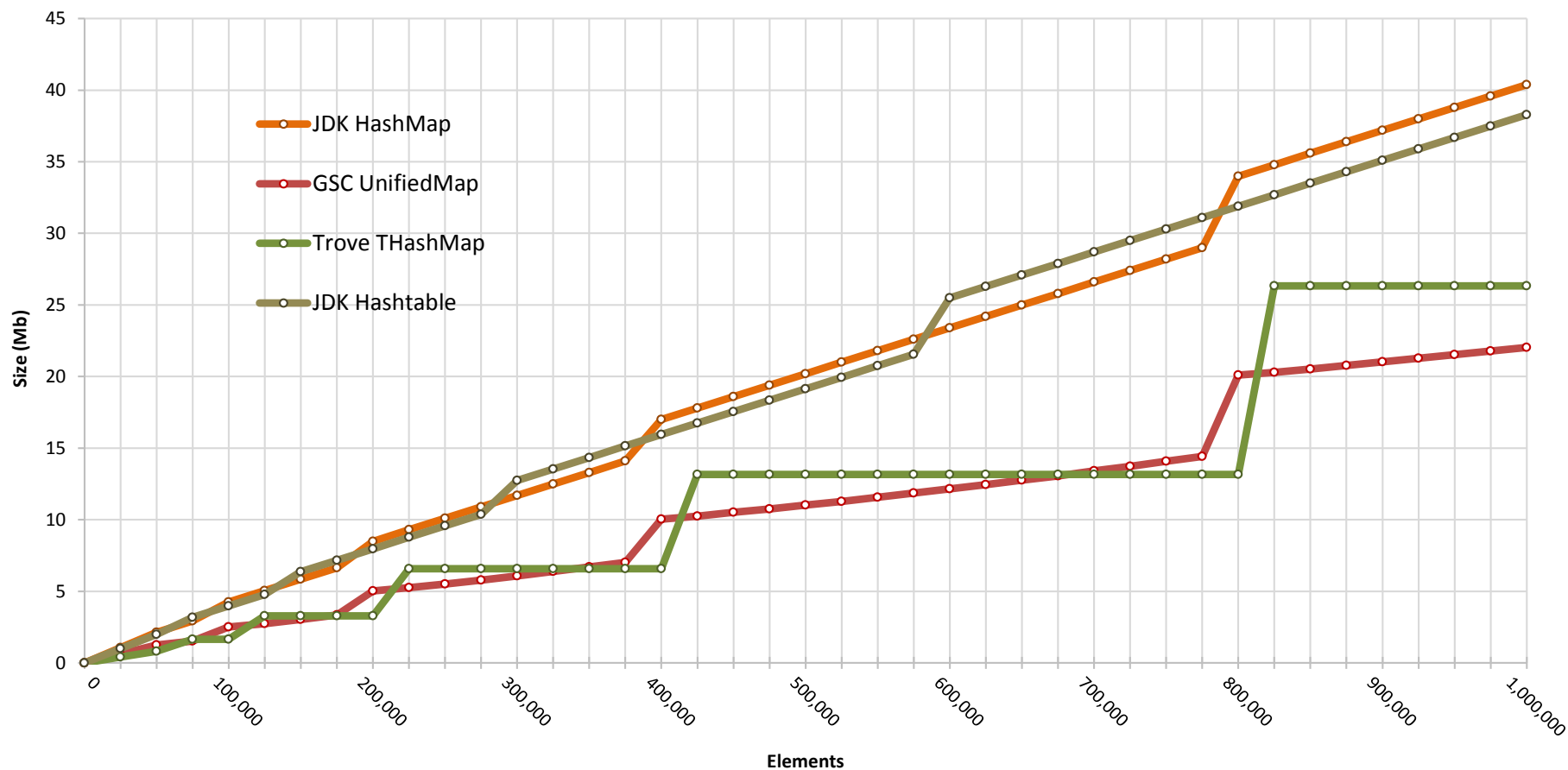
**GS Collections adds 16 interfaces x 8 primitive types
(boolean, byte, char, double, float, int, long, short)**

Save memory with UnifiedSet



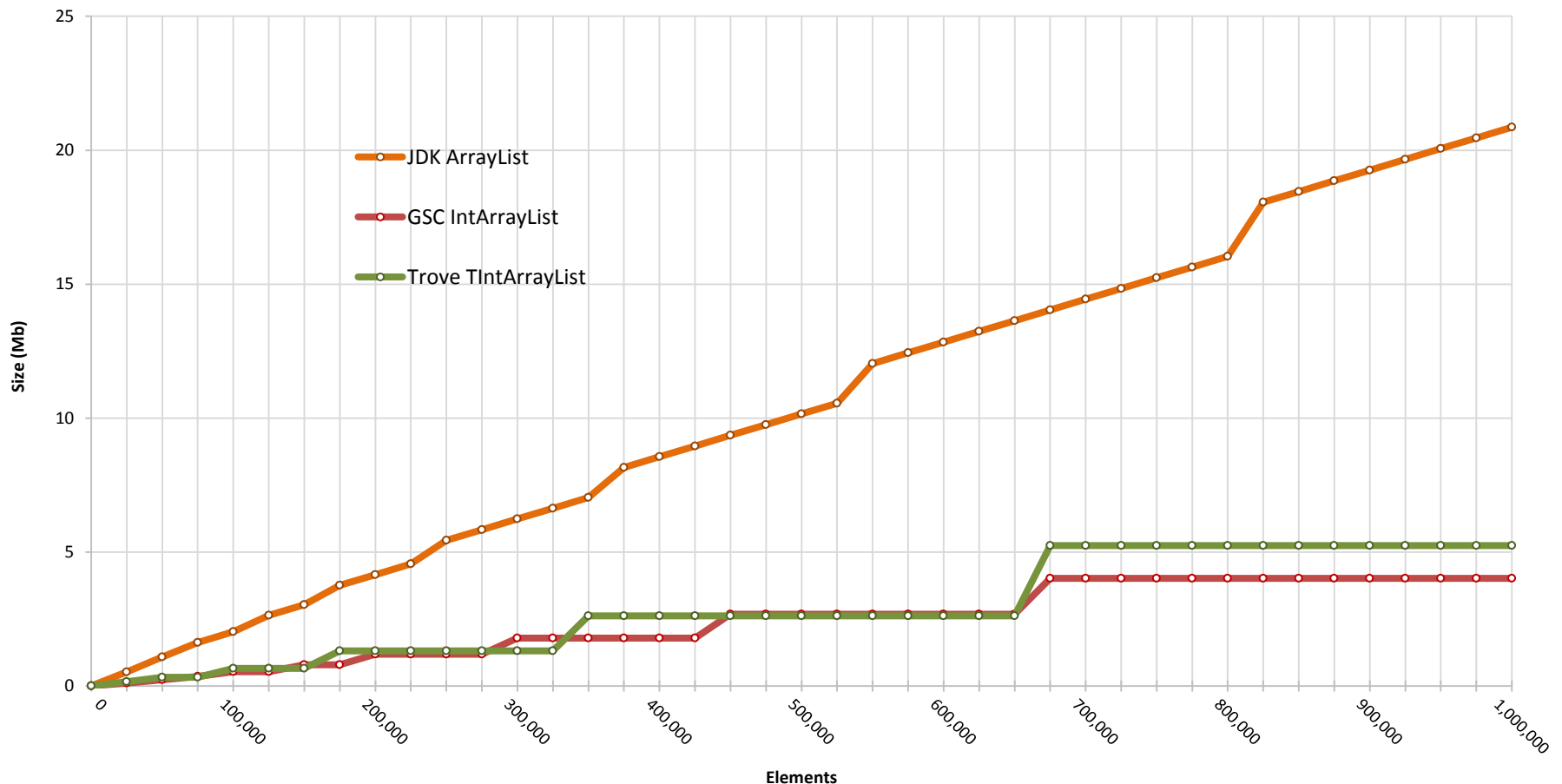
Save memory with UnifiedMap

Mutable Map



Save memory with Primitive Collections

IntList



Reducing code using lambdas

- Converted most of our anonymous inner classes in unit 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
- Biggest single class savings ~700 lines of code.

Eager vs. Lazy Iteration

Iteration Style	GSC Example	JDK 8 Example
Serial Eager (collect)	<pre>MutableList<Address> addresses = people.collect(Person::getAddress);</pre>	
Serial Lazy (collect / map)	<pre>LazyIterable<Address> addresses = people.asLazy() .collect(Person::getAddress);</pre>	<pre>Stream<Address> addresses = people.stream() .map(Person::getAddress);</pre>
Serial Lazy (collect / map, toList)	<pre>MutableList<Address> addresses = people.asLazy() .collect(Person::getAddress) .toList();</pre>	<pre>List<Address> addresses = people.stream() .map(Person::getAddress) .collect(Collectors.toList());</pre>
Parallel Eager	<pre>Collection<Address> addresses = ParallelIterate.collect(people, Person::getAddress);</pre>	
Parallel Lazy	<pre>ParallelListIterable<Address> addresses = people.asParallel(executor, batchSize) .collect(Person::getAddress);</pre>	<pre>Stream<Address> addresses = people.parallelStream() .map(Person::getAddress);</pre>

GSC Kata Example#1 w/ JDK 5-7

```
@Test
public void getCustomerNames()
{
    Function<Customer, String> nameFunction = new Function<Customer, String>()
    {
        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> customerNames = customers.collect(nameFunction);
    MutableList<String> expectedNames =
        FastList.newListWith("Fred", "Mary", "Bill");
    Assert.assertEquals(expectedNames, customerNames);
}
```

GSC Kata Example#1 w/ JDK 8

@Test

```
public void getCustomerNames()
```

```
{
```

```
    Function<Customer, String> fn = c -> c.getName();
```

```
    /**
```

```
     * Get the name of each of the company's customers.
```

```
     */
```

```
    MutableList<Customer> customers = this.company.getCustomers();
```

```
    MutableList<String> customerNames = customers.collect(fn);
```

```
    MutableList<String> expectedNames =
```

```
        FastList.newListWith("Fred", "Mary", "Bill");
```

```
    Assert.assertEquals(expectedNames, customerNames);
```

```
}
```

GSC Kata Example#2 w/ JDK 5-7

```
@Test
public void getLondonCustomers()
{
    MutableList<Customer> customers = this.company.getCustomers();

    MutableList<Customer> selected =
        customers.select(new Predicate<Customer>()
        {
            public boolean accept(Customer customer)
            {
                return "London".equals(customer.getCity());
            }
        }));

    Verify.assertSize("Should be 2 London customers", 2, selected);
}
```

GSC Kata Example#2 w/ JDK 8

```
@Test
public void getLondonCustomers()
{
    MutableList<Customer> customers = this.company.getCustomers();

    MutableList<Customer> selected =
        customers.selectWith(Customer::livesIn, "London");

    Verify.assertSize("Should be 2 London customers", 2, selected);
}

// On Customer class
public boolean livesIn(String city)
{
    return city.equals(this.city);
}
```

GSC Kata Example#3 w/ JDK 5-7

```
@Test
public void filterOrderValues()
{
    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#3 w/ JDK 8

@Test

```
public void filterOrderValues()
{
    DoubleList filtered = this.company.getMostRecentCustomer()
        .getOrders()
        .asLazy()
        .collectDouble(Order::getValue)
        .select(DoublePredicates.greaterThan(1.5))
        .toSortedList();
    Assert.assertEquals(
        DoubleArrayList.newListWith(1.75, 372.5),
        filtered);
}
```

GSC Kata Example#4 w/ JDK 5-7 (1)

*// Create a multimap where the values are customers and the key is the price of
// the most expensive item that the customer ordered.*

@Test

```
public void mostExpensiveItem()
{
    MutableListMultimap<Double, Customer> multimap =
        this.company.getCustomers().groupBy(new Function<Customer, Double>()
        {
            public Double valueOf(Customer customer)
            {
                return customer.getOrders()
                    .asLazy()
                    .flatCollect(Order.TO_LINE_ITEMS)
                    .collectDouble(LineItem.TO_VALUE)
                    .max();
            }
        });
    Assert.assertEquals(3, multimap.size());
    Assert.assertEquals(2, multimap.keysView().size());
    Assert.assertEquals(
        FastList.newListWith(
            this.company.getCustomerNamed("Fred"),
            this.company.getCustomerNamed("Bill")),
        multimap.get(50.0));
}
```

GSC Kata Example#4 w/ JDK 5-7 (2)

// In Order Class

```
public static final Function<Order, Iterable<LineItem>> TO_LINE_ITEMS =  
    new Function<Order, Iterable<LineItem>>()  
    {  
        public Iterable<LineItem> valueOf(Order order)  
        {  
            return order.lineItems;  
        }  
    };
```

// In LineItem Class

```
public static final DoubleFunction<LineItem> TO_VALUE =  
    new DoubleFunction<LineItem>()  
    {  
        public double doubleValueOf(LineItem lineItem)  
        {  
            return lineItem.value;  
        }  
    };
```

GSC Kata Example#4 w/ JDK 8

*// Create a multimap where the values are customers and the key is the price of
// the most expensive item that the customer ordered.*

@Test

public void mostExpensiveItem()

```
{  
    MutableListMultimap<Double, Customer> multimap = this.company.getCustomers()  
        .groupBy(customer -> customer.getOrders()  
            .asLazy()  
            .flatCollect(Order::getLineItems)  
            .collectDouble(LineItem::getValue)  
            .max());  
  
    Assert.assertEquals(3, multimap.size());  
    Assert.assertEquals(2, multimap.keySet().size());  
    Assert.assertEquals(  
        FastList.newListWith(  
            this.company.getCustomerNamed("Fred"),  
            this.company.getCustomerNamed("Bill")),  
        multimap.get(50.0));  
}
```

GSC Kata Example#5 w/ JDK 8

```
@Test
public void totalOrderValuesByCity()
{
    MutableMap<String, Double> map = this.company.getCustomers()
        .aggregateBy(Customer::getCity,
            () -> 0.0,
            (result, customer) -> result + customer.getTotalOrderValue());
    Assert.assertEquals(2, map.size());
    Assert.assertEquals(446.25, map.get("London"), 0.0);
    Assert.assertEquals(857.0, map.get("Liphook"), 0.0);
}
```

```
@Test
public void totalOrderValuesByItem()
{
    MutableMap<String, Double> map = this.company.getOrders()
        .flatCollect(Order::getLineItems)
        .aggregateBy(LineItem::getName,
            () -> 0.0,
            (result, lineItem) -> result + lineItem.getValue());
    Verify.assertSize(12, map);
    Assert.assertEquals(100.0, map.get("shed"), 0.0);
    Assert.assertEquals(10.5, map.get("cup"), 0.0);
}
```

GSC Kata Example#6 w/ JDK 8

```
@Test
public void doubleSummaryStatistics()
{
    DoubleSummaryStatistics stats = new DoubleSummaryStatistics();
    LazyDoubleIterable iterable = this.company.getCustomers()
        .asLazy()
        .flatCollect(Customer::getOrders)
        .flatCollect(Order::getLineItems)
        .collectDouble(LineItem::getValue);
    iterable.forEach(stats::accept);

    Assert.assertEquals(iterable.min(), stats.getMin(), 0.0);
    Assert.assertEquals(iterable.max(), stats.getMax(), 0.0);
    Assert.assertEquals(iterable.average(), stats.getAverage(), 0.0);
    Assert.assertEquals(iterable.sum(), stats.getSum(), 0.0);
    Assert.assertEquals(iterable.size(), stats.getCount(), 0.0);
    Assert.assertEquals(iterable.median(), 7.5, 0.0);
}
```

Is GS Collection Fast?

Jon Brisbin @j_brisbin Mar 28

@GoldmanSachs gs-collections multimap is 2x faster than Guava's in my JMH benches: <https://github.com/reactor/reactor-benchmark/blob/master/src/main/java/org/projectreactor/bench/Collection/CacheBenchmarks.java> ... **@ProjectReactor**

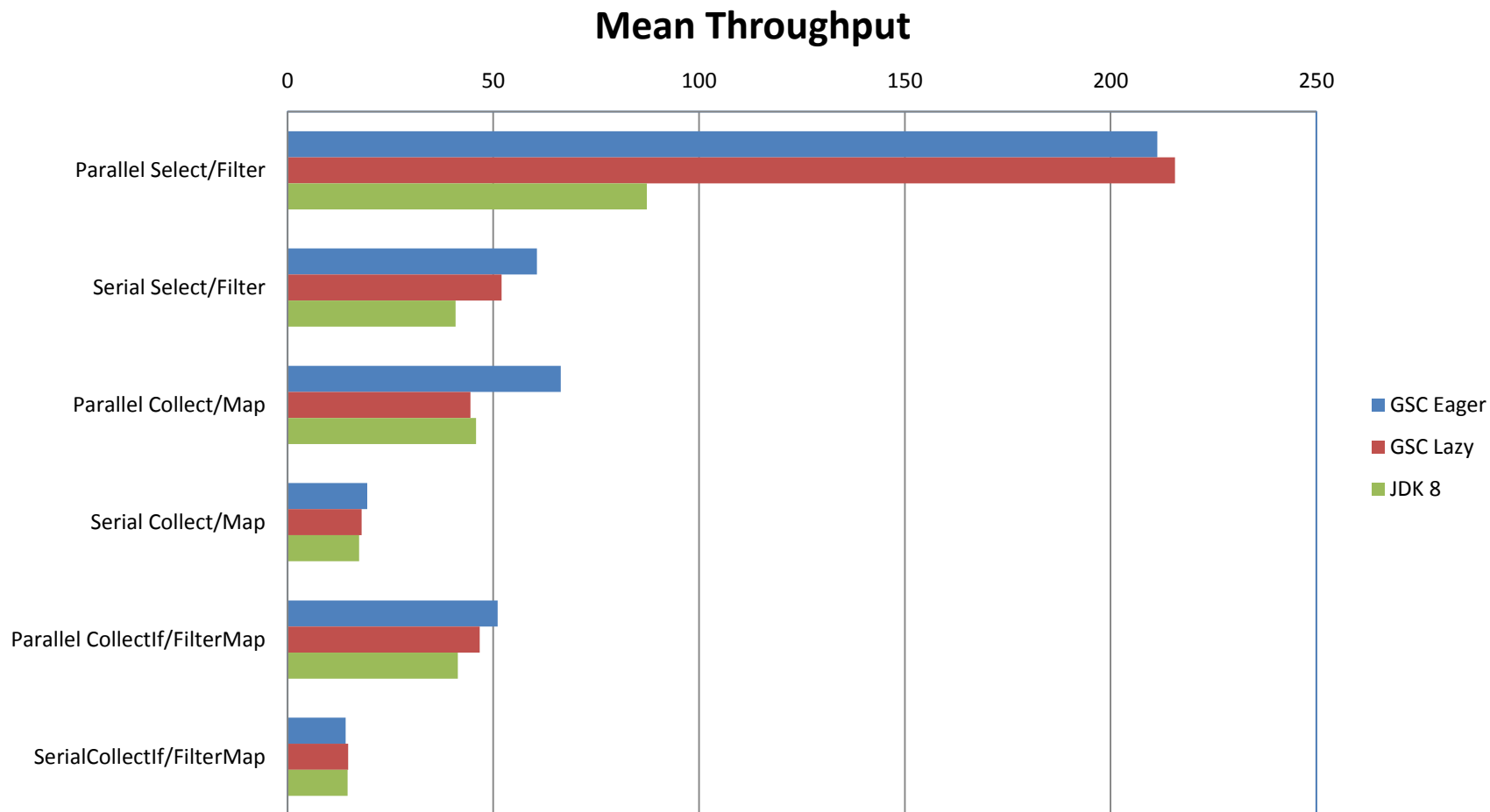
What is JMH?

- JMH = Java Microbenchmark Harness
- An Open JDK Code Tools Project

“JMH is a Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targetting the JVM.”

<http://openjdk.java.net/projects/code-tools/jmh/>

JDK 8 vs. GS Collections Benchmarks



Select Benchmark Code using JMH

```

@State(Scope.Thread) @BenchmarkMode(Mode.Throughput) @OutputTimeUnit(TimeUnit.SECONDS)
public class SelectTest
{
    private static final int SIZE = 1_000_000;
    private final List<Integer> integersJDK = new ArrayList<>(Interval.oneTo(SIZE));
    private final FastList<Integer> integersGSC = FastList.newList(Interval.oneTo(SIZE));

    @GenerateMicroBenchmark
    public void serial_lazy_jdk()
    {
        List<Integer> evens = this.integersJDK.stream().filter(each -> each % 2 == 0).collect(Collectors.toList());
        Assert.assertEquals(SIZE / 2, evens.size());
    }

    @GenerateMicroBenchmark
    public void serial_eager_gsc()
    {
        MutableList<Integer> evens = this.integersGSC.select(each -> each % 2 == 0);
        Assert.assertEquals(SIZE / 2, evens.size());
    }
    ...
}

```

Output:

Benchmark	Mode	Samples	Mean	Mean error	Units
c.g.c.i.s.SelectTest.gscEagerSerialSelect	thrpt	25	59.580	3.881	ops/s
c.g.c.i.s.SelectTest.jdk8SerialFilter	thrpt	25	38.676	0.527	ops/s

- Java 8 Release Notes
<http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html>
- GS Collections on GitHub
<https://github.com/goldmansachs/gs-collections>
- GS Collections Kata on GitHub
<https://github.com/goldmansachs/gs-collections-kata>
- GS Collections Memory Benchmark
http://www.goldmansachs.com/gs-collections/presentations/GSC_Memory_Tests.pdf
- Functional Interfaces
<http://docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html>
- NY JUG Presentation, March 2013
http://www.goldmansachs.com/gs-collections/presentations/NYJUG_March_18_2013_GSCollections.pdf

References

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