A Java™ collections framework design
GS Collections™ explained
JVM Language Summit 2012

<table>
<thead>
<tr>
<th>Version</th>
<th>By</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Don Raab</td>
<td>July 2012</td>
</tr>
</tbody>
</table>
Who am I?

A Vice President and Tech Fellow @ Goldman Sachs

Programming Language Experience

- BASIC, Fortran, COBOL, Pascal, Dbase III+, Prolog, Clipper, Assembly, Lisp, CA-Visual Objects, CA-Realizer, REXX, Smalltalk, C++, Java™, Visual Basic, JavaScript, Scala

Creator of GS Collections

- An open source Java collections framework available on GitHub
  - https://github.com/goldmansachs/gs-collections
- Training materials and exercises for GS Collections also on GitHub
  - https://github.com/goldmansachs/gs-collections-kata
- Initial work started in 2004 focused on memory efficient structures
  - Fixed Sized Lists/Sets/Maps and Pools for basic immutable types
- Rich iterable API work started in 2005
Collections are fundamental components of a programming language
  – Up there with statements and operators
  – Used in all but the most trivial of applications

Think in terms of behavioral patterns, not in terms of loops and iterators
  – GS Collections builds upon the Java interfaces adding a rich protocol with many behavioral implementation patterns
  – GS Collections will be able to leverage and benefit from Java 8 Lambda syntax when it is released

There and back again:
  – Java replaced rich collection libraries of its predecessors with minimalist approach
    • Java has collection types – Java is missing iteration patterns
    • Java 8 will have lambdas, which will make functional APIs easier to use
  – New languages recreate the richness of humane interfaces of old
  – And so do dozens of Java collection libraries
GS Collections Explained

- Overall Design Goals and Challenges
- Humane API
  - API’s Compared (Java, Smalltalk, Scala and GS Collections)
  - How rich is your iterable?
- Reduce Memory Footprint
  - Reduce Static Memory
  - Empty should be empty
- Improve Performance
  - Optimize methods by type
  - Optimize for no-ops (like Empty)
  - Prevent unnecessary garbage
- Appendix
  - How did we get here?
  - Example Iteration Patterns
  - The Java Collections Tutorial
  - Anagrams Reloaded
Overall Design Goals

- Bring back the expressiveness of Smalltalk-80
  - Implement basic Smalltalk iteration patterns (select, reject, collect, detect, etc.)
  - Implement converter methods for container types (toList, toSet)
  - Add humane api for common use cases we find (e.g. groupBy, partition, etc.)
  - Behavior is eager by default, and lazy by request (asLazy)
  - Make the API OO/functional and fluent to aid in framework discoverability

- Make compatible with Java Collections Framework interfaces
  - Extend Iterable, Collection, List, Set, SortedSet, Map
  - Add support for Unmodifiable and Synchronized constructs on API
  - Compatible back to Java 5 to increase usability

- Add missing collection types
  - Bag, Multimap, Interval, Stack (as interface)
  - Lazy iterables
  - Primitive Collections
  - Implement Immutable versions of mutable container types
    - Mutable types are builders for their immutable types -> toImmutable() 
  - Implement parallel versions of common iteration patterns
Minimal API vs. Humane API

GS Collections

Smalltalk-80

- asSet toSet
- asBag toBag
- asSortedCollection toSortedList

- do: forEach forEach
- select: filter select
- reject: filterNot reject
- collect: map collect
- detect: find detect
- inject: into: foldLeft injectInto
- printString: on: mkString makeString
- printString: on: addString appendString

Scala

- partition partition
- flatMap flatCollect
- groupBy groupBy
- reduceLeft dropWhile takeWhile

Java

- asArray toArray toArray
- toArray
- toArray
- toArray
- toArray
- asArray
- toBag
- toSortedList

- allSatisfy: forAll allSatisfy
- anySatisfy: exists anySatisfy
- occurrence sOf: count count
- asOrderedCollection toList toList
- first head getFirst
- last last getLast

- partition partition
- take take
- drop drop
- iterator iterator iterator
- min Collections.min() min
- max Collections.max() max

7/30/12 © Copyright 2012 Goldman Sachs. All Rights Reserved
### Humane API - How Rich is Your Iterable?

<table>
<thead>
<tr>
<th><code>RichIterable</code> interface</th>
<th>50+ Unique Methods</th>
<th>70+ Total Methods (w/overloads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allSatisfy</td>
<td>forEachWith</td>
<td>rejectWith</td>
</tr>
<tr>
<td>anySatisfy</td>
<td>forEachWithIndex</td>
<td>select</td>
</tr>
<tr>
<td>appendString</td>
<td>getFirst</td>
<td>selectWith</td>
</tr>
<tr>
<td>asLazy</td>
<td>getLast</td>
<td>size</td>
</tr>
<tr>
<td>chunk</td>
<td>groupBy</td>
<td>toArray</td>
</tr>
<tr>
<td>collect</td>
<td>groupByEach</td>
<td>toBag</td>
</tr>
<tr>
<td>collectIf</td>
<td>injectInto</td>
<td>toList</td>
</tr>
<tr>
<td>collectWith</td>
<td>isEmpty</td>
<td>toMap</td>
</tr>
<tr>
<td>contains</td>
<td>iterator</td>
<td>toSet</td>
</tr>
<tr>
<td>containsAll</td>
<td>makeString</td>
<td>toSortedList</td>
</tr>
<tr>
<td>containsAllArguments</td>
<td>max</td>
<td>toSortedListBy</td>
</tr>
<tr>
<td>containsAllIterable</td>
<td>maxBy</td>
<td>toSortedMap</td>
</tr>
<tr>
<td>count</td>
<td>min</td>
<td>toSortedSet</td>
</tr>
<tr>
<td>detect</td>
<td>minBy</td>
<td>toSortedSetBy</td>
</tr>
<tr>
<td>detectIfNone</td>
<td>notEmpty</td>
<td>toString</td>
</tr>
<tr>
<td>flatCollect</td>
<td>partition</td>
<td>zip</td>
</tr>
<tr>
<td>forEach</td>
<td>reject</td>
<td>zipWithIndex</td>
</tr>
</tbody>
</table>
GS Collections Hierarchy

White: JDK Interfaces
Green: Readable Interfaces
Blue: Mutable Interfaces
Brown: Immutable Interfaces
Purple: FixedSize Interfaces
GS Collections – Multimap Hierarchy
Reducing Static Memory Footprint

• The obvious
  • Don’t build your Set class using a Map
    • Look at GS Collections UnifiedSet for an alternative
  • Don’t build your Map using Entry objects
    • Look at GS Collections UnifiedMap for an alternative
• The not so obvious
  • Don’t extend AbstractList -> save 8 bytes per List for modCount int
    • Look at GS Collections FastList for an alternative
  • Create memory efficient lists, sets and maps for small sized collections
    • Akin to Arrays -> mutable but not growable without a different API
    • These can yield huge memory savings for large static caches
• Empty should be empty
  • Do not create a default sized array of size 10 for empty lists
    • FastList lazy initializes the array on the first add
Improving Performance

- The obvious
  - Optimize basic methods by type
    - FastList optimizes sort, equals, hashCode, removeAll, retainAll, addAll
  - Implement Externalizable for containers types whenever possible
- The not so obvious
  - Optimize iteration methods by type
    - Do not implement serial functional apis using forEach or injectInto
    - All iteration methods on FastList iterate over the internal array directly
  - Optimize eager algorithms based on available information
    - E.g. Collect pre-sizes the target collection
  - Optimize for Empty in methods
    - Methods like addAll can become essentially a garbage-free no-op, for very little extra cost
  - Optimize common parallel methods that use forEach by casting functions to avoid megamorphic calls in forEach
    - We could really use JVM magic here!
Improving Performance Continued

• More not so obvious
  • Implement methods that deal with primitives explicitly to avoid autoboxing
  • Use the Commando Anti-pattern to iterate over a `java.util.ArrayList > size 100`
    • Violate encapsulation carefully in a utility class
  • Make your concrete container classes final
    • Provide abstractions specific for extension
  • Make your blocks static - Don’t generate garbage if you don’t have to
    • Add internal iteration APIs that allow for more static blocks by taking an extra parameter to each function, predicate or procedure

• Occasionally
  • Use lazy iteration to avoid temporary garbage
    • This may have an unexpected and hard to reason about performance impact, but less transient garbage is usually a good thing

• Avoid at all costs
  • Do not use `entrySet` in your code – it assumes that your map eagerly stores Entry objects
    • This can turn a static memory savings into a dynamic memory nightmare
Java SE 8 – Finally we get Lambdas!

OpenJDK Project Lambda

Binary Snapshots available for testing
  – [http://jdk8.java.net/lambda/](http://jdk8.java.net/lambda/)

Public mailing list for info on progress
  – [http://mail.openjdk.java.net/mailman/listinfo/lambda-dev](http://mail.openjdk.java.net/mailman/listinfo/lambda-dev)
Why GS Collections?

• Ready for JDK 8 Lambdas today and works with Java 5 – 7 as well
• Improves readability and reduces duplication of iteration code (enforces DRY/OAOO)
• Implements several, high-level iteration patterns (select, reject, collect, inject into, etc.) on "humane" container interfaces which are extensions of the JDK interfaces
• Provides a consistent mechanism for iterating over Collections, Arrays, Maps, and Strings
• Provides replacements for ArrayList, HashSet, and HashMap optimized for memory usage and performance
• Performs more "behind-the-scene" optimizations in utility classes
• Encapsulates a lot of the structural complexity of parallel iteration and lazy evaluation
• Adds new containers including Bag, Interval, Multimap, and immutable versions of all types
• Has been under active development since 2005 and is a mature library
Conclusions

A Collection Library one uses imprints a particular culture

- “The framework should inspire, encourage the good things and discourage the bad. It should allow you to express what you want to program and not get in the way.”
  (Mr. Noodle, noodlesoft.com and formerly of Sun)

Study history so that you do not repeat old mistakes or make new ones

- Learn from others (Smalltalk, Scala, Haskell, Groovy, Ruby, Python, Clojure)

You can download GS Collections and the GS Collections Kata today

- [https://github.com/goldmansachs/gs-collections](https://github.com/goldmansachs/gs-collections)
- [https://github.com/goldmansachs/gs-collections-kata](https://github.com/goldmansachs/gs-collections-kata)

Focus on behavioral patterns in your code

- Say what you want to do rather than how you do it
- Don’t let yourself and the readers of your code get lost in the jungle of curly braces
Appendix

- Class Hierarchies Compared
  - Smalltalk
  - Java
  - Scala
  - GS Collections
- A loop is never just a loop
- Some Iteration Patterns Explained
  - Select
  - Collect
  - GroupBy
- Java Collections Tutorial
- Anagrams Algorithm
  - Explained
  - Smalltalk solution
  - Scala solution
  - GS Collections solution(s)
1997 - Smalltalk Best Practice Patterns

- do:
- select:
- reject:
- collect:
- detect:
- detect:ifNone:
- inject:into:
- ...
  (Dr. Seuss methods)

2007 - Implementation Patterns

- Map
- List
- Set

- The behavior patterns have disappeared from the language library
- “You get what you get and you don’t get upset”
  (A nursery school rule)
Java Collection Interface Hierarchy

- Enumeration (1.0)
- Iterator (1.2)
- Iterable (1.5)
- RandomAccess (1.4)
- Map (1.2)
- String (1.0)
- Array (1.0)

- Naming Enumeration (1.3)
- ListIterator (1.2)
- Collection (1.2)
- SortedMap (1.2)
- ConcurrentHashMap (1.2)

- List (1.2)
- Set (1.2)
- SortedSet (1.6)
- NavigableSet (1.6)

- Queue (1.5)
- Deque (1.6)
- BlockingDeque (1.6)

- NavigableMap (1.6)
- ConcurrentHashMap (1.6)
Scala 2.8 Collection Hierarchy

- Traversable
  - Iterable
    - Seq
      - Vector
      - LinearSeq
    - Set
      - SortedSet
      - BitSet
    - Map
      - SortedMap
    - Buffer

GS Collections Hierarchy High Level

- Iterable
  - Internal Iterable
  - Rich Iterable
    - Immutable Collection
    - Lazy Iterable
      - Immutable Bag
      - Immutable Bag
      - Immutable Set
      - Immutable List
      - Immutable Set
    - MapIterable
      - Mutable Bag
      - Mutable Bag
      - Immutable Bag
      - Immutable Map
### A Loop Is Never Just A Loop

#### Iteration Pattern

<table>
<thead>
<tr>
<th>Select</th>
<th>Pattern in Idiomatic Java</th>
<th>Pattern in Java 8 w/ GS Collections</th>
</tr>
</thead>
<tbody>
<tr>
<td>detecet</td>
<td>for each <code>&lt;element&gt;</code> of <code>&lt;collection&gt;</code> if condition(&lt;element&gt;) return <code>&lt;element&gt;</code></td>
<td><code>list.detect( each -&gt; each &gt; 50 );</code></td>
</tr>
<tr>
<td>create &lt;newcollection&gt;</td>
<td>for each <code>&lt;element&gt;</code> of <code>&lt;collection&gt;</code> if condition(&lt;element&gt;) add <code>&lt;element&gt;</code> to &lt;newcollection&gt;</td>
<td><code>List result = new ArrayList(); for (Integer each : list) { if (each &gt; 50) result.add(each); }</code></td>
</tr>
<tr>
<td>reject</td>
<td>for each <code>&lt;element&gt;</code> of <code>&lt;collection&gt;</code> if not condition(&lt;element&gt;) add <code>&lt;element&gt;</code> to &lt;newcollection&gt;</td>
<td><code>List result = new ArrayList(); for (Integer each : list) { if (each &lt;= 50) result.add(v); }</code></td>
</tr>
<tr>
<td>any satisfy</td>
<td>for each <code>&lt;element&gt;</code> of <code>&lt;collection&gt;</code> if condition(&lt;element&gt;) return true otherwise return false</td>
<td><code>list.anySatisfy( each -&gt; each &gt; 50 );</code></td>
</tr>
<tr>
<td>all satisfy</td>
<td>for each <code>&lt;element&gt;</code> of <code>&lt;collection&gt;</code> if not condition(&lt;element&gt;) return false otherwise return true</td>
<td><code>list.allSatisfy( each -&gt; each &gt; 50 );</code></td>
</tr>
<tr>
<td>collect</td>
<td>create &lt;newcollection&gt; for each <code>&lt;element&gt;</code> of <code>&lt;collection&gt;</code> <code>&lt;result&gt; = transform(&lt;element&gt;) add &lt;result&gt; to &lt;newcollection&gt;</code></td>
<td><code>list.collect( each -&gt; each.toString() );</code></td>
</tr>
<tr>
<td>inject into</td>
<td>set <code>&lt;result&gt;</code> to <code>&lt;initialvalue&gt;</code> for each <code>&lt;element&gt;</code> of <code>&lt;collection&gt;</code> <code>&lt;result&gt; = apply(&lt;result&gt;, &lt;element&gt;) return </code>&lt;result&gt;`</td>
<td><code>int result = FastList.newListWith(1, 2) .injectInto(3, (x, y) -&gt; x + y );</code></td>
</tr>
</tbody>
</table>
Iteration Pattern: Select

**JDK Select**

```java
List<Integer> results = new ArrayList<>();

for (Integer each : list) {
    if (each > 50) {
        results.add(each);
    }
}
```

**JDK Select Optimized for RandomAccess**

```java
List<Integer> results = new ArrayList<>();

for (int i = 0; i < list.size(); i++) {
    Integer each = list.get(i);
    if (each > 50) {
        results.add(each);
    }
}
```

**GS Collections Select w/ Java 8 Lambdas**

```java
list.select(each -> each > 50);
```

**GS Collections Select w/ Predicates Factory**

```java
list.select(Predicates.greaterThan(50));
```

**GS Collections Select w/ Java 5 - 7**

```java
list.select(new Predicate<Integer>() {
    public boolean accept(Integer each) {
        return each > 50;
    }
});
```
Iterate Pattern: Collect

**JDK Collect**

```java
List<String> names = new ArrayList<>();
for (Person each : people)
{
    names.add(each.getName());
}
```

**JDK Collect Optimized for RandomAccess**

```java
List<String> names = new ArrayList<>();
for (int i = 0; i < people.size(); i++)
{
    Person each = people.get(i);
    names.add(each.getName());
}
```

**GS Collections Collect w/ Java 8 Lambdas**

```java
people.collect(each -> each.getName());
```

**GS Collections Collect w/ Java 5 - 7**

```java
people.collect(new Function<Person, String>()
{
    public String valueOf(Person each)
    {
        return each.getName();
    }
});
```
Map<String, List<Person>> statesToPeople = new HashMap<>();
for (Person person : people) {
    String state = person.getAddress().getState();
    List<Person> peopleInState = statesToPeople.get(state);
    if (peopleInState == null) {
        peopleInState = new ArrayList<>();
        statesToPeople.put(state, peopleInState);
    }
    peopleInState.add(person);
}
Multimap<String, Person> statesToPeople =

people.groupBy(new Function<Person, String>()
{
    public String valueOf(Person each)
    {
        return each.getAddress().getState();
    }
});

Multimap<String, Person> statesToPeople =

people.groupBy(each -> each.getAddress().getState());
Lesson: Algorithms
http://docs.oracle.com/javase/tutorial/collections/algorithms/index.html

Example program for calculating anagrams:
http://docs.oracle.com/javase/tutorial/collections/algorithms/examples/Anagrams2.java
Anagram Algorithm: How It Works

Input

tools loots stool loost otols abra dart trad artd alerts stelra seltra traels lestra aetrls

GroupBy (Multimap)

loost: [tools, loots, stool, loost, otols]
aabr: [abra]
adrt: [dart, trad, artd]
aelrst: [alerts, stelra, seltra, traels, lestra, aetrls]

Select and Sort

[abra]
[alerts, stelra, seltra, traels, lestra, aetrls]
[tools, loots, stool, loost, otols]
[dart, trad, artd]

Output

6: [alerts, stelra, seltra, traels, lestra, aetrls]
5: [tools, loots, stool, loost, otols]
3: [dart, trad, artd]
Anagrams in Smalltalk (Open Source Pharo Smalltalk)

anagrams := words groupedBy: [:each | each asAlphagram].
winners := anagrams select: [:each | each size >= minGroupSize].
sorted := winners asSortedCollection: [:a :b | a size > b size].
sorted do: [:each |
    Transcript cr;
    show: (each size printString, ':', each printString)].

**Note:** I modified the String class by adding the following method

```smalltalk
asAlphagram
    ^self class streamContents:
        [:stream |
            self asSortedCollection do: [:each |
                stream nextPut: each]]
```
words
  .groupBy(_.sorted)
  .values
  .filter(_.length > minGroupSize)
  .toList
  .sortBy( - _.size )
  .elements
  .foreach(winner => println(winner.size + "": " + winner.toList))
Anagrams w/ GS Collections

```
Multimap<Alphagram, String> multimap =
    this.getWords().groupBy(Alphagram::new);

MutableList<RichIterable<String>> results =
    multimap.multiValuesView()
        .select(it -> it.size() >= minSize)
        .toSortedListBy(it -> -it.size());

results.asLazy()
    .collect(it -> it.size() + ": " + it)
    .forEach(Procedures.println(System.out));
```
```java
this.getWords()
  .groupBy(Alphagram::new)
  .multiValuesView()
  .select(it -> it.size() >= minSize)
  .toSortedListBy(it -> -it.size())
  .asLazy()
  .collect(it -> it.size() + ": " + it)
  .forEach(Procedures.println(System.out));
```
private static final class Alphagram {
    private final char[] key;

    private Alphagram(String string) {
        this.key = string.toCharArray();
        Arrays.sort(this.key);
    }

    public boolean equals(Object o) {
        return this == o ||
                Arrays.equals(this.key, ((Alphagram) o).key);
    }

    public int hashCode() {
        return Arrays.hashCode(this.key);
    }
}
• Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.