Goldman Sachs

A Java™ collections framework design

GS Collections™ explained

JVM Language Summit 2012

Version	By	Date
1.0	Don Raab	July 2012

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

Preface

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

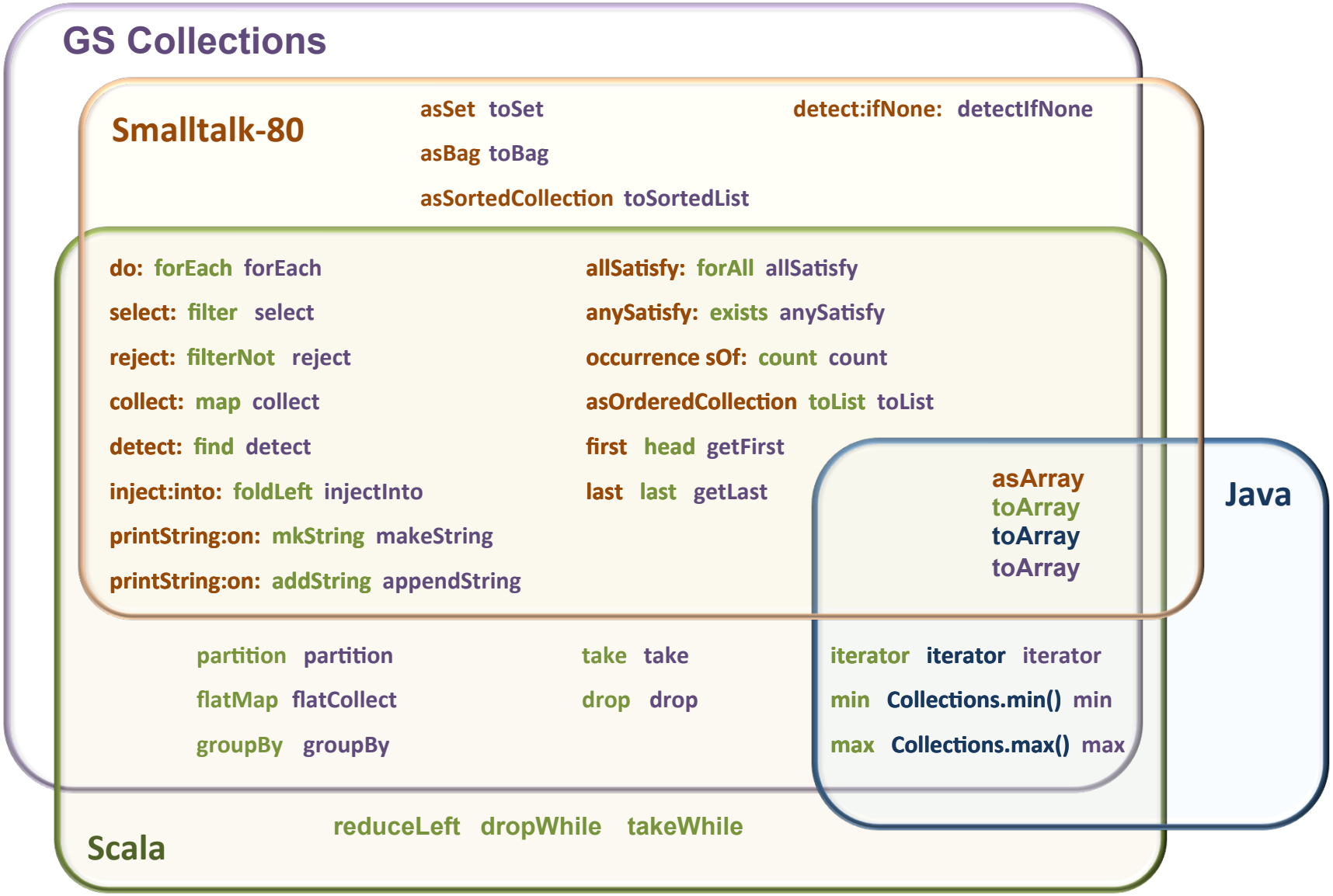
GS Collections Explained

- 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

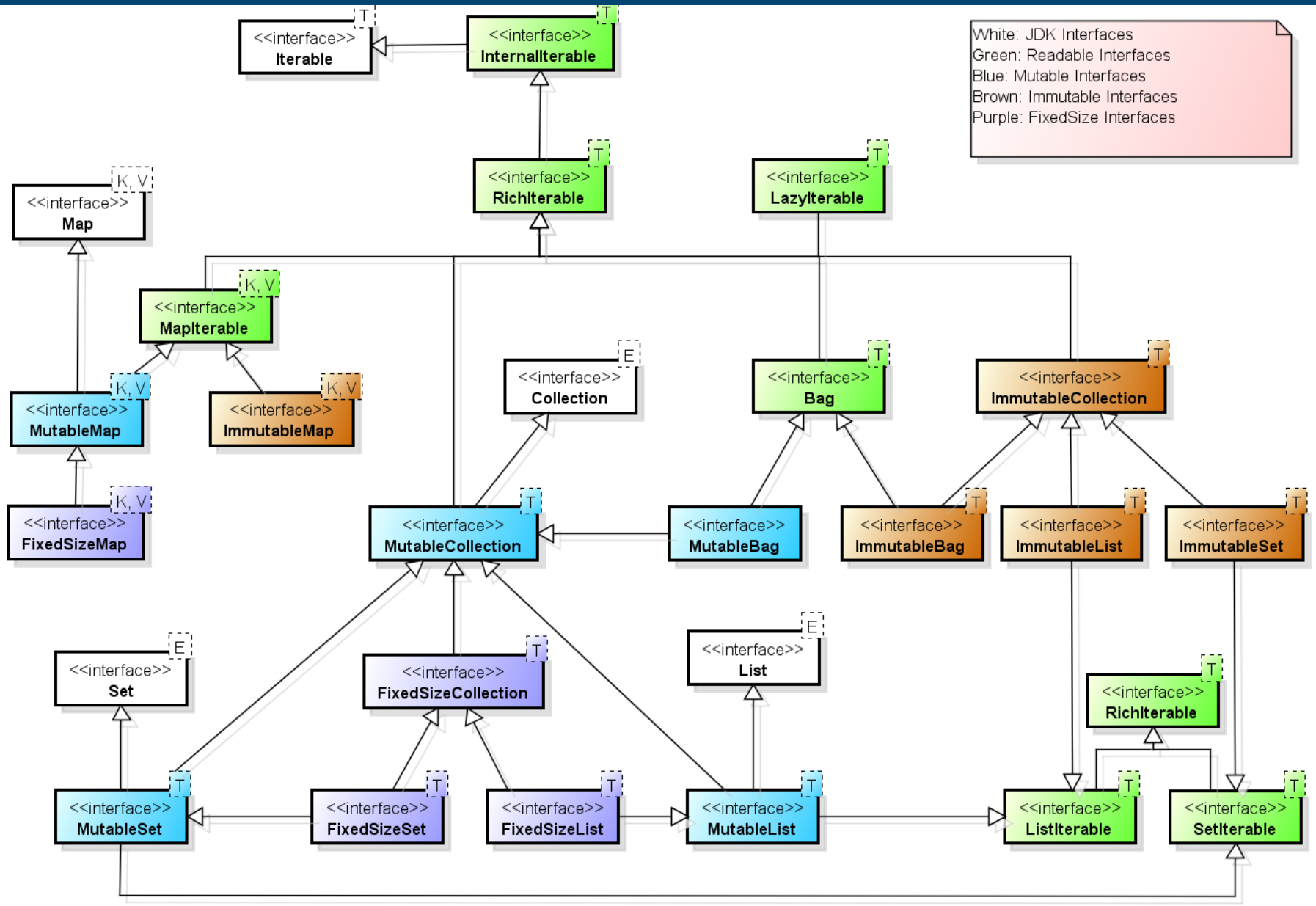


Humane API - How Rich is Your Iterable?

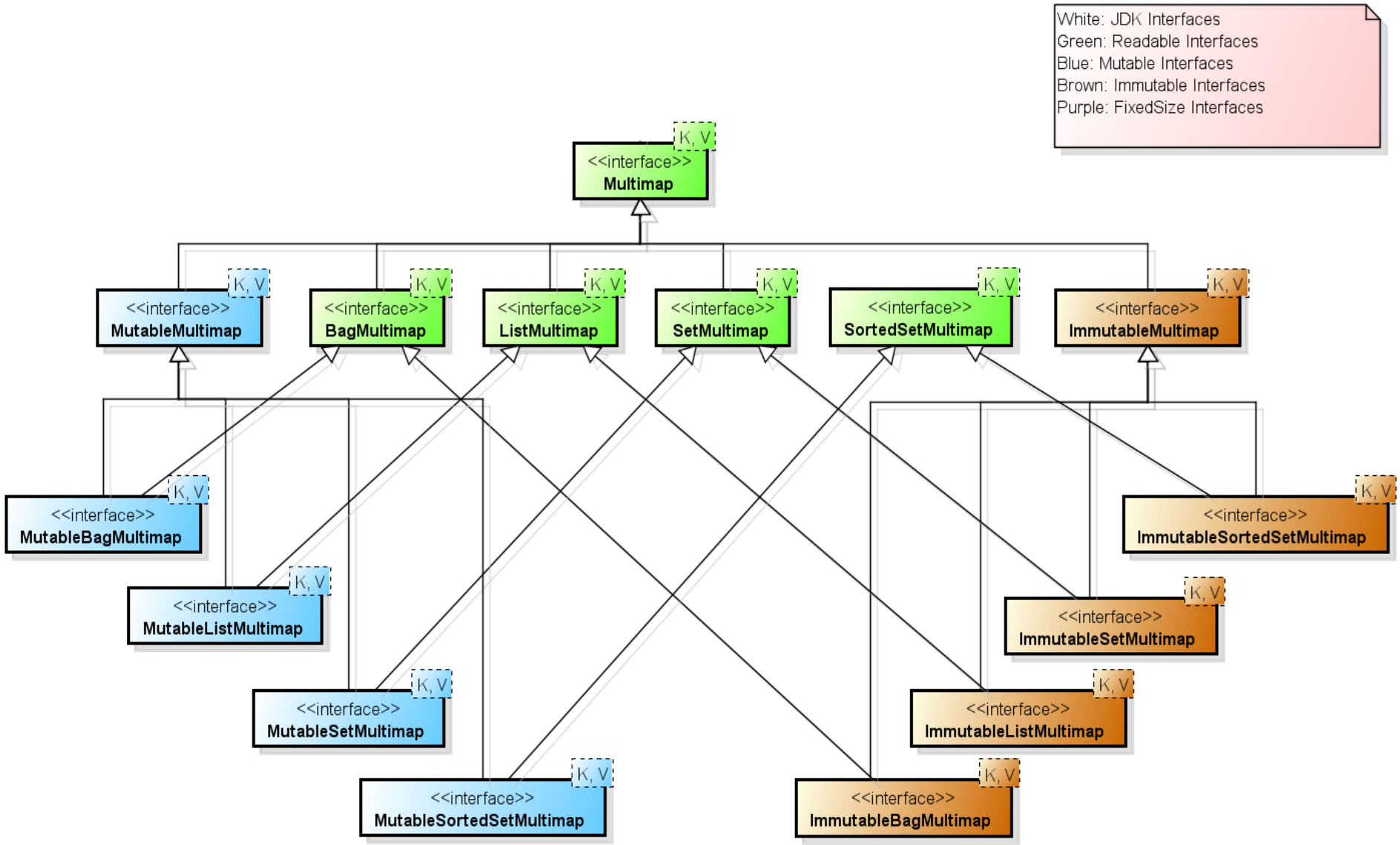
Goldman Sachs

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

GS Collections Hierarchy



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!

Goldman Sachs

OpenJDK Project Lambda

- http://openjdk.java.net/projects/lambda/

Binary Snapshots available for testing

- http://jdk8.java.net/lambda/

Public mailing list for info on progress

- http://mail.openjdk.java.net/mailman/listinfo/lambda-dev

Why GS Collections?

Goldman Sachs

- 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-kata

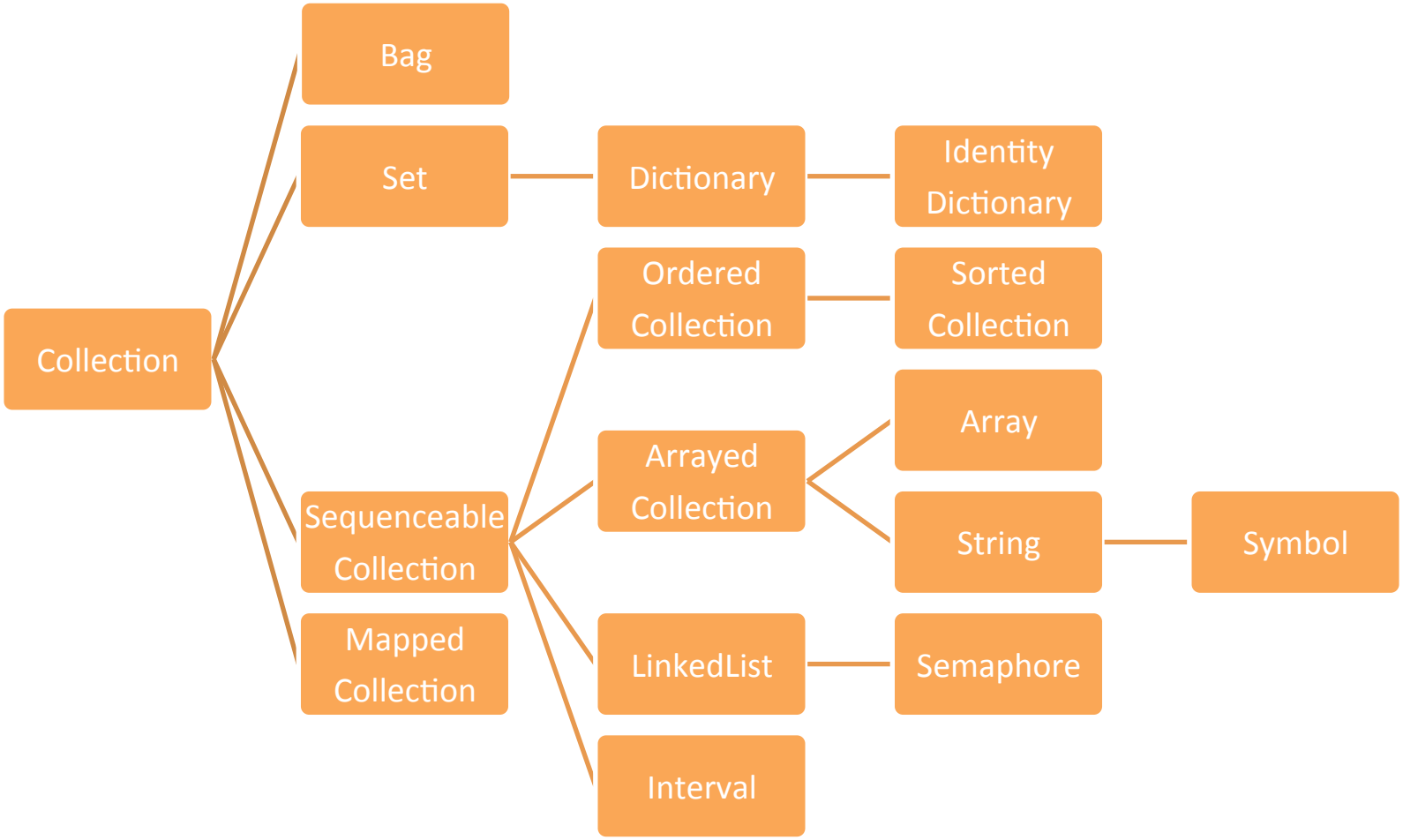
Focus on behavioral patterns in your code

- 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)

Smalltalk-80 Collection Hierarchy



Kent Beck Documents Collection [De-]Evolution

Goldman Sachs

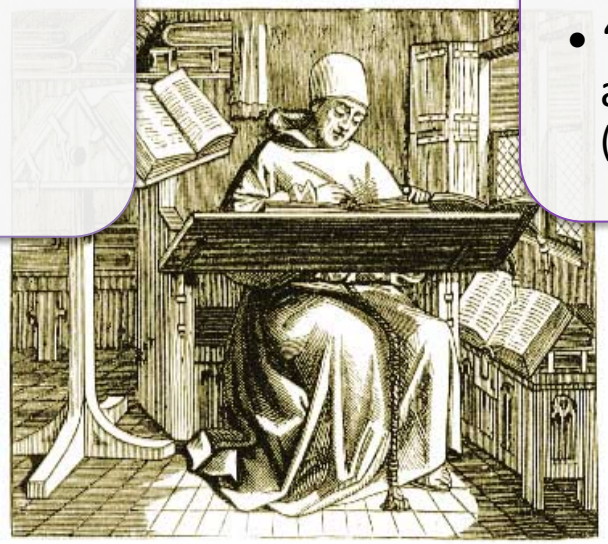
1997 - Smalltalk Best Practice Patterns

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

2007 - Implementation Patterns

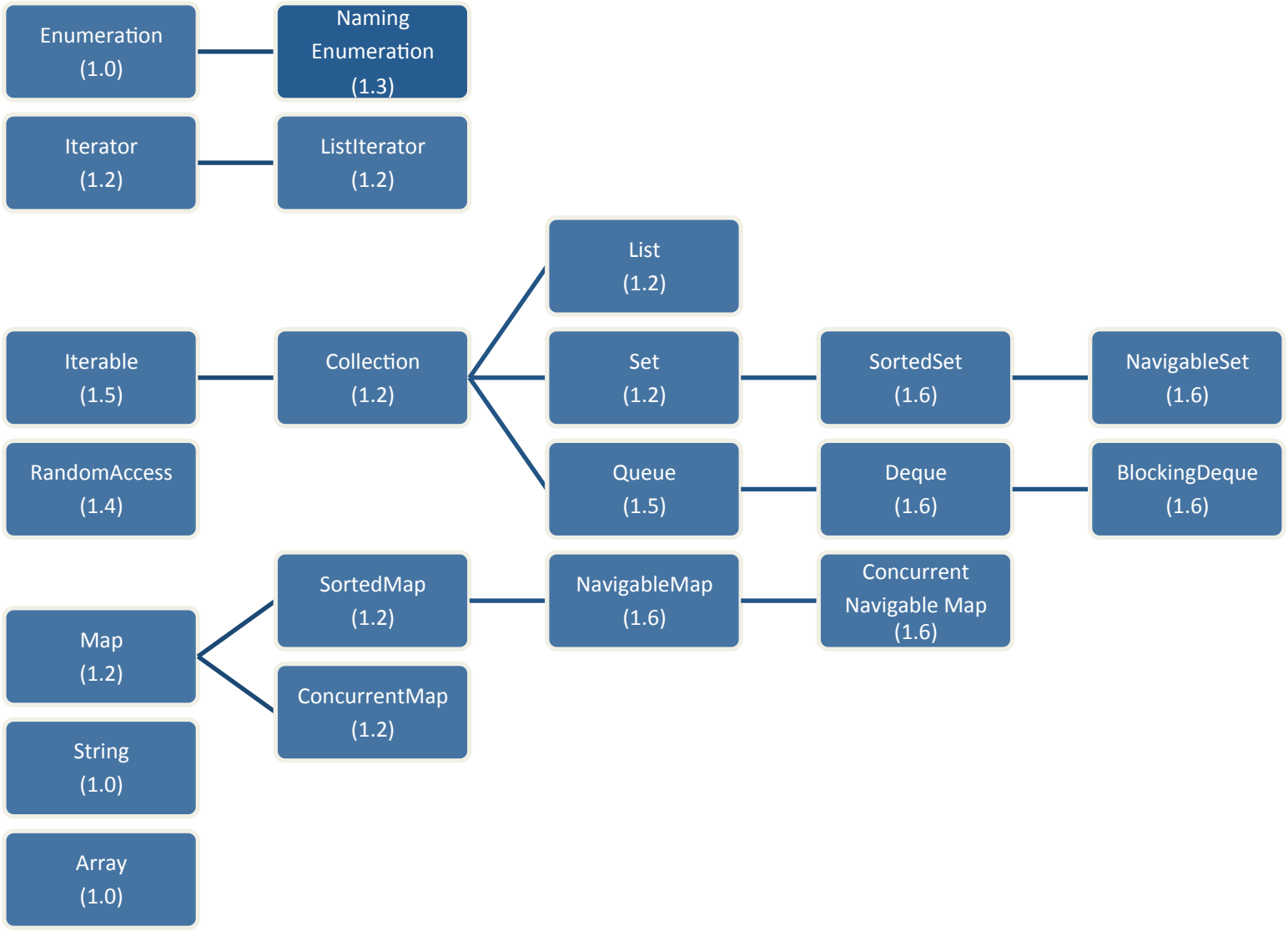
- Map
- List
- Set

- The behavior patterns have disappeared from the language library
- "You get what you get and you don't get upset"
  (A nursery school rule)

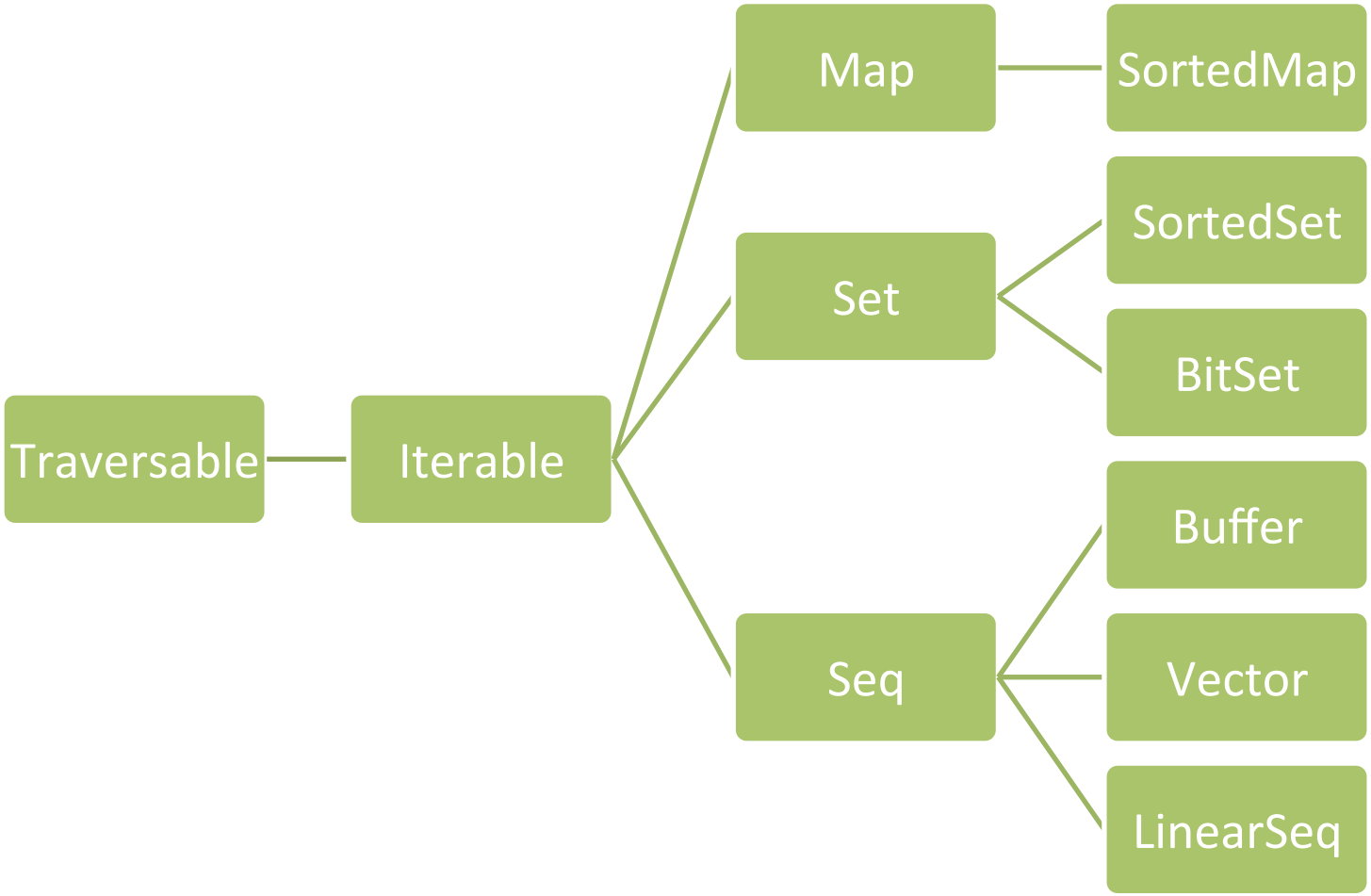


Java Collection Interface ~~Hierarchy~~

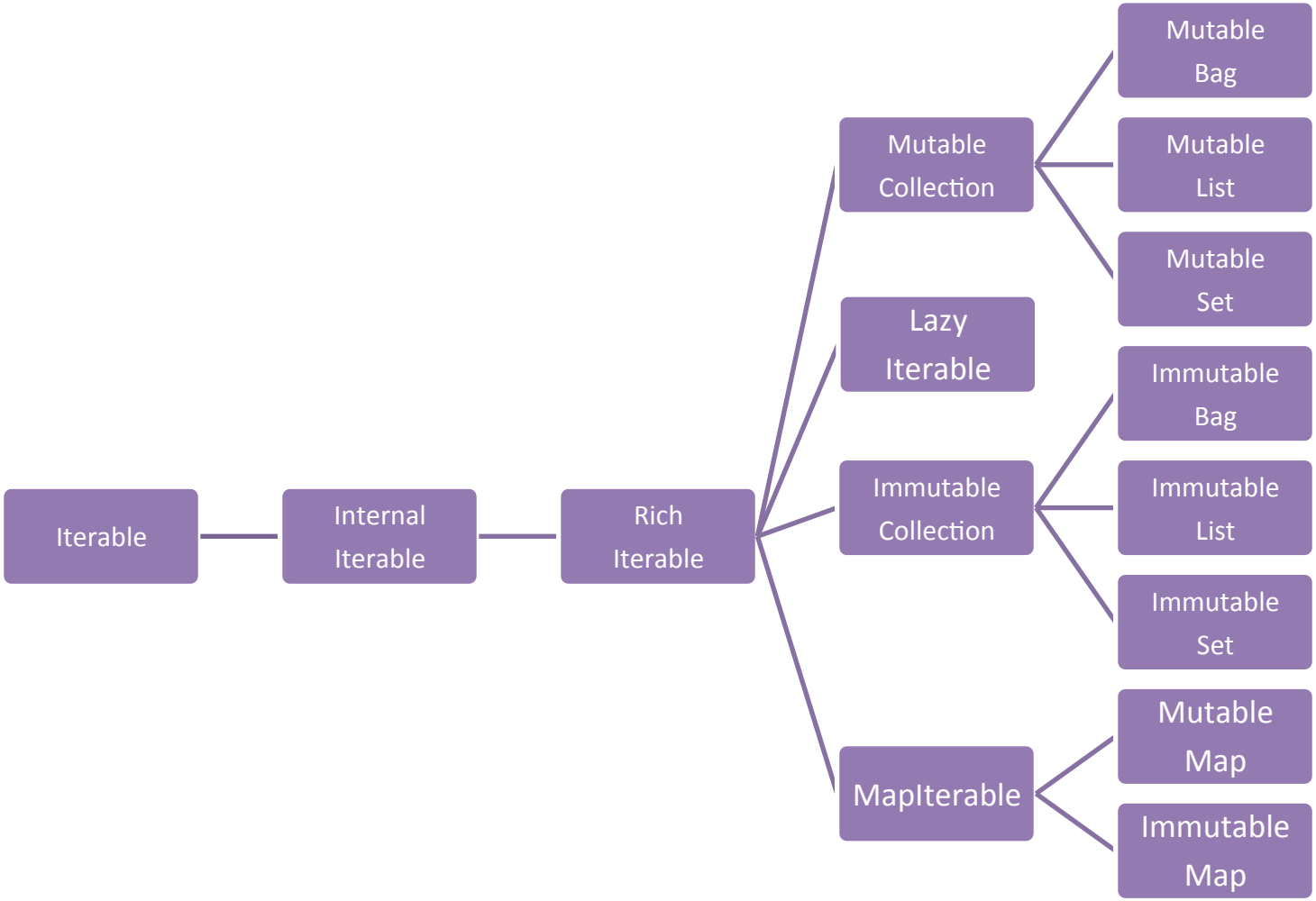
Heterarchy



Scala 2.8 Collection Hierarchy

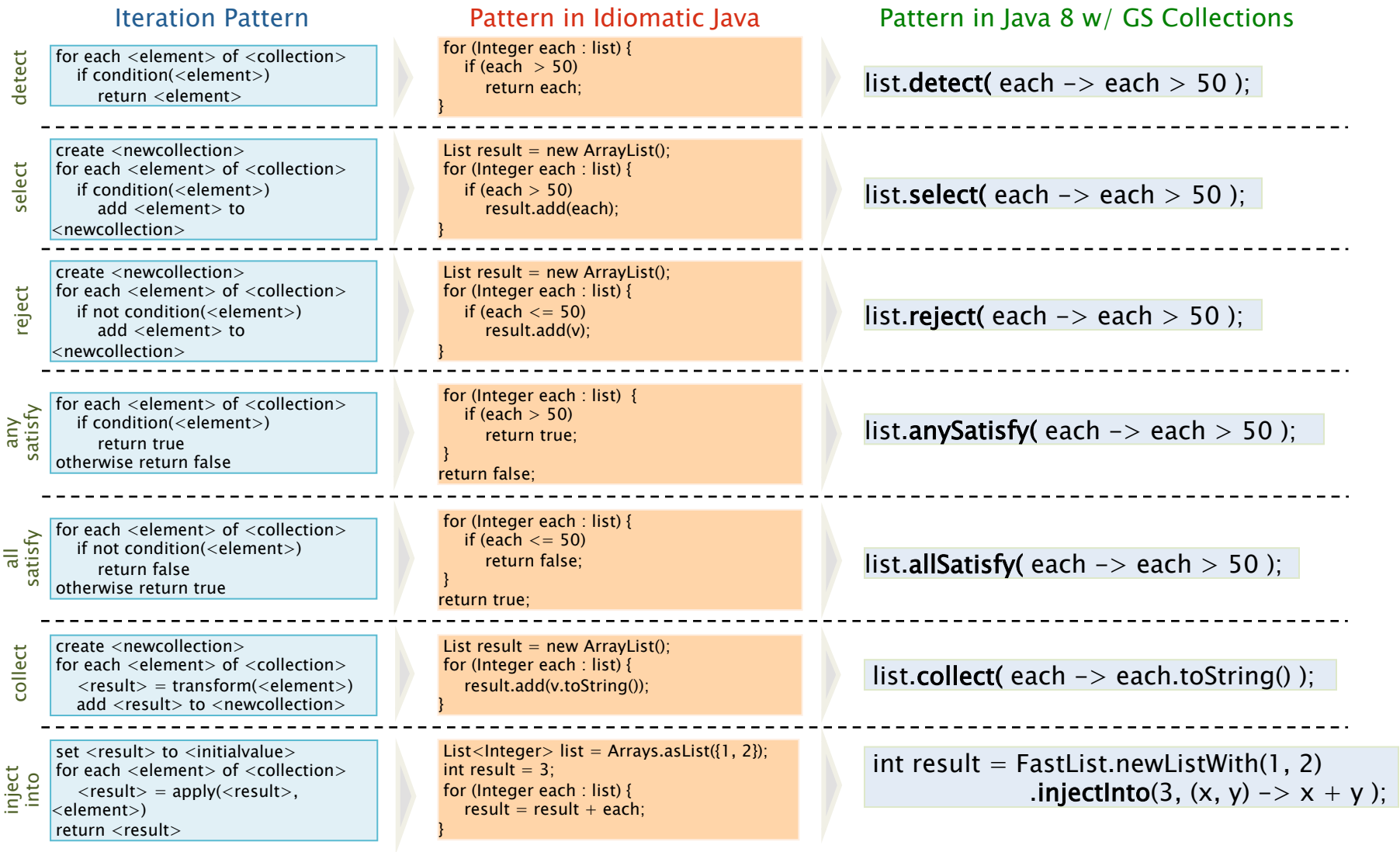


GS Collections Hierarchy High Level



A Loop Is Never Just A Loop

Goldman Sachs



Iteration Pattern: Select

JDK Select

```
List<Integer> results = new ArrayList<>();
for (Integer each : list)
{
    if (each > 50)
    {
        results.add(each);
    }
}
```

JDK Select Optimized for RandomAccess

```
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

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

GS Collections Select w/ Predicates Factory

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

GS Collections Select w/ Java 5 - 7

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

Iteration Pattern: Collect

Goldman Sachs

JDK Collect

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

JDK Collect Optimized for RandomAccess

```
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

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

w/ JDK 8 Method References

```
people.collect(Person::getName);
```

GS Collections Collect w/ Java 5 - 7

```
people.collect(new Function<Person, String>()
{
    public String valueOf(Person each)
    {
        return each.getName();
    }
});
```

Iteration Pattern: GroupBy (Java 7)

```
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);
}
```

Iteration Pattern: GroupBy (GS Collections w/ JDK 5-7 & JDK8)

Goldman Sachs

```
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());
```

The Java Collections Tutorial

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

Goldman Sachs

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

```
asAlphagram
   ^self class streamContents: [:stream |
        self asSortedCollection do: [:each |
              stream nextPut: each]]
```

Anagrams in Scala

```
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));
```

Anagrams w/ GS Collections fully fluent

```
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));
```

Alphagram class used in GS Collections examples

```
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);
    }
}
```

Notes

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