

Eclipse Collections by Example

GS.com/Engineering
Fall, 2015

Christian Glencross

Introductions – EclipseCon Europe 2015



Agenda

- **What is GS Collections?**
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- Hobson's Choice
- Eclipse Collections Examples
- To Use or Reuse?
- More Features

What is GS Collections?

- Open source Java collections framework developed in Goldman Sachs
 - Inspired by Smalltalk Collections Framework
 - In development since **2004**
 - Hosted on GitHub w/ Apache 2.0 License in **January 2012**
 - github.com/goldmansachs/gs-collections
- GS Collections Kata
 - Internal training developed in 2007
 - Taught to > 2,000 GS Java developers
 - Hosted on GitHub w/ Apache 2.0 License in January 2012
 - github.com/goldmansachs/gs-collections-kata

Trending positively

Spring Reactor includes GS Collections as a dependency

RELEASES

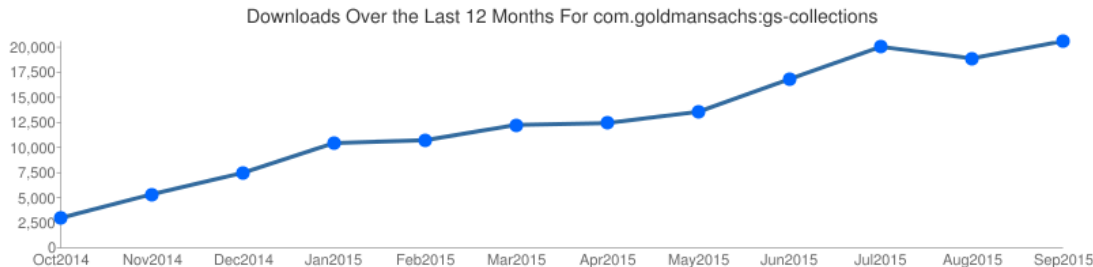
Reactor 1.1.0.RELEASE now available

RELEASES JON BRISBIN MAY 06, 2014

The Reactor team is pleased to announce that some significant updates to the Reactor framework are now available in the 1.1.0.RELEASE version of Reactor's flexible, asynchronous, fast data framework. This version includes numerous bug fixes and rewrites of key components to make them faster and, maybe more importantly, more efficient in terms of memory usage. Reactor 1.1 now includes the fantastic **gs-collections** library from Goldman Sachs [1] which provides a very fluent API for dealing with maps and collections of all kinds.

quote from: <https://spring.io/blog/2014/05/06/reactor-1-1-0-release-now-available>

Monthly Maven central downloads have risen since JavaOne 2014



Awesome Java awesome

A curated list of awesome Java frameworks, libraries and software.

High Performance

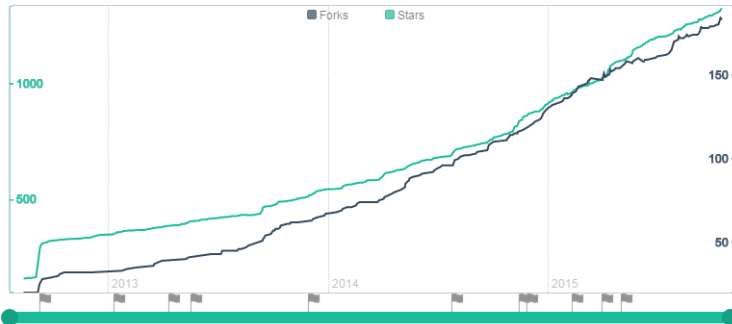
Everything about high performance computation, from collections to specific libraries.

- **Agrona** - Data structures and utility methods that are common in high-performance.
- **Disruptor** - Inter-thread messaging library.
- **fastutil** - Fast and compact type-specific collections.
- **GS Collections** - Collection framework inspired by Smalltalk.
- **HPPC** - Primitive collections.
- **Javolution** - Library for real-time and embedded systems.
- **JCTools** - Concurrency tools currently missing from the JDK.
- **Koloboke** - Hash sets and hash maps.
- **Trove** - Primitive collections.

In the Top 300 Java GitHub projects based on number of stars

goldmansachs/gs-collections

A supplement or replacement for the Java Collections Framework.



★ Star 1,324

🔗 Fork 183

News 12

1. Choosing a Collection Library in Java
Apr 30th 2015  engineering.datarank.com
2. Java performance tips
Mar 30th 2015  java
3. Large HashMap Overview: JDK, FastUtil, Goldman Sachs, HPPC, Koloboke, Trove
Feb 0th 2015  java
4. GS Collections by Example – Part 2
Nov 24th 2014  www.infoq.com
5. GS Collections by Example
Nov 11th 2014  www.infoq.com
6. Time – memory tradeoff with the example of Java Maps

Agenda

- What is GS Collections?
- **What is Eclipse Collections?**
- JCF and EC Memory Comparisons
- Hobson's Choice
- Eclipse Collections Examples
- To Use or Reuse?
- More Features

What is Eclipse Collections?

- Eclipse Collections 7.0
 - Project @ Eclipse Foundation
 - Feature set is the same as GS Collections 7.0
 - Packages renamed
 - com.gs -> org.eclipse
 - Released under dual licenses
 - Eclipse Public License 1.0
 - Eclipse Distribution License 1.0 (BSD)
- Eclipse Collections is open for contributions!
 - Drop by the Goldman Sachs booth in the EclipseCon exhibitor hall to find out more!



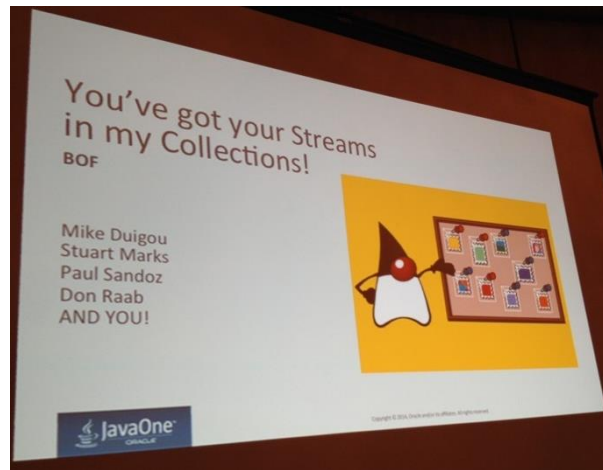
Eclipse Collections Features

Eclipse Collections 7.0

Java 8 Streams

- Functional APIs
- Lazy only
- Single use
- Serial & Parallel
- Primitive streams (3 types)
- Extensible Collectors

- Eager & Lazy, Serial & Parallel
- Memory efficient containers
 - Primitive containers (all 8)
 - Immutable containers
 - More container types
 - More iteration patterns
 - “With” method patterns
 - “target” method patterns
 - Covariant return types
 - Java 5+ compatible



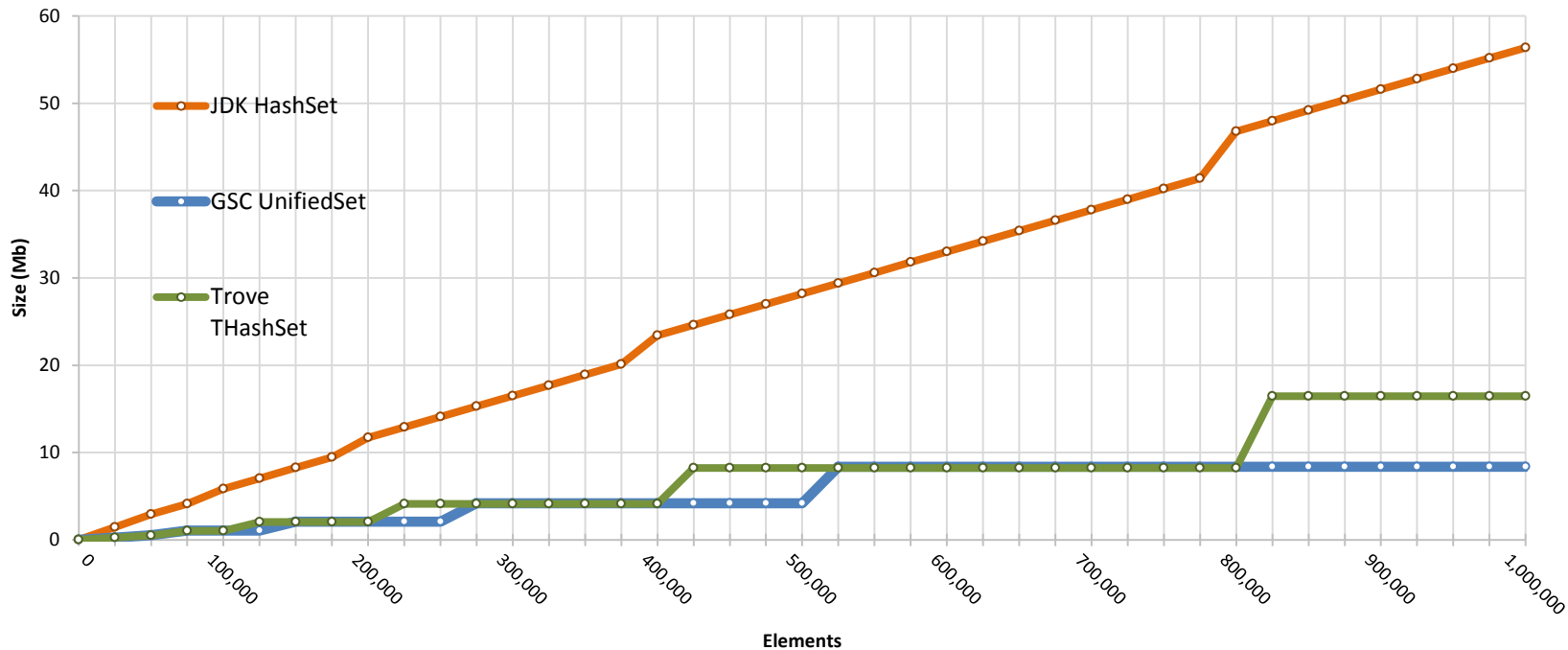
Eclipse Collections has Streams...
And much much more!

And you can contribute too!!!

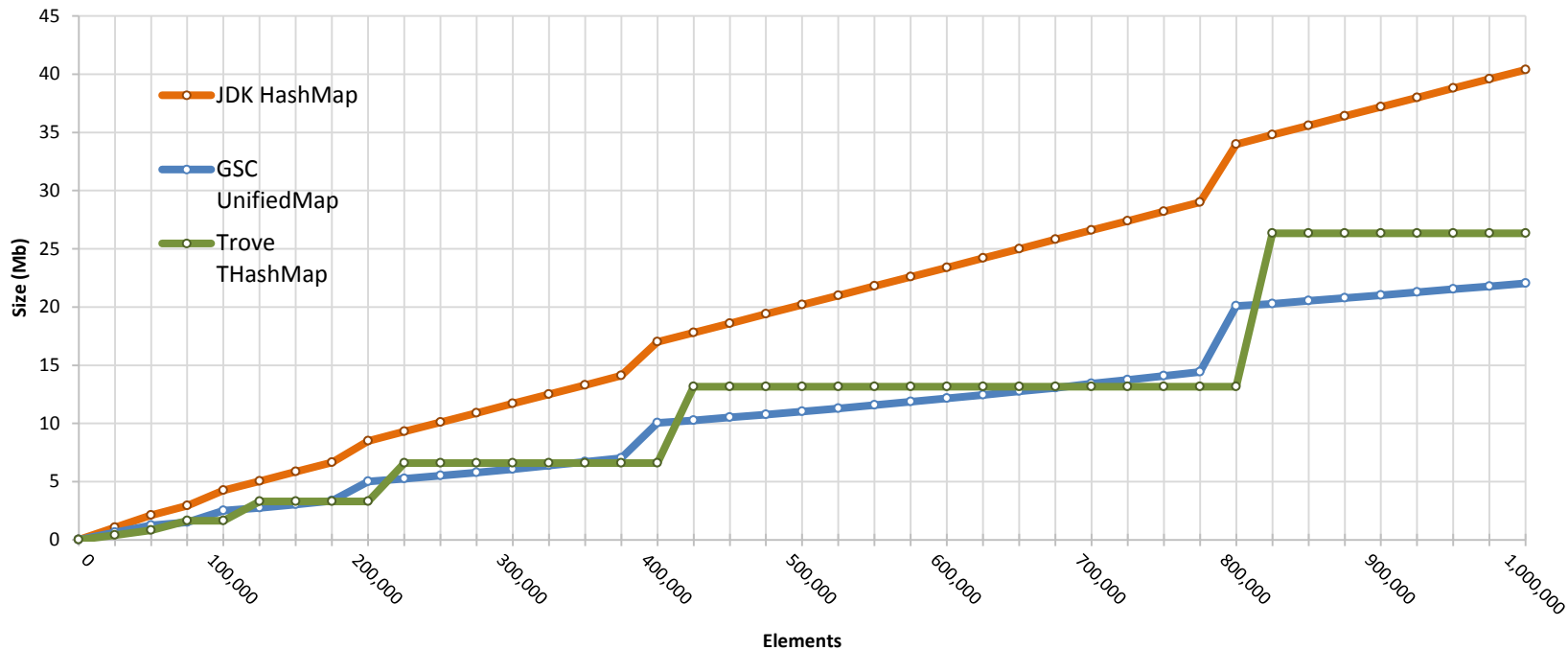
Agenda

- What is GS Collections?
- What is Eclipse Collections?
- **JCF and EC Memory Comparisons**
- Hobson's Choice
- Eclipse Collections Examples
- To Use or Reuse?
- More Features

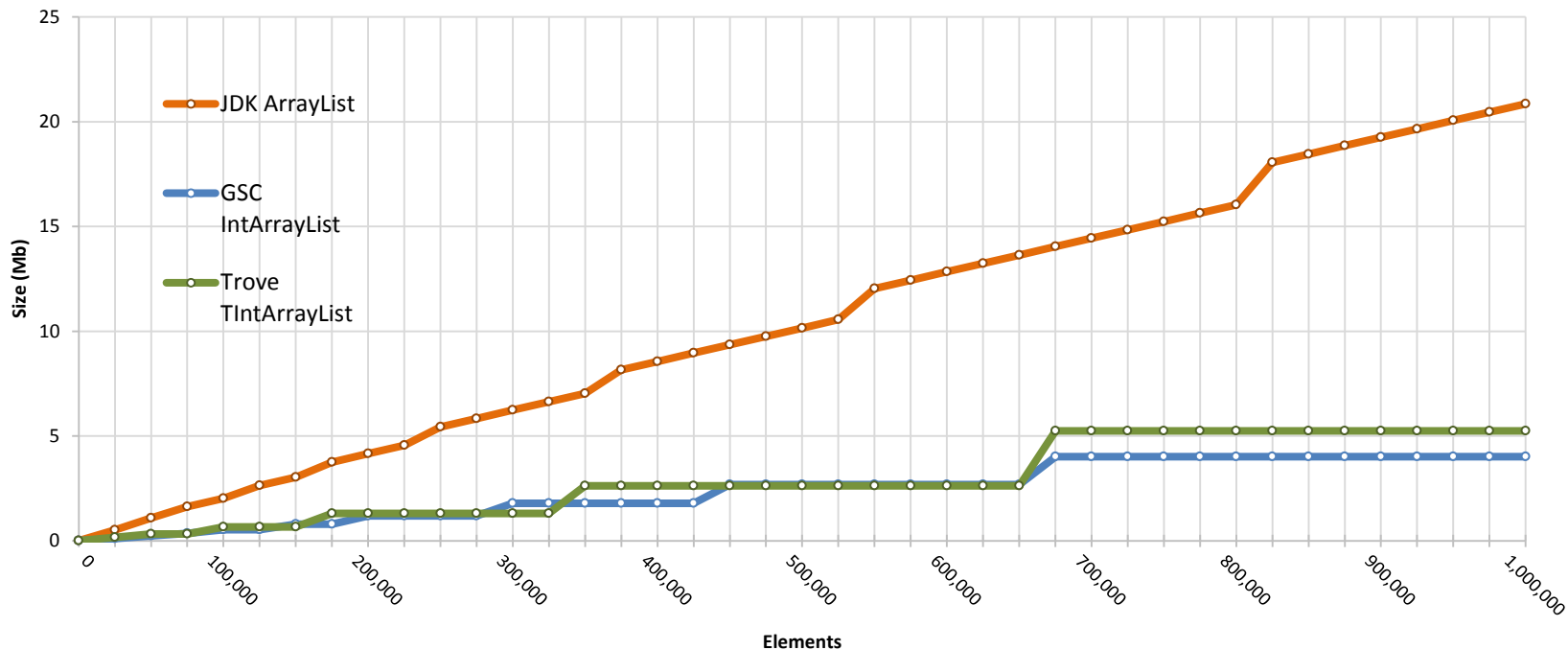
Comparing Sets



Comparing Maps



Why Primitive Collections?



Agenda

- What is GS Collections?
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- **Hobson's Choice**
- Eclipse Collections Examples
- To Use or Reuse?
- More Features

Hobson's Choice

- “A Hobson's choice is a free choice in which only one option is actually offered.”
– Wikipedia

*“You get what you get
and you don't get upset!”*

Hobson's Choice - Iteration Patterns

- Lazy or Eager?
 - Java Collections Framework = Eager

```
Assert.assertEquals(Integer.valueOf(1),  
    Collections.min(Lists.mutable.with(1, 2, 3)));
```

- Streams = Lazy

```
Assert.assertEquals(1,  
    Lists.mutable.with(1, 2, 3).stream().mapToInt(i -> i).min().getAsInt());
```

- Eclipse Collections = You choose

```
Assert.assertEquals(1, IntLists.mutable.with(1, 2, 3).min());  
Assert.assertEquals(1, IntLists.mutable.with(1, 2, 3).asLazy().min());
```

Hobson's Choice - Map

Type	Eclipse Collections	JDK Collections
Bag	Bag<T>	Use Map<T, Integer or Long>
Multimap	Multimap<K, V>	Use Map<K, Collection<V>>
BiMap	BiMap<K, V>	Use two maps.
Partition	PartitionIterable<T>	Use Map<Boolean, Collection<T>>
Pair	Pair<T1, T2>	Use Map.Entry<T1, T2>

When your only tool is a Map, everything is either a key, a value or null.

Hobson's Choice - Primitives

Type	Eclipse Collections	JDK Collections
Primitive List	Yes	Boxed
Primitive Set	Yes	Boxed
Primitive Map	Yes	Boxed
Primitive Stack	Yes	Boxed
Primitive Bag	Yes	Map and Boxed
Primitive Lazy / Stream	Yes (all 8 primitives)	Int, Long, Double only

Hobson's Choice - Parallel

```
Stream<Address> addresses =  
    people.parallelStream()  
        .map(Person::getAddress);
```

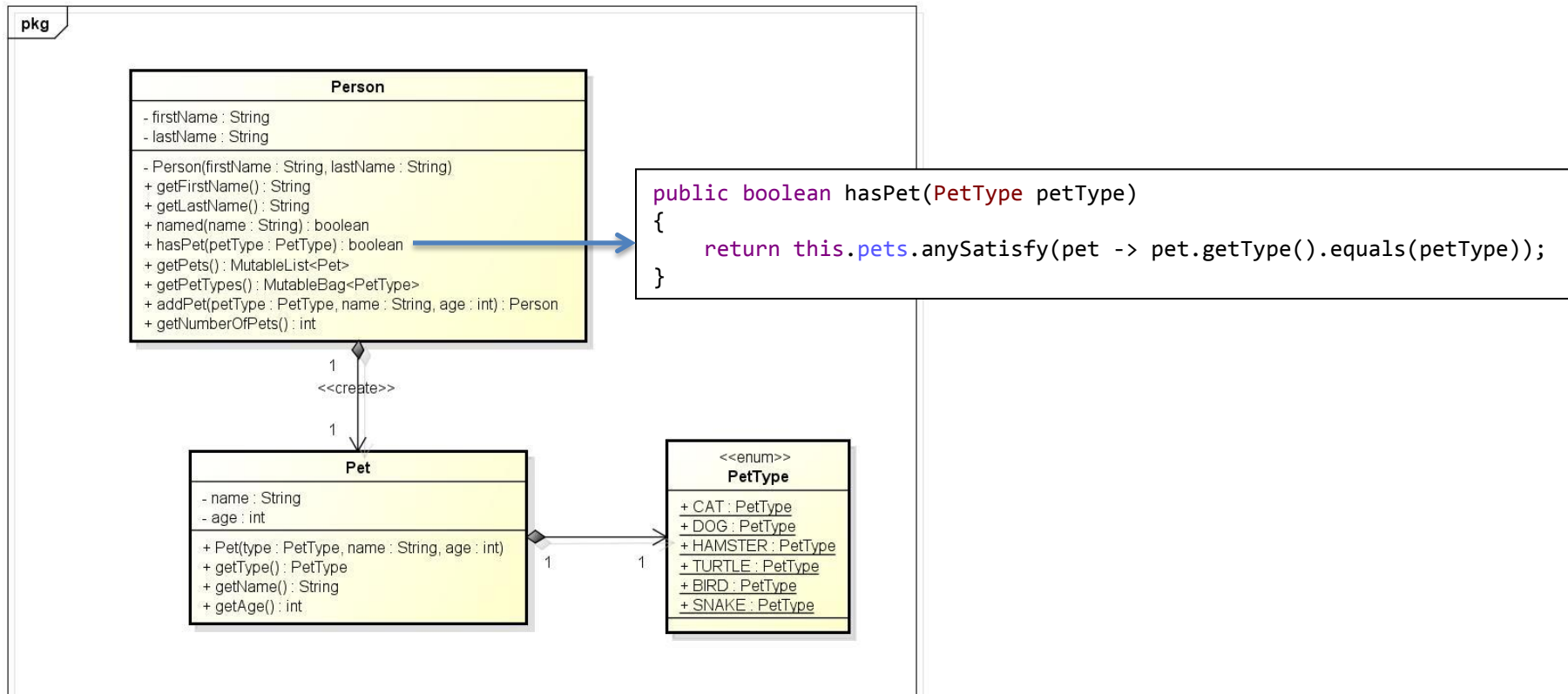
```
ParallelListIterable<Address> addresses =  
    people.asParallel(executor, batchSize)  
        .collect(Person::getAddress);
```

<http://www.infoq.com/presentations/java-streams-scala-parallel-collections>

Agenda

- What is GS Collections?
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- Hobson's Choice
- **Eclipse Collections Examples**
- To Use or Reuse?
- More Features

Eclipse Collections by Example



Setup

```
private MutableList<Person> people;

@Before
public void setUp() throws Exception
{
    this.people = Lists.mutable.with(
        new Person("Mary", "Smith").addPet(PetType.CAT, "Tabby", 2),
        new Person("Bob", "Smith")
            .addPet(PetType.CAT, "Dolly", 3)
            .addPet(PetType.DOG, "Spot", 2),
        new Person("Ted", "Smith").addPet(PetType.DOG, "Spike", 4),
        new Person("Jake", "Snake").addPet(PetType.SNAKE, "Serpy", 1),
        new Person("Barry", "Bird").addPet(PetType.BIRD, "Tweety", 2),
        new Person("Terry", "Turtle").addPet(PetType.TURTLE, "Speedy", 1),
        new Person("Harry", "Hamster")
            .addPet(PetType.HAMSTER, "Fuzzy", 1)
            .addPet(PetType.HAMSTER, "Wuzzy", 1)
    );
}
```

Do any people have cats?

stream

```
boolean result =  
    this.people.stream().anyMatch(person -> person.hasPet(PetType.CAT));
```

eager

```
boolean result =  
    this.people.anySatisfy(person -> person.hasPet(PetType.CAT));
```

```
boolean resultMethodRef =  
    this.people.anySatisfyWith(Person::hasPet, PetType.CAT);
```

asLazy

```
boolean result =  
    this.people.asLazy().anySatisfy(person -> person.hasPet(PetType.CAT));
```

```
boolean resultMethodRef =  
    this.people.asLazy().anySatisfyWith(Person::hasPet, PetType.CAT);
```

How many people have cats?

stream

```
long result =  
    this.people.stream().filter(person -> person.hasPet(PetType.CAT)).count();
```

eager

```
int result =  
    this.people.count(person -> person.hasPet(PetType.CAT));
```

```
int resultMethodRef =  
    this.people.countWith(Person::hasPet, PetType.CAT);
```

asLazy

```
int result =  
    this.people.asLazy().count(person -> person.hasPet(PetType.CAT));
```

```
int resultMethodRef =  
    this.people.asLazy().countWith(Person::hasPet, PetType.CAT);
```

How does “count” stack up?

stream

```

Debug  Servers
Thread [main] (Suspended (breakpoint at line 699 in PersonAndPetKataTest$Person))
  PersonAndPetKataTest$Person.hasPet(PersonAndPetKataTest$PetType) line: 699
  PersonAndPetKataTest.lambda$17(PersonAndPetKataTest$Person) line: 193
  521081105.test(Object) line: not available
  ReferencePipeline$2$1.accept(P_OUT) line: 174
  MutableIterator<T>(Iterator<E>).forEachRemaining(Consumer<? super E>) line: 116
  Spliterators$IteratorSpliterator<T>.forEachRemaining(Consumer<? super T>) line: 1801
  ReferencePipeline$5(AbstractPipeline<E_IN,E_OUT,S>).copyInto(Sink<P_IN>, Spliterator<P_IN>) line: 512
  ReferencePipeline$5(AbstractPipeline<E_IN,E_OUT,S>).wrapAndCopyInto(S, Spliterator<P_IN>) line: 502
  ReduceOps$8(ReduceOps$ReduceOp<T,R,S>).evaluateSequential(PipelineHelper<T>, Spliterator<P_IN>) line: 708
  ReferencePipeline$5(AbstractPipeline<E_IN,E_OUT,S>).evaluate(TerminalOp<E_OUT,R>) line: 234
  ReferencePipeline$5(LongPipeline<E_IN>).reduce(long, LongBinaryOperator) line: 438
  ReferencePipeline$5(LongPipeline<E_IN>).sum() line: 396
  ReferencePipeline$2(ReferencePipeline<P_IN,P_OUT>).count() line: 526
  PersonAndPetKataTest.howManyPeopleHaveCatsUsingStreams() line: 193

```

```
this.people.stream().filter(person -> person.hasPet(PetType.CAT)).count();
```

eager

```

Debug  Servers
Thread [main] (Suspended (breakpoint at line 699 in PersonAndPetKataTest$Person))
  PersonAndPetKataTest$Person.hasPet(PersonAndPetKataTest$PetType) line: 699
  PersonAndPetKataTest.lambda$15(PersonAndPetKataTest$Person, PersonAndPetKataTest$PetType) line: not available
  1597655940.accept(Object, Object) line: not available
  InternalArrayIterate.countWith(T[], int, Predicate2<? super T,? super P>, P) line: 640
  FastList<T>.countWith(Predicate2<? super T,? super P>, P) line: 1169
  PersonAndPetKataTest.howManyPeopleHaveCats() line: 177

```

```
this.people.countWith(Person::hasPet, PetType.CAT);
```

asLazy

```

Debug  Servers
Thread [main] (Suspended (breakpoint at line 699 in PersonAndPetKataTest$Person))
  PersonAndPetKataTest$Person.hasPet(PersonAndPetKataTest$PetType) line: 699
  PersonAndPetKataTest.lambda$16(PersonAndPetKataTest$Person, PersonAndPetKataTest$PetType) line: not available
  497359413.accept(Object, Object) line: not available
  Predicates$BindPredicate2<T,P>.accept(T) line: 1480
  CountProcedure<T>.value(T) line: 45
  FastList<T>.each(Procedure<? super T>) line: 557
  FastList<T>(AbstractRichIterable<T>).forEach(Procedure<? super T>) line: 550
  Iterate.forEach(Iterable<T>, Procedure<? super T>) line: 129
  LazyIterableAdapter<T>.each(Procedure<? super T>) line: 50
  LazyIterableAdapter<T>(AbstractRichIterable<T>).forEach(Procedure<? super T>) line: 550
  LazyIterableAdapter<T>(AbstractRichIterable<T>).count(Predicate<? super T>) line: 447
  LazyIterableAdapter<T>(AbstractRichIterable<T>).countWith(Predicate2<? super T,? super P>, P) line: 453
  PersonAndPetKataTest.howManyPeopleHaveCats() line: 185

```

```
this.people.asLazy().countWith(Person::hasPet, PetType.CAT);
```

Who has cats?

stream

```
List<Person> peopleWithCats =  
    this.people.stream().filter(person -> person.hasPet(PetType.CAT))  
        .collect(Collectors.toList());
```

eager

```
MutableList<Person> peopleWithCats =  
    this.people.select(person -> person.hasPet(PetType.CAT));  
  
MutableList<Person> peopleWithCatsMethodRef =  
    this.people.selectWith(Person::hasPet, PetType.CAT); // select: descriptive API
```

asLazy

```
MutableList<Person> peopleWithCats =  
    this.people.asLazy().select(person -> person.hasPet(PetType.CAT)).toList();  
  
MutableList<Person> peopleWithCatsMethodRef =  
    this.people.asLazy().selectWith(Person::hasPet, PetType.CAT).toList();
```

Who doesn't have cats?

stream

```
List<Person> peopleWithoutCats = // not!  
    this.people.stream().filter(person -> !person.hasPet(PetType.CAT))  
        .collect(Collectors.toList());
```

eager

```
MutableList<Person> peopleWithoutCats =  
    this.people.reject(person -> person.hasPet(PetType.CAT));  
  
MutableList<Person> peopleWithoutCatsMethodRef =  
    this.people.rejectWith(Person::hasPet, PetType.CAT); // detect: descriptive API
```

asLazy

```
MutableList<Person> peopleWithoutCats =  
    this.people.asLazy().reject(person -> person.hasPet(PetType.CAT)).toList();  
  
MutableList<Person> peopleWithoutCatsMethodRef =  
    this.people.asLazy().rejectWith(Person::hasPet, PetType.CAT).toList();
```

Partition people with/without cats

stream

```
Map<Boolean, List<Person>> catsAndNoCats =  
    this.people.stream().collect(  
        Collectors.partitioningBy(person -> person.hasPet(PetType.CAT)));
```

eager

```
PartitionMutableList<Person> catsAndNoCats =  
    this.people.partition(person -> person.hasPet(PetType.CAT));  
  
PartitionMutableList<Person> catsAndNoCatsMethodRef =  
    this.people.partitionWith(Person::hasPet, PetType.CAT);  
  
// PartitionIterable supports getSelected() and getRejected()
```

asLazy

```
PartitionIterable<Person> catsAndNoCats =  
    this.people.asLazy().partition(person -> person.hasPet(PetType.CAT));  
  
PartitionIterable<Person> catsAndNoCatsMethodRef =  
    this.people.asLazy().partitionWith(Person::hasPet, PetType.CAT);
```

Get the names of Bob's pets

stream

```
Person person =
    this.people.stream()
                .filter(each -> each.named("Bob Smith"))
                .findFirst().get();

Assert.assertEquals("Dolly & Spot",
    person.getPets()
        .stream()
        .map(Pet::getName)
        .collect(Collectors.joining(" & ")));
```

eager

```
Person person =
    this.people.detectWith(Person::named, "Bob Smith");

Assert.assertEquals("Dolly & Spot",
    person.getPets()
        .collect(Pet::getName)
        .makeString(" & "));
```

Get the set of all pet types

stream

```
Set<PetType> allPetTypes =  
    this.people.stream()  
        .flatMap(person -> person.getPetTypes().stream())  
        .collect(Collectors.toSet());
```

eager

```
MutableSet<PetType> allPetTypes =  
    this.people.flatCollect(Person::getPetTypes).toSet(); // copies and iterates twice
```

```
MutableSet<PetType> allPetTypesTarget =  
    this.people.flatCollect(Person::getPetTypes, Sets.mutable.empty());  
// Better performance with target collection
```

asLazy

```
MutableSet<PetType> allPetTypes =  
    this.people.asLazy().flatCollect(Person::getPetTypes).toSet();
```

```
MutableSet<PetType> allPetTypesTarget =  
    this.people.asLazy().flatCollect(Person::getPetTypes, Sets.mutable.empty());
```

Group people by their last name

stream

```
Map<String, List<Person>> byLastName =  
    this.people.stream().collect(  
        Collectors.groupingBy(Person::getLastName));  
  
Map<String, MutableBag<Person>> byLastNameTargetBag =  
    this.people.stream().collect(  
        Collectors.groupingBy(Person::getLastName,  
            Collectors.toCollection(Bags.mutable::empty)));  
// Interop with Eclipse Collections Bag
```

eager

```
MutableListMultimap<String, Person> byLastName =  
    this.people.groupBy(Person::getLastName); // Multimap  
  
MutableBagMultimap<String, Person> byLastNameTargetBagMultimap =  
    this.people.groupBy(Person::getLastName, Multimaps.mutable.bag.empty());  
// Native target collection handling
```

Get the age statistics of pets

stream

```
List<Integer> agesList = this.people.stream()
    .flatMap(person -> person.getPets().stream())
    .map(Pet::getAge)
    .collect(Collectors.toList());
IntSummaryStatistics stats = agesList.stream().collect(Collectors.summarizingInt(i -> i));

Assert.assertEquals(stats.getMin(), agesList.stream().mapToInt(i -> i).min().getAsInt());
Assert.assertEquals(stats.getMax(), agesList.stream().mapToInt(i -> i).max().getAsInt());
Assert.assertEquals(stats.getSum(), agesList.stream().mapToInt(i -> i).sum());
```

asLazy

```
IntList agesList = this.people.asLazy() // Primitive collection type
    .flatCollect(Person::getPets)
    .collectInt(Pet::getAge) // collect method available for all 8 primitive types
    .toList();
IntSummaryStatistics stats = new IntSummaryStatistics();
agesList.each(stats::accept);

Assert.assertEquals(stats.getMin(), agesList.min()); // Native support for statistics APIs
Assert.assertEquals(stats.getMax(), agesList.max());
Assert.assertEquals(stats.getSum(), agesList.sum());
```

Counts by pet age

stream

```
Map<Integer, Long> counts = Collections.unmodifiableMap( // Unmodifiable => throws at runtime mutation
    this.people.stream()
        .flatMap(person -> person.getPets().stream())
        .collect(Collectors.groupingBy(Pet::getAge,
            Collectors.counting())));

Assert.assertEquals(Long.valueOf(4), counts.get(1));
Assert.assertEquals(Long.valueOf(3), counts.get(2));
Assert.assertNull(counts.get(5));
Verify.assertThrows(UnsupportedOperationException.class, () -> counts.put(5, 0L));
```

asLazy

```
ImmutableIntBag counts = // Immutable type => no mutation APIs available
    this.people.asLazy()
        .flatCollect(Person::getPets)
        .collectInt(Pet::getAge)
        .toBag() // Bag native support
        .toImmutable();

Assert.assertEquals(4, counts.occurrencesOf(1));
Assert.assertEquals(3, counts.occurrencesOf(2));
Assert.assertEquals(0, counts.occurrencesOf(5)); // Bag returns 0 for non-existing occurrence
```

Agenda

- What is GS Collections?
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- Hobson's Choice
- Eclipse Collections Examples
- **To Use or Reuse?**
- More Features

Use or Reuse?

- Streams – like Iterator

```
Stream<Integer> stream = Lists.mutable.with(1, 2, 3).stream();  
Assert.assertEquals(1, stream.mapToInt(i -> i).min().getAsInt());  
Assert.assertEquals(3, stream.mapToInt(i -> i).max().getAsInt()); // throws
```

```
java.lang.IllegalStateException: stream has already been operated upon or closed  
    at java.util.stream.AbstractPipeline.<init>(AbstractPipeline.java:203)  
    at java.util.stream.IntPipeline.<init>(IntPipeline.java:91)  
    at java.util.stream.IntPipeline$StatelessOp.<init>(IntPipeline.java:592)  
    at java.util.stream.ReferencePipeline$4.<init>(ReferencePipeline.java:204)  
    at java.util.stream.ReferencePipeline.mapToInt(ReferencePipeline.java:203)
```

- LazyIterable – Iterable

```
LazyIterable<Integer> lazy = Lists.mutable.with(1, 2, 3).asLazy();  
Assert.assertEquals(1, lazy.collectInt(i -> i).min());  
Assert.assertEquals(3, lazy.collectInt(i -> i).max());
```

Agenda

- What is GS Collections?
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- Hobson's Choice
- Eclipse Collections Examples
- To Use or Reuse?
- **More Features**

More Features

- “as” methods ($O(1)$ cost) return an adapter to an existing collection
 - asLazy, asUnmodifiable, asSynchronized, asParallel, asReversed
- “to” methods ($O(n)$ cost) return a copy of the collection
 - toImmutable, toList, toBag, toSortedMap, toArray, toReversed, etc.
- Convenient support for method references that throw checked exceptions

```
eclipseList.each(Procedures.throwing(fileWriter::write));
```

Resources

- Eclipse Collections Proposal
<https://projects.eclipse.org/proposals/eclipse-collections>
- GS Collections on GitHub
<https://github.com/goldmansachs/gs-collections>
<https://github.com/goldmansachs/gs-collections/wiki>
<https://github.com/goldmansachs/gs-collections-kata>
- GS Collections Memory Benchmark
http://www.goldmansachs.com/gs-collections/presentations/GSC_Memory_Tests.pdf
- JavaOne 2014 – GS Collections and Java 8 Presentation
http://www.goldmansachs.com/gs-collections/presentations/2014-09-29_JavaOne_GSC.pptx
- Parallel-lazy Performance: Java 8 vs Scala vs GS Collections
<http://www.infoq.com/presentations/java-streams-scala-parallel-collections>

we
BUILD

Learn more at [GS.com/Engineering](https://www.gs.com/engineering)