Eclipse Collections by Example

GS.com/Engineering
Fall, 2015

#j1ec

Hiroshi Ito
Donald Raab

Introductions – JavaOne 2015

Goldman Sachs

Agenda

- **What is GS Collections?**
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- Hobson’s Choice
- Method Reference Preference
- 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

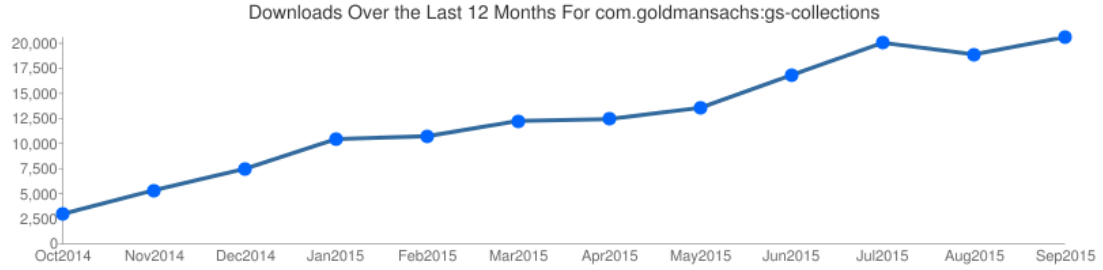
Reactor 1.1.0.RELEASE now available

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

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

High Performance

Everything about high performance computation, from collections to specific librar

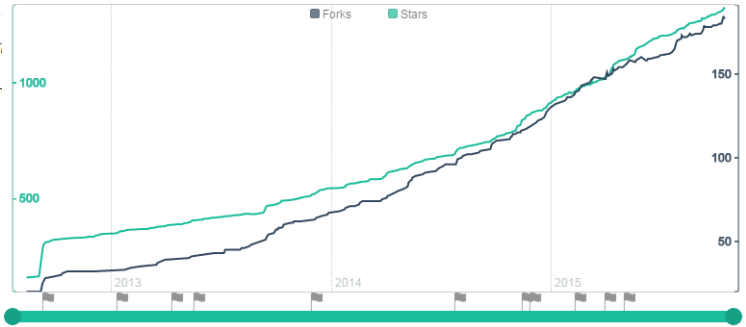
- Agrona - Data structures and utility methods that are common in high-perform
- Disruptor - Inter-thread messaging library.
- fastutil - Fast and compact type-specific collections.
- GS Collections - Collection framework inspired by Smalltalk.
- HPPC - Primitive collections.
- Javolution - Library for 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

Star 1,324 Fork 183

A supplement or replacement for the Java Collections Framework.



News 12

1. Choosing a Collection Library in Java
2. Java performance tips
3. Large HashMap Overview: JDK, FastUtil, Goldman Sachs, HPPC, Koloboke, Trove
4. GS Collections by Example – Part 2
5. GS Collections by Example
6. Time – memory tradeoff with the example of Java Maps

Goldman Sachs

Agenda

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

Goldman Sachs

What is Eclipse Collections?

- Eclipse Collections 7.0
  - Project @ Eclipse Foundation
    - https://projects.eclipse.org/proposals/eclipse-collections
  - Feature set is the same as GS Collections 7.0
  - Packages renamed
    - com.gs -> org.eclipse

- Eclipse Collections - Open for contributions!
  - Drop by the Eclipse Foundation or Goldman Sachs booths in the JavaOne exhibitor hall to find out more!

Goldman Sachs

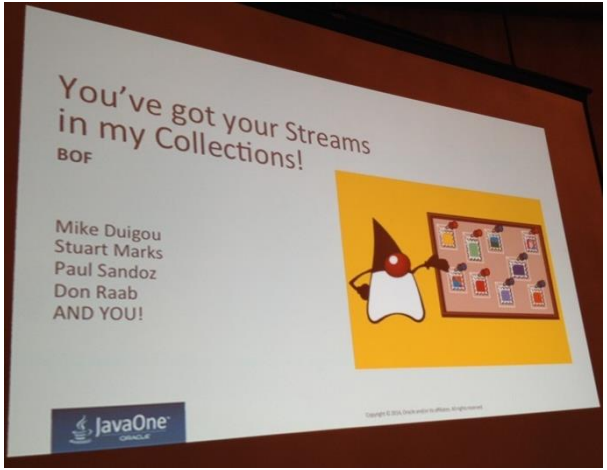
Eclipse Collections Features

Eclipse Collections 7.0

- 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

Java 8 Streams

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



Eclipse Collections has Streams…
And much much more!

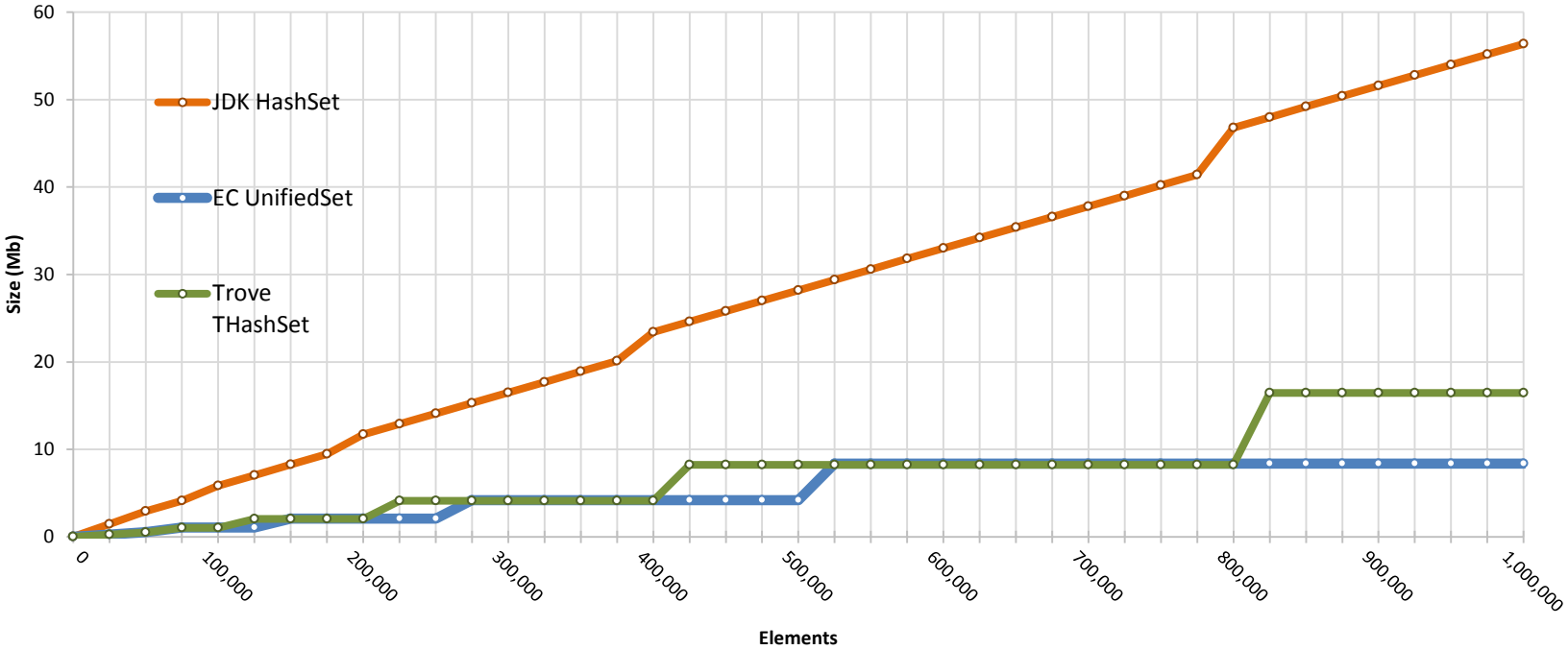
And you can contribute too!!!

Goldman Sachs

Agenda

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

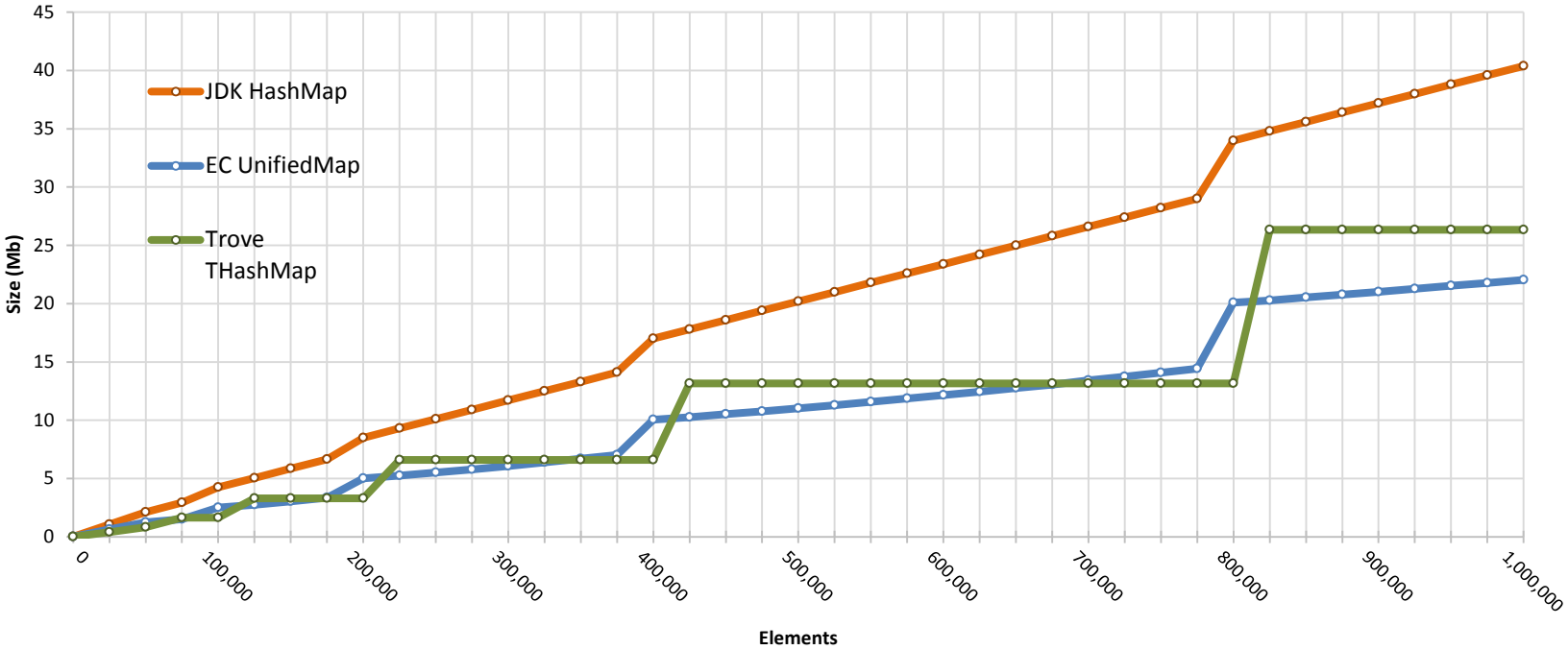
Comparing Sets



Data collected by https://github.com/goldmansachs/gs-collections/tree/master/jmh-tests

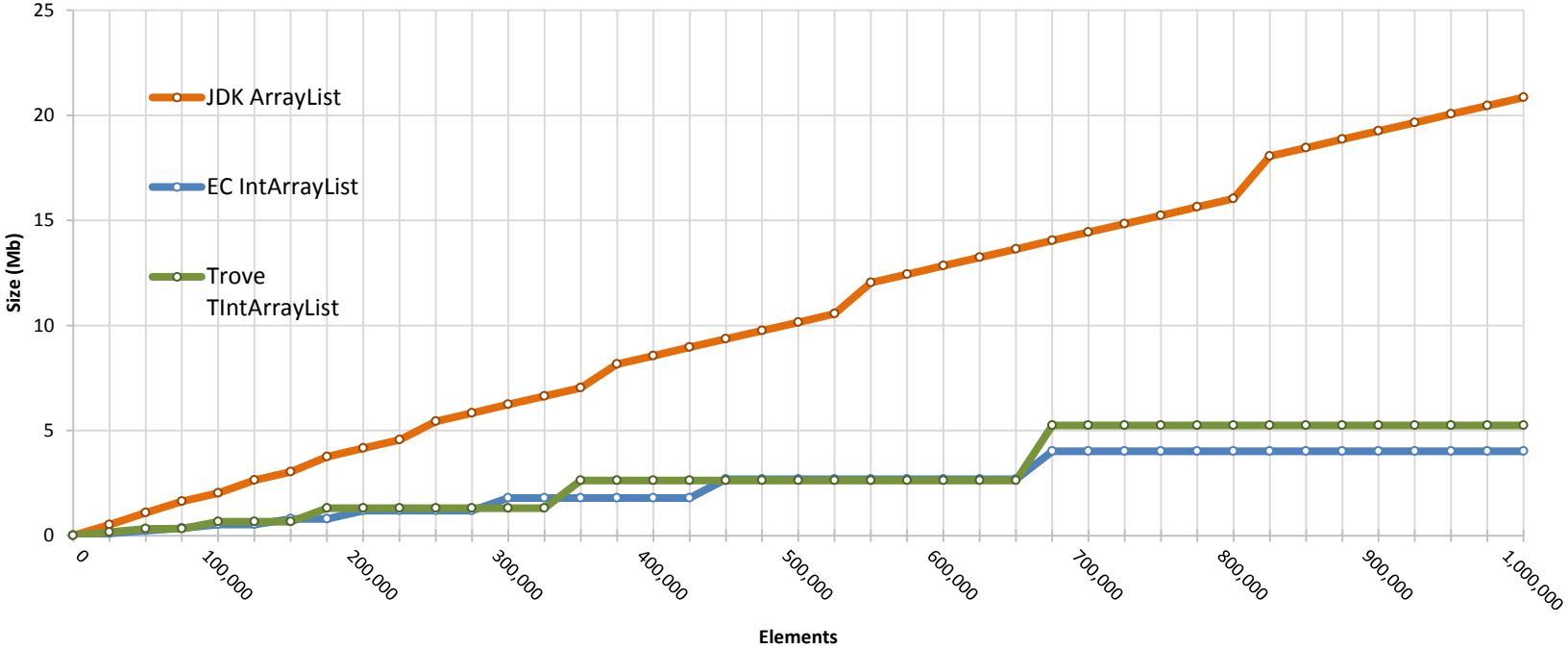
Goldman Sachs

Comparing Maps



Data collected by https://github.com/goldmansachs/gs-collections/tree/master/jmh-tests

Why Primitive Collections?



Data collected by https://github.com/goldmansachs/gs-collections/tree/master/jmh-tests

Goldman Sachs

Agenda

- What is GS Collections?
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- **Hobson's Choice**
- Method Reference Preference
- 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

Goldman Sachs

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

Goldman Sachs

Agenda

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

Method Reference Preference

- I prefer to use method references whenever it is possible
- They are clear, concise and self-documenting

```
Lists.mutable.with("you", "say", "goodbye!")
        .asLazy()
        .collect(each -> each.toUpperCase())
        .select(each -> each.equals("GOODBYE!"))
        .each(each -> System.out.println(each));

Lists.mutable.with("i", "say", "hello!", "hello!")
        .asLazy()
        .collect(String::toUpperCase)
        .select("HELLO!"::equals)
        .each(System.out::println);
```

Outputs:

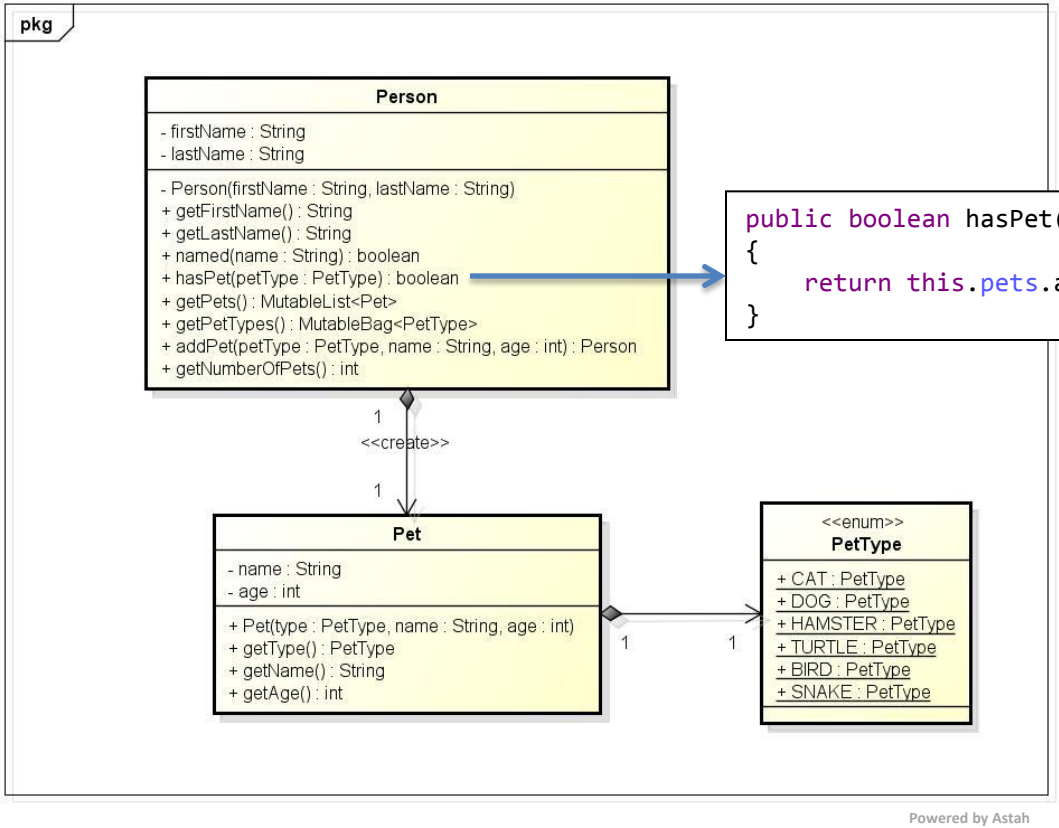
GOODBYE!
HELLO!
HELLO!

Goldman Sachs

Agenda

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

Eclipse Collections by Example




```
public boolean hasPet(PetType petType)
{
    return this.pets.anySatisfy(pet -> pet.getType().equals(petType));
}
```

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)
    );
}
```

Goldman Sachs

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);
```

Goldman Sachs

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);
```

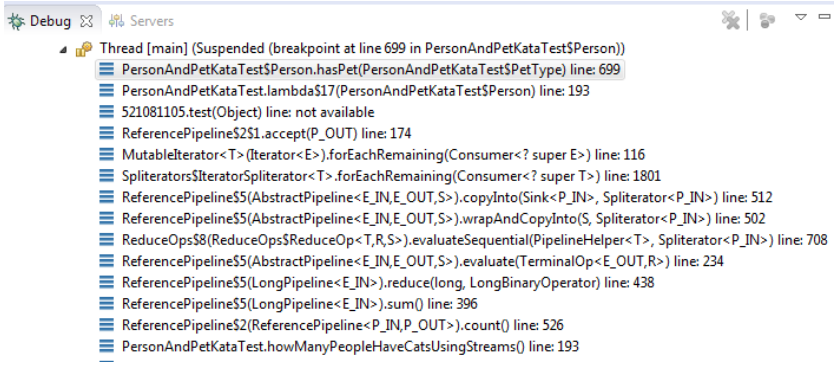
asLazy

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

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

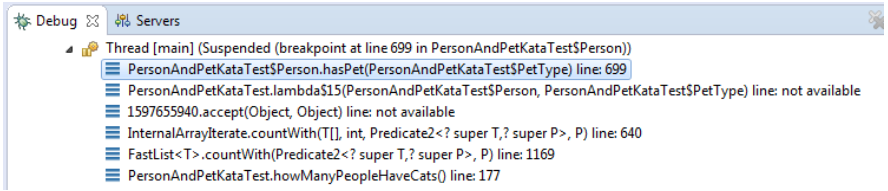
How does “count” stack up?

stream



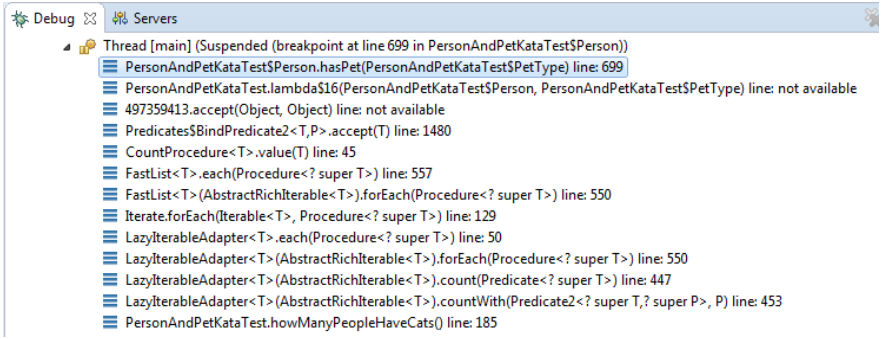

```
this.people.stream().filter(person -> person.hasPet(PetType.CAT)).count();
```

eager




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

asLazy




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

Follow the ~~Rabbit~~ Stream count()

```
public final long count() {
    return mapToLong(e -> 1L).sum();
}
```

```
public final long sum() {
    // use better algorithm to compensate for intermediate overflow?
    return reduce(0, Long::sum);
}
```

```
public final long reduce(long identity, LongBinaryOperator op) {
    return evaluate(ReduceOps.makeLong(identity, op));
}
```

```
final <R> R evaluate(TerminalOp<E_OUT, R> terminalOp) {
    assert getOutputShape() == terminalOp.inputShape();
    if (linkedOrConsumed)
        throw new IllegalStateException(MSG_STREAM_LINKED);
    linkedOrConsumed = true;

    return isParallel()
           ? terminalOp.evaluateParallel(this, sourceSpliterator(terminalOp.getOpFlags()))
           : terminalOp.evaluateSequential(this, sourceSpliterator(terminalOp.getOpFlags()));
}
```

...

```
public final LongStream mapToLong(ToLongFunction<? super P_OUT> mapper) {
    Objects.requireNonNull(mapper);
    return new LongPipeline.StatelessOp<P_OUT>(this,
StreamShape.REFERENCE,
                                               StreamOpFlag.NOT_SORTED |
StreamOpFlag.NOT_DISTINCT) {
        @Override
        Sink<P_OUT> opWrapSink(int flags, Sink<Long> sink) {
            return new Sink.ChainedReference<P_OUT, Long>(sink) {
                @Override
                public void accept(P_OUT u) {
                    downstream.accept(mapper.applyAsLong(u));
                }
            };
        }
    };
}
```

countWith() Eclipse Collections

```
public <P> int countWith(Predicate2<? super T, ? super P> predicate, P parameter)
{
    return InternalArrayIterate.countWith(this.items, this.size, predicate, parameter);
}
```

```
public static <T, P> int countWith(T[] array, int size, Predicate2<? super T, ? super P> predicate, P parameter)
{
    int count = 0;
    for (int i = 0; i < size; i++)
    {
        if (predicate.accept(array[i], parameter))
        {
            count++;
        }
    }
    return count;
}
```

Goldman Sachs

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();
```

Goldman Sachs

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();
```

Goldman Sachs

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
```

Goldman Sachs

Agenda

- What is GS Collections?
- What is Eclipse Collections?
- JCF and EC Memory Comparisons
- Hobson's Choice
- Method Reference Preference
- 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());
```

Goldman Sachs

Agenda

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

The “as” methods (O(1) cost)

Method	Visible on Type	Returns
asLazy	RichIterable, {**primitive**}Iterable	LazyIterable, Lazy{**primitive**}Iterable
asUnmodifiable	Mutable{**Collection**}, Mutable{**primitive**}{**Collection**}	Mutable{**Collection**}, Mutable{**primitive**}{**Collection**}
asSynchronized	Mutable{**Collection**} , Mutable{**primitive**}{**Collection**}	Mutable{**Collection**}, Mutable{**primitive**}{**Collection**}
asParallel	ListIterable, SetIterable	ParallelListIterable, ParallelSetIterable
asReversed	ReversibleIterable, Reversible{**primitive**}Iterable	LazyIterable, Lazy{**primitive**}Iterable

{**Collection**} = Collection, List, Set, Bag, Map
{**primitive**} = boolean, byte, char, short, int, float, long, double

The “to” methods (O(n) cost)

Method	Available on Type	Returns
toImmutable	{**Collection**}Iterable, Mutable{**primitive**}{**Collection**}	Immutable{**Collection**}, Immutable{**primitive**}{**Collection**}
toList, toSortedList (By)	RichIterable, {**primitive**}Iterable	MutableList, Mutable{**primitive**}List
toSet	RichIterable, {**primitive**}Iterable	MutableSet, Mutable{**primitive**}Set
toSortedSet (By)	RichIterable	MutableSortedSet
toBag	RichIterable, {**primitive**}Iterable	MutableBag, Mutable{**primitive**}Bag
toSortedBag (By)	RichIterable	MutableSortedBag
toMap	RichIterable	MutableMap
toSortedMap	RichIterable	MutableSortedMap
toArray	RichIterable, {**primitive**}Iterable	Object[], {**primitive**}[]
toReversed	ReversibleIterable, Reversible{**primitive**}Iterable	ReversibleIterable, Reversible{**primitive**}Iterable

{**Collection**} = Collection, List, Set, Bag, Map

{**primitive**} = boolean, byte, char, short, int, float, long, double

Handling Exceptions

Java Collections

```
Appendable appendable = new StringBuilder();
List<String> jdkList = Arrays.asList("one");
   // jdkList.forEach(appendable::append);
   // Compile Error: incompatible thrown types java.io.IOException in method reference
jdkList.forEach(each -> {
    try
    {
        appendable.append(each);
    }
    catch (IOException e)
    {
        throw new RuntimeException(e);
    }
});
Assert.assertEquals("one", appendable.toString());
```

Eclipse Collections

```
ImmutableList<String> eclipseList = Lists.immutable.with("two");
eclipseList.each(Procedures.throwing(appendable::append));
Assert.assertEquals("onetwo", appendable.toString());
```

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

Goldman Sachs

we

BUILD

Learn more at GS.com/Engineering