The Java Evolution of Eclipse Collections

GS.com/Engineering
QCon NY 2017

Kristen O'Leary

Goldman Sachs

Agenda

- **Intro**
- What is Eclipse Collections
- New features in 8.0
- Optional
- Collectors
- Default methods
- Primitive Collections
- Java 9
- Conclusion

Goldman Sachs

Kristen O'Leary

- Software developer at Goldman Sachs
- Contributor to GS/Eclipse Collections
  - http://www.goldmansachs.com/what-we-do/engineering/see-our-work/growing-with-gs-collections.html
- EC Kata Instructor and Contributor
  - https://www.eclipsecon.org/na2016/session/eclipse-collections-kata-fun-way-learn-feature-rich-collections-framework
- Eclipse Collections Advocate
  - NY Java SIG 2016, JavaOne 2016, EclipseCon 2016, QCon NY 2014

Goldman Sachs

Agenda

- Intro
- **What is Eclipse Collections**
- New features in 8.0
- Optional
- Collectors
- Default methods
- Primitive Collections
- Java 9
- Conclusion

Goldman Sachs

What is Eclipse Collections?

- Feature rich, memory efficient Java Collections framework
- Open sourced in 2012 as GS Collections
- New – 8.0 compatible with JDK 8+
- eclipse.org/collections
- Eclipse Collections is open for contributions!
  - https://github.com/eclipse/eclipse-collections/blob/master/CONTRIBUTING.md

Goldman Sachs

Agenda

- Intro
- What is Eclipse Collections
- **New features in 8.0**
- Optional
- Collectors
- Default methods
- Primitive Collections
- Java 9
- Conclusion

Goldman Sachs

Version 8 for Java 8

- Before, EC worked with Java 8
- But did not require or embrace it
- V 8 changes that. Now:
  - If you have JDK 8 code → easy transition to EC
  - If you have EC code → easier to use JDK 8 Collections/Streams features

Goldman Sachs

What's new in EC 8.0?

- EC **functional interfaces** extend **Java 8 functional interfaces**
- New interop with Java 8 features directly in API
  - reduce, reduceInPlace, summarize, detectOptional and more
- Collectors + Eclipse Collections Types = Collectors2
- Default methods = More Features, Less Code
- Primitive Collections and Java 8

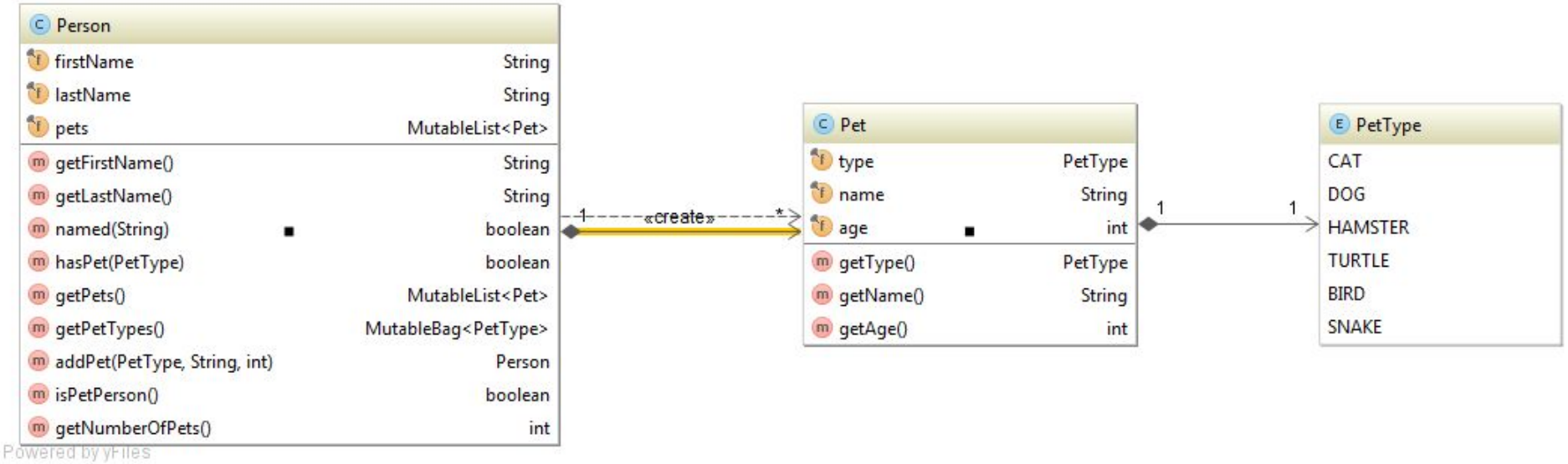
Goldman Sachs

Agenda

- Intro
- What is Eclipse Collections
- New features in 8.0
- **Optional**
- Collectors
- Default methods
- Primitive Collections
- Java 9
- Conclusion

Goldman Sachs

Our Domain



Goldman Sachs

Optional

- New JDK 8 feature
- From the javadoc:
  - "*A container object which may or may not contain a non-null value. If a value is present, isPresent() will return true and get() will return the value*."

Goldman Sachs

Where can we use this in EC?

- RichIterable “detectWith” is one example
- From the javadoc:
  - *“Returns the first element that evaluates to true for the specified predicate2 and parameter, or null if none evaluate to true.”*

Goldman Sachs

So what happens?

Kata exercise - find Mary Smith

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

Assert.assertEquals("Mary", person.getFirstName());
Assert.assertEquals("Smith", person.getLastName());
```

java.lang.NullPointerException

Goldman Sachs

Pre 8.0 – Null check

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

if(person == null)
{
    person = new Person("Mary", "Smith");
}

Assert.assertEquals("Mary", person.getFirstName());
Assert.assertEquals("Smith", person.getLastName());
```

Goldman Sachs

Pre 8.0 – detectWithIfNone

Find Mary Smith, or create her

```
Person person = this.people.detectWithIfNone(
    Person::named, "Mary Smith",
    () -> new Person("Mary", "Smith"));


Assert.assertEquals("Mary", person.getFirstName());
Assert.assertEquals("Smith", person.getLastName());
```

Test passes!

8.0 - Optional

Find Mary Smith, or create her

```
Optional<Person> optional = this.people
        .detectWithOptional(Person::named, "Mary Smith");

Person person = optional.orElseGet(
        () -> new Person("Mary", "Smith"));

Assert.assertEquals("Mary", person.getFirstName());
Assert.assertEquals("Smith", person.getLastName());
```

Test passes!

Goldman Sachs

Agenda

- Intro
- What is Eclipse Collections
- New features in 8.0
- Optional
- **Collectors**
- Default methods
- Primitive Collections
- Java 9
- Conclusion

Goldman Sachs

What is a Collector?

- New feature in Java 8
- A way to implement a mutable reduction operation
  - Accumulating into a collection
- JDK 8 has several “built in” Collectors

Goldman Sachs

JDK 8 Collector Examples

```
List<String> names = this.people.stream()
    .map(Person::getFirstName).collect(Collectors.toList());
```

```
// Output:
[Bob, Ted, Jake]
```

```
int total = this.people.stream()
     .collect(Collectors.summingInt(Person::getNumberOfPets));
```

```
// Output:
4
```

Goldman Sachs

Eclipse Collections and Collectors

- Starting with Eclipse Collections 8.0, we can leverage Collectors
- Also has some “built in” – Collectors2

Goldman Sachs

Collectors2 Examples

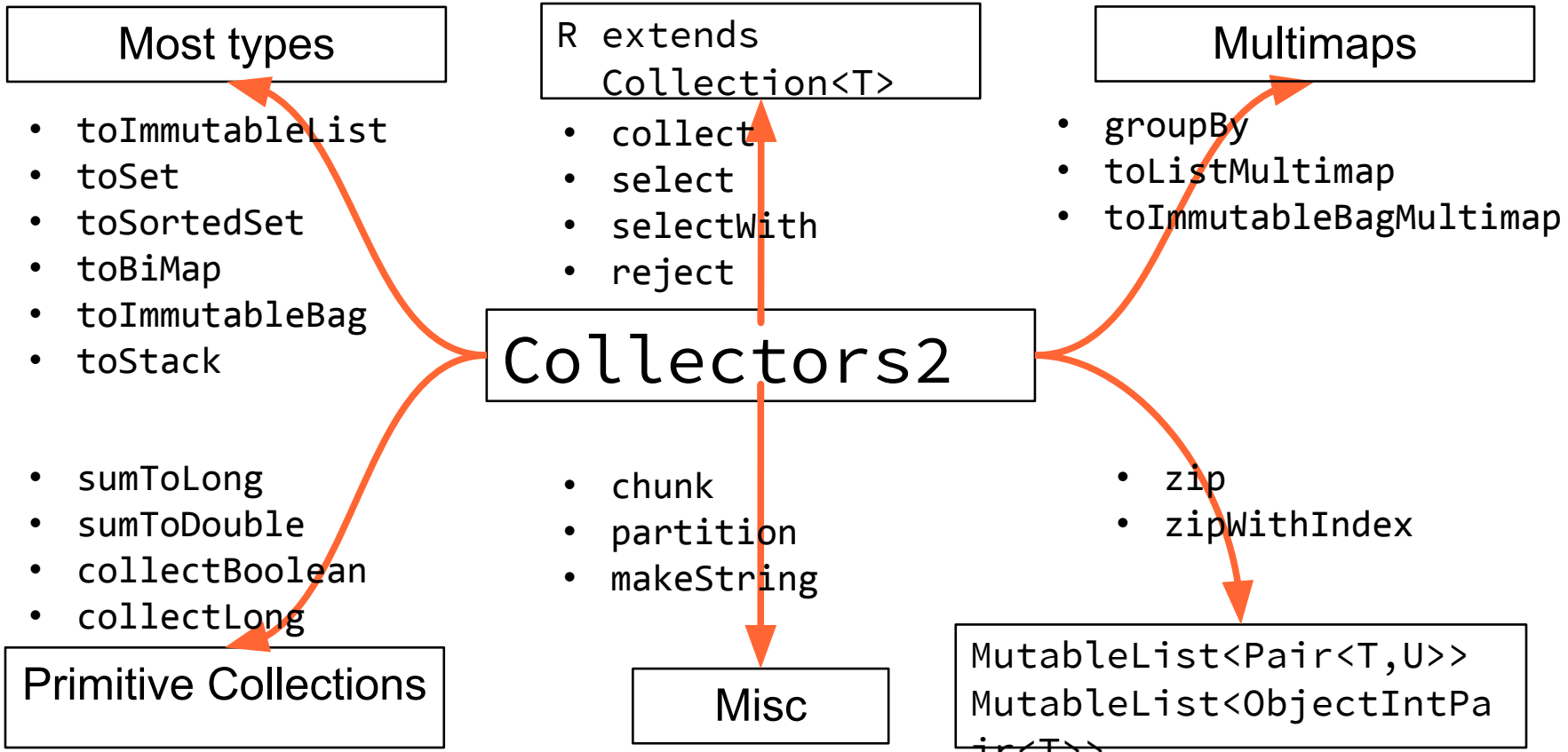
```
MutableBag<Pet> bag = this.people.stream()
        .map(Person::getPets).flatMap(List::stream)
        .collect(Collectors2.toBag());
```

```
// Bag Contents:
Dog   → 2
Cat   → 1
Snake → 1
```

```
ImmutableSet<Person> catPeople = this.people.stream()
        .filter(person -> person.hasPet(PetType.CAT))
        .collect(Collectors2.toImmutableSet());
```

```
// Set Contents:
Bob
```

Collectors2



Interop w/Collectors & Collectors2

```
Map<Integer, String> people = this.people
        .stream()
        .collect(Collectors.groupingBy(
                Person::getNumberOfPets,
                Collectors2.makeString()));
```

```
// Output:
{1=Ted, Jake,
2=Bob}
```

```
Map<Integer, String> collect = this.people
        .stream()
        .collect(Collectors.groupingBy(
                Person::getNumberOfPets,
                Collectors.mapping(
                        Object::toString,
                        Collectors.joining(","))));
```

Goldman Sachs

Agenda

- Intro
- What is Eclipse Collections
- New features in 8.0
- Optional
- Collectors
- **Default methods**
- Primitive Collections
- Java 9
- Conclusion

Goldman Sachs

RichIterable.reduceInPlace

```
/**
 * This method produces the equivalent result as {@link
Stream#collect(Collector)}.
 *
 * @since 8.0
 */
default <R, A> R reduceInPlace(Collector<? super T, A, R> collector)
{
    A mutableResult = collector.supplier().get();
    BiConsumer<A, ? super T> accumulator = collector.accumulator();
    this.each(each -> accumulator.accept(mutableResult, each));
    return collector.finisher().apply(mutableResult);
}
```

Goldman Sachs

Example

```
//using Java 8 and Collectors2
ImmutableSet<Person> people = this.people
        .stream()
        .filter(person -> person.hasPet(PetType.DOG))
        .collect(Collectors2.toImmutableSet());

//using Eclipse Collections and Collectors
ImmutableSet<Person> peopleReduce = this.people
        .selectWith(Person::hasPet, PetType.DOG)
        .reduceInPlace(Collectors2.toImmutableSet());
```

```
// Set Contents:
[Bob, Ted]
```

Goldman Sachs

Lazy and immutable API

```
this.people.toImmutable().stream()


this.people.asLazy().stream()
```

```
Set<Person> peopleReduced = this.people
        .asLazy() // stream() is no longer available
        .selectWith(Person::hasPet, PetType.DOG)
        .reduceInPlace(Collectors.toSet());
```

Goldman Sachs

Agenda

- Intro
- What is Eclipse Collections
- New features in 8.0
- Optional
- Collectors
- Default methods
- **Primitive Collections**
- Java 9
- Conclusion

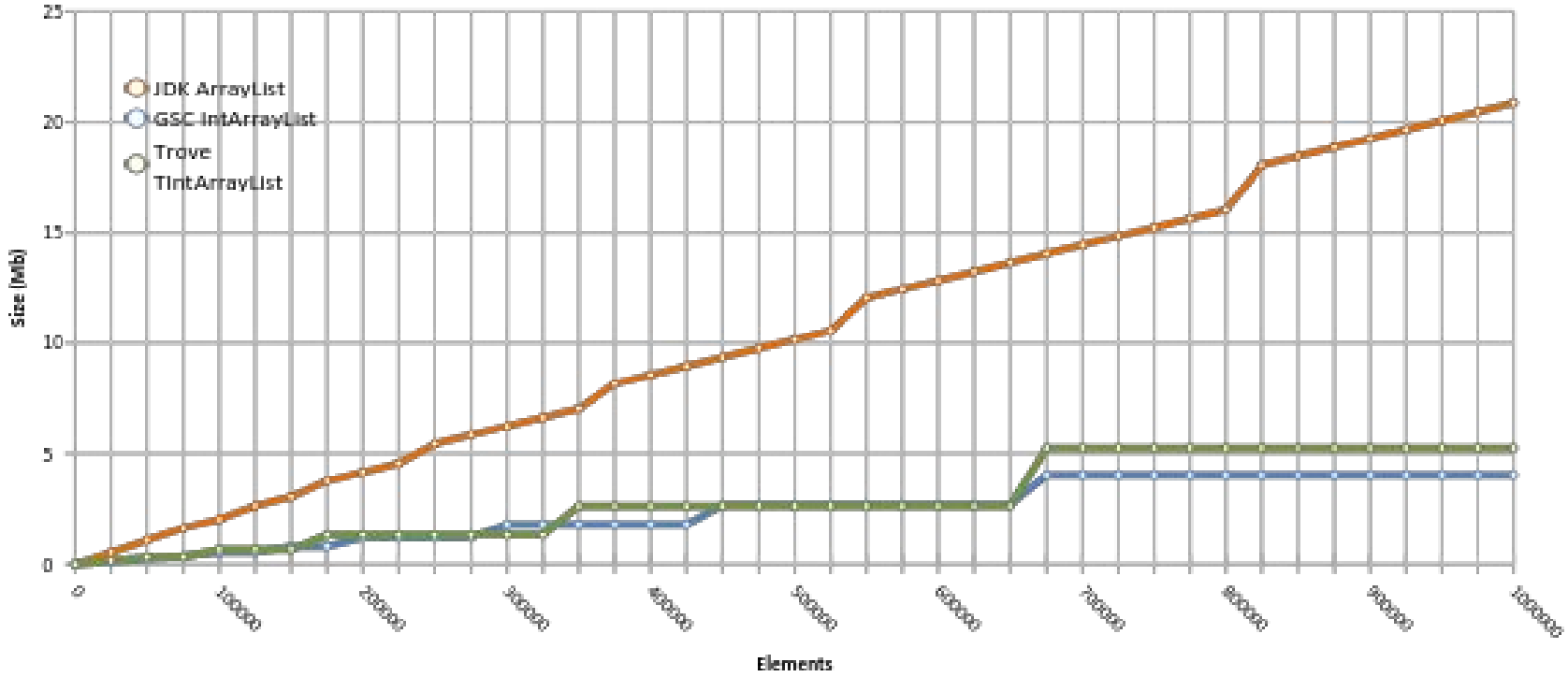
Goldman Sachs

Primitive Collections

- Primitive Collections since GS Collections 3.0.
- Memory Optimized Collections for primitive types.
- Similar Interface hierarchy to the Object collections.
- Symmetry among the primitive types.

Goldman Sachs

How much should an **int** cost?



Goldman Sachs

List<Integer> vs. IntList

- Java has object and primitive arrays
  - Primitive arrays have no behaviors
- Java does not have primitive Lists, Sets or Maps
  - Primitives must be boxed
  - Boxing is expensive
    - Reference + Header + alignment

Goldman Sachs

Primitive vs. JDK Collections

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

IntStream vs. LazyIntIterable

- Java has only 3 primitive stream: Int, Long, Double
  - IntStream.of vs. Stream.of for creation.
  - Specialized lambda expressions
  - Interchangable between Primitive and Obj stream

Goldman Sachs

IntStream vs. LazyIntIterable cont.

- LazyIterable for all 8 primitive types.
  - AsLazy API for creation
  - Rich APIs both similar to object and type specific.
  - Specialized functional interfaces

  These functional interfaces now can be used in Java 8 streams too!
  - Reusability

Goldman Sachs

Use or Reuse?

- Streams – like Iterator

```
IntStream stream = IntStream.of(1, 2, 3);
Assert.assertEquals(1, stream.min().getAsInt());
Assert.assertEquals(3, stream.max().getAsInt());          // throws IllegalStateException
```

```
java.lang.IllegalStateException: stream has already been operated upon or closed
      at java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:229)
      at java.util.stream.IntPipeline.reduce(IntPipeline.java:461)
      at java.util.stream.IntPipeline.max(IntPipeline.java:424))
```

- LazyIterable – Iterable

```
LazyIntIterable lazy = IntArrayList.newListWith(1, 2, 3).asLazy();
Assert.assertEquals(1, lazy.min());
Assert.assertEquals(3, lazy.max());                       // reuse!
```

Filter Pet Ages Appearing Once

stream

```
List<Map.Entry<Integer, Long>> counts = this.people.stream()
        .flatMap(person -> person.getPets().stream())
        .collect(Collectors.groupingBy(Pet::getAge,
Collectors.counting()))
        .entrySet()
        .stream()
        .filter(e -> e.getValue().equals(Long.valueOf(1)))
        .collect(Collectors.toList());
```

```
// Output:
[3=1, 4=1]
```

asLazy

```
MutableIntBag counts = this.people.asLazy()
     .flatCollect(Person::getPets)
     .collectInt(Pet::getAge)
     .toBag()
     .selectByOccurrences(IntPredicates.equal(1));
```

```
// Output:
[3, 4]
```

Goldman Sachs

Find the Mode of Pet Age

stream

```
List<Map.Entry<Integer, Long>> topOccurrenceList = this.people.stream()
        .flatMap(person -> person.getPets().stream())
        .collect(Collectors.groupingBy(Pet::getAge, Collectors.counting()))
        .entrySet()
        .stream()
        .sorted(Map.Entry.<Integer, Long>comparingByValue().reversed())
        .limit(1)
        .collect(Collectors.toList());
```

```
// Output:
[2=2]
```

asLazy

```
MutableList<IntIntPair> topOccurrences =
                this.people.asLazy()
                   .flatCollect(Person::getPets)
                   .collectInt(Pet::getAge)
                   .toBag()
                   .topOccurrences(1);
```

```
// Output:
[2:2]
```

Goldman Sachs

Find the Least Frequent Pet Age

stream

```
List<Map.Entry<Integer, Long>> bottomOccurrenceList = this.people.stream()
        .flatMap(person -> person.getPets().stream())
        .collect(Collectors.groupingBy(Pet::getAge, Collectors.counting()))
        .entrySet()
        .stream()
        .sorted(Map.Entry.<Integer, Long>comparingByValue())
        .limit(1)
        .collect(Collectors.toList());
```

```
// Output:
[3=1]
```

asLazy

```
MutableList<IntIntPair> bottomOccurrences =
                this.people.asLazy()
                    .flatCollect(Person::getPets)
                    .collectInt(Pet::getAge)
                    .toBag()
                    .bottomOccurrences(1);
```

```
// Output:
[3:1]
```

Goldman Sachs

Agenda

- Intro
- What is Eclipse Collections
- New features in 8.0
- Optional
- Collectors
- Default methods
- Primitive Collections
- **Java 9**
- Conclusion

Goldman Sachs

What's next?

- Java 9 will introduce many interesting changes to the Java ecosystem
  - Module System
  - Internal API Encapsulation
- Eclipse Collections **must** change in order to be compatible

Goldman Sachs

Methods using reflection

- We need to change our methods that use reflection to build

Goldman Sachs

What happens?

- *Compile, run tests, deprecate*
- Demo

Goldman Sachs

ArrayListIterate

```
public final class ArrayListIterate
{
    private static final Field ELEMENT_DATA_FIELD;
    private static final Field SIZE_FIELD;
    private static final int MIN_DIRECT_ARRAY_ACCESS_SIZE = 100;

    static
    {
        Field data = null;
        Field size = null;
        try
        {
            data = ArrayList.class.getDeclaredField("elementData");
            size = ArrayList.class.getDeclaredField("size");
            data.setAccessible(true);
            size.setAccessible(true);
        }
        catch (Exception ignored)
        {
            data = null;
            size = null;
        }
        ELEMENT_DATA_FIELD = data;
        SIZE_FIELD = size;
    }
```

Goldman Sachs

Workarounds for reflection

- http://mail.openjdk.java.net/pipermail/jigsaw-dev/2017-March/011763.html
- Can avoid via command line arguments
- EC proactively solved the problem

Goldman Sachs

Agenda

- Intro
- What is Eclipse Collections
- New features in 8.0
- Optional
- Collectors
- Default methods
- Primitive Collections
- Java 9
- **Conclusion**

Goldman Sachs

Resources

- Eclipse Collections 8.2 release

  https://github.com/eclipse/eclipse-collections/releases/tag/8.2.0
- Collectors Fair: Developing and Marketing a New Kind of Component
  https://static.rainfocus.com/oracle/oow16/sess/1462848741026001JnVD/ppt/CollectorsFair_J1_2016_final.pdf
- GS Collections Memory Benchmark
  http://www.goldmansachs.com/gs-collections/presentations/GSC_Memory_Tests.pdf

Goldman Sachs

we

BUILD

Learn more at GS.com/Engineering