Guava

Daniel Hinojosa

Making Java Bearable With Guava (2014 Edition)

Dulles Airport

Who is this for?

- Any Java Developer not familiar with Guava
- People who have to use Java by company fiat

2014 Edition

- What’s new?
  - Integration with Java 8
  - Broom Filters
  - Concurrency
- What went away
  - Optional
  - Splitters
  - Joiners

Where can I get the code?

http://www.github.com/dhinojosa/usingguava (http://www.github.com/dhinojosa/usingguava)

What is it?

- Indispensable set of utilities
- Additional and Immutable collections built upon JDK
- Open Source *
- Fully Generic Collections (unlike Apache Commons)
- Continually Growing (@Beta)
- Embrace DRY principle even more!

Guava Collections

Bi-Map

```
HashBiMap<String, String> englishSpanishMap =
   HashBiMap.<String, String>create();

englishSpanishMap.put("book", "libro");
englishSpanishMap.put("cloud", "nube");
englishSpanishMap.put("school", "escuela");
englishSpanishMap.put("computer", "ordenador");
```

Bi-Map

```
HashBiMap<String, String> englishSpanishMap =
   HashBiMap.<String, String>create();

englishSpanishMap.put("book", "libro");
englishSpanishMap.put("cloud", "nube");
englishSpanishMap.put("school", "escuela");
englishSpanishMap.put("computer", "ordenador");

englishSpanishMap.get("computer") => "ordenador"
```

Bi-Map

```
HashBiMap<String, String> englishSpanishMap =
   HashBiMap.<String, String>create();

englishSpanishMap.put("book", "libro");
englishSpanishMap.put("cloud", "nube");
englishSpanishMap.put("school", "escuela");
englishSpanishMap.put("computer", "ordenador");

englishSpanishMap.get("computer") => "ordenador"
englishSpanishMap.put("fill", "llenar");
```

Bi-Map

```
HashBiMap<String, String> englishSpanishMap =
   HashBiMap.<String, String>create();

englishSpanishMap.put("book", "libro");
englishSpanishMap.put("cloud", "nube");
englishSpanishMap.put("school", "escuela");
englishSpanishMap.put("computer", "ordenador");

englishSpanishMap.get("computer") => "ordenador"
englishSpanishMap.put("fill", "llenar");
englishSpanishMap.put("feed", "llenar");
```

Bi-Map

```
HashBiMap<String, String> englishSpanishMap =
   HashBiMap.<String, String>create();

englishSpanishMap.put("book", "libro");
englishSpanishMap.put("cloud", "nube");
englishSpanishMap.put("school", "escuela");
englishSpanishMap.put("computer", "ordenador");

englishSpanishMap.get("computer") => "ordenador"
englishSpanishMap.put("fill", "llenar");
englishSpanishMap.put("feed", "llenar");
```

IllegalArgumentException

Bi-Map

```
HashBiMap<String, String> englishSpanishMap =
   HashBiMap.<String, String>create();

englishSpanishMap.put("book", "libro");
englishSpanishMap.put("cloud", "nube");
englishSpanishMap.put("school", "escuela");
englishSpanishMap.put("computer", "ordenador");

englishSpanishMap.get("computer") => "ordenador"
englishSpanishMap.put("fill", "llenar");
englishSpanishMap.forcePut("feed", "llenar");
```

Bi-Map

```
HashBiMap<String, String> englishSpanishMap =
   HashBiMap.<String, String>create();

englishSpanishMap.put("book", "libro");
englishSpanishMap.put("cloud", "nube");
englishSpanishMap.put("school", "escuela");
englishSpanishMap.put("computer", "ordenador");

englishSpanishMap.get("computer") => "ordenador"
englishSpanishMap.put("fill", "llenar");
englishSpanishMap.forcePut("feed", "llenar");

englishSpanishMap.toString() =>
   {computer=ordenador, school=escuela,
    book=libro, cloud=nube, feed=llenar}
```

Bi-Map

```
englishSpanishMap.toString() =>
{computer=ordenador, school=escuela, book=libro, cloud=nube, feed=llenar}

BiMap<String, String> spanishEnglishMap = englishSpanishMap.inverse();

spanishEnglishMap.toString() =>
{escuela=school, nube=cloud, ordenador=computer, llenar=feed, libro=book}
```

Bi-Map

```
spanishEnglishMap.put("futbol", "soccer");

spanishEnglishMap.toString() =>
{escuela=school, nube=cloud, futbol=soccer, ordenador=computer, llenar=feed, libro=book}
```

Bi-Map

```
spanishEnglishMap.put("futbol", "soccer");

spanishEnglishMap.toString() =>
{escuela=school, nube=cloud, futbol=soccer, ordenador=computer, llenar=feed, libro=book}

englishSpanishMap.toString() =>
{computer=ordenador, school=escuela, book=libro, cloud=nube, soccer=futbol, feed=llenar}
```

Bi-Map

```
BiMap<String, String> spanishEnglishMap =
    englishSpanishMap.inverse();
```

Multimap

```
ArrayListMultimap<String, Integer>
  superBowlMap =
     ArrayListMultimap.create();
```

Different Flavors: `LinkedHashMultimap`, `LinkedListMultimap`, `TreeMultimap`, `HashMultimap`, `ListMultimap`, `SetMultimap`, `SortedSetMultimap`

Multimap

```
ArrayListMultimap<String, Integer> superBowlMap =
   ArrayListMultimap.create();
superBowlMap.put("Dallas Cowboys", 1972);
superBowlMap.put("Dallas Cowboys", 1978);
superBowlMap.put("Dallas Cowboys", 1993);
superBowlMap.put("Dallas Cowboys", 1994);
superBowlMap.put("Dallas Cowboys", 1996);
superBowlMap.put("Pittsburgh Steelers", 1975);
superBowlMap.put("Pittsburgh Steelers", 1976);
superBowlMap.put("Pittsburgh Steelers", 1979);
superBowlMap.put("Pittsburgh Steelers", 1980);
superBowlMap.put("Pittsburgh Steelers", 2006);
superBowlMap.put("Pittsburgh Steelers", 2009);
```

Multimap

```
superBowlMap.get("Dallas Cowboys").size() => 5
superBowlMap.get("Pittsburgh Steelers").size() => 6
superBowlMap.get("Buffalo Bills").size() => 0
```

Multiset (Bag)

```
Multiset<String> worldCupChampionships =
       HashMultiset.<String>create();
```

Different Flavors: `EnumMultiset`, `HashMultiset`, `ImmutableMultiset`, `LinkedHashMultiset`, `TreeMultiset`

Multiset (Bag)

```
Multiset<String> worldCupChampionships =
   HashMultiset.<String>create();

worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");

worldCupChampionships.add("Italy");
worldCupChampionships.add("Italy");
worldCupChampionships.add("Italy");
worldCupChampionships.add("Italy");

=> ["Brazil x 5", "Italy x 4"]
```

Multiset (Bag)

```
Multiset<String> worldCupChampionships =
    HashMultiset.<String>create();

worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");
worldCupChampionships.add("Brazil");

worldCupChampionships.add("Italy");
worldCupChampionships.add("Italy");
worldCupChampionships.add("Italy");
worldCupChampionships.add("Italy");

worldCupChampionships.add("Germany", 3); //explicitly add count

=> ["Brazil x 5", "Italy x 4", "Germany x 3"]
```

Multiset (Bag)

```
worldCupChampionships.count("Brazil") => 5
worldCupChampionships.count("Italy") => 4
worldCupChampionships.count("Germany") => 3
worldCupChampionships.count("United States") => 0 //Grr!
```

Java 8 and Multisets

```
worldCupChampionships.stream().forEach(t -> System.out.println(t));
```

```
worldCupChampionships.stream().forEach(System.out::println);
```

```
Multiset<String> updatedWorldCupChampionships =
  worldCupChampionships.stream().map((s) -> String.format("Team %s", s))
  .collect(Collectors.toCollection(HashMultiset::create));

=> [Team Brazil x 5, Team Italy x 4, Team Germany x 3]
```

Immutable vs. Unmodifiable

Unmodifiability of the JDK

```
Set<Integer> intSet = new HashSet<Integer>();
intSet.add(4);
intSet.add(5);
intSet.add(6);
intSet.add(7);

Set<Integer> unmodifiableSet =
   Collections.unmodifiableSet(intSet);
unmodifiableSet.add(10);
```

UnsupportedOperationException

Unmodifiability of the JDK

```
Set<Integer> intSet = new HashSet<Integer>();
intSet.add(4);
intSet.add(5);
intSet.add(6);
intSet.add(7);

Set<Integer> unmodifiableSet =
   Collections.unmodifiableSet(intSet);
intSet.add(10);
```

Unmodifiability of the JDK

```
Set<Integer> intSet = new HashSet<Integer>();
intSet.add(4);
intSet.add(5);
intSet.add(6);
intSet.add(7);

Set<Integer> unmodifiableSet =
   Collections.unmodifiableSet(intSet);
intSet.add(10); // allowed
unmodifiableSet.toString() => [4, 5, 6, 7, 10]
```

Not Immutable

> You can’t modify the collection, but I can!

Immutability

Guava contains factories to create actual immutable collections for:

- `Map`
- `MultiSet`
- `MultiMap`
- `SortedSet`
- `SortedMap`
- `List`
- `Set`
- `BiMap`

Immutablilty with `of`

`Immutable<CollectionType>.of(E1, E2, E3, E4)`

Immutability with `List`

```
List<Integer> integerList =
     ImmutableList.of(4, 4, 5, 6, 7);

integerList.toString() => [4, 4, 5, 6, 7]
```

Immutability with `Set`

```
Set<Integer> intSet =
    ImmutableSet.of(6, 7, 7, 8, 9, 10);

intSet.toString() => [6, 7, 8, 9, 10]
```

Immutability with `Map`

```
Map<String, String> capitalMap =
      ImmutableMap.of(
         "New Mexico", "Santa Fe",
         "Texas", "Austin",
         "Arizona", "Phoenix");

capitalMap.toString() =>
     New Mexico -> Santa Fe,
     Texas -> Austin, Arizona -> Phoenix
```

Immutability with `BiMap`

```
BiMap<String, String> biMap = ImmutableBiMap.of(
                "book", "libro",
                "cloud", "nube",
                "school", "escuela",
                "computer", "ordenador");

biMap.toString() =>
     {book=libro, cloud=nube, school=escuela,
         computer=ordenador}
```

Immutability with `Multimap`

```
Multimap<String, Integer> multiMap =
   ImmutableMultimap.of
      ("Dallas Cowboys", 1972,
       "Dallas Cowboys", 1993,
       "Dallas Cowboys", 1994,
       "Dallas Cowboys", 1994,
       "Dallas Cowboys", 1996);
```

Immutability with `Multimap`

```
Multimap<String, Integer> multiMap =
   ImmutableMultimap.of
      ("Dallas Cowboys", 1972,
       "Dallas Cowboys", 1993,
       "Dallas Cowboys", 1994,
       "Dallas Cowboys", 1994,
       "Dallas Cowboys", 1996);
```

But Dallas won in 1978 where is it? Where are the Steelers information I had earlier?

The Limits of `of`

```
Multimap<String, String> multiMap = ImmutableMultimap.of(
      "Dallas Cowboys", 1972, "Dallas Cowboys", 1993,
      "Dallas Cowboys", 1994, "Dallas Cowboys", 1994,
      "Dallas Cowboys", 1996, "Dallas Cowboys", 1978,
      "Pittsburgh Steelers", 1975, "Pittsburgh Steelers", 1976,
      "Pittsburgh Steelers", 1979, "Pittsburgh Steelers", 1980,
      "Pittsburgh Steelers", 2006, "Pittsburgh Steelers", 2009);
```

Compile Time Exception Cannot Resolve Method

Immutability with Builders

```
Immutable<CollectionType>.builder()
```

List Immutablity with Builders

```
List<Integer> intList =
    ImmutableList.<Integer>builder()
       .add(1, 2, 3, 4, 5)
       .addAll(Arrays.asList(6, 7, 8, 9, 10))
       .build();

intList.toString() =>
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Set Immutability with Builders

```
Set<Integer> intSet =
   ImmutableSet.<Integer>builder()
       .add(1, 2, 3, 4, 5)
       .addAll(Arrays.asList(5, 6, 7, 8, 9, 10))
       .build();

intSet.toString() => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Map Immutability with Builders

```
Map<String, String> capitals =
   new ImmutableMap.Builder<String, String>()
       .put("Brazil", "Brasilia")
       .put("United States", "Washington, DC")
       .put("Portugal", "Lisbon")
       .build();

capitals.toString() =>
{Brazil=Brasilia, United States=Washington, DC, Portugal=Lisbon}
```

Bi-Map Immutability with Builders

```
BiMap<String, String> biMap = ImmutableBiMap.
      <String, String>builder()
        .put("book", "libro")
        .put("cloud", "nube")
        .put("school", "escuela")
        .put("computer",
"ordenador").build();

capitals.toString() =>
{Brazil=Brasilia, United States=Washington, DC, Portugal=Lisbon}
```

Multimap Immutability with Builders

```
Multimap<String, Integer> multiMap =
  ImmutableMultimap.<String, Integer>builder()
    .put("Dallas Cowboys", 1972).put("Dallas Cowboys", 1993)
    .put("Dallas Cowboys", 1994).put("Dallas Cowboys", 1994)
    .put("Dallas Cowboys", 1996).put("Dallas Cowboys", 1978)
    .put("Pittsburgh Steelers", 1975)
    .put("Pittsburgh Steelers", 1976)
    .put("Pittsburgh Steelers", 1979)
    .put("Pittsburgh Steelers", 1980)
    .put("Pittsburgh Steelers", 2006)
    .put("Pittsburgh Steelers", 2009).build();
```

Multimap Immutability with Builders

```
multiMap.toString() => {Dallas Cowboys=[1972, 1993, 1994, 1994, 1996, 1978],
   Pittsburgh Steelers=[1975, 1976, 1979, 1980, 2006, 2009]}
```

Predicates and Functions

Predicates

Predicates

```
Predicate<Integer> isOdd = new Predicate<Integer>(){
     public boolean apply(Integer input) {
       return input % 2 != 0;
     }
};
```

Predicates

```
Predicate<Integer> isOdd = new Predicate<Integer>(){
     public boolean apply(Integer input) {
       return input % 2 != 0;
     }
};

Collection<Integer> unfiltered =
    Lists<Integer>.newArrayList
      (1, 5, 6, 8, 9, 10, 44, 55, 19);
```

Predicates

```
Predicate<Integer> isOdd = new Predicate<Integer>(){
     public boolean apply(Integer input) {
       return input % 2 != 0;
     }
};

Collection<Integer> unfiltered =
    Lists<Integer>.newArrayList
      (1, 5, 6, 8, 9, 10, 44, 55, 19);

Collections2.filter(unfiltered, isOdd).toString()
=>[1, 5, 9, 55, 19]
```

Predicates

```
Predicate<Integer> isOdd = new Predicate<Integer>(){
     public boolean apply(Integer input) {
       return input % 2 != 0;
     }
};

Collection<Integer> unfiltered =
    Lists<Integer>.newArrayList
      (1, 5, 6, 8, 9, 10, 44, 55, 19);

Collections2.filter(unfiltered, isOdd).toString()
=>[1, 5, 9, 55, 19]

unfiltered.toString()
=> [1, 5, 6, 8, 9, 10, 44, 55, 19]
```

Predicates

```
Predicate<Integer> isOdd = new Predicate<Integer>(){
     public boolean apply(Integer input) {
       return input % 2 != 0;
     }
};

Collection<Integer> unfiltered =
    Lists<Integer>.newArrayList
      (1, 5, 6, 8, 9, 10, 44, 55, 19);

Collections2.filter(unfiltered, isOdd).toString()
=>[1, 5, 9, 55, 19]

unfiltered.toString()
=> [1, 5, 6, 8, 9, 10, 44, 55, 19]

filtered.add(23); //Good
unfiltered.contains(23) //Yes!
```

Functions

Functions

```
Function<Integer, Integer> doubleIt = new
    Function<Integer, Integer>() {
       public Integer apply(Integer from) {
          return from * 2;
       }
    };
```

Functions

```
Function<Integer, Integer> doubleIt = new
    Function<Integer, Integer>() {
       public Integer apply(Integer from) {
          return from * 2;
       }
    };

Collection<Integer> untransformed = Lists
  .newArrayList
     (1, 5, 6, 8, 9, 10, 44, 55, 19);
```

Functions

```
Function<Integer, Integer> doubleIt = new
    Function<Integer, Integer>() {
       public Integer apply(Integer from) {
          return from * 2;
       }
    };

Collection<Integer> untransformed = Lists
  .newArrayList
     (1, 5, 6, 8, 9, 10, 44, 55, 19);

Collections2.transform(untransformed, doubleIt).toString()

=> [2, 10, 12, 16, 18, 20, 88, 110, 38]
```

Functions

```
Function<Integer, Integer> doubleIt = new
    Function<Integer, Integer>() {
       public Integer apply(Integer from) {
          return from * 2;
       }
    };

Collection<Integer> untransformed = Lists
  .newArrayList
     (1, 5, 6, 8, 9, 10, 44, 55, 19);

Collections2.transform(untransformed, doubleIt).toString()

=> [2, 10, 12, 16, 18, 20, 88, 110, 38]

untransformed.toString() => [1, 5, 6, 8, 9, 10, 44, 55, 19]");
```

Utilities

Utilities

Simple Rule: Use the Plural of the Class for the utility you need.

Utilities

Simple Rule: Use the Plural of the Class for the utility you need.

`Booleans`, `Longs`, `Ints`, `Floats`, `Iterables`, `Iterators`, `Lists`, `Longs`, `Maps`, `Objects`, `Multimaps`, `ObjectArrays`, `Strings`, `Shorts`, `SignedBytes`, `Sets`, `Predicates`, `Multisets`, `Multimaps`, `BiMaps`, `Functions`, `Bytes`

Objects

Instead of:

```
  if (a != null) return a.equals(b);
  return b != null && b.equals(a);
```

Prefer:

```
 Objects.equal(a,b)
```

Lists

```
Lists.newArrayList ("one, "two", "three")
```

```
Lists.newLinkedList(1, 2, 3, 4, 5)
```

```
Lists.reverse(someList)
```

```
Lists.transform(list, function)
```

Maps

```
Maps.newHashMap();
```

```
Maps.newEnumMap();
```

```
Maps.newLinkedHashMap();
```

```
Maps.newConcurrentMap();
```

```
Maps.newTreeMap();
```

Maps

```
Maps.difference(map1,map2).entriesInCommon();
```

```
Maps.filterEntries(map, predicate);
```

```
Maps.filterKeys(map, predicate);
```

```
Maps.filterValues(map, predicate);
```

```
Maps.transformEntries(map, function);
```

```
Maps.transformValues(map, function);
```

Finding Differences

```
Map<String, String> stateCaps =
     ImmutableMap.<String, String>builder()
        .put("Tallahassee", "Florida")
        .put("Santa Fe", "New Mexico")
        .put("Trenton", "New Jersey")
        .put("Olympia", "Washington")
        .put("Albany", "New York").build();
Map<String, String> stateCaps2 =
     ImmutableMap.<String, String>builder()
        .put("Tallahassee", "Florida")
        .put("Raleigh", "North Carolina")
        .put("Bismarck", "North Dakota").build();
MapDifference<String, String> diff =
     Maps.difference(stateCaps, stateCaps2);
diff.entriesOnlyOnLeft().size() => 4
diff.entriesOnlyOnRight().size() => 2
```

Finding Common Entries

```
Map<String, String> stateCaps =
     ImmutableMap.<String, String>builder()
        .put("Tallahassee", "Florida")
        .put("Santa Fe", "New Mexico")
        .put("Trenton", "New Jersey")
        .put("Olympia", "Washington")
        .put("Albany", "New York").build();
Map<String, String> stateCaps2 =
     ImmutableMap.<String, String>builder()
        .put("Tallahassee", "Florida")
        .put("Raleigh", "North Carolina")
        .put("Bismarck", "North Dakota").build();
Map<String, String> common = Maps.difference(stateCaps,
    stateCaps2).entriesInCommon();
common.size() => 1
common.get("Tallahassee") => Florida
```

Using Predicate and Filter Values

```
Map<String, String> stateCaps =
    ImmutableMap.<String,String>builder()
    .put("Tallahassee", "Florida")
    .put("Santa Fe", "New Mexico")
    .put("Trenton", "New Jersey")
    .put("Olympia", "Washington")
    .put("Albany", "New York").build();
Predicate<CharSequence> startsWithNew =
     Predicates.containsPattern("New.*");
Map<String, String> filtered =
      Maps.filterValues(stateCaps,startsWithNew);
filtered.size(); => 3
```

Iterables

```
Iterables.concat(list1, list2);
```

```
Iterables.elementsEqual(list1, list2);
```

```
Iterables.cycle(list);
```

```
Iterables.filter(list, clazz);
```

Iterables (Continued)

```
Iterables.filter(list, predicate);
```

```
Iterables.partition(list, size);
```

```
Iterables.paddedPartition(list, size);
```

```
Iterables.transform(list, function);
```

```
Iterables.tryFind(list, predicate);
```

Using Cycle

```
List<Integer> list = Lists.newArrayList(1, 2, 3, 4, 5);
Iterable iterable = Iterables.cycle(list);
Iterator it = iterable.iterator();
for (int i = 0; i < 1000; i++){
  it.next();
} => 1
```

Using Partition

```
List<Integer> list =
   Lists.newArrayList(1, 2, 3, 4, 5);
Iterable iterable =
   Iterables.partition(list, 2);
Iterator it = iterable.iterator();
it.next(); => List(1, 2)
it.next(); => List(3, 4)
it.next(); => List(5);
```

Using Padded Partition

```
List<Integer> list =
   Lists.newArrayList(1, 2, 3, 4, 5);
Iterable iterable =
   Iterables.partition(list, 2);
Iterator it = iterable.iterator();
it.next(); => List(1, 2)
it.next(); => List(3, 4)
it.next(); => List(5);
```

Using Strings

```
Strings.isNullOrEmpty(string)
Strings.nullToEmpty(string)
Strings.padEnd(string, minLength, char)
Strings.padStart(string, minLength, char)
Strings.repeat(string, times)
```

Moral of the Story

> If it feels like someone else has already developed what you are trying to, do look it up.

Ordering

Ordering

```
public class StarWarsEpisode {
       private String name;
       private int number;
       private int year;

       //getters, toString, hashCode, equals
}
```

Ordering

```
public class StarWarsCharacter implements
     Comparable<StarWarsCharacter> {
        private String name;
        private StarWarsEpisode firstAppearance;

        //getters, toString, hashCode, equals

        public int compareTo(StarWarsCharacter o) {
            return this.name.compareTo(o.name) +
                    this.firstAppearance.getYear() -
                    o.firstAppearance.getYear();
    }
}
```

Ordering

```
aNewHope = new StarWarsEpisode
          ("A New Hope", 4, 1977);
empireStrikesBack = new StarWarsEpisode
          ("The Empire Strikes Back", 5, 1980);
returnOfTheJedi = new StarWarsEpisode
          ("Return Of The Jedi", 6, 1983);
phantomMenace = new StarWarsEpisode
          ("The Phantom Menace", 1, 1999);
attackOfTheClones = new StarWarsEpisode
          ("Attack Of The Clones", 2, 2002);
revengeOfTheSith = new StarWarsEpisode
          ("Revenge Of The Sith", 3, 2005);
```

Ordering

```
hanSolo = new StarWarsCharacter
          ("Han Solo", aNewHope);
lukeSkywalker = new StarWarsCharacter
          ("Luke Skywalker", aNewHope);
princessLeia = new StarWarsCharacter
          ("Princess Leia", aNewHope);
landoCalrissian = new StarWarsCharacter
          ("Lando Calrissian", empireStrikesBack);
bobaFett = new StarWarsCharacter
          ("Boba Fett", empireStrikesBack);
```

Ordering

```
public class StarWarsEpisodeYearComparator
   implements Comparator<StarWarsEpisode> {
      public int compare (StarWarsEpisode o1,
                          StarWarsEpisode o2) {
            return o1.getYear() - o2.getYear();
      }
}
```

Ordering

```
Ordering.from(
     new StarWarsEpisodeYearComparator())
        .max(aNewHope, phantomMenace)

=> phantomMenace
```

Ordering

```
public class StarWarsCharacterNameComparator implements
   Comparator<StarWarsCharacter> {
      public int compare(StarWarsCharacter o1,
                         StarWarsCharacter o2) {
         return o1.getName().compareTo(o2.getName());
     }
}
```

Ordering

```
Ordering.from(new StarWarsCharacterYearComparator())
    .compound(new StarWarsCharacterNameComparator())
    .sortedCopy(Lists.newArrayList(
                bobaFett,princessLeia,
                landoCalrissian, lukeSkywalker,
                hanSolo)).toString();

=> ["Han Solo", "Luke Skywalker", "Princess Leia",
    "Boba Fett", "Lando Calrissian"]
```

Ordering

```
Ordering<String> byLengthOrdering =
       new Ordering<String>() {
            public int compare(String left, String right) {
                return (left.length() - right.length());
            }
        };

byLengthOrdering.max(hanSolo.getName(),
                     lukeSkywalker.getName(),
                     princessLeia.getName())

=> "Luke Skywalker"
```

Ordering

```
Ordering.explicit(phantomMenace,
    attackOfTheClones, revengeOfTheSith,
    returnOfTheJedi, aNewHope,
    empireStrikesBack).max(revengeOfTheSith, aNewHope);

=> aNewHope
```

Ordering

```
byLengthOrdering = new Ordering<String>() {
            public int compare(String left, String right) {
                return (left.length() - right.length());
            }
        };

byLengthOrdering.nullsLast()
       .sortedCopy(Arrays.asList(hanSolo.getName(), null,
            lukeSkywalker.getName(), null,
                princessLeia.getName())).toString()

=> ["Han Solo", "Princess Leia", "Luke Skywalker", null, null]
```

Ordering

```
byLengthOrdering = new Ordering<String>() {
            public int compare(String left, String right) {
                return (left.length() - right.length());
            }
};

byLengthOrdering.isOrdered
        (Arrays.asList(hanSolo.getName(),
                       princessLeia.getName(),
                       lukeSkywalker.getName(),
                       lukeSkywalker.getName())
=> true
```

Ordering

```
byLengthOrdering = new Ordering<String>() {
            public int compare(String left, String right) {
                return (left.length() - right.length());
            }
};

byLengthOrdering.isStrictlyOrdered
        (Arrays.asList(hanSolo.getName(),
                       princessLeia.getName(),
                       lukeSkywalker.getName(),
                       lukeSkywalker.getName())
=> false
```

Ordering

```
StarWarsCharacterNameComparator
    starWarsCharacterNameComparator = new
         StarWarsCharacterNameComparator();

StarWarsCharacter key = new
       StarWarsCharacter("Princess Leia", null);

Ordering.from(starWarsCharacterNameComparator)
    .binarySearch(Arrays.asList(bobaFett, hanSolo,
                  landoCalrissian, lukeSkywalker,
                  princessLeia), key)
=> 4
```

Ordering

```
public class StarWarsCharacter implements
     Comparable<StarWarsCharacter> {
        private String name;
        private StarWarsEpisode firstAppearance;

        //getters, toString, hashCode, equals

        public int compareTo(StarWarsCharacter o) {
            return this.name.compareTo(o.name) +
                    this.firstAppearance.getYear() -
                    o.firstAppearance.getYear();
    }
}
```

Ordering

```
Ordering.natural()
     .sortedCopy(Arrays.asList
         (bobaFett, hanSolo, lukeSkywalker,
            princessLeia, landoCalrissian)).toString()

=>["Boba Fett", "Han Solo", "Lando Calrissian", "Luke Skywalker", "Princess Leia"]
```

Event Stream

Event Stream

- Dispatches Events
- Easier than the `java.util.Observer` and `java.util.Observable`
- Requires the components to explicitly register with one another
- Posters, Handlers, Dead Events

Broadcast Event

```
public class BroadcastEvent {
  private String message;

  public BroadcastEvent(String message) {
     this.message = message;
  }

  public String getMessage() {
     return message;
  }

  //equals, hashcode, toString
}
```

Broadcaster

```
public class Broadcaster {
   private EventBus eventBus;

   public void setEventBus(EventBus eventBus) {
      this.eventBus = eventBus;
   }

   public void broadcastToAll() {
      this.eventBus.post(
      new BroadcastEvent("The Guava Revolution
          will not be televised"));
   }
}
```

Brodcast Event

```
public class BroadcastEvent {
   private String message;

   public BroadcastEvent(String message) {
      this.message = message;
   }

   public String getMessage() {
      return message;
   }

   //equals, hashcode, toString
}
```

Subscriber

```
public class Subscriber {
   private List<String> messages =
        Lists.newArrayList();

   @Subscribe
   public void eventOccured(BroadcastEvent event) {
      messages.add(event.getMessage());
   }

   public int getCount() {
      return messages.size();
   }

   public List<String> getMessages() {
      return ImmutableList.copyOf(messages);
   }
}
```

Using the Event Bus

```
EventBus eventBus = new EventBus();
Subscriber subscriber = new Subscriber();
eventBus.register(subscriber);

Broadcaster broadcaster = new Broadcaster();
broadcaster.setEventBus(eventBus);

broadcaster.broadcastToAll();
broadcaster.broadcastToAll();
broadcaster.broadcastToAll();

subscriber.getCount() => 3
```

Questions?

Thank You

- Email: dhinojosa@evolutionnext.com (mailto:dhinojosa@evolutionnext.com)
- Github: https://www.github.com/dhinojosa (https://www.github.com/dhinojosa)
- Twitter: http://twitter.com/dhinojosa (http://twitter.com/dhinojosa)
- Google Plus: http://gplus.to/dhinojosa (http://gplus.to/dhinojosa)
- Linked In: http://www.linkedin.com/in/dhevolutionnext (http://www.linkedin.com/in/dhevolutionnext)

Last updated 2014-05-16 12:15:52 MDT