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
  - Enhanced `LoadingCache`
  - 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.create();

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

Bi-Map

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

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

Multiset (Bag)

```
Multiset<String> worldCupChampionships = HashMultiset.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.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);
unmodifiableSet.toString() => [4, 5, 6, 7, 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
```

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`

Immutabilty 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 Immutability with Builders

```
List<Integer> intList =
    ImmutableList.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.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.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!
```

About Java 8 Lambdas

Functional Interface Definition

> A functional interface is any interface that contains only one abstract method. (A functional interface may contain one or more default methods or static methods.) Because a functional interface contains only one abstract method, you can omit the name of that method when you implement it.

(`equals` is an explicit declaration of a concrete method inherited from `Object` that, without this declaration, would otherwise be implicitly declared.)

Default Method

```
public interface Human {
   public String getFirstName();
   public String getLastName();
   default public String getFullName() {
     return String.format("%s %s", getFirstName(), getLastName());
   }
}
```

Predicate definition

```
public interface Predicate<T> {
  boolean apply(@Nullable T input);

  @Override
  boolean equals(@Nullable Object object);
}
```

Converting the Predicate Example for Java 8

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

Collection<Integer> filteredList =
        Collections2.filter(unfiltered, input -> input % 2 != 0);
```

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]");
```

Functions with Java 8

```
public interface Function<F, T> {
  @Nullable T apply(@Nullable F input);
  @Override
  boolean equals(@Nullable Object object);
}
```

Converting the Functions Example for Java 8

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

List<Integer> transformed = Lists.transform(untransformed, from -> from * 2);
```

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)
```

Objects.equal

```
public class Employee {
   private String firstName;
   private String lastName;

   public String getFirstName() {...}
   public String getLastName() {...}

   @Override
   public boolean equals(Object o) {
      if (this == o) return true;
      if (o == null || getClass() != o.getClass()) return false;

      Employee employee = (Employee) o;

      if (firstName != null ? !firstName.equals(employee.firstName) :
                               employee.firstName != null) return false;
      if (lastName != null ? !lastName.equals(employee.lastName) :
                               employee.lastName != null) return false;
      return true;
   }
}
```

Objects.equal

```
public class Employee {
   private String firstName;
   private String lastName;

   public String getFirstName() {...}
   public String getLastName() {...}

   @Override
   public boolean equals(Object o) {
      if (!(o instanceof Employee)) return false;
        Employee employee = (Employee) o;
        return Objects.equal(this.firstName, employee.firstName) &&
                Objects.equal(this.lastName, employee.lastName);
      }
}
```

What about Java 7’s `Object.equal`?

Same Implementation

Guava’s implementation

```
@CheckReturnValue //JSR305 javax.annotation.CheckReturnValue
public static boolean equal(@Nullable Object a, @Nullable Object b) {
    return a == b || (a != null && a.equals(b));
}
```

`java.util.Object`'s implementation

```
public static boolean equals(Object a, Object b) {
   return (a == b) || (a != null && a.equals(b));
}
```

What other good things does Java 7 `Objects` have?

```
public static boolean equals(Object a, Object b)
public static boolean deepEquals(Object a, Object b)
public static int hashCode(Object o)
public static int hash(Object... values)
public static String toString(Object o)
public static String toString(Object o, String nullDefault)
public static <T> int compare(T a, T b, Comparator<? super T> c)
public static <T> T requireNonNull(T obj)
public static <T> T requireNonNull(T obj, String message)
public static boolean isNull(Object obj)
public static boolean nonNull(Object obj)
public static <T> T requireNonNull(T obj, Supplier<String> messageSupplier)
```

Which should you use?

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)
```

Preconditions

```
Integer grade = null;
addGrade(grade);
```

```
java.lang.NullPointerException:
  Grade cannot be null
    at com.google.common.base.Preconditions
    .checkNotNull(Preconditions.java:204)
```

Preconditions

```
import static com.google.common.base.Preconditions.*;

public void addGrade(Integer grade) {
    checkNotNull(grade,
       "grade cannot be null");
    checkArgument
       (grade >= 0 && grade < 101,
          "Grade must be between 0 and 101");
    this.grades.add(grade);
}
```

Preconditions

```
Integer grade = 133;
addGrade(grade);
```

```
java.lang.IllegalArgumentException:
   Grade must be between 0 and 101 at
    com.google.common.base.Preconditions.
     checkArgument(Preconditions.java:88)
```

Preconditions

```
import static com.google.common.base.Preconditions.*;
public class Book {
       private List grades;
       ...
       public Integer getHighestGrade() {
              checkState(grades != null, "Grades
                  are not set");
              checkState(grades.size() > 0, "No
                  grades are entered");
              return Collections.max(this.grades);
          }
}
```

Preconditions

```
Book book = new Book();
book.getHighestGrade();
```

```
java.lang.IllegalStateException:
  Grades are not set at
    com.google.common.base.Preconditions.
       checkState(Preconditions.java:145)
```

Preconditions

```
import static com.google.common.base.Preconditions.*;

public class Book {
    private List grades;
    ...
    public Integer getHighestGrade() {
        checkState(grades != null, "Grades
            are not set");
        checkState(grades.size() > 0, "No
            grades are entered");
        return
         Collections.max
            (this.grades);
    }
}
```

Preconditions

```
Book book = new Book(Lists.
  <Integer>newArrayList());
book.getHighestGrade();
```

```
java.lang.IllegalStateException: No grades
  are entered at
   com.google.common.base.Preconditions.
       checkState(Preconditions.java:145)
```

Moral of the Story

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

LoadCache

`LoadCache`

- A simple concurrent cache
- All features optional
- Great features include
  - A concurrent `Map`
  - Automatic loading into the cache
  - Least-Recently-Used (LRU) Eviction of entries
  - `Weak` or `Soft` Reference Eviction
  - Eviction by Expiration since Last read and Last written
  - Add your own custom Eviction Listener
  - Automatically does small amounts of maintenance over time
  - Statistics

`LoadCache` Concurrency

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                              .concurrencyLevel(4);
```

- Set the anticipated number of thread that will need to **update** the `Cache`
- Overestimates and Underestimates can affect performance
- Since reads are concurrent, this is much faster than doing your own **synchronization**

`LoadCache` Automatic Loading

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                            .concurrencyLevel(4)
                                            .build(
   new CacheLoader<Integer, BigInteger>() {
         public BigInteger load(Integer source) throws InterruptedException {
            Thread.sleep(5000);
            return BigInteger.valueOf(source).multiply(new BigInteger("500"));
         }
      }
   );
```

Takes slightly over 5 seconds

```
graphs.get(4); => BigInteger(2000)
```

Takes slightly over 1 second

```
graphs.get(4); => BigInteger(2000)
```

`LoadCache` Maximum Size

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                            .concurrencyLevel(4)
                                            .maximumSize(1000)
                                            .build(...)
```

- Maximum number of entries in the map
- Evictions may occur before the maximum size is reached
- Typically will evict Last Recently Used (LRU) entries

`LoadCache` Expiration After Write

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                            .concurrencyLevel(4)
                                            .maximumSize(1000)
                                            .expireAfterWrite(5000, TimeUnit.SECONDS)
                                            .build(...)
```

- Length of time after an entry is updated that it should be automatically removed
- TimeUnit is an `enum` from `java.util.concurrent`

`LoadCache` Expiration After Access

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                             .concurrencyLevel(4)
                                             .maximumSize(1000)
                                             .expireAfterWrite(5000, TimeUnit.SECONDS)
                                             .expireAfterAccess(5000, TimeUnit.SECONDS)
                                             .build(...)
```

- Length of time after an entry is read that it should be automatically removed
- TimeUnit is an `enum` from `java.util.concurrent`

`LoadCache` Initial Capacity

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                            .concurrencyLevel(4)
                                            .maximumSize(1000)
                                            .expireAfterWrite(5000, TimeUnit.SECONDS)
                                            .expireAfterAccess(5000, TimeUnit.SECONDS)
                                            .initialCapacity(50)
                                            .build(...)
```

- Set the minimum size of `HashTable`s that back up the `Cache`
- The number of `HashTable` segments is the closest power of two of the concurrency level
  - e.g. A concurrency level of `4` will have `4 Hashtable` segments
  - e.g. A concurrency level of `20` with have `32 Hashtable` segments
- The size of each segment will be the `concurrency level - initial capacity` rounded to the next power of two
- Overestimating and Underestimating will affect performance

`LoadCache` Weak Keys

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                            .concurrencyLevel(4)
                                            .maximumSize(1000)
                                            .expireAfterWrite(5000, TimeUnit.SECONDS)
                                            .expireAfterAccess(5000, TimeUnit.SECONDS)
                                            .initialCapacity(50)
                                            .weakKeys()
                                            .build(...)
```

- Specifies that the key be wrapped in a `WeakReference`
- By default strong references are used
- A `WeakReference` will be reclaimed at the `gc`
- Uses `==` to check equality
- Entries that have been garbage collected will still show in size, but will not be accessible.

`LoadCache` Weak Values

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                            .concurrencyLevel(4)
                                            .maximumSize(1000)
                                            .expireAfterWrite(5000, TimeUnit.SECONDS)
                                            .expireAfterAccess(5000, TimeUnit.SECONDS)
                                            .initialCapacity(50)
                                            .weakKeys()
                                            .weakValues()
                                            .build(...)
```

- Specifies that the value be wrapped in a `WeakReference`
- By default strong references are used
- Opt for `softValues()` instead
- Uses `==` to check equality
- Entries that have been garbage collected will still show in size, but will not be accessible.

`LoadCache` Soft Values

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                           .concurrencyLevel(4)
                                           .maximumSize(1000)
                                           .expireAfterWrite(5000, TimeUnit.SECONDS)
                                           .expireAfterAccess(5000, TimeUnit.SECONDS)
                                           .initialCapacity(50)
                                           .weakKeys()
                                           .softValues()
                                           .build(...)
```

- Specifies that the value be wrapped in a `SoftReference`
- By default strong references are used
- Preferred over `weakValues()`
- Will be garbage collected in Least Recently Used (LRU) Manner globally throughout
- Uses `==` to check equality
- Entries that have been garbage collected will still show in size, but will not be accessible.

`LoadCache` Removal Listener

```
public class MyRemovalListener implements RemovalListener<Integer, BigInteger> {
  public void onRemoval(RemovalNotification<Integer, BigInteger>
                                        keyGraphRemovalNotification) {
  }
}
```

`LoadCache` Removal Listener

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                            .concurrencyLevel(4)
                                            .maximumSize(1000)
                                            .expireAfterWrite(5000, TimeUnit.SECONDS)
                                            .expireAfterAccess(5000, TimeUnit.SECONDS)
                                            .initialCapacity(50)
                                            .weakKeys()
                                            .softValues()
                                            .removeListener(new MyRemovalListener())
                                            .build(...)
```

- Listener invoked for various reasons
  - `RemovalCause.EXPLICIT` Explicitly removed with `invalidate()` or `invalidateAll()`
  - `RemovalCause.REPLACED` Value replaced
  - `RemovalCause.COLLECTED` Removed because key of value was garbage collected
  - `RemovalCause.SIZE` Removed because it didn't mean size constraints
  - `RemovalCause.EXPIRED` Removed because it expired

Enabling Stats

```
LoadingCache<Integer, BigInteger> map = CacheBuilder.newBuilder()
                                           .concurrencyLevel(4)
                                           .maximumSize(1000)
                                           .expireAfterWrite(5000, TimeUnit.SECONDS)
                                           .expireAfterAccess(5000, TimeUnit.SECONDS)
                                           .initialCapacity(50)
                                           .weakKeys()
                                           .softValues()
                                           .removeListener(new MyRemovalListener())
                                           .recordStats()
                                           .build(...)
```

- Was automatically enable pre-12.0
- Enables statistics to be recorded for the cache created

Viewing Stats:

- `map.stats()` returns `Cache.stats` object containing various statistics
    - `map.stats().averageLoadCount()` -
        - Returns the average time spent loading new values.
        - `totalLoadTime / (loadSuccessCount + loadExceptionCount)`
    - `map.stats().evictionCount()` - How many evictions have ocurred
    - `map.stats().hitCount()` - Number of times `Cache` lookup methods have returned a cached value
    - `map.stats().missCount()` - Number of times Cache lookup methods have returned an uncached (newly loaded) value, or null.
    - `map.stats().requestCount()`
        - Number of times Cache lookup methods have returned either a cached or uncached value
        - `hitCount + missCount`
    - `map.stats().hitRate()` - `hitCount / requestCount`
    - `map.stats().loadCount()`
        - Cache lookup methods attempted to load new values.
        - Includes both successful load operations
        - Includes those that threw exceptions
        - `loadSuccessCount + loadExceptionCount`

Remove Entries

Any and all entries can be invalidated

- Individually, using `Cache.invalidate(key)`
- In bulk, using `Cache.invalidateAll(keys)`
- To all entries, using `Cache.invalidateAll()`

Futures & Concurrency

`java.util.concurrent`

- Contains an `Executor`
- `Executor` manages independent threading tasks and decouples task submission from task execution
- Contains an `ExecutorService` that takes `Runnable` or `Callable` tasks and comes complete with a lifecycle and monitoring systems

The `Executor`

```
public interface Executor {
   void execute(Runnable command);
}
```

`ExecutorService`

- An `ExecutorService` is a subinterface of `Executor`
- Provides the ability to create and track `Future`s from `Runnable` and `Callable`s
- Services can be started up and shut down
- Can create your own
- Likely will use `Executors` class to create `ExecutorServices`

Snippet of `ExecutorService`

```
public interface ExecutorService extends Executor {
  ...
   <T> Future<T> submit(Callable<T> task);
   <T> Future<T> submit(Runnable task, T result);
   Future<?> submit(Runnable task);
  ...
}
```

Executors

- Utility Factory that can create `ExecutionServices`

`FixedThreadPool`

- `Executors.newFixedThreadPool()`
- *“Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue.”* according to the API
- Keeps threads constant and uses the queue to manage tasks waiting to be run
- If a thread fails, a new one is created in its stead
- If all threads are taken up, it will wait on an unbounded queue for the next available thread

`SingleThreadExecutor`

- `Executors.newSingleThreadExecutor()`
- *“Creates an Executor that uses a single worker thread operating off an unbounded queue.”*
- If a thread terminates due to a failure during execution prior to shutdown, a new one will take its place if needed to execute subsequent tasks.

`CachedThreadPool`

- `Executors.newCachedThreadPool()`
- Factory method is a flexible thread pool implementation that will reuse previously constructed threads if they are available
- If no existing thread is available, a new thread is created and added to the pool
- Threads that have not been used for sixty seconds are terminated and removed from the cache

`ScheduledThreadPool`

- `Executors.newScheduledThreadPool()`
- Can run your tasks after a delay or periodically
- This method does not return an `ExecutorService`
- Returns a ScheduledExecutorService which contains methods to help you set not only the task but the delay or periodic schedule

`Future`

- An asynchronous computation
- Contains methods determine completion of said computation
- Can be in running state, completed state, or have thrown an Exception

`Future` interface

```
public interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning);
    boolean isCancelled();
    boolean isDone();
    V get() throws
         InterruptedException, ExecutionException;
    V get(long timeout, TimeUnit unit)
         throws InterruptedException,
           ExecutionException, TimeoutException;
}
```

Blocking `Future`

```
ExecutorService executorService =
        Executors.newCachedThreadPool();

Callable<String> asynchronousTask
        = new Callable<String>() {
    @Override
    public String call() throws Exception {
        //something expensive
        Thread.sleep(5000);
        return "Asynchronous String Result";
    }
};

java.util.concurrent.Future<String> future =
        executorService.submit(asynchronousTask);
System.out.println("Processing 1");
System.out.println(future.get()); //waits if necessary
System.out.println("Processing 2");
```

```
Processing 1
Asynchronous String Result
Processing 2
```

Asynchronous `Future`

```java
ExecutorService executorService =
        Executors.newCachedThreadPool();

Callable<String> asynchronousTask = new Callable<String>() {
    @Override
    public String call() throws Exception {
        //something expensive
        Thread.sleep(1000);
        return "Asynchronous String Result";
    }
};

java.util.concurrent.Future<String> future =
        executorService.submit(asynchronousTask);


System.out.println("Processing Asynchronously 1");
while (!future.isDone()) {
    System.out.println("Doing something else");
}
System.out.println(future.get()); //waits if necessary
System.out.println("Processing Asynchronously 2");
```

```
Processing Asynchronously 1
Doing something else
Doing something else
Doing something else
Doing something else
Asynchronous String Result
Processing Asynchronously 2
```

Guava Listeners

```
ListeningExecutorService service = MoreExecutors.listeningDecorator(
                Executors.newCachedThreadPool());

Callable<String> asynchronousTask = new Callable<String>() {
    @Override
    public String call() throws Exception {
        //something expensive
        Thread.sleep(1000);
        return "Asynchronous String Result";
    }
};

ListenableFuture<String> listenableFuture = service.submit(asynchronousTask);

Futures.addCallback(listenableFuture,
        new FutureCallback<String>() {
            @Override
            public void onSuccess(String result) {
                System.out.println(
                        "Got the result and the answer is? " + result);
            }

            @Override
            public void onFailure(Throwable t) {
                System.out.println("Things happened man. Bad things" + t.getMessage());
            }
        }
);
```

Guava Listeners with Java 8

```
ListeningExecutorService service = MoreExecutors.listeningDecorator(
        Executors.newCachedThreadPool());

ListenableFuture<String> listenableFuture = service.submit (() -> {
    //something expensive
    Thread.sleep(1000);
    return "Asynchronous String Result";
});

Futures.addCallback(listenableFuture,
   new FutureCallback<String>() {
       @Override
       public void onSuccess(String result) {
           System.out.println(
                   "Got the result and the answer is? " + result);
       }

       @Override
       public void onFailure(Throwable t) {
           System.out.println("Things happened man. Bad things" + t.getMessage());
       }
   }
);
```

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

```java
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 2014-05-21 12:05:16 MDT