An Overview of Guava: Google Core Libraries for Java

Kevin Bourrillion
Java Core Libraries Team
Google, Inc.

Presented at QCon November 8, 2012

What's Guava?

Google™

Free open-source library for Java, GWT, Android.

14 packages chock full of utility classes and methods:

annotations	io
base	math
cache	net
collect	primitives
collect.testing	reflect
eventbus	testing
hash	util.concurrent

(These packages live under `com.google.common`.)

Where does it come from?

Google™

We are the Java Core Libraries Team at Google.

What do we do?

Why do we release Guava?

It's a junk drawer then?

Why are you here talking about it?

Actual* User Quotes

Google

"On any new Java project, the first thing I do is add a dependency to Guava; I just know I'm going to need it."

"Writing Java without Guava was like coding with one hand tied behind my back."

"Guava makes Java bearable again."

(We happen to think Java is more than bearable *anyway*, but still: Guava makes writing Java programs easier.)

*paraphrased. But I swear people really said these things. Honest.

About this presentation

com.google.common.base

String joining

Google

Who here has ever written this utility?

```
public class StringUtil {
  public static String join(
      String separator, Iterable<String> pieces) {

    // any of ~5 common implementations goes here

  }
}
```

String joining 2

Google

But what about all the *variations?*

- What to do with nulls?
  - skip over them? skip but leave separator? substitute `"null"` or some other string? Just die?
- Joining an Iterable, Iterator, varargs/array?
- Return a String, or append to an Appendable?

We could be looking at 18 to 48 different methods here.

To cover all these bases we made `Joiner`:

```
return Joiner.on(", ")
    .skipNulls()
    .join("one", null, "two", "three");
```

Joiner: what's going on here? Google™

To *get* a Joiner:

- `static Joiner on(String)`
- `static Joiner on(char)`

To get an *altered* Joiner from that:

- `Joiner skipNulls()`
- `Joiner useForNull(String)`

To *actually join stuff*:

- `String join(Itera___)`
- `String join(Object...)`
- `Appendable appendTo(Appendable, Itera___)`
- `Appendable appendTo(Object...)`

Splitter

Google™

Similar! But in the other direction.

```
return Splitter.on("|")
    .omitEmptyStrings()
    .split("|Harry||Ron|||Hermione  ||");

// returns "Harry", "Ron", "Hermione  ";
```

These classes use what we (tentatively) call the "Utility Object pattern."

CharMatcher (motivation)

Google™

Once upon a time we had a **`StringUtil`** class. Soon it was overflowing with static methods:

```
allAscii, collapse, collapseControlChars,
collapseWhitespace, indexOfChars, lastIndexNotOf,
numSharedChars, removeChars, removeCrLf, replaceChars,
retainAllChars, strip, stripAndCollapse,
stripNonDigits...
```

Each dealt with two *orthogonal* concerns:

(a) what does it consider a *matching* character?
(b) what do we *do* with these matching characters?

One static method per combination would not scale.

CharMatcher

Google

Once again we use the Utility Object pattern.

A `CharMatcher` instance represents the set of "matching" characters (part "a"). Methods on that instance provide the operations (part "b").

```
// "_34-425==" becomes "34425"
String sanitized =
    CharMatcher.anyOf("-=_")
        .removeFrom(input);
```

Separates "configuration" from "processing".

Getting a CharMatcher

Google™

Use a predefined constant (examples)

- `CharMatcher.WHITESPACE` (Unicode)
- `CharMatcher.ASCII`
- `CharMatcher.ANY`

Use a factory method (examples)

- `CharMatcher.is('x')`
- `CharMatcher.isNot('_')`
- `CharMatcher.oneOf("aeiou")`
- `CharMatcher.inRange('a', 'z')`
  `.or(inRange('A', 'Z')).negate()`

Or subclass `CharMatcher`, implement `matches(char)`.

Using your new CharMatcher

Google

```
boolean matchesAllOf(CharSequence)
boolean matchesAnyOf(CharSequence)
boolean matchesNoneOf(CharSequence)

int indexIn(CharSequence, int)
int lastIndexIn(CharSequence, int)
int countIn(CharSequence)

String removeFrom(CharSequence)
String retainFrom(CharSequence)
String trimFrom(CharSequence)
String trimLeadingFrom(CharSequence)
String trimTrailingFrom(CharSequence)
String collapseFrom(CharSequence, char)
String trimAndCollapseFrom(CharSequence, char)
String replaceFrom(CharSequence, char)
```

CharMatcher (last)

Google

Putting it back together... to scrub an id number, you might use

```
String seriesId =
    CharMatcher.DIGIT.or(CharMatcher.is('-'))
        .retainFrom(input);
```

In a loop? Move the definition above or to a constant.

```
static final CharMatcher ID_CHARS =
    CharMatcher.DIGIT.or(CharMatcher.is('-'));
 ...
String id = SERIES_ID_CHARS.retainFrom(input);
```

One problem with `null`

Google

Consider looking up a phone number.

```
PhoneNumber phone = phoneBook.lookUp("Barack");
if (phone == null) {
  // what does this mean?
}
```

No entry? Or entry exists but the phone number is unlisted?

Or consider a `nickname` field. `null` means no nickname, or we just don't know?

Optional<T>

Google

Guava's `Optional` class lets you have a "second kind of not-there" -- a "positive negative."

```
// Yes, has a nickname
Optional<String> nickname = Optional.of("Barry");

// Yes, we have no nickname
Optional<String> nickname = Optional.absent();

// Information missing/unknown
Optional<String> nickname = null;

// wat? Throws an exception.
Optional<String> nickname = Optional.of(null);
```

Optional<T> basic usage

Google™

```
Optional<String> nickname = person.nickname();

if (nickname == null) return;

if (nickname.isPresent()) {
  say("I hear people call you " + nickname.get());

} else {
  // calling get() would throw an exception!
  say("Your friends are not creative.");
}
```

Optional<T> cooler usages

Google™

```
for (String actualNick : nickname.asSet()) {
  doSomething(actualNick);
}
```

or

```
String firstName = person.firstName();
say("Hello, " + nickname.or(firstName));
```

For null-unfriendly collections

Google™

Many collections, including the JDK's `Queue` and `ConcurrentMap` implementations, don't allow null elements.

`Queue<Optional<Foo>>` is a simple and natural solution!

Optional<T>: the anti-null?

Google

Some users use Optional even when they have only one "kind of not-there". They use it as a `null` replacement.

Before:

```
Foo foo = someMethodThatMightReturnNull();
foo.whoops(); // I just forgot to check!
```

After:

```
Optional<Foo> foo = someMethod();
foo.get().whoops(); // same mistake!
```

Same mistake possible, but less likely to *some* degree.

Especially appropriate for public method return types.

Stopwatch

Google

For measuring *elapsed time*. **Don't** use `System.currentTimeMillis()`!

```
Stopwatch watch = new Stopwatch().start();
doSomeOperation();
long micros = watch.elapsedTime(MICROSECONDS);
```

- `Stopwatch` uses `nanoTime()` but exposes only relative timings, a meaningless absolute value
- an alternate time source can be substituted using `Ticker`
- has the same functions as a physical stopwatch
- `toString()` gives human readable format

Other things in base

Google™

... that we're not really going into ...

- `Preconditions`
- `Objects.toStringHelper()`
- `Objects.firstNonNull(T, T)`
- `Throwables.propagate(Throwable)`
- `CaseFormat`
- `Strings.repeat(String, int)`
- `Equivalence<T>`
- `Function, Predicate, Supplier`

com.google.common.collect

Collection Types (review)

Google

`Set`:

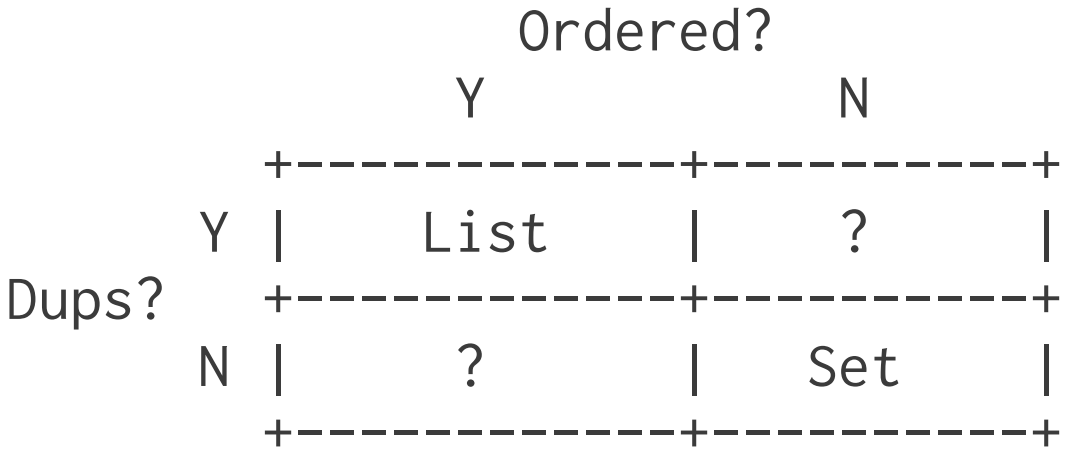
- doesn't guarantee order, has "unordered equality"
- collapses duplicates

`List`:

- guarantees order, has "ordered equality"
- allows duplicates (multiple "occurrences")

Aren't these two orthogonal concerns?

Basic Collection Types



Basic Collection Types

Google™

Dups? \ Ordered?	Y	N
Y	List	**Multiset**
N	(UniqueList)	Set

Multiset<E>

Google

Implements `Collection<E>`.

```
List:      [a, c, b, b, c, a, a, b]
Set:       [a, c, b]
Multiset:  [a, a, a, c, c, b, b, b]
```

So a Multiset implementation only needs to store one occurrence of each element, plus a count!

```
[a x 3, c x 2, b x 3]
```

Counting without Multiset

Google™

```
Map<String, Integer> tags = new HashMap<>();
for (BlogPost post : getAllBlogPosts()) {
  for (String tag : post.getTags()) {
    int value = tags.containsKey(tag) ? tags.get(tag) : 0;
    tags.put(tag, value + 1);
  }
}
```

distinct tags: `tags.keySet()`

count for "java" tag:

```
tags.containsKey("java") ? tags.get("java") : 0;
```

total count: `// uh oh...`

Counting with Multiset

Google

```
Multiset<String> tags = HashMultiset.create();
for (BlogPost post : getAllBlogPosts()) {
  tags.addAll(post.getTags());
}
```

distinct tags: `tags.elementSet();`
count for "java" tag: `tags.count("java");`
total count: `tags.size();`

Impls include `ConcurrentHashMultiset`, `EnumMultiset`...

Next...

Google™

Map:

a → 1

b → 2

c → 3

d → 4

Multimap<K, V>

Google™

Map:

a → 1
b → 2
c → 3
d → 4

Multimap:

a → 1
b → 2
c → 3
a → 4

Of course, we often also want to view this as:

a → 1, 4
b → 2
c → 3

Multimap<K, V>

Google

- Like a `Map` (key-value pairs), but may have duplicate keys
- The values related to a single key can be viewed as a collection (set or list)
- Consistent design to `Map<K, V>`
  (analogy holds: `Map` : `Set` :: `Multimap` : `Multiset`)
- Typically use instead of a `Map<K, Collection<V>>`
  - can view as that type using `asMap()`
- Almost always want variable type to be either `ListMultimap` or `SetMultimap` (and not `Multimap`)
- Implementations include `HashMultimap`, `ArrayListMultimap`...

Not going to say much more about these...

Immutable Collections

Google™

```
ImmutableSet<Integer> luckyNumbers =
    ImmutableSet.of(4, 8, 15, 16, 23, 42);
```

- unlike `Collections.unmodifiableXXX`, they
  - perform a copy (not a view / wrapper)
  - type conveys immutability
- offered for all collection types, including JDK ones
- inherently thread-safe
- deterministic, specified iteration order
- reduced memory footprint
- slightly improved CPU performance

Prefer immutability!

FluentIterable<T>

Google

You should all know Iterable<T>:

```
public interface Iterable<T> {
  Iterator<T> iterator();
}
```

Guava can turn your iterables into *fluent* iterables:

```
FluentIterable<Thing> things =
    FluentIterable.from(getThings());
```

... but why? ...

FluentIterable

Google

Because operations! (note: "lazy")

```
return FluentIterable.from(database.getClientList())
    .filter(
        new Predicate<Client>() {
          public boolean apply(Client client) {
            return client.activeInLastMonth();
          }
        })
    .transform(Functions.toStringFunction())
    .limit(10)
    .toList();
```

FluentIterable API

Google

Chaining methods (return `FluentIterable<T>`)

- **`filter`**`(Predicate)`
- **`transform`**`(Function)`
- **`skip`**`(int)`, `limit(int)`
- **`cycle`**`()`

Query methods (return `boolean`)

- **`allMatch`**`(Predicate)`, **`anyMatch`**`(Predicate)`
- **`contains`**`(Object)`
- **`isEmpty`**`()`

Extraction methods (return `Optional<T>`)

- **`first`**`()`, **`last`**`()`, **`firstMatch`**`(Predicate)`, **`get`**`(int)`

Conversion methods (return a copied `Collection<T>`)

- **`toList`**`()`, **`toSet`**`()`, **`toSortedSet`**`()`, **`toArray`**`()`

Functional Programming

Google

```
Multiset<Integer> lengths = HashMultiset.create(
    FluentIterable.from(strings)
        .filter(new Predicate<String>() {
            @Override public boolean apply(String s) {
              return JAVA_UPPER_CASE.matchesAllOf(s);
            }
          })
        .transform(new Function<String, Integer>() {
            @Override public Integer apply(String s) {
              return s.length();
            }
          }));
```

w/o Functional Programming

Google™

```
Multiset<Integer> lengths = HashMultiset.create();
for (String s : strings) {
  if (UPPER_CASE.matchesAllOf(s)) {
    lengths.add(s.length());
  }
}
```

Moral: just be careful out there.

Functional style can be great but it's not *automatically* the better way to go.

JDK 8 will make functional programming in Java **way** better (JSR-335).

Comparators

Google™

Who loves implementing comparators by hand?

```
Comparator<String> byReverseOffsetThenName =
    new Comparator<String>() {
      public int compare(String tzId1, String tzId2) {
        int offset1 = getOffsetForTzId(tzId1);
        int offset2 = getOffsetForTzId(tzId2);
        int result = offset2 - offset1; // careful!
        return (result == 0)
            ? tzId1.compareTo(tzId2)
            : result;
      }
    };
```

ComparisonChain example

Google

Here's one way to rewrite this:

```
Comparator<String> byReverseOffsetThenName =
    new Comparator<String>() {
      public int compare(String tzId1, String tzId2) {
        return ComparisonChain.start()
            .compare(getOffset(tzId2), getOffset(tzId1))
            .compare(tzId1, tzId2)
            .result();
      }
    };
```

Never allocates. Short-circuits (*kind of*).

Ordering example

Here's another:

```
Ordering<String> byReverseOffsetThenName =
    Ordering.natural()
        .reverse()
        .onResultOf(tzToOffsetFn())
        .compound(Ordering.natural());

// okay, this should actually go above
Function<String, Integer> tzToOffsetFn =
  new Function<String, Integer>() {
    public Integer apply(String tzId) {
      return getOffset(tzId);
    }
  };
```

Ordering details 1/3

Google™

Implements `Comparator`, and adds delicious goodies!
(Could have been called `FluentComparator`.)

Common ways to *get* an Ordering to start with:

- `Ordering.natural()`
- `new Ordering() { public int compare(...) {...} }`
- `Ordering.from(preExistingComparator);`
- `Ordering.explicit("alpha", "beta", "gamma", "delta");`

... then ...

Ordering details 2/3

Google

... then you can use the chaining methods to get an altered version of that `Ordering`:

- **`reverse`**`()`
- **`compound`**`(Comparator)`
- **`onResultOf`**`(Function)`
- **`nullsFirst`**`()`
- **`nullsLast`**`()`

... now you've got your `Comparator`! But also ...

Ordering details 3/3

Google

Ordering has some handy operations:

- **immutableSortedCopy**(Iterable)
- **isOrdered**(Iterable)
- **isStrictlyOrdered**(Iterable)
- **min**(Iterable)
- **max**(Iterable)
- **leastOf**(int, Iterable)
- **greatestOf**(int, Iterable)

... which is better?

Google

`Ordering` or `ComparisonChain`?

Answer: it depends, and that's why we have both.

Either is better than writing comparators by hand (why?).

When implementing a `Comparator` with `ComparisonChain`, still extend `Ordering` anyway!

Glimpses of other stuff

Static utils for primitives

Google™

Method	Longs	Ints	Shorts	Chars	Doubles	Bytes	S.Bytes	U.Bytes	Booleans
hashCode	X	X	X	X	X	X			X
compare	X	X	X	X	X		X	X	X
checkedCast		X	X	X			X	X	
saturatedCast		X	X	X			X	X	
contains	X	X	X	X	X	X			
indexOf	X	X	X	X	X	X			X
lastIndexOf	X	X	X	X	X	X			X
min	X	X	X	X	X		X	X	
max	X	X	X	X	X		X	X	
concat	X	X	X	X	X	X			X
join	X	X	X	X	X		X	X	X
toArray	X	X	X	X	X	X			X
asList	X	X	X	X	X	X			X
lexComparator	X	X	X	X	X		X	X	X
toByteArray	X	X	X	X					
fromByteArray	X	X	X	X					

Concurrency libraries

Google

First learn the contents of `java.util.concurrent`.

Then check out our:

- `ListenableFuture<V>`, `ListeningExecutorService`
- `CheckedFuture<V, X>`
- `Service`, `ServiceManager`
- `RateLimiter`
- `ThreadFactoryBuilder`
- `MoreExecutors`
- `AtomicLongMap<K>`
- `AtomicDouble`
- `Uninterruptibles`
- `...`

Caching

Google

Guava has a powerful on-heap key→value cache.

```
LoadingCache<Key, Graph> cache = CacheBuilder.newBuilder()
    .maximumSize(50000)
    .expireAfterWrite(33, MINUTES)
    .removalListener(notifyMe)
    .build(
        new CacheLoader<Key, Graph>() {
          public Graph load(Key key) throws AnyException {
            return createExpensiveGraph(key);
          }
        });
  . . .
return cache.getUnchecked(myKey);
```

Unified Hashing API

Google

`Object.hashCode()` is good enough for in-memory hash tables. What about more advanced hashing use cases (fingerprinting, cryptographic, bloom filters...)?

```
HashCode hash = Hashing.murmur3_128().newHasher()
    .putInt(person.getId())
    .putString(person.getFirstName())
    .putBytes(person.getSomeBytes())
    .putObject(person.getPet(), petFunnel)
    .hash(); // asLong(), asBytes(), toString()...
```

You can use murmur3 or JDK-provided algorithms (sha1, crc32, etc.) with one consistent, user-friendly API.

Coda

How to contact us

Google™

Need help with a specific problem?

- Post to Stack Overflow! Use the "guava" tag.

Report a defect, request an enhancement?

- http://code.google.com/p/guava-libraries/issues/entry

Start an email discussion?

- Send to guava-discuss@googlegroups.com

Requesting a new feature

Google™

It is *hard* to sell a new feature to Guava -- even for *me!*
Your best bet: a really well-documented feature request.

Try to search closed issues first, but don't worry (dups happen).

1. What are you trying to do?
2. What's the best code you can write to accomplish that using only today's Guava?
3. What would that same code look like if we added your feature?
4. How do we know this problem comes up often enough in *real life* to belong in Guava?

(Don't rush to code up a patch yet.)

Q & A

Google™

Some FAQs

... in case you don't ask enough questions!

1. Y U NO Apache Commons? Google™

Should you use Guava or Apache Commons?

We may be biased, so consult this question on Stack Overflow:

`http://tinyurl.com/guava-vs-apache`

2. Why is Guava monolithic?

Google™

Why don't we release separate "guava-collections", "guava-io", "guava-primitives", etc.?

For many usages, binary size isn't an issue.

When it is, a split like this *won't actually help!* Most users use a little bit from each package.

Our favorite solution: ProGuard.

A single release is *much* simpler for us *and* you.

3. What about GWT?

Google™

A significant subset is available for use with Google Web Toolkit.

Every class marked `@GwtCompatible`... minus the members marked `@GwtIncompatible`.

Note: We haven't spent much time *optimizing* for GWT.

4. What about Android?

Google™

Everything should work on Android.

Guava 13.0 requires Gingerbread (has been out 2 years).

Targeting Froyo or earlier? There's a backport, `guava-jdk5`.

Note: We haven't spent much time *optimizing* for Android.

5. What is "google-collect"?

Google

Heard of the **Google Collections Library 1.0**?

It's **Guava 0.0**, essentially. It's the *old name* of the project before we had to *rename* it to Guava.

So **please** do not use google-collect.jar! Seek and destroy! Classpath catastrophe will ensue otherwise!

(More history: if you see a funny version like "guava-r07", it actually means Guava 7.0. Very sorry!)

6. Where are fold, reduce, etc? Google™

We're not trying to be the end-all be-all of functional programming in Java!

We have Predicates and Functions, filter() and transform(), because we had specific needs for them.

If you code in Java but love the FP ethos, then Java 8 will be the first thing to really make you happy.

Thanks!