An Overview of Guava: Google Core Libraries for Java

Kurt Alfred Kluever (kak@google.com)

Silicon Valley Java Users Group
September 2012

What is Guava?

Google™

Extensive suite of common libraries that we have found the most useful in Google's many Java projects.

These utilities relate to: collections, concurrency, primitives, reflection, comparison, I/O, hashing, networking, strings, math, in-memory caching, in-memory publish/subscribe... and various basic data types.

The "JDK-plus"! *Requires JDK 1.6+

Released to the public under Apache license 2.0.

Why Guava?

Google™

Our goal is for you to write *less* code ...

And for the code you do write to be simpler, cleaner, and more readable.

Guava releases

Google™

About every 3 months, with significant new functionality and fixes.

Posted 13.0.1 on August 3, 2012. (in maven central: `com.google.guava:guava:13.0.1`)

Guava's classes and methods are a mixture of *API-frozen* and *non-API-frozen* (marked with `@Beta`).

All releases are either *major* or *patch* releases; never *minor*.

What is this presentation?

Google™

A very incomplete overview of *some* of the handiest libraries.

And hopefully: *your questions!*

com.google.common.base

Preconditions

Google

Used to validate assumptions at the start of methods or constructors (and fail-fast if they aren't valid)

```
public Car(Engine engine) {
  this.engine = checkNotNull(engine); // NPE
}
public void drive(double speed) {
  checkArgument(speed > 0.0,
      "speed (%s) must be positive", speed); // IAE
  checkState(engine.isRunning(),
      "engine must be running"); // ISE
  ...
}
```

Objects.toStringHelper()

Google™

Makes implementing `Object.toString()` cleaner:

```
return Objects.toStringHelper(this)
    .add("name", name)
    .add("id", userId)
    .add("pet", petName) // petName is @Nullable!
    .omitNullValues()
    .toString();

// "Person{name=Kurt Kluever, id=42}"
```

Or without `.omitNullValues()`:

```
// "Person{name=Kurt Kluever, id=42, pet=null}"
```

Stopwatch

Google

Prefer `Stopwatch` over `System.nanoTime()`

- (and *definitely* over `currentTimeMillis()`!)
- exposes relative timings, not absolute time
- alternate time sources can be substituted using `Ticker` (`read()` returns nanoseconds)
- `toString()` gives human readable format

```
Stopwatch stopwatch =
    new Stopwatch().start();
doSomeOperation();
long nanos = stopwatch.elapsedTime(
    TimeUnit.NANOSECONDS);
```

CharMatcher

Google

- What's a *matching* character?
  - `WHITESPACE`, `ASCII`, `ANY` (many pre-defined sets)
  - `is('x')`
  - `isNot('_')`
  - `oneOf("aeiou")`
  - `inRange('a', 'z')`
  - Or subclass `CharMatcher`, implement `matches(char)`
- What to do with those matching characters?
  - `matchesAllOf`, `matchesAnyOf`, `matchesNoneOf`
  - `indexIn`, `lastIndexIn`, `countIn`
  - `removeFrom`, `retainFrom`
  - `trimFrom`, `trimLeadingFrom`, `trimTrailingFrom`
  - `collapseFrom`, `trimAndCollapseFrom`, `replaceFrom`

CharMatcher

Google

- Scrubbing a user ID from user input:

```
private static final CharMatcher ID_MATCHER =
    CharMatcher.DIGIT.or(CharMatcher.is
('-'));

...

ID_MATCHER.retainFrom(userInput);
```

String Joining

Google

`Joiner` concatenates strings using a delimiter

- throws a NPE on null objects, unless:
  - `.skipNulls()`
  - `.useForNull(String)`

```
private static final Joiner JOINER =
    Joiner.on(", ").skipNulls();

JOINER.join("Kurt", "Kevin", null, "Chris");
// yields: "Kurt, Kevin, Chris"
```

String Splitting Quiz

Google

Question: What does this return?

```
" foo, ,bar, quux,".split(",");
```

Answer:

a. `[" foo", " ", "bar", " quux"]`
b. `["foo", "bar", "quux"]`
c. `["foo", "", "bar", "quux"]`
d. `[" foo", " ", "bar", " quux", ""]`

String Splitting Quiz

Google

Question: What does this return?

```
" foo, ,bar, quux,".split(",");
```

Answer:

a. `[" foo", " ", "bar", " quux"]`
b. `["foo", "bar", "quux"]`
c. `["foo", "", "bar", "quux"]`
d. `[" foo", " ", "bar", " quux", ""]`

String Splitting Quiz

Google

Question: What does this return?

```
" foo, ,bar, quux,".split(",");
```

Answer:

a. `[" foo", " ", "bar", " quux"]`
b. `["foo", "bar", "quux"]`
c. `["foo", "", "bar", "quux"]`
d. `[" foo", " ", "bar", " quux", ""]`

We *probably* want (b). How do we get it?

```
Splitter.on(',')
    .trimResults()
    .omitEmptyStrings()
    .split(" foo, ,bar, quux,");
```

String Splitting

Google

`Splitter` divides strings into substrings with a delimiter

- A better, more intuitive `String.split()`
  - doesn't silently discard trailing separators
  - handles empty pieces predictably
- By default, assumes nothing about whitespace
  - `.trimResults()`
  - `.omitEmptyStrings()`

```
private static final Splitter SPLITTER =
    Splitter.on(',').trimResults();

SPLITTER.split("Kurt, Kevin, Chris");
// yields: ["Kurt", "Kevin", "Chris"]
```

Optional<T>

Google

- An immutable wrapper that is either:
  - *present*: contains a non-null reference
  - *absent*: contains nothing
  - Note that it never "contains null"
- possible uses:
  - return type (vs. null)
    - "a `T` that must be present"
    - "a `T` that might be absent"
  - distinguish between
    - "unknown" (for example, not present in a map)
    - "known to have no value" (present in the map, with value `Optional.absent()`)
  - wrap nullable references for storage in a collection that does not support null

Optional<T>

Google

- Creating an `Optional<T>`

```
Optional.of(notNull);
Optional.absent();
Optional.fromNullable(maybeNull);
```

- Unwrapping an `Optional<T>`

```
mediaType.charset().get(); // maybe ISE
mediaType.charset().or(Charsets.UTF_8);
mediaType.charset().orNull();
```

- Other useful methods:

```
mediaType.charset().asSet(); // 0 or 1
mediaType.charset().transform(stringFunc);
```

Functional Programming

Google

`Function<F, T>`

- one way transformation of `F` into `T`
- `T apply(F input)`
- most common use: *transforming* collections (view)

`Predicate<T>`

- determines `true` or `false` for a given `T`
- `boolean apply(T input)`
- most common use: *filtering* collections (view)

com.google.common.collect

FP Example

Google

```
private static final Predicate<Client> ACTIVE_CLIENT =
    new Predicate<Client>() {
      public boolean apply(Client client) {
        return client.activeInLastMonth();
      }
    };

// Returns an immutable list of the names of
// the first 10 active clients in the database.

FluentIterable.from(database.getClientList())
    .filter(ACTIVE_CLIENT)
    .transform(Functions.toStringFunction())
    .limit(10)
    .toList();
```

FluentIterable API

Google™

- Chaining (returns `FluentIterable`)
  - `skip`
  - `limit`
  - `cycle`
  - `filter`, `transform`
- Querying (returns `boolean`)
  - `allMatch`, `anyMatch`
  - `contains`, `isEmpty`
- Converting
  - `to{List, Set, SortedSet}`
  - `toArray`
- Extracting
  - `first`, `last`, `firstMatch` (returns `Optional<E>`)
  - `get` (returns `E`)

Functional Programming

Google™

```
// with functional programming...be careful!
Function<String, Integer> lengthFunction =
    new Function<String, Integer>() {
      public Integer apply(String string) {
        return string.length();
      }
    };
Predicate<String> allCaps = new Predicate<String>() {
  public boolean apply(String string) {
    return CharMatcher.JAVA_UPPER_CASE
        .matchesAllOf(string);
  }
};
Multiset<Integer> lengths = HashMultiset.create(
    Iterables.transform(
        Iterables.filter(strings, allCaps),
        lengthFunction));
```

Functional Programming

Google

```
// without functional programming
Multiset<Integer> lengths = HashMultiset.create();
for (String string : strings) {
  if (CharMatcher.JAVA_UPPER_CASE.matchesAllOf(string)) {
    lengths.add(string.length());
  }
}
```

Multiset<E>

Google

- often called a *bag*
- add multiple instances of a given element
- counts how many occurrences exist

- *similar* to a `Map<E, Integer>`, but...
  - only positive counts
  - `size()` returns total # of items, not # keys
  - `count()` for a non-existent key is 0
  - `iterator()` goes over each element in the Multiset
    - `elementSet().iterator()` unique elements

- *similar* to our `AtomicLongMap<E>`, which is like a `Map<E, AtomicLong>`

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)

- *similar* to a `Map<K, Collection<V>>`, but...
  - `get()` never returns null (returns an empty collection)
  - `containsKey()` is true only if 1 or more values exist
  - `entries()` returns all entries for all keys
  - `size()` returns total number of entries, not keys
  - `asMap()` to view it as a `Map<K, Collection<V>>`

- Almost always want variable type to be either `ListMultimap` or `SetMultimap` (and not `Multimap`)

BiMap<K1, K2>

Google™

- bi-directional map
- both keys *and* values are unique
- can view the inverse map with `inverse()`
- use instead of maintaining two separate maps
  - `Map<K1, K2>`
  - `Map<K2, K1>`

Table<R, C, V>

Google

A "two-tier" map, or a map with two keys (called the "row key" and "column key").

- can be sparse or dense
  - `HashBasedTable`: uses hash maps (sparse)
  - `TreeBasedTable`: uses tree maps (sparse)
  - `ArrayTable`: uses `V[][]` (dense)
- many views on the underlying data are possible
  - row or column map (of maps)
  - row or column key set
  - set of all cells (as `<R, C, V>` entries)
- use instead of `Map<R, Map<C, V>>`

Immutable Collections

Google

- offered for all collection types (including JDK ones)
- inherently thread-safe
- elements are never null
- specified iteration order
- reduced memory footprint
- slightly improved performance
- unlike `Collections.unmodifiableXXX`, they
  - perform a copy (not a view / wrapper)
  - type conveys immutability

Always prefer immutability!

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

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();
      }
    };
```

Short-circuits, never allocates, is fast. Also has `compare(T, T, Comparator<T>)`, `compareFalseFirst`, `compareTrueFirst`

Ordering example

Google

Here's another:

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

private Function<String, Integer> tzToOffset() {
  return 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`, like `FluentIterable`.)

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

- `Ordering.natural()`
- `new Ordering() { ... }`
- `Ordering.from(existingComparator);`
- `Ordering.explicit("alpha", "delta", "beta");`

... 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)`

Some are even optimized for the specific kind of comparator you have.

... 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, you should still extend Ordering.

com.google.common.hash

Why a new hashing API?

Google™

Think of `Object.hashCode()` as "only good enough for in-memory hash maps." But:

- Strictly limited to 32 bits
- Worse, composed hash codes are "collared" down to 32 bits during the computation
- No separation between "which data to hash" and "which algorithm to hash it with"
- Implementations have very bad bit dispersion

These make it not very useful for a multitude of hashing applications: a document "fingerprint", cryptographic hashing, cuckoo hashing, Bloom filters...

JDK solution

Google

To address this shortcoming, the JDK introduced ~~an interface~~ *two* interfaces:

- `java.security.MessageDigest`
- `java.util.zip.Checksum`

Each named after a specific *use case* for hashing.

Worse than the split: neither is remotely easy to use when you aren't hashing raw byte arrays.

Guava Hashing Example

Google™

```
HashCode hash =
    Hashing.murmur3_128().newHasher()
        .putInt(person.getAge())
        .putLong(person.getId())
        .putString(person.getFirstName())
        .putString(person.getLastName())
        .putBytes(person.getSomeBytes())
        .putObject(person.getPet(), petFunnel)
        .hash();
```

`HashCode` has `asLong()`, `asBytes()`, `toString()`...

Or put it into a Set, return it from an API, etc.

(It also implements `equals()` in a secure way.)

Hashing Overview

Google™

The `com.google.common.hash` API offers:

- A unified, user-friendly API for all hash functions
- Seedable 32- and 128-bit implementations of murmur3
- `md5()`, `sha1()`, `sha256()`, `sha512()` adapters
  - change only one line of code to switch between these and murmur etc.
- `goodFastHash(int bits)`, for when you don't care what algorithm you use and aren't persisting hash codes
- General utilities for `HashCode` instances, like `combineOrdered` / `combineUnordered`

BloomFilter

Google™

A *probabilistic set*.

- `public boolean mightContain(T);`
  - `true` == *"probably there"*
  - `false` == *"definitely not there"*

WHY? Consider a spell-checker:

- If `syzygy` gets red-underlined, that's annoying
- But if `yarrzcw` *doesn't* get red-underlined... oh well!
- And the memory savings can be very large

Primary use: short-circuiting an expensive boolean query

com.google.common.cache

Caching

Google™

Guava has a powerful on-heap cache (key -> value)

```
LoadingCache<Key, Graph> cache = CacheBuilder.
newBuilder()
     .maximumSize(10000)
     .expireAfterWrite(10, TimeUnit.MINUTES)
     .removalListener(MY_LISTENER)
     .build(
         new CacheLoader<Key, Graph>() {
           public Graph load(Key key) throws AnyException
{
             return createExpensiveGraph(key);
           }
         });
```

Caching really deserves it's own hour long talk.

Search for "Java Caching with Guava"

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.

To save time, search closed issues first. Then file.

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.)

How to help

Google

- *Use* it and evangelize it to your team!
- High-quality issue reports! (as just described)
- High-quality comments/votes on existing issue reports!
- Suggest improvements to the GuavaExplained wiki
- Blog posts, videos, tweets, etc. -- we love these!
- Volunteer to fix an "Accepted" issue (procedure: `http://code.google.com/p/guava-libraries/wiki/HowToContribute`)

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`

The large number of upvotes for the top answers shows a pretty strong community consensus.

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!*

Our favorite solution: ProGuard.

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

3. What's the deal with `@Beta`? Google™

`@Beta` can appear on a method or a class (in which case it implicitly applies to all methods as well).

It simply means "not API-frozen". We might rename or otherwise change the API in ways that may break compilation. In *extreme* cases APIs can be removed without replacement.

If you are a *library*, beta APIs can cause your users major headaches! Request that an API be frozen, then either *avoid or repackage* (e.g. jarjar, proguard). Check out the findbugs plugin "library-detectors" by Overstock.com!

4. What about deprecation?

Google™

Even a non-beta API can be removed if it's been deprecated for **18 months** (typically *six* releases) first!

A beta API can be removed with no deprecation, but we do always *try* to deprecate it for one release first.

So, when upgrading from 9.0 to 12.0, it may be easier to single-step it.

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. 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.

7. What about Android?

Google™

Everything should work on Android.

Guava 12.0 requires Gingerbread (has been out 16 months).

Targeting Froyo or earlier? Stick to Guava 11.0.2. (backport should be out soon)

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

8. 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!

Who is Guava?

Google™

The Java Core Libraries Team @ Google:

- Kevin Bourrillion
- Chris Povirk
- Kurt Alfred Kluever
- Gregory Kick
- Colin Decker
- Christian Gruber
- Louis Wasserman

+ Lots of 20% contributors and consultants

Josh Bloch, Martin Buchholz, Jesse Wilson, Jeremy Manson, Crazy Bob Lee, Doug Lea, etc.

+ The open source community!