

Using ProGuard with Guava

Guava is a fairly sizable JAR file, and your app may be using only a small portion of it. If you'd like an easy way to include just the parts of Guava you really need in your own application, we recommend you look into [ProGuard](#).

Here's a Proguard configuration for Guava:

```
-dontwarn javax.lang.model.element.Modifier

# Note: We intentionally don't add the flags we'd need to make Enums work.
# That's because the Proguard configuration required to make it work on
# optimized code would preclude lots of optimization, like converting enums
# into ints.

# Throwables uses internal APIs for lazy stack trace resolution
-dontnote sun.misc.SharedSecrets
-keep class sun.misc.SharedSecrets {
    *** getJavaLangAccess(...);
}
-dontnote sun.misc.JavaLangAccess
-keep class sun.misc.JavaLangAccess {
    *** getStackTraceElement(...);
    *** getStackTraceDepth(...);
}

# FinalizableReferenceQueue calls this reflectively
# Proguard is intelligent enough to spot the use of reflection onto this, so we
# only need to keep the names, and allow it to be stripped out if
# FinalizableReferenceQueue is unused.
-keepnames class com.google.common.base.internal.Finalizer {
    *** startFinalizer(...);
}
# However, it cannot "spot" that this method needs to be kept IF the class is.
-keepclassmembers class com.google.common.base.internal.Finalizer {
    *** startFinalizer(...);
}
-keepnames class com.google.common.base.FinalizableReference {
    void finalizeReferent();
}
-keepclassmembers class com.google.common.base.FinalizableReference {
    void finalizeReferent();
}

# Striped64, LittleEndianByteArray, UnsignedBytes, AbstractFuture
-dontwarn sun.misc.Unsafe

# Striped64 appears to make some assumptions about object layout that
# really might not be safe. This should be investigated.
-keepclassmembers class com.google.common.cache.Striped64 {
    *** base;
    *** busy;
```

```

}
-keepclassmembers class com.google.common.cache.Striped64$Cell {
    <fields>;
}

-dontwarn java.lang.SafeVarargs

-keep class java.lang.Throwable {
    *** addSuppressed(...);
}

# Futures.getChecked, in both of its variants, is incompatible with proguard.

# Used by AtomicReferenceFieldUpdater and sun.misc.Unsafe
-keepclassmembers class com.google.common.util.concurrent.AbstractFuture** {
    *** waiters;
    *** value;
    *** listeners;
    *** thread;
    *** next;
}
-keepclassmembers class com.google.common.util.concurrent.AtomicDouble {
    *** value;
}
-keepclassmembers class com.google.common.util.concurrent.AggregateFutureState {
    *** remaining;
    *** seenExceptions;
}

# Since Unsafe is using the field offsets of these inner classes, we don't want
# to have class merging or similar tricks applied to these classes and their
# fields. It's safe to allow obfuscation, since the by-name references are
# already preserved in the -keep statement above.
-keep,allowshrinking,allowobfuscation class
com.google.common.util.concurrent.AbstractFuture** {
    <fields>;
}

# Futures.getChecked (which often won't work with Proguard anyway) uses this. It
# has a fallback, but again, don't use Futures.getChecked on Android regardless.
-dontwarn java.lang.ClassValue

# MoreExecutors references AppEngine
-dontnote com.google.appengine.api.ThreadManager
-keep class com.google.appengine.api.ThreadManager {
    static *** currentRequestThreadFactory(...);
}
-dontnote com.google.apphosting.api.ApiProxy
-keep class com.google.apphosting.api.ApiProxy {
    static *** getCurrentEnvironment (...);
}

```

We would like to hear about your experiences using ProGuard with Guava so we can improve this page.