To: core-libs-dev@openjdk.java.net, concurrency-interest@cs.oswego.edu
Subject: Durations in existing JDK APIs

Hi core-libs-dev and concurrency-interest,

Recently we've been closely studying date/time code inside of Google, and found a surprising number of places with time unit mismatches due to overuse of primitives to represent date/time concepts. And unlike "off-by-one" errors, these are often "off-by-1000x" or worse...yikes! For internal APIs, we're strongly encouraging folks to use the appropriate `java.time` types (i.e., `Duration` or `Instant`) because this greatly reduces the occurrence of these problems.

There are several improvements in the JDK we'd like to propose:

1. **Rename ALL existing unitless primitive method parameters to include their time unit.** At Google, we have static analysis tools to detect unit mismatches that work based on method names, variable names, etc. However, when methods and method parameters are unitless, the only way to know what correct units are is to read and understand the javadocs (which our tools obviously can't do). Many IDEs also show method signatures with parameter names only. There are a handful of examples in the JDK that we'd like to update. For example, `Object.wait(long timeout)` would become `Object.wait(long timeoutMillis)` (or similar). An incomplete list of these APIs is included below [1].
2. **Add a `java.time` overload to *some* APIs that currently represent date/time concepts using primitives.** While static analysis helps find errors, ideally people wouldn't have to decompose their `Duration` instances to call these APIs. Adding a `Duration` overload of each of these APIs would remove the need to decompose durations, and would encourage developers to plumb durations through more layers of their application. The old primitive-accepting APIs would be softly discouraged. Note that new default implementations will have to delegate to the existing overloads, and will have to choose between losing precision or capping large values at e.g. 292 years (for `long nanos`), but it is hard to imagine this being a serious problem in practice for any of them.
    a. Note: it's probably not worth adding `Duration` overloads to *legacy* APIs (e.g., `java.util.Timer`) or low-level APIs (e.g., `java.lang.Object.wait()`). Determining which APIs make the cut is certainly open to discussion.
3. **Add a `java.time` overload to *most* APIs that currently accept a `<long, TimeUnit>` pair.** Prior to Java8, the recommended advice for accepting a logical duration was to use a `<long, TimeUnit>` pair, as most of `java.util.concurrent` currently does. Similarly, we've seen unit mismatch bugs with these APIs as well (e.g., `future.get(timeout.toNanos(), MILLISECONDS)`). Adding a `Duration` overload for each of these APIs would remove the need to decompose durations, and would encourage broader `Duration` adoption. The old APIs would be softly discouraged.

    a. Note: a few of the `<long, TimeUnit>` APIs are *already* overloaded, which would make this a 2x2 explosion of methods. Whether that sort of API explosion is acceptable is certainly open for discussion.
    b. Note: We've already started adding these overloads in Guava and have plans to do so across the board. Caffeine has also added `Duration` overloads.
4. **Add APIs to convert between `TimeUnit` and `Duration`.** As users transition to the new `Duration`-centric world, these APIs will come in handy. They will also be necessary for overload-implementers. We're proposing `Duration.of(long, TimeUnit)` and `TimeUnit.convert(Duration)`.

These recommendations, of course, should also apply to new APIs added in the future. Note that we are not expressing an opinion at this time on whether new APIs should or shouldn't *also* have a `<long, TimeUnit>` overload.

Of course, even comprehensive adoption of `Duration` in a codebase will not eliminate every possible source of unit mismatch bugs. But it would confine them to only the places where `Duration`s are created. This would reduce the risk in itself, but also makes it much easier for static analyses to detect any remaining bugs. It also lowers the cognitive burden faced by every developer interacting with logical durations.

We realize that these are significant changes, so we'd love to hear your thoughts. We'd also be happy to work with Martin Buchholz to make these changes.

Thanks,

-Kurt Alfred Kluever
(on behalf of the Java Core Libraries Team @ Google)

Appendix

[1] Unitless primitive parameters

- Probably worth adding a `Duration` overload
    - `java.lang.Thread.sleep(long millis)`
    - `java.lang.Thread.sleep(long millis, long nanos)`
        - Note: this API is particularly weird since it accepts millis and nanos!
    - `java.lang.Thread.join(long millis)`
    - `java.lang.Thread.join(long millis, long nanos)`
        - Note: this API is particularly weird since it accepts millis and nanos!
    - `java.nio.channel.Selector.select(long timeout)`
- Probably too low level to worry about adding a `java.time` overloads
    - `java.lang.Object.wait(long timeout)`
    - `java.lang.Object.wait(long timeout, long nanos)`

        - Note: this API is particularly weird since it accepts millis and nanos!
    - `java.lang.ReferenceQueue.remove(long timeout)`
    - `java.util.concurrent.locks.LockSupport.parkUntil(long deadline)`
        - Note: this parameter represents milliseconds since epoch (an `Instant`)
    - `java.util.concurrent.locks.LockSupport.parkUntil(Object blocker, long deadline)`
        - Note: this parameter represents milliseconds since epoch (an `Instant`)
    - `java.util.concurrent.locks.LockSupport.parkNanos(long nanos)`
    - `java.util.concurrent.locks.LockSupport.parkNanos(Object blocker, long nanos)`
    - `java.util.concurrent.locks.AbstractQueuedSynchronizer.ConditionObject.awaitUntil(Date)`
        - Note: this would be overloaded with an `Instant`
    - `java.util.logging.LogRecord.setMillis(long millis)`
- "Legacy" APIs that are probably not worth adding a `java.time` overloads
    - `java.util.Timer.schedule(TimerTask task, Date firstTime, long period)`
    - `java.util.Timer.schedule(TimerTask task, long delay)`
    - `java.util.Timer.schedule(TimerTask task, Date time)`
    - `java.util.Timer.schedule(TimerTask task, long delay, long period)`
    - `java.util.Timer.scheduleAtFixedRate(TimerTask task, Date firstTime, long period)`
    - `java.util.Timer.scheduleAtFixedRate(TimerTask task, long delay, long period)`
    - `java.sql.Connection.isValid(int timeout)`
        - Note: this parameter is in *seconds*
    - `java.sql.DriverManager.setLoginTimeout(int seconds)`
        - Note: this parameter is in *seconds*
- Networking APIs (perhaps not worth adding a `Duration` overload?)
    - `java.net.InetAddress.isReachable(int timeout)`
    - `java.net.InetAddress.isReachable(NetworkInterface netif, int ttl, int timeout)`
    - `java.net.URLConnection.setConnectTimeout(int timeout)`
    - `java.net.URLConnection.setReadTimeout(int timeout)`
    - `java.net.Socket.setSoTimeout(int timeout)`
    - `java.net.Socket.connect(SocketAddress endpoint, int timeout)`
    - `java.net.ServerSocket.setSoTimeout(int timeout)`
    - `java.net.DatagramSocket.setSoTimeout(int timeout)`