Background on Concurrency & Parallelism in Java (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand the meaning of the terms concurrency & parallelism
- Be aware of the history of Java concurrency & parallelism

JAVA HISTORY

• Understand the meaning of the terms concurrency & parallelism
• Be aware of the history of Java concurrency & parallelism

Applications

Additional Frameworks & Languages

Threading & Synchronization Packages

Java Execution Environment (e.g., JVM)

System Libraries

Operating System Kernel

Java/JNI

C++/C

C
Learning Objectives in this Part of the Lesson

- Understand the meaning of the terms concurrency & parallelism
- Be aware of the history of Java concurrency & parallelism

Additional Frameworks & Languages

Applications

Threading & Synchronization Packages

Java Execution Environment (e.g., JVM)

System Libraries

Operating System Kernel

C++/C

Java/JNI

C

Hopefully, you’ll already know some of this!!!
Learning Objectives in this Part of the Lesson

• Understand the meaning of the terms concurrency & parallelism
• Be aware of the history of Java concurrency & parallelism
• Know which Java mechanism(s) to understand & apply
A Brief History of Concurrency & Parallelism in Java
A Brief History of Concurrency & Parallelism in Java

• Foundational concurrency support

- e.g., Java threads & built-in monitor objects available in Java 1.0
A Brief History of Concurrency & Parallelism in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

See docs.oracle.com/javase/tutorial/essential/concurrency
A Brief History of Concurrency & Parallelism in Java

- Foundational concurrency support
- Focus on basic multi-threading & synchronization primitives

```java
SimpleBlockingBoundedQueue<Integer> simpleQueue = new SimpleBlockingBoundedQueue<>();

Thread[] threads = new Thread[] {
    new Thread(new Producer<>(simpleQueue)),
    new Thread(new Consumer<>(simpleQueue))
};

for (Thread thread : threads) thread.start();

for (Thread thread : threads) thread.join();

See github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue
```

Allow multiple threads to communicate via a bounded buffer.
A Brief History of Concurrency & Parallelism in Java

- Foundational concurrency support
- Focus on basic multi-threading & synchronization primitives

```java
SimpleBlockingBoundedQueue<Integer> simpleQueue = new SimpleBlockingBoundedQueue<>();

Thread[] threads = new Thread[] {
    new Thread(new Producer<> (simpleQueue)),
    new Thread(new Consumer<> (simpleQueue))
};

for (Thread thread : threads) thread.start();

for (Thread thread : threads) thread.join();
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue)
A Brief History of Concurrency & Parallelism in Java

- Foundational concurrency support
- Focus on basic multi-threading & synchronization primitives

```java
class SimpleBlockingBoundedQueue<E> {
    public E take() {...
        synchronized(this) {
            while (mList.isEmpty())
                wait();
            notifyAll();
            return mList.poll();
        }
    }
}
```

See [github.com/douglasraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue]
A Brief History of Concurrency & Parallelism in Java

• Foundational concurrency support
  • Focus on basic multi-threading & synchronization primitives
  • Efficient, but low-level & very limited in capabilities
A Brief History of Concurrency & Parallelism in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives
- Efficient, but low-level & very limited in capabilities
  - Many accidental complexities

See en.wikipedia.org/wiki/No_Silver_Bullet

Accidental complexities arise from limitations with software techniques, tools, & methods
A Brief History of Concurrency & Parallelism in Java

- Advanced concurrency support

- e.g., Java executor framework, synchronizers, blocking queues, atomics, & concurrent collections available in Java 1.5+
A Brief History of Concurrency & Parallelism in Java

- Advanced concurrency support
- Focus on course-grained “task parallelism” whose computations can run concurrently

See en.wikipedia.org/wiki/Task_parallelism
A Brief History of Concurrency & Parallelism in Java

- Advanced concurrency support
- Focus on course-grained “task parallelism” whose computations can run concurrently

Create a fixed-sized thread pool & also coordinate the starting & stopping of multiple tasks that acquire/release shared resources

```
ExecutorService executor = Executors.newFixedThreadPool
                         (numOfBeings,
                          mThreadFactory);
...
CyclicBarrier entryBarrier =
   new CyclicBarrier(numOfBeings+1);
CountDownLatch exitBarrier =
   new CountDownLatch(numOfBeings);
for (int i=0; i < beingCount; ++i)
   executor.execute
      (makeBeingRunnable(i,
                       entryBarrier,
                       exitBarrier));
```

See [github.com/douglas craigschmidt/LiveLessons/tree/master/PalantiriManagerApplication](https://github.com/douglasfraicschmidt/LiveLessons/tree/master/PalantiriManagerApplication)
A Brief History of Concurrency & Parallelism in Java

• Advanced concurrency support
  • Focus on course-grained “task parallelism” whose computations can run concurrently
• Feature-rich & optimized, but also tedious & error-prone to program
A Brief History of Concurrency & Parallelism in Java

• Foundational parallelism support

- Operating System Kernel
- System Libraries
- Java Execution Environment (e.g., JVM)
- Threading & Synchronization Packages
- Additional Frameworks & Languages
- Applications

e.g., Java fork-join pool available in Java 1.7
A Brief History of Concurrency & Parallelism in Java

- Foundational parallelism support
- Focus on data parallelism that runs the same task on different data elements

See [en.wikipedia.org/wiki/Data_parallelism](en.wikipedia.org/wiki/Data_parallelism)
A Brief History of Concurrency & Parallelism in Java

- Foundational parallelism support
- Focus on data parallelism that runs the same task on different data elements

Use a common fork-join pool to search input strings to locate phrases that match

List<List<SearchResults>>
listOfListOfSearchResults = ForkJoinPool
    .commonPool()
    .invoke(new
    SearchWithForkJoinTask
    (inputList,
     mPhrasesToFind, ...));

See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchForkJoin
A Brief History of Concurrency & Parallelism in Java

- Foundational parallelism support
  - Focus on data parallelism that runs the same task on different data elements
- Powerful & scalable, but tedious to program directly
A Brief History of Concurrency & Parallelism in Java

- Advanced parallelism support

*e.g., Java parallel streams & completable futures available in Java 1.8*
A Brief History of Concurrency & Parallelism in Java

- Advanced parallelism support
- Focus on functional programming for data parallelism

See [en.wikipedia.org/wiki/Data_parallelism](en.wikipedia.org/wiki/Data_parallelism)
A Brief History of Concurrency & Parallelism in Java

- Advanced parallelism support
- Focus on functional programming for data parallelism & reactive asynchrony

See gist.github.com/staltz/868e7e9bc2a7b8c1f754
A Brief History of Concurrency & Parallelism in Java

- Advanced parallelism support
- Focus on functional programming for *data parallelism* & reactive asynchrony

```java
List<Image> images = urls
    .parallelStream()
    .filter(not(this::urlCached))
    .map(this::downloadImage)
    .flatMap(this::applyFilters)
    .collect(toList());
```

*Synchronously download images that aren’t already cached from a list of URLs & process/store the images in parallel*

A Brief History of Concurrency & Parallelism in Java

- Advanced parallelism support
- Focus on functional programming for data parallelism & reactive asynchrony

```java
CompletableFuture<Stream<Image>> resultsFuture = urls
    .stream()
    .map(this::checkUrlCachedAsync)
    .map(this::downloadImageAsync)
    .flatMap(this::applyFiltersAsync)
    .collect(toFuture())
    .thenApply(stream ->
        log(stream.flatMap(Optional::stream), urls.size())
    )
    .join();
```

Asynchronously download images that aren’t already cached from a list of URLs & process/store the images in parallel

A Brief History of Concurrency & Parallelism in Java

- Advanced parallelism support
  - Focus on functional programming for data parallelism & reactive asynchrony
- Strikes an effective balance between productivity & performance
A Brief History of Concurrency & Parallelism in Java

- Advanced parallelism support
  - Focus on functional programming for data parallelism & reactive asynchrony
- Strikes an effective balance between productivity & performance
- However, may be overly prescriptive
Which Java Mechanism(s) to Understand & Apply
Which Java Mechanism(s) to Understand & Apply

- Java’s concurrency & parallelism mechanisms span multiple layers in the software stack.
Which Java Mechanism(s) to Understand & Apply

- Java’s concurrency & parallelism mechanisms span multiple layers in the software stack

- Choosing best mechanism(s) depend on various factors
Which Java Mechanism(s) to Understand & Apply

- Developers of low-level classes & performance-sensitive apps may prefer shared object mechanisms

```
Package java.util.concurrent
Description
Utility classes commonly useful in concurrent programming. This package includes a few small standardized extensible frameworks, as well as some classes that provide useful functionality and are otherwise tedious or difficult to implement. Here are brief descriptions of the main components. See also the java.util.concurrent.locks and java.util.concurrent.atomic packages.
```

- e.g., java.util.concurrent as per [www.youtube.com/watch?v=sq0MX3fHkro](https://www.youtube.com/watch?v=sq0MX3fHkro)
Which Java Mechanism(s) to Understand & Apply

- Developers of low-level classes & performance-sensitive apps may prefer shared object mechanisms
  - **Pros**: Efficient & lightweight
  - **Cons**: Tedious & error-prone
Which Java Mechanism(s) to Understand & Apply

• Framework developers may want to use the Java message passing mechanisms

e.g., Android AsyncTask/HaMeR frameworks or Java ExecutorCompetitionService
Which Java Mechanism(s) to Understand & Apply

- Framework developers may want to use the Java message passing mechanisms
  - **Pros**: Flexible & decoupled
  - **Cons**: Time/space overhead
Which Java Mechanism(s) to Understand & Apply

- Mobile app developers may want to program with higher-level frameworks.

  e.g., Java 8 parallel streams & completable futures, RxJava, etc.
### Which Java Mechanism(s) to Understand & Apply

- Mobile app developers may want to program with higher-level frameworks
  - **Pros**: Productivity & robustness
  - **Cons**: Time/space overhead & overly prescriptive

### Additional Application Frameworks

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Application Frameworks</td>
</tr>
<tr>
<td>Concurrency/Parallelism Frameworks</td>
</tr>
<tr>
<td>Java Threads &amp; Synchronizers</td>
</tr>
<tr>
<td>Execution Environment (JVM, Dalvik/ART, etc.)</td>
</tr>
<tr>
<td>System Libraries</td>
</tr>
<tr>
<td>Operating System Kernel</td>
</tr>
</tbody>
</table>
“Full stack” developers should understand concepts & mechanisms at each layer.
End of Background on Java Concurrency & Parallelism (Part 2)