Background on Concurrency & Parallelism in Java (Part 1)

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
An Overview of Concurrency
An Overview of Concurrency

- Concurrency is a form of computing where threads can run simultaneously.

See en.wikipedia.org/wiki/Concurrency_(computer_science)
An Overview of Concurrency

- Concurrency is a form of computing where threads can run simultaneously

```java
new Thread(() ->
    someComputations());
```

A Java threads are units of execution for instruction streams that can run concurrently on processor cores

See [docs.oracle.com/javase/tutorial/essential/concurrency/threads.html](docs.oracle.com/javase/tutorial/essential/concurrency/threads.html)
An Overview of Concurrency

- Concurrency is a form of computing where threads can run simultaneously.
- Often used to offload work from the user interface (UI) thread to background thread(s).

Concurrency is a form of computing where threads can run simultaneously.

Often used to offload work from the user interface (UI) thread to background thread(s), e.g.

- Background thread(s) can block
- The UI thread does not block

See developer.android.com/training/multiple-threads/communicate-ui.html
An Overview of Concurrency

- Concurrent Java threads interact via shared objects and/or message passing

See docs.oracle.com/javase/8/docs/api/?java/util/concurrent/package-summary.html
An Overview of Concurrency

- Concurrent Java threads interact via shared objects and/or message passing
  
- **Shared objects**
  - Synchronize concurrent operations on objects so object state remains coherent after each operation

See tutorials.jenkov.com/java-concurrency/thread-safety.html
An Overview of Concurrency

- Concurrent Java threads interact via shared objects and/or message passing

- **Shared objects**
  - Synchronize concurrent operations on objects so object state remains coherent after each operation
  - Examples of Java synchronizers:
    - Synchronized statements/methods
    - Reentrant locks & intrinsic locks
    - Atomic operations
    - Semaphores & condition objects
    - “Compare-and-swap” (CAS) operations in sun.misc.unsafe

See [dzone.com/articles/the-java-synchronizers](http://dzone.com/articles/the-java-synchronizers)
An Overview of Concurrency

• Concurrent Java threads interact via shared objects and/or message passing

• Shared objects

• Message passing
  • Send message(s) from producer thread(s) to consumer thread(s) via a thread-safe queue
An Overview of Concurrency

- Concurrent Java threads interact via shared objects and/or message passing
  
  **Shared objects**
  
  **Message passing**
  
  - Send message(s) from producer thread(s) to consumer thread(s) via a thread-safe queue
  
  - Examples of Java thread-safe queues
    - Array & linked blocking queues
    - Priority blocking queue
    - Synchronous queue
    - Concurrent linked queue

See [docs.oracle.com/javase/tutorial/collections/implementations/queue.html](docs.oracle.com/javase/tutorial/collections/implementations/queue.html)
An Overview of Concurrency

- Key goals of using shared objects and/or message passing are to share resources safely/efficiently & avoid hazards

See [en.wikipedia.org/wiki/Thread_safety](en.wikipedia.org/wiki/Thread_safety)
An Overview of Concurrency

- Key goals of using shared objects and/or message passing are to share resources safely/efficiently & avoid hazards, e.g.
  - Race conditions
    - Race conditions occur when a program depends upon the sequence or timing of threads for it to operate properly

See en.wikipedia.org/wiki/Race_condition#Software
An Overview of Concurrency

- Key goals of using shared objects and/or message passing are to share resources safely/efficiently & avoid hazards, e.g.
  - Race conditions
    - Race conditions occur when a program depends upon the sequence or timing of threads for it to operate properly

This test program induces race conditions due to lack of synchronization between producer & consumer threads accessing a bounded queue

See [github.com/douglascraigschmidt/LiveLessons/tree/master/BuggyQueue](github.com/douglascraigschmidt/LiveLessons/tree/master/BuggyQueue)
An Overview of Concurrency

- Key goals of using shared objects and/or message passing are to share resources safely/efficiently & avoid hazards, e.g.
  - Race conditions
  - Memory inconsistencies
    - These errors occur when different threads have inconsistent views of what should be the same data

An Overview of Concurrency

- Key goals of using shared objects and/or message passing are to share resources safely/efficiently & avoid hazards, e.g.
  - Race conditions
  - Memory inconsistencies
  - Deadlocks
    - Occur when two or more competing actions are each waiting for the other to finish, & thus none ever do

See en.wikipedia.org/wiki/Deadlock
An Overview of Parallelism
An Overview of Parallelism

- Parallelism is a form of computing that performs several steps on multiple processor cores.

See en.wikipedia.org/wiki/Parallel_computing
An Overview of Parallelism

- Parallelism is a form of computing that performs several steps on multiple processor cores, i.e.
- Partitions tasks into sub-tasks
Parallelism is a form of computing that performs several steps on multiple processor cores, i.e.

- Partitions tasks into sub-tasks
- Run independent sub-tasks in parallel

An Overview of Parallelism
Parallelism is a form of computing that performs several steps on multiple processor cores, i.e.

- Partitions tasks into sub-tasks
- Run independent sub-tasks in parallel
- Combines sub-results to form one final result
A key goal of parallelism is to *efficiently* partition tasks into sub-tasks & combine results.
An Overview of Parallelism

- A key goal of parallelism is to efficiently partition tasks into sub-tasks & combine results
- Parallelism thus focuses on optimizing performance
  - e.g., throughput, scalability, & latency

A key goal of parallelism is to efficiently partition tasks into sub-tasks & combine results.

Parallelism thus focuses on optimizing performance.

Parallelism works best when threads share no mutable state & don’t block.

See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html
An Overview of Parallelism

- A key goal of parallelism is to efficiently partition tasks into sub-tasks & combine results
- Parallelism thus focuses on optimizing performance
- Parallelism works best when threads share no mutable state & don’t block
  - Hence Java 8’s emphasis on “fork-join” & “work-stealing”

An Overview of Parallelism

• Brian Goetz has an excellent talk about the evolution of Java from concurrent to parallel computing

See www.youtube.com/watch?v=NsDE7E8sldQ
An Overview of Parallelism

- Brian Goetz has an excellent talk about the evolution of Java from concurrent to parallel computing.

His talk emphasizes that Java 8 combines functional programming with fine-grained data parallelism to leverage many-core processors.

See [www.infoq.com/presentations/parallel-java-se-8](http://www.infoq.com/presentations/parallel-java-se-8)
End of Background on Java Concurrency & Parallelism (Part 1)
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
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

Topics:

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

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

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

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

```
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/douglasraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue](https://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

```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 {
    public E take() {
        synchronized(this) {
            while (mList.isEmpty())
                wait();
            notifyAll();
            return mList.poll();
        }
    }
}
```

Built-in monitor object mutual exclusion & coordination primitives

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
  • 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

```java
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));
```

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

See [github.com/douglascraigschmidt/LiveLessons/tree/master/PalantiriManagerApplication](https://github.com/douglascraigschmidt/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

![Diagram showing the hierarchy of Java environments and frameworks](image)

- 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
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/douglasraigschmidt/LiveLessons/tree/master/SearchForkJoin](https://github.com/douglasraigschmidt/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

Additional Frameworks & Languages

Threading & Synchronization Packages

Java Execution Environment (e.g., JVM)

System Libraries

Operating System Kernel

*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

**Parallel Streams**

- filter(not(this::urlCached))
- map(this::downloadImage)
- flatMap(this::applyFilters)
- collect(toList())
A Brief History of Concurrency & Parallelism in Java

• Advanced parallelism support
• Focus on functional programming for data parallelism & asynchrony
A Brief History of Concurrency & Parallelism in Java

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

List<Image> images = urls
    .parallelStream()
    .filter(not(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

See github.com/douglasraigschmidt/LiveLessons/tree/master/ImageStreamGang
Asynchronously download images that aren’t already cached from a list of URLs & process/store the images in parallel

```java
collect(toFuture());
```
A Brief History of Concurrency & Parallelism in Java

• Advanced parallelism support
  • Focus on functional programming for data parallelism & 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 & 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

Operating System Kernel
Execution Environment (JVM, Dalvik/ART, etc.)
System Libraries
Concurrency/Parallelism Frameworks
Java Threads & Synchronizers
Additional Application Frameworks
Applications
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
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

- Example frameworks or Java Mechanisms:
  - e.g., Android AsyncTask/HaMeR frameworks or Java Executor/CompetitionService
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
Mobile app developers may want to program with higher-level frameworks.

- Additional Application Frameworks
- Concurrency/Parallelism Frameworks
- Execution Environment (JVM, Dalvik/ART, etc.)
- System Libraries
- Operating System Kernel

- 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

---

**Applications**

**Additional Application Frameworks**

**Concurrency/Parallelism Frameworks**
- Java Threads & Synchronizers

**Execution Environment (JVM, Dalvik/ART, etc.)**

**System Libraries**

**Operating System Kernel**
Which Java Mechanism(s) to Understand & Apply

- **Applications**
- **Additional Application Frameworks**
- **Concurrency/Parallelism Frameworks**
  - Java Threads & Synchronizers
- **Execution Environment (JVM, Dalvik/ART, etc.)**
- **System Libraries**
- **Operating System Kernel**

“Full stack” developers should understand concepts & mechanisms at each layer.
End of Background on Java Concurrency & Parallelism (Part 2)