The History of Concurrency & Parallelism Support in Java

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

- Be aware of the history of Java concurrency & parallelism
Learning Objectives in this Part of the Lesson

• Be aware of the history of Java concurrency & parallelism

Hopefully, you’ll already know some of this!!!
A Brief History of Concurrency in Java
A Brief History of Concurrency in Java

- Foundational concurrency support

---

e.g., Java threads & built-in monitor objects available in Java 1

---

See en.wikipedia.org/wiki/Java_version_history#JDK_1.0
A Brief History of Concurrency in Java

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

See [docs.oracle.com/javase/tutorial/essential/concurrency](docs.oracle.com/javase/tutorial/essential/concurrency)
A Brief History of Concurrency 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)

Allow multiple threads to communicate via a bounded buffer.
A Brief History of Concurrency 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
```
A Brief History of Concurrency 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();
        }
    }
}
```

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 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 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 in Java

- Advanced concurrency support

See en.wikipedia.org/wiki/Java_version_history#J2SE_5.0
A Brief History of Concurrency in Java

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

See en.wikipedia.org/wiki/Task_parallelism
Create a fixed-sized thread pool & also coordinate the starting & stopping of multiple tasks that acquire/release shared resources

A Brief History of Concurrency 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));
```

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

• Foundational parallelism support

See [en.wikipedia.org/wiki/Java_version_history#Java_SE_7](en.wikipedia.org/wiki/Java_version_history#Java_SE_7)
A Brief History of Parallelism in Java

- Foundational parallelism support
- Focus on data parallelism that runs the same task on different data elements by applying the split-apply-combine model

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

- Foundational parallelism support
- Focus on data parallelism that runs the same task on different data elements by applying the split-apply-combine model

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

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

A Brief History of Parallelism in Java

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

- Advanced parallelism support

* e.g., Java parallel streams & completable futures made available in Java 8

See [en.wikipedia.org/wiki/Java_version_history#Java_SE_8](en.wikipedia.org/wiki/Java_version_history#Java_SE_8)
A Brief History of Parallelism in Java

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

See en.wikipedia.org/wiki/Data_parallelism
A Brief History of Parallelism in Java

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

```
 supplyAsync(getStartPage())
 .thenComposeAsync(crawlHyperLinks(page))
 .thenApplyAsync(countImages(page))
 .thenApply(List::size)
 .thenCombine(imgNum2, (imgNum1, imgNum2) -> Integer::sum)

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

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

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

See github.com/douglasraigschmidt/LiveLessons/tree/master/ImageStreamGang
A Brief History of 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 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 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
End of History of Concurrency & Parallelism in Java