Java Streams:
Sequential vs. Parallel Streams

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 structure & functionality of Java streams, e.g.,
  - Fundamentals of streams
  - Benefits of streams
  - Creating a stream
  - Aggregate operations in a stream
  - Applying streams in practice
  - Sequential vs. parallel streams

See radar.oreilly.com/2015/02/java-8-streams-api-and-parallelism.html
Sequential vs. Parallel Streams
We’ll cover sequential streams first

See docs.oracle.com/javase/tutorial/collections/streams
Sequential vs. Parallel Streams

- Stream operations run sequentially or in parallel

```
parallelStream()
```

Aggregate operation (behavior f)

Aggregate operation (behavior g)

We’ll cover parallel streams later

See [docs.oracle.com/javase/tutorial/collectionsstreams/parallelism.html](docs.oracle.com/javase/tutorial/collectionsstreams/parallelism.html)
Sequential vs. Parallel Streams

- A parallel stream splits its elements into multiple chunk & uses the common fork-join pool to process these chunks independently.

Sequential vs. Parallel Streams

- A parallel stream splits its elements into multiple chunk & uses the common fork-join pool to process these chunks independently.

A parallel stream is often more efficient and scalable than a sequential stream.

Tests conducted on a quad-core Lenovo P50 with 32 Gbytes of RAM
Sequential vs. Parallel Streams

- Ideally, a behavior’s output in a stream depends only on its input arguments

![Diagram showing sequential vs. parallel streams](https://en.wikipedia.org/wiki/Side_effect_(computer_science))

Sequential vs. Parallel Streams

- Ideally, a behavior’s output in a stream depends only on its input arguments.

```java
String capitalize(String s) {
    if (s.length() == 0)
        return s;
    return s.substring(0, 1).toUpperCase() + s.substring(1).toLowerCase();
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12](github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12)
Ideally, a behavior’s output in a stream depends only on its input arguments.

Behaviors with side-effects can incur race conditions in parallel streams.

Race conditions arise in software when an application depends on the sequence or timing of threads for it to operate properly.

See [en.wikipedia.org/wiki/Race_condition#Software](en.wikipedia.org/wiki/Race_condition#Software)
Sequential vs. Parallel Streams

- Ideally, a behavior’s output in a stream depends only on its input arguments
- Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

```java
long factorial(long n) {
    Total t = new Total();
    LongStream
        .rangeClosed(1, n)
        .parallel()
        .forEach(t::mult);
    return t.mTotal;
}
```

A buggy attempt to compute the ’nth’ factorial in parallel

Sequential vs. Parallel Streams

• Ideally, a behavior’s output in a stream depends only on its input arguments

• Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

```java
long factorial(long n) {
    Total t = new Total();
    LongStream
        .rangeClosed(1, n)
        .parallel()
        .forEach(t::mult);
    return t.mTotal;
}
```

See [henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html](http://henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html)
Sequential vs. Parallel Streams

• Ideally, a behavior’s output in a stream depends only on its input arguments
• Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```java
long factorial(long n) {
    Total t = new Total();
    LongStream
        .rangeClosed(1, n)
        .parallel()
        .forEach(t::mult);
    return t.mTotal;
}
```

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

- Run in parallel

- Sequential vs. Parallel Streams

- Behaviors with side-effects can incur race conditions in parallel streams, e.g.
Sequential vs. Parallel Streams

- Ideally, a behavior’s output in a stream depends only on its input arguments.
- Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```java
long factorial(long n) {
    Total t = new Total();
    LongStream
        .rangeClosed(1, n)
        .parallel() // Beware of race conditions!!!
        .forEach(t::mult);
    return t.mTotal;
}
```

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

See en.wikipedia.org/wiki/Race_condition#Software
Sequential vs. Parallel Streams

- Ideally, a behavior’s output in a stream depends only on its input arguments.
- Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```java
long factorial(long n) {
    Total t = new Total();
    LongStream
        .rangeClosed(1, n)
        .parallel()
        .forEach(t::mult);
    return t.mTotal;
}
```

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

Beware of inconsistent memory visibility

**Sequential vs. Parallel Streams**

- Ideally, a behavior’s output in a stream depends only on its input arguments
- Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```java
long factorial(long n) {
    Total t = new Total();
    LongStream
        .rangeClosed(1, n)
        .parallel()
        .forEach(t::mult);
    return t.mTotal;
}
```

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

*Only you can prevent concurrency hazards!*

In Java *you* must avoid these hazards, i.e., the compiler & JVM won’t save you...
End of Java Streams: Sequential vs. Parallel Streams