Java 8 Functional Interfaces

Introduction

Douglas C. Schmidt
Learning Objectives in This Lesson

- Recognize foundational functional programming features in Java 8, e.g.,
  - Lambda expressions
  - Method & constructor references
  - Key functional interfaces
Learning Objectives in This Lesson

• Recognize foundational functional programming features in Java 8, e.g.,
  • Lambda expressions
  • Method & constructor references
  • Key functional interfaces

These features are the basis for Java streams & concurrency/parallelism frameworks.
Learning Objectives in This Lesson

• Recognize foundational functional programming features in Java 8.
• Understand how to apply these Java 8 features in concise example programs.

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8)
Learning Objectives in This Lesson

- Recognize foundational functional programming features in Java 8.
- Understand how to apply these Java 8 features in concise example programs.
- The examples showcase the Java collections framework.

See docs.oracle.com/javase/8/docs/technotes/guides/collections
Overview of Common Functional Interfaces

Douglas C. Schmidt
Overview of Common Functional Interfaces

• A *functional interface* contains only one abstract method.

Overview of Common Functional Interfaces

A functional interface is the type used for a parameter when a lambda expression or method reference is passed as an argument to a method.

```java
<T> void runTest(Function<T, T> fact, T n) {
    long startTime = System.nanoTime();
    T result = fact.apply(n));
    long stopTime = (System.nanoTime() - startTime) / 1_000_000;
    ...
}
runTest(ParallelStreamFactorial::factorial, n);
runTest(SequentialStreamFactorial::factorial, n);
...
Overview of Common Functional Interfaces

- A functional interface is the type used for a parameter when a lambda expression or method reference is passed as an argument to a method.

```java
<T> void runTest(Function<T, T> fact, T n) {
    long startTime = System.nanoTime();
    T result = fact.apply(n));
    long stopTime = (System.nanoTime() - startTime) / 1_000_000;
    ...
}
```

Records & prints the time taken to compute 'n' factorial.

See [github.com/douglas craigschmidt/LiveLessons/tree/master/Java8/ex16](https://github.com/douglas craigschmidt/LiveLessons/tree/master/Java8/ex16)
Overview of Common Functional Interfaces

- A functional interface is the type used for a parameter when a lambda expression or method reference is passed as an argument to a method.

```java
<T> void runTest(Function<T, T> fact, T n) {
    long startTime = System.nanoTime();
    T result = fact.apply(n));
    long stopTime = (System.nanoTime() - startTime) / 1_000_000;
    ...
}
runTest(ParallelStreamFactorial::factorial, n);
runTest(SequentialStreamFactorial::factorial, n);
...
```

'fact' parameterizes the factorial implementation.

See [docs.oracle.com/javase/8/docs/api/java/util/function/Function.html](http://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)
Overview of Common Functional Interfaces

- A functional interface is the type used for a parameter when a lambda expression or method reference is passed as an argument to a method.

```java
<T> void runTest(Function<T, T> fact, T n) {
    long startTime = System.nanoTime();
    T result = fact.apply(n));
    long stopTime = (System.nanoTime() - startTime) / 1_000_000;
    ...
}
runTest(ParallelStreamFactorial::factorial, n);
runTest(SequentialStreamFactorial::factorial, n);
...
```

Different factorial implementations can be passed as parameters to `runTest()`.
Overview of Common Functional Interfaces

A functional interface is the type used for a parameter when a lambda expression or method reference is passed as an argument to a method.

```java
<T> void runTest(Function<T, T> fact, T n) {
    long startTime = System.nanoTime();
    T result = fact.apply(n);
    long stopTime = (System.nanoTime() - startTime) / 1_000_000;
    ...
}
runTest(ParallelStreamFactorial::factorial, n);

static BigInteger factorial(BigInteger n) {
    return LongStream.rangeClosed(1, n)
        .parallel()
        .mapToObj(BigInteger::valueOf)
        .reduce(BigInteger.ONE, BigInteger::multiply);
}
```
Java 8 defines many types of functional interfaces.

### Overview of Common Functional Interfaces

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiConsumer&lt;T,U&gt;</td>
<td>Represents an operation that accepts two input arguments and returns no result.</td>
</tr>
<tr>
<td>BiFunction&lt;T,U,R&gt;</td>
<td>Represents a function that accepts two arguments and produces a result.</td>
</tr>
<tr>
<td>BinaryOperator&lt;T&gt;</td>
<td>Represents an operation upon two operands of the same type, producing a result of the same type as the operands.</td>
</tr>
<tr>
<td>BiPredicate&lt;T,U&gt;</td>
<td>Represents a predicate (boolean-valued function) of two arguments.</td>
</tr>
<tr>
<td>BooleanSupplier</td>
<td>Represents a supplier of boolean-valued results.</td>
</tr>
<tr>
<td>Consumer&lt;T&gt;</td>
<td>Represents an operation that accepts a single input argument and returns no result.</td>
</tr>
<tr>
<td>DoubleBinaryOperator</td>
<td>Represents an operation upon two double-valued operands and producing a double-valued result.</td>
</tr>
<tr>
<td>DoubleConsumer</td>
<td>Represents an operation that accepts a single double-valued argument and returns no result.</td>
</tr>
<tr>
<td>DoubleFunction&lt;R&gt;</td>
<td>Represents a function that accepts a double-valued argument and produces a result.</td>
</tr>
<tr>
<td>DoublePredicate</td>
<td>Represents a predicate (boolean-valued function) of one double-valued argument.</td>
</tr>
<tr>
<td>DoubleSupplier</td>
<td>Represents a supplier of double-valued results.</td>
</tr>
<tr>
<td>DoubleToLongFunction</td>
<td>Represents a function that accepts a double-valued argument and produces a long-valued result.</td>
</tr>
<tr>
<td>DoubleUnaryOperator</td>
<td>Represents an operation on a single double-valued operand that produces a double-valued result.</td>
</tr>
<tr>
<td>Function&lt;T,R&gt;</td>
<td>Represents a function that accepts one argument and produces a result.</td>
</tr>
</tbody>
</table>

See [docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html](docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html)
Overview of Common Functional Interfaces

- Java 8 defines many types of functional interfaces.
- Some of these interfaces handle reference types.
Java 8 defines many types of functional interfaces.

Some of these interfaces handle reference types.

Other interfaces support primitive types.

See docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
Overview of Common Functional Interfaces

- Java 8 defines many types of functional interfaces.
  - Some of these interfaces handle reference types.
  - Other interfaces support primitive types.
  - Avoids “auto-boxing” overhead.

### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntegerConsumer</td>
<td>Represents an operation that accepts a single int-valued argument and returns no result.</td>
</tr>
<tr>
<td>IntegerFunction&lt;R&gt;</td>
<td>Represents a function that accepts an int-valued argument and produces a result.</td>
</tr>
<tr>
<td>IntegerPredicate</td>
<td>Represents a predicate (boolean-valued function) of one int-valued argument.</td>
</tr>
<tr>
<td>IntegerSupplier</td>
<td>Represents a supplier of int-valued results.</td>
</tr>
<tr>
<td>IntegerToDoubleFunction</td>
<td>Represents a function that accepts an int-valued argument and produces a double-valued result.</td>
</tr>
<tr>
<td>IntegerToLongFunction</td>
<td>Represents a function that accepts an int-valued argument and produces a long-valued result.</td>
</tr>
<tr>
<td>IntegerUnaryOperator</td>
<td>Represents an operation on a single int-valued operand that produces an int-valued result.</td>
</tr>
<tr>
<td>LongBinaryOperator</td>
<td>Represents an operation upon two long-valued operands and producing a long-valued result.</td>
</tr>
<tr>
<td>LongConsumer</td>
<td>Represents an operation that accepts a single long-valued argument and returns no result.</td>
</tr>
<tr>
<td>LongFunction&lt;R&gt;</td>
<td>Represents a function that accepts a single long-valued argument and produces a result.</td>
</tr>
<tr>
<td>LongPredicate</td>
<td>Represents a predicate (boolean-valued function) of one long-valued argument.</td>
</tr>
<tr>
<td>LongSupplier</td>
<td>Represents a supplier of long-valued results.</td>
</tr>
<tr>
<td>LongToDoubleFunction</td>
<td>Represents a function that accepts a long-valued argument and produces a double-valued result.</td>
</tr>
<tr>
<td>LongToIntFunction</td>
<td>Represents a function that accepts a long-valued argument and produces an int-valued result.</td>
</tr>
<tr>
<td>LongUnaryOperator</td>
<td>Represents an operation on a single long-valued operand that produces a long-valued result.</td>
</tr>
<tr>
<td>ObjDoubleConsumer&lt;T&gt;</td>
<td>Represents an operation that accepts an object-valued and a double-valued argument, and returns no result.</td>
</tr>
<tr>
<td>ObjIntConsumer&lt;T&gt;</td>
<td>Represents an operation that accepts an object-valued and a int-valued argument, and returns no result.</td>
</tr>
</tbody>
</table>

See [rules.sonarsource.com/java/tag/performance/RSPEC-4276](rules.sonarsource.com/java/tag/performance/RSPEC-4276)
Overview of Common Functional Interfaces

- Java 8 defines many types of functional interfaces.
- Some of these interfaces handle reference types.
- Other interfaces support primitive types.
- There’s an explosion of Java functional interfaces!

See dzone.com/articles/whats-wrong-java-8-part-ii
Java 8 defines many types of functional interfaces.

- Some of these interfaces handle reference types.
- Other interfaces support primitive types.
- There’s an explosion of Java functional interfaces!
- However, learn these interfaces before trying to customize your own.
Overview of Common Functional Interfaces

- Java 8 defines many types of functional interfaces.
  - Some of these interfaces handle reference types.
  - Other interfaces support primitive types.
  - There’s an explosion of Java functional interfaces!

*We focus on the most common types of functional interfaces.*
Overview of Common Functional Interfaces

- Java 8 defines many types of functional interfaces.
  - Some of these interfaces handle reference types.
  - Other interfaces support primitive types.
  - There’s an explosion of Java functional interfaces!

All usages of functional interfaces in the upcoming examples are “stateless”!