Java 8 Lambda Expressions

Douglas C. Schmidt
Learning Objectives in This Lesson

• Recognize foundational functional programming features in Java 8, e.g.,
  • Lambda expressions

Several concise examples are used to showcase foundational Java 8 features.
Overview of Java 8
Lambda Expressions

Douglas C. Schmidt
Overview of Java 8 Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later

```java
new Thread(() -> System.out.println("hello world"))
    .start();
```

Overview of Java 8 Lambda Expressions

- A lambda expression is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later

```java
new Thread(() -> System.out.println("hello world"))
    .start();
```

The Thread constructor expects an instance of Runnable.

See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html
Overview of Java 8 Lambda Expressions

- A lambda expression is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
    .start();
```

This lambda expression takes no parameters, i.e., "()".
Overview of Java 8 Lambda Expressions

- A lambda expression is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
    .start();
```

*The arrow separates the param list from the body of the lambda.*
Overview of Java 8 Lambda Expressions

• A lambda expression is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
    .start();
```

The body of the lambda defines the computation.
Overview of Java 8 Lambda Expressions

- A **lambda expression** is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
  .start();
```

Java 8 lambda expressions support more concise types of "behavioral parameterization."

See blog.indrek.io/articles/java-8-behavior-parameterization
Overview of Java 8 Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
   .start();
```

This lambda defines a computation that runs in a separate Java thread.

See [docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html](http://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html)
A *lambda expression* is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
   .start();
```

Runnable `r = () -> System.out.println("hello world");
new Thread(r).start();

You can also store a lambda expression into a variable & pass that variable to a method.

See [docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html](http://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html)
Overview of Java 8 Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
    .start();
```

*Lambda expressions are compact since they just focus on computation(s) to perform.*
A lambda expression is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() -> System.out.println("hello world"))
    .start();
```

VS.

Conversely, this anonymous inner class requires more code to write each time.

```java
new Thread(new Runnable() {
    public void run() {
        System.out.println("hello world");
    }
}).start();
```
Overview of Java 8 Lambda Expressions

• Lambda expressions can work with multiple parameters in a much more compact manner than anonymous inner classes

String[] nameArray = {"Barbara", "James", "Mary", "John", 
"Robert", "Michael", "Linda", "james", "mary"};

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s,String t) { return 
    s.toLowerCase().compareTo(t.toLowerCase()); }});

VS.

Arrays.sort(nameArray, 
    (s, t) -> s.compareToIgnoreCase(t));
Overview of Java 8 Lambda Expressions

• Lambda expressions can work with multiple parameters in a much more compact manner than anonymous inner classes, e.g.,

```

Arrays.sort(nameArray, new Comparator<String>() {
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }
});
```

```
Arrays.sort(nameArray,
    (s, t) -> s.compareToIgnoreCase(t));
```

Array of names represented as strings.
Overview of Java 8 Lambda Expressions

- Lambda expressions can work with multiple parameters in a much more compact manner than anonymous inner classes, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }});
```

Extraneous syntax for anonymous inner class.
Lambda expressions can work with multiple parameters in a much more compact manner than anonymous inner classes, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>() {
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }
});
```

**VS.**

```java
Arrays.sort(nameArray, (s, t) -> s.compareToIgnoreCase(t));
```

*(s, t) is short for (String s, String t), which leverages Java 8’s type inference capabilities.*

See [docs.oracle.com/javase/tutorial/java/generics/genTypeInference.html](docs.oracle.com/javase/tutorial/java/generics/genTypeInference.html)
Overview of Java 8 Lambda Expressions

- Lambda expressions can work with multiple parameters in a much more compact manner than anonymous inner classes, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }
});
```

vs.

```java
Arrays.sort(nameArray,
    (s, t) -> s.compareToIgnoreCase(t));
```

This lambda expression omits the method name & extraneous syntax.
Overview of Java 8 Lambda Expressions

• Lambda expressions can work with multiple parameters in a much more compact manner than anonymous inner classes, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }
});
```

VS.

```java
Arrays.sort(nameArray, (s, t) -> s.compareToIgnoreCase(t));
```

Therefore, it’s good practice to use lambda expressions whenever you can!
A lambda expression can access the (effectively) final variables from the enclosing scope.

```java
int answer = 42;
new Thread(() -> System.out.println("The answer is " + answer)).start();
```

This lambda expression can access the value of “answer,” which is an effectively final variable whose value never changes after it’s initialized.

See [www.linkedin.com/pulse/java-8-effective-final-gaurhari-dass](http://www.linkedin.com/pulse/java-8-effective-final-gaurhari-dass)
Overview of Java 8 Lambda Expressions

• Lambda expressions are most effective when they are “stateless” & have no shared mutable data.

    int answer = 42;
    new Thread(() -> System.out.println("The answer is " + answer))
        .start();

See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html
Overview of Java 8 Lambda Expressions

- Lambda expressions are most effective when they are “stateless” & have no shared mutable data.

Stateless lambda expressions are particularly useful when applied to Java parallel streams.

See [docs.oracle.com/javase/tutorial/collections/streams/parallelism.html](docs.oracle.com/javase/tutorial/collections/streams/parallelism.html)
Douglas C. Schmidt

Implementing Closures With Java 8 Lambda Expressions
Implementing Closures with Java 8 Lambda Expressions

- Lambda expressions can implement (simplified) variants of “closures”

```java
class ClosureExample {
    private int mRes;

    Thread makeThreadClosure(String s, int n) {
        return new Thread(() -> System.out.println(s + (mRes += n)));
    }

    ClosureExample() throws InterruptedException {
        Thread t = makeThreadClosure("result = ", 10);
        t.start(); t.join();
    }
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex1](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex1)
Lambda expressions can implement (simplified) variants of “closures”

```java
class ClosureExample {
    private int mRes;

    Thread makeThreadClosure(String s, int n) {
        return new Thread(() -> System.out.println(s + (mRes += n)));
    }

    ClosureExample() throw InterruptedException {
        Thread t = makeThreadClosure("result = ", 10);
        t.start(); t.join();
    }
}
```

A closure is an object storing a method together with an environment that has least one bound variable.

See [en.wikipedia.org/wiki/Closure_(computer_programming)](en.wikipedia.org/wiki/Closure_(computer_programming))
Lambda expressions can implement (simplified) variants of “closures”

```java
class ClosureExample {
  private int mRes;

  Thread makeThreadClosure(String s, int n) {
    return new Thread(() -> System.out.println(s + (mRes += n)));
  }

  ClosureExample() throws InterruptedException {
    Thread t = makeThreadClosure("result = ", 10);
    t.start(); t.join();
  }
}
```

This private field & the method params are “bound variables.”

A bound variable is name that has a value, such as a number or a string.
Implementing Closures with Java 8 Lambda Expressions

- Lambda expressions can implement (simplified) variants of “closures”

```java
class ClosureExample {
    private int mRes;

    Thread makeThreadClosure(String s, int n) {
        return new Thread(() -> System.out.println(s + (mRes += n)));
    }

    ClosureExample() throws InterruptedException {
        Thread t = makeThreadClosure("result = ", 10);
        t.start(); t.join();
    }
}
```

This lambda implements a closure that captures a private field & method params.

See bruceeckel.github.io/2015/10/17/are-java-8-lambdas-closures
Implementing Closures with Java 8 Lambda Expressions

• Lambda expressions can implement (simplified) variants of “closures”

```java
class ClosureExample {
    private int mRes;

    Thread makeThreadClosure(String s, int n) {
        return new Thread(() -> System.out.println(s + (mRes += n)));
    }

    ClosureExample() throws InterruptedException {
        Thread t = makeThreadClosure("result = ", 10);
        t.start(); t.join();
    }
}
```

Values of private fields can be updated in a lambda, but not params or local vars (which are read-only).

See [dzone.com/articles/java-8-lambdas-limitations-closures](dzone.com/articles/java-8-lambdas-limitations-closures)
Lambda expressions can implement (simplified) variants of “closures”

```java
class ClosureExample {
    private int mRes;

    Thread makeThreadClosure(String s, int n) {
        return new Thread(() -> System.out.println(s + (mRes += n)));
    }

    ClosureExample() throws InterruptedException {
        Thread t = makeThreadClosure("result = ", 10);
        t.start(); t.join();
    }
}
```

This factory method creates a closure that then runs in a background thread.
