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.

Douglas C. Schmidt

Overview of Java 8 Lambda Expressions

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

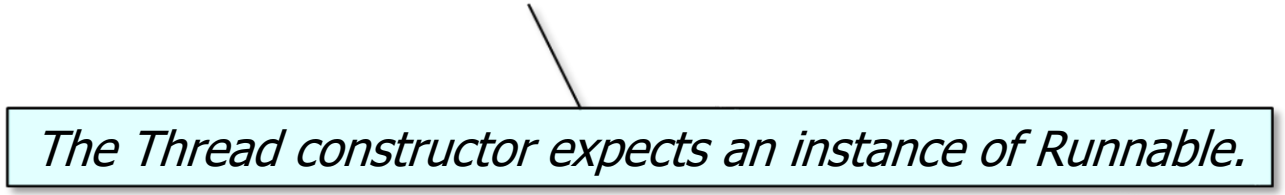
```
new Thread(() -> System.out.println("hello world"))
    .start();
```

See www.drdobbs.com/jvm/lambda-expressions-in-java-8/240166764

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

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



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.,

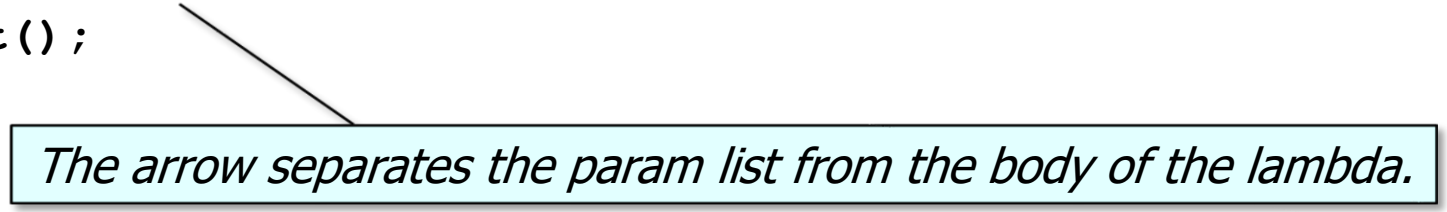
```
new Thread(() -> System.out.println("hello world"))
    .start();
```

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.,

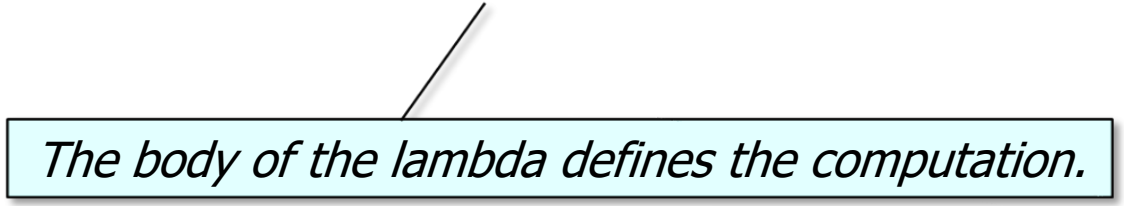
```
new Thread(() -> System.out.println("hello world"))
    .start();
```



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, e.g.,

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

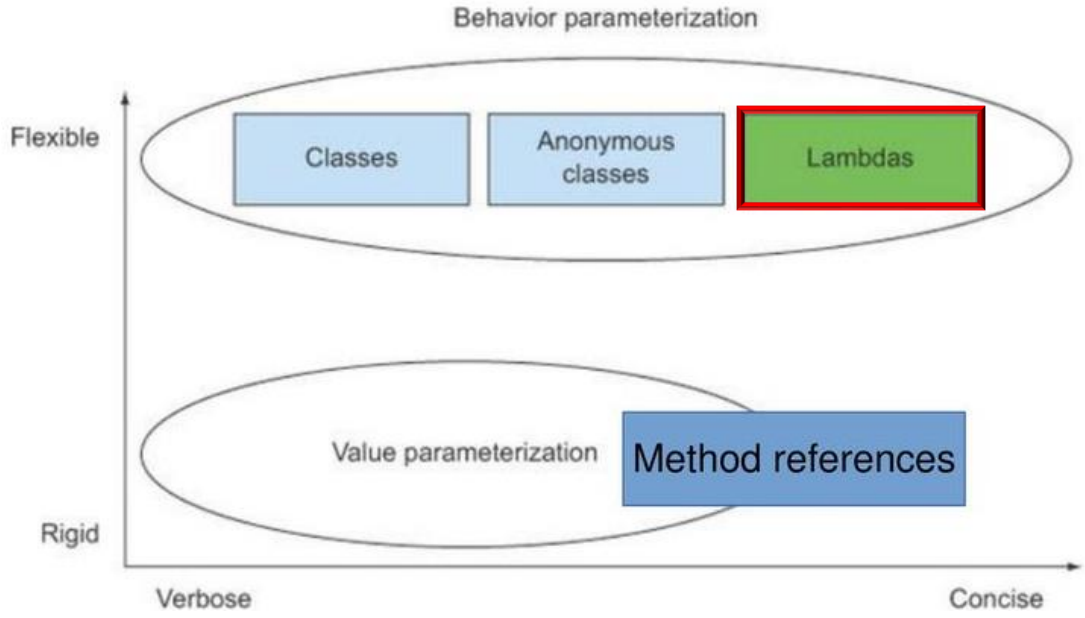


Overview of Java 8 Lambda Expressions

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

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

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



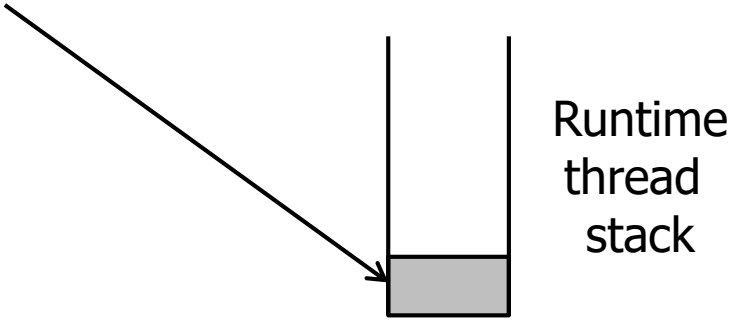
See blog.indrek.io/articles/java-8-behavior-parameterization

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.,

```
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

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.,

```
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

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.,

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

Lambda expressions are compact since they just focus on computation(s) to perform.

Overview of Java 8 Lambda Expressions

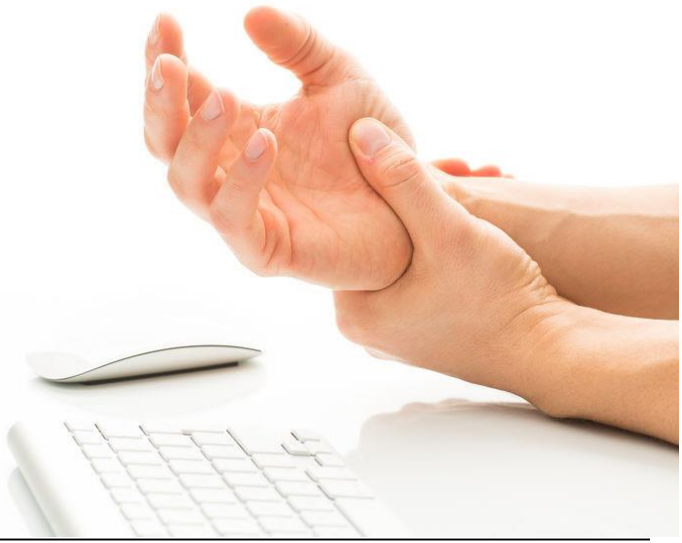
- A *lambda expression* is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

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

vs.

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

```
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));
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex1

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.,

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

Array of names represented as strings.

```
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.,

```
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()); }});
```

Extraneous syntax for anonymous inner class.

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.,

```
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));
```

(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

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.,

```
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));
```

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.,

```
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));
```

Therefore, it's good practice to use lambda expressions whenever you can!

Overview of Java 8 Lambda Expressions

- A lambda expression can access the (effectively) final variables from the enclosing scope.

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

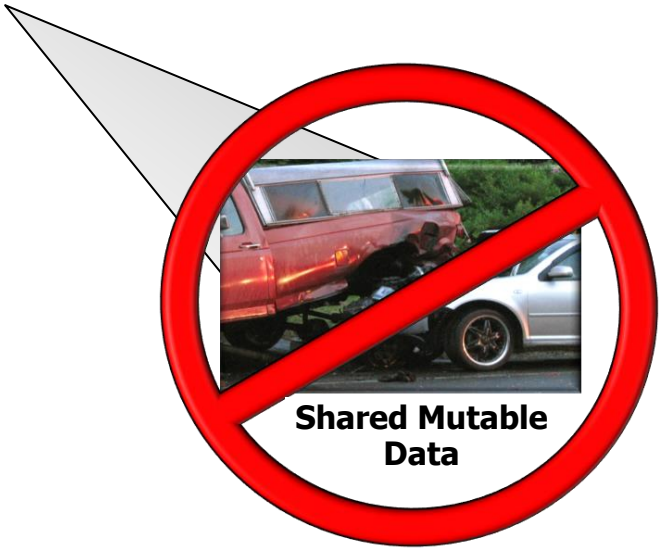
This lambda expression can access the value of "answer," which is an effectively final variable whose value never changes after it's initialized.

See www.linkedin.com/pulse/java-8-effective-final-gaurhari-dass

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();
```

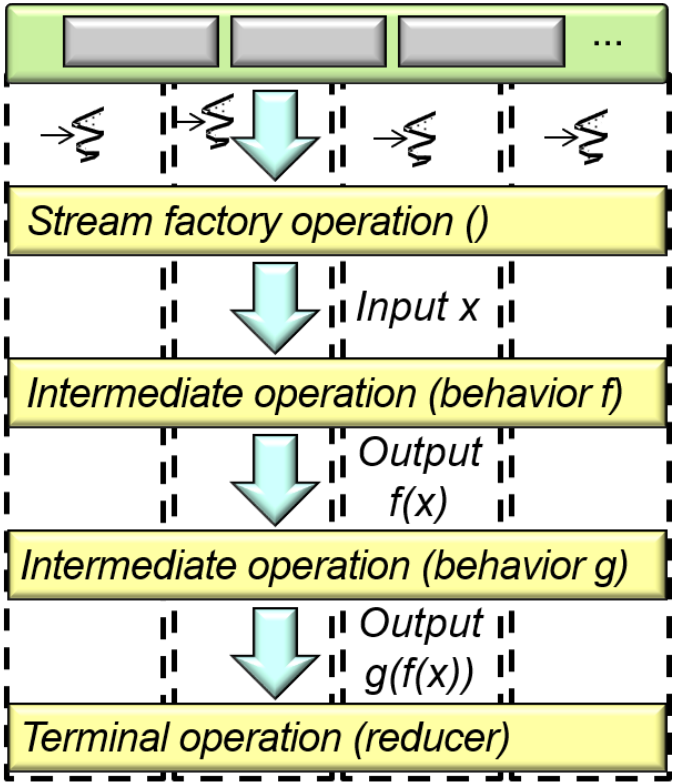


See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html

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

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"

```
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();
  }
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex1

Implementing Closures with Java 8 Lambda Expressions

- Lambda expressions can implement (simplified) variants of "closures"

```
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 w/ an environment that has least one bound variable.

See en.wikipedia.org/wiki/Closure_(computer_programming)

Implementing Closures with Java 8 Lambda Expressions

- Lambda expressions can implement (simplified) variants of "closures"

```
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();
  }
}
```

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"

```
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();
  }
}
```

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"

```
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();
  }
}
```

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-lambas-limitations-closures

Implementing Closures with Java 8 Lambda Expressions

- Lambda expressions can implement (simplified) variants of "closures"

```
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();
  }
}
```

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

See en.wikipedia.org/wiki/Factory_method_pattern