Working with Streams and Lambda Expressions in Java

(Java SE 11 Developer Certification 1Z0-819)

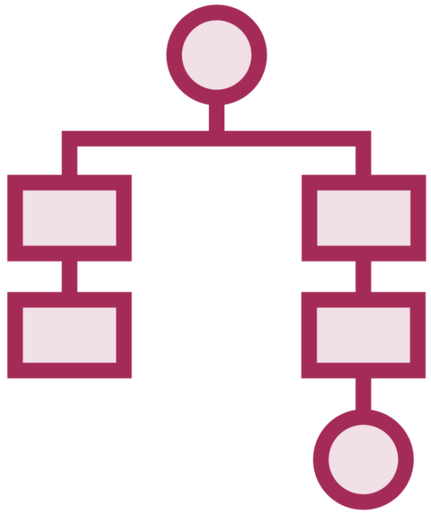
Working with Lambda Expressions

Jesper de Jong
Software Architect

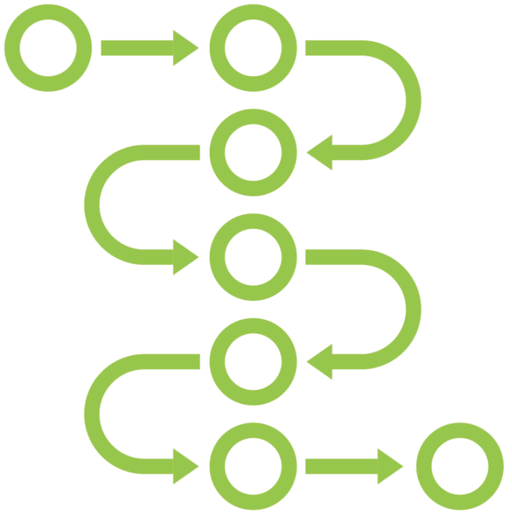
@jesperdj www.jesperdj.com

Working with Streams and Lambda Expressions

Lambda Expressions



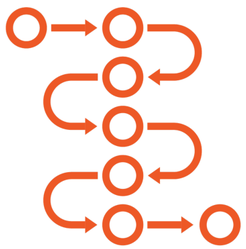
Functional Interfaces



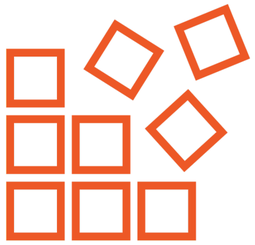
Streams

Developer Certification Exam Skills

Implement functional interfaces using lambda expressions, including interfaces from the java.util.function package.



Use Java streams to filter, transform and process data.



Perform decomposition and reduction, including grouping and partitioning on sequential and parallel streams.

Working with the Exercises

Download examples and exercises from the Pluralsight course page

Java Development Kit for Java SE 11

- **https://oracle.com/javase**
- **https://adoptopenjdk.net**

IDE – IntelliJ IDEA

- **https://jetbrains.com**

Lambda Expressions – First Look

What Is a Lambda Expression?

A lambda expression is an anonymous method

Functional programming: Passing code as data

Functional Interfaces – First Look

Functional Interfaces

```
new Comparator<Product> {
    @Override
    public int compare(Product p1, Product p2) {
        return p1.getPrice().compareTo(p2.getPrice());
    }
}
```

```
(p1, p2) -> p1.getPrice().compareTo(p2.getPrice())
```

```
interface Comparator<T> {
    int compare(T o1, T o2);
}
```

Functional Interfaces

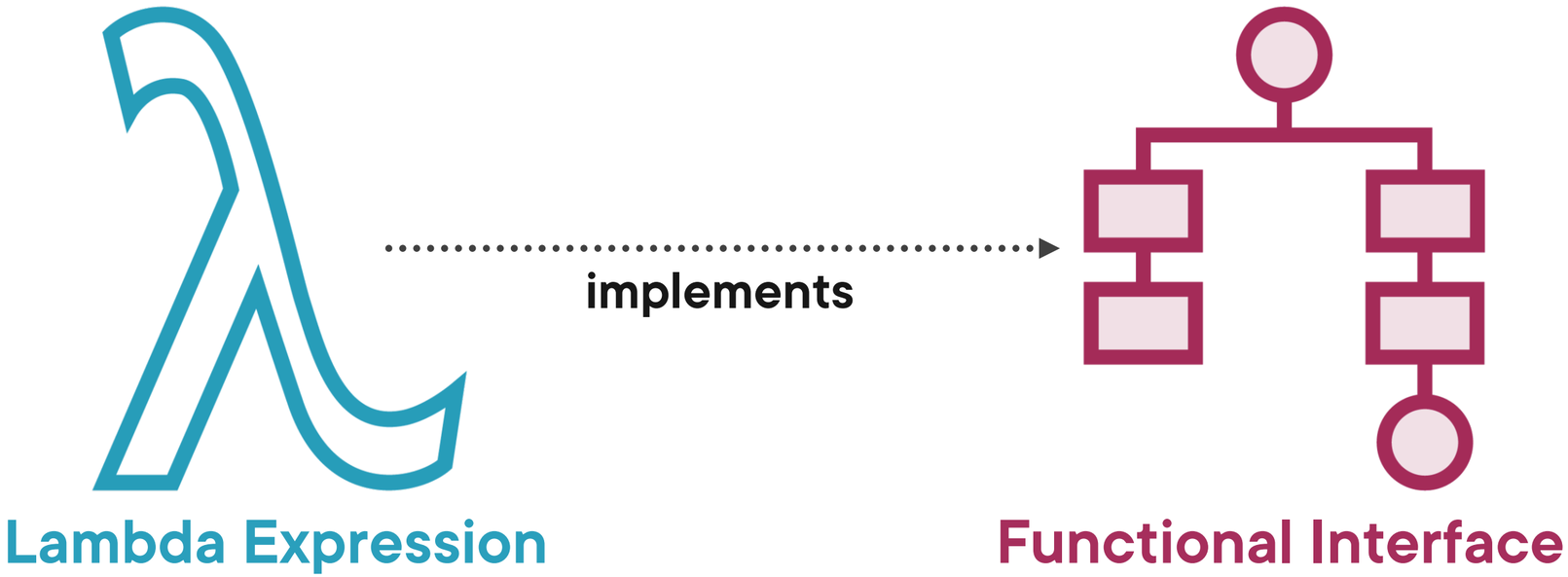
```
interface Comparator<T> {
    int compare(T o1, T o2);
}
```

```
interface Runnable {
    void run();
}
```

```
interface FileFilter {
    boolean accept(File f);
}
```

```
interface ActionListener {
    void actionPerformed(ActionEvent e);
}
```

Lambda Expressions and Functional Interfaces



Standard functional interfaces
in package java.util.function

Syntax of Lambda Expressions

Syntax of Lambda Expressions

```
public int compare(Product p1, Product p2) {
    return p1.getPrice().compareTo(p2.getPrice());
}
```

Access specifier **Return type** **Name** **Parameter list** **Body**

```
(Product p1, Product p2) -> p1.getPrice().compareTo(p2.getPrice())
```

Parameter list **Arrow** **Body**

Syntax of Lambda Expressions

```
Runnable runnable = () -> System.out.println(“Hello World”);
```

No parameters

```
FileFilter filter = file -> file.isHidden();
```

Single parameter

Syntax of Lambda Expressions

```
public int compare(Product p1, Product p2) {
    return p1.getPrice().compareTo(p2.getPrice());
}
```

```
(p1, p2) -> {
    return p1.getPrice().compareTo(p2.getPrice());
}
```

```
(p1, p2) -> p1.getPrice().compareTo(p2.getPrice())
```

Syntax of Lambda Expressions

parameters **->** **body**

- **Parameter types are optional**
- **No parameters: empty parentheses**
- **Single parameter: parentheses are optional**

- **Block or single expression**

Capturing Variables in Lambda Expressions

Interaction of Lambda Expressions with Enclosing Code

The Meaning of “this” in an Anonymous Class

```
public class Example {

    public void printMessage() {
        Runnable runnable = new Runnable() {
            String message = “Hello World”;

            public void run() {
                System.out.println(this.message);
            }
        };

        new Thread(runnable).start();
    }
}
```

The Meaning of “this” in a Lambda Expression

```
public class Example {
    private String message = “Hello World”;

    public void printMessage() {
        Runnable runnable =
            () -> System.out.println(this.message);

        new Thread(runnable).start();
    }
}
```

Working with Exceptions in Lambda Expressions

Method References

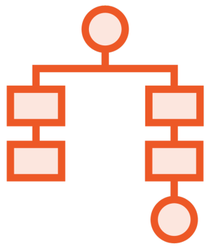
Four Types of Method References

ClassName :: methodName	**Static method**
objectRef :: methodName	**Instance method of a specific object**
ClassName :: methodName	**Instance method not of a specific object**
ClassName :: new	**Constructor**

Summary of Lambda Expressions

Lambda Expressions and Functional Interfaces

A lambda expression is an anonymous method



A lambda expression implements a functional interface

A functional interface has a single abstract method

Syntax of Lambda Expressions

```
(p1, p2) -> p1.getPrice().compareTo(p2.getPrice())
```

```
Runnable runnable = () -> System.out.println("Hello World");
```

```
FileFilter filter = file -> file.isHidden();
```

Lambda Expressions

Captured local variables must be effectively final

Avoid side effects in lambda expressions

this and super have the same meaning as in surrounding code

Dealing with exceptions can be inconvenient

Use a method reference instead of a lambda expression

Working with the Exercises

Take a look at the example code

Work with the exercises

Up Next:
Working with Functional Interfaces