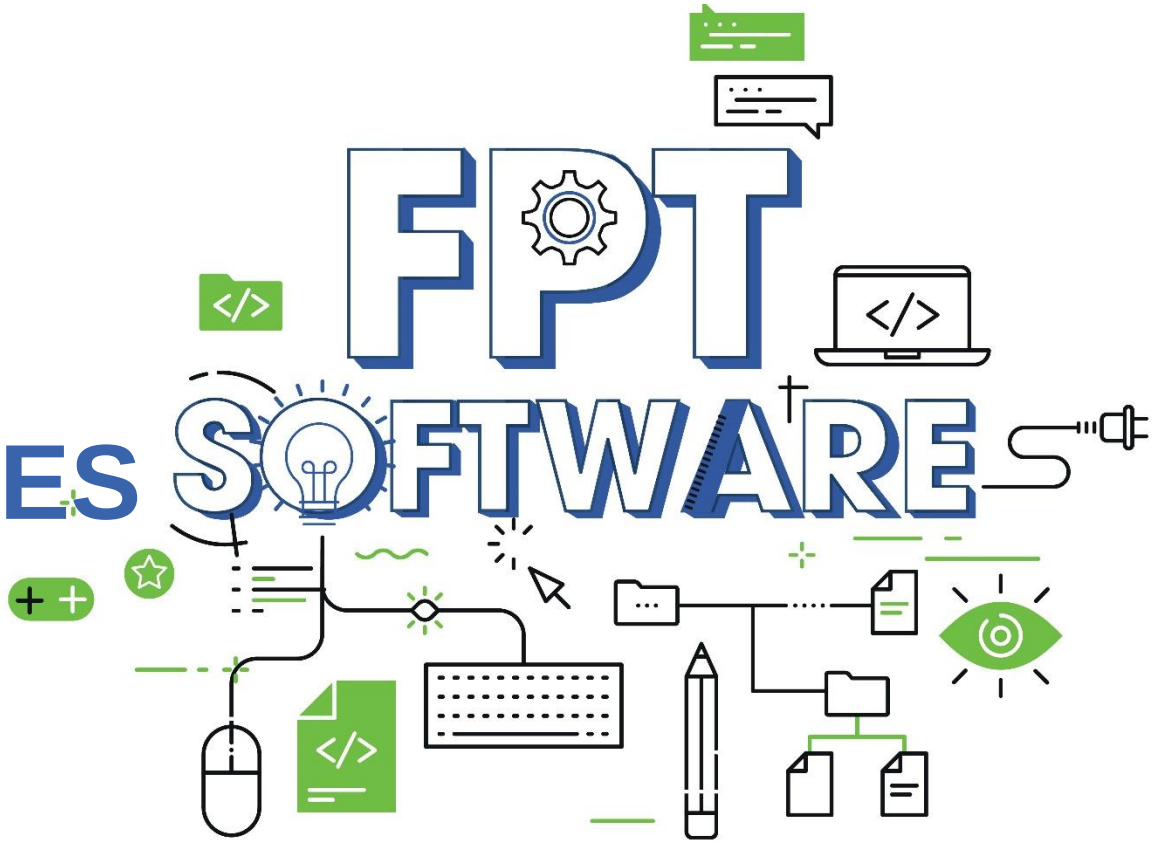
LAMBDA EXPRESSIONS
FUNCTIONAL INTERFACES



Design by: DieuNT1

Lesson Objectives

- *Understand the **concept** and **purpose of Lambda Expressions** in programming.*
- *Recognize the **syntax** and **structure of Lambda Expressions**.*
- *Differentiate Lambda Expressions from traditional anonymous inner classes.*
- *Learn about interfaces and **Functional Interfaces***
- *Define and utilize Functional Interfaces in conjunction with Lambda Expressions.*

Agenda

- Lambda Expressions
- Functional Interfaces

Section 1

Lambda Expressions in Java

Lambda Expressions

A lambda expression is a short block of code which are designed to accept a set of parameters as input and return a value as an output

- **A Lambda expression** like an anonymous method, they do not need a name and they can be implemented right in the body of a method.
- **Lambda expressions** are often referred to as lambdas for short.
- Lambda Expressions implement the only abstract function and therefore implement functional interfaces lambda expressions are added in **Java 8** and provide the below functionalities.
  - ✓ Enable to treat functionality as a **method argument**, or **code as data**.
  - ✓ A function that can be created **without belonging to any class**.
  - ✓ A lambda expression can be passed around as if it was an object and executed on demand.

Lambda Expressions

Syntax

```
(parameter_list) -> { function_body }
```

The lambda expression contains three required parts as below:

1. ***parameter_list:*** (that can also be empty). If more than one argument is included, the arguments must be comma separated and enclosed inside the parenthesis.

2. ***arrow operator: ->***

3. ***function_body:*** that contains expressions and statements for execution (or is empty). If the function body consists of only one line the **curly braces** are optional

Lambda Expressions Examples

- **Example 1:**

```
public interface FuncInterface {
  // An abstract function
  void abstractFun(int x);

  // A non-abstract (or default) function
  default void normalFun() {
     System.out.println("Hello");
  }
}
```

```
public interface Test {
  public static void main(String args[]) {
  // lambda expression to implement above functional interface.
  // This interface by default implements abstractFun()
  FuncInterface fobj = (int x) -> System.out.println(2 * x);

  // This calls above lambda expression and prints 10.
  fobj.abstractFun(5);
  fobj.normalFun();
  }
}
```

- **Output:**

```
10
Hello
```

Lambda Expression Parameters

- There are three Lambda Expression Parameters are mentioned below:

1. **Lambda Expression with Zero parameter**

```
() -> System.out.println("Zero parameter lambda");
```

2. **Lambda Expression with Single parameter**

```
(p) -> System.out.println("One parameter: " + p);
```

3. **Lambda Expression with Multiple parameters**

```
(p1, p2) -> {
   Double total = p1 + p1;
   System.out.println("Multiple parameters: " + p1 + ", " + p2 + ":" + total);
};
```

Lambda Expressions Examples

- **Example 2:** Use a lambda expression in the ArrayList's `forEach()` method to print every item in the list:

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(5);
numbers.add(9);
numbers.add(8);
numbers.add(1);

numbers.forEach((n) -> {
        System.out.println(n);
});
```

- **Output:**

```
5 9 8 1
```

Lambda Expressions Examples

- **Example 3:** Use Java's **Consumer** interface to store a lambda expression in a variable:

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(5);
numbers.add(9);
numbers.add(8);
numbers.add(1);

Consumer<Integer> method = (n) -> { System.out.println(n); };
numbers.forEach( method );
```

Lambda Expressions Examples

- **Example 4:** Create a method which takes a lambda expression as a parameter:

```
interface StringFunction {
    String run(String str);
}

public class StringSample {
    public static void main(String[] args) {
        StringFunction exclaim = (s) -> s + "!";
        StringFunction ask = (s) -> s + "?";
        print("Hello", exclaim);
        print("Hello", ask);
    }
    public static void print(String str, StringFunction format) {
        String result = format.run(str);
        System.out.println(result);
    }
}
```

Lambda Expressions Examples

Sample: Suppose that we are creating an interview management app. The candidate is classified to: Qualify (Interview result >= 7) and Unqualify (Interview result < 7)

```
public class Candidate {
    private final String name;
    private final int interviewResult; // From 0 -> 10

    public Candidate(String name, int interviewResult) {
        this.name = name;
        this.interviewResult = interviewResult;
    }

    public String getName() {
        return name;
    }

    public int getInterviewResult() {
        return interviewResult;
    }
}
```

```
public interface Check {
    boolean test(Candidate c);
}
```

Lambda Expressions Examples

- **Solution 01:** Provide concrete classes implement method of Check interface

```
public class QualifyCandidate implements Check {
    @Override
    public boolean test(Candidate c) {
        return c.getInterviewResult() >= 7;
    }
}
```

```
public class UnQualifyCandidate implements Check{
    @Override
    public boolean test(Candidate c) {
        return c.getInterviewResult() < 7;
    }
}
```

Lambda Expressions Examples

```
public static void main(String[] args) {
    List<Candidate> candidates = Arrays.asList(new Candidate("Peter", 4),
                                               new Candidate("David", 8));

    System.out.println("Qualify Candidate: ");
    print(candidates, new QualifyCandidate());

    System.out.println("UnQualify Candidate: ");
    print(candidates, new UnQualifyCandidate());

}

public static void print(List<Candidate> candidates, Check check) {
    for (Candidate candidate : candidates) {
        if (check.test(candidate)) {
            System.out.println(candidate.getName() + "\n");
        }
    }
}
```

```
=== Qualify Candidate ===
David

=== UnQualify Candidate ===
Peter
```

Lambda Expressions Examples

- **Solution 02:** Using Anonymous class

```
public static void main(String[] args) {
    List<Candidate> candidates = Arrays.asList(new Candidate("Peter", 4),
                                               new Candidate("David", 8));

    System.out.println("Qualify Candidate: ");
    print(candidates, new Check() {
        @Override
        public boolean test(Candidate a) {
            return a.getInterviewResult() >= 7;
        }
    });

    System.out.println("UnQualify Candidate: ");
    print(candidates, new Check() {
        @Override
        public boolean test(Candidate a) {
            return a.getInterviewResult() < 7;
        }
    });
}
```

```
=== Qualify Candidate ===
David

=== UnQualify Candidate ===
Peter
```

Lambda Expressions Discussion

- **Solution 01:** We need to create many concrete classes. For each condition, we have to create a concrete class. **Source code will be verbose, redundant.**

- **Solution 02:** Implementation of anonymous class such as an interface that contains only one method. **The syntax may seem unwieldy and unclear.**

In these cases, you're usually trying to pass functionality as an argument to another method. Lambda expressions enable you to do this.

Lambda Expressions Examples

- **Solution 03:** Using lambda expressions

```
public static void main(String[] args) {
    List<Candidate> candidates = Arrays.asList( new Candidate("Peter", 4),
                                                new Candidate("David", 8));

    System.out.println("Qualify Candidate: ");
    print(candidates, (Candidate c) -> { return c.getInterviewResult() >= 7; });

    System.out.println("UnQualify Candidate: ");
    print(candidates, (Candidate c) -> { return c.getInterviewResult() < 7; });
}
```

```
=== Qualify Candidate ===
David

=== UnQualify Candidate ===
Peter
```

Lambda Expressions

- **Caution:** The syntax of lambdas is tricky because many parts are optional.
  - ✓ The parentheses can only be omitted if there is a **single parameter** and its type is not explicitly stated

```
c -> { return c.getInterviewResult() >= 7; }
```

  - ✓ Can omit braces when we only have a single statement and ***doesn't require*** you to type **return** or use a **semicolon** when no braces are used

```
c -> c.getInterviewResult() >= 7
```

  - ✓ If body expression has two or more statements, must be using {} to create blocks of code.

Lambda Expressions

- **Let's look at some examples of valid lambdas**

```
() -> true                                  // 0 parameter
a -> a.startsWith("test")                   // 1 parameter
(String a) -> a.startsWith("test")          // 1 parameter
(a, b) -> a.startsWith("test")              // 2 parameters
(String a, String b) -> a.startsWith("test")  // 2 parameters
```

- And some examples of invalid lambdas

```
a, b -> a.startsWith("test")                // [1] DOES NOT COMPILE
a -> { a.startsWith("test"); }              // [2] DOES NOT COMPILE
a -> { return a.startsWith("test") }        // [3] DOES NOT COMPILE
```

[1] The first line needs parentheses around the parameter list
[2] The second line is missing the return keyword
[3] The last line is missing the semicolon

Lambda Expression vs Regular Methods

Lambda Expression	Method
✓ Lambda Expressions do not require naming	• Methods require the method name to be declared
✓ **Syntax**: ([comma separated argument-list]) -> {body}	• **Syntax**: <classname> :: <methodname> or <objectname>.<methodname>
✓ Lambda Expression may not include parameters	• Methods may not include parameters as well
✓ Lambda Expression does not require a return type	• The return type for methods is mandatory
✓ The Lambda Expression is itself the complete code segment	• The method body is just another code segment of the program

Section 2

Functional Interfaces

Functional Interfaces

A **functional interface** as an interface that contains a single abstract method.

- **Syntax**:

```
@FunctionalInterface
public interface Check {
    boolean test(Candidate c);
}
```

✓ The **Check** is a functional interface, because it contains **exactly one abstract method** `test()`.

Functional Interfaces

- **Applying the `@FunctionalInterface` annotation**
  - ✓ It is not required with functional programming.
  - ✓ The Java compiler **implicitly assumes** that any *interface that contains exactly one abstract method is a functional interface*.
  - ✓ If a class marked with the **@FunctionalInterface** annotation contains **more than one abstract method**, or **no abstract methods** at all, then the compiler will **detect this error** and **not compile**.

Functional Interfaces

- **Create a Functional Interface**

```
@FunctionalInterface
interface MyInterface {
     // abstract method
     String reverse(String n);
}
```

- **Using lambda expression with parameters**

```
public class TestFunc {
     public static void main(String[] args) {
          // Declare a reference to MyInterface
          // Assign a lambda expression to the reference
          MyInterface ref = (str) -> {
               String result = "";
               for (int i = str.length() - 1; i >= 0; i--) {
                    result += str.charAt(i);
               }
               return result;
          };
          // call the method of the interface
          System.out.println("Lambda reversed = " + ref.reverse("Lambda"));
     }
}
```

Output:

```
Lambda reversed = adbmaL
```

Generic Functional Interfaces

- **Create a generic Functional Interface**

```
//GenericInterface.java
@FunctionalInterface
interface GenericInterface<T> {
     // generic method
     T func(T t);
}
```

- **Generic Functional Interface and Lambda Expressions**

```
// the GenericInterface operates on String data and assign a lambda expression to it
GenericInterface<String> reverse = (str) -> {
     String result = "";
     for (int i = str.length() - 1; i >= 0; i--) {
          result += str.charAt(i);
     }
     return result;
};
System.out.println("Lambda reversed = " + reverse.func("Lambda"));
```

Generic Functional Interfaces

- **Generic Functional Interface and Lambda Expressions**

```
// the GenericInterface operates on Integer data and assign a lambda expression to it
GenericInterface<Integer> factorial = (n) -> {
     int result = 1;
     for (int i = 1; i <= n; i++) {
          result = i * result;
     }
     return result;
};

System.out.println("Factorial of 5 = " + factorial.func(5));
```

- **Output:**

```
Lambda reversed = adbmaL
Factorial of 5 = 120
```

Built-in Functional Interfaces

- **Lambdas** work with **functional interfaces**.
- Java define many **built-In Functional Interfaces** in the *java.util.function* package:

Functional Interface	Parameters	Return Type	Single Abstract Method
Supplier<T>	0	T	get
Consumer<T>	1 (T)	void	accept
BiConsumer<T, U>	2 (T, U)	void	accept
Predicate<T>	1 (T)	boolean	test
BiPredicate<T, U>	2 (T, U)	boolean	test
Function<T, R>	1 (T)	R	apply
BiFunction<T, U, R>	2 (T, U)	R	apply
UnaryOperator<T>	1 (T)	T	apply
BinaryOperator<T>	2 (T, T)	T	apply

Built-in Functional Interfaces

▪ Implementing Predicate

✓ In our earlier example, we created an interface with one method:

```
public interface Check {
    boolean test(Candidate c);
}
```

✓ Luckily, Java recognizes that this is a common problem and provides such an interface for us

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

Built-in Functional Interfaces

```
public static void main(String[] args) {
    List<Candidate> candidates = Arrays.asList(    new Candidate("Peter", 4), new Candidate("David", 8));

    System.out.println("Qualify Candidate: ");
    print(candidates, (c) -> c.getInterviewResult() >= 7);

    System.out.println("UnQualify Candidate: ");
    print(candidates, (c) -> c.getInterviewResult() < 7);
}

public static void print(List<Candidate> candidates, Predicate<Candidate> check) {
    for (Candidate candidate : candidates) {
        if (check.test(candidate)) {
            System.out.println(candidate.getName() + "\n");
        }
    }
}
```

Built-in Functional Interfaces

- **Implementing Examples:**

```
Supplier<Boolean> supplier = () -> true; // 0 parameter
System.out.println(supplier.get());

Predicate<String> predicate = a -> a.startsWith("test"); // 1 parameter
System.out.println(predicate.test("test lambda expression!"));

Function<String, Boolean> function = (String a) -> a.startsWith("test"); // 1 parameter
System.out.println(function.apply("lambda expression testing"));

BiPredicate<String, String> biPredicate = (a, b) -> a.startsWith("test"); // 2 parameters
System.out.println(biPredicate.test("automation test", "test quiz"));

BiFunction<String, String, Boolean> biFunction = (String a, String b) -> a.startsWith("test");
System.out.println(biFunction.apply("test quiz", "automation test"));
```

- **Output:**

```
true
true
false
false
true
```

Functional Interfaces

- **Implementing Comparator:**

```
public class Product {
     private int id;
     private String name;
     private float price;
     public Product(int id, String name, float price) {
          super();
          this.id = id;
          this.name = name;
          this.price = price;
     }

     // Setter, getter methods
}
```

Functional Interfaces

- **Implementing Comparator:**

```
public class ProductManagement {
	public static void main(String[] args) {
		List<Product> list = new ArrayList<Product>();
		// Adding Products
		list.add(new Product(1, "HP Laptop", 25000f));
		list.add(new Product(3, "Keyboard", 300f));
		list.add(new Product(2, "Dell Mouse", 150f));
		System.out.println("Sorting on the basis of name...");
		// implementing lambda expression
		Collections.sort(list, (p1, p2) -> {
		    return p1.getName().compareTo(p2.getName());
		});

		for (Product p : list) {
		    System.out.println(p.getId() + " " + p.getName() + " " + p.getPrice());
		}
	}
}
```

- **Output:**

```
Sorting on the basis of name...
2 Dell Mouse 150.0
1 HP Laptop 25000.0
3 Keyboard 300.0
```

Summary

- Lambda Expressions
- Functional Interfaces

References

- *https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html*
- *https://www.baeldung.com/java-8-lambda-expressions-tips*
- *https://www.geeksforgeeks.org/lambda-expressions-java-8/*

THANK YOU!