Lambda Expressions & Functional Interface

Dasar – Dasar Pemrograman 2

Pudy Prima

Lambda Expressions

- An expression that can be used to simplify code
- Lambda expression is a new feature in Java 8
- Lambda expression can be seen as an anonymous function
  - It is not associated with a class
- A lambda expression can be used wherever the type is a functional interface
  - to provide the implementation of the abstract method

Functional Interface

- An interface that has only one abstract method
- Previously known as Single Abstract Method (SAM)
- It is recommended that a functional interface utilizes @FunctionalInterface annotation
- A functional interface may have multiple default methods
- Examples of functional interface in Java API
  - `interface ActionListener { void actionPerformed(ActionEvent e); }`
  - `interface Runnable { void run(); }`
  - `interface Comparable<T> { int compareTo(T o) }`

Lambda Expressions: Basic Syntax

```
(type1 param1, type2 param2, …) -> expression
```

or

```
(type1 param1, type2 param2, …) -> { statements; }
```

Example use of lambda expressions:

- Variable assignment

```
Callable c = ( ) -> process();
```

- Method parameter

```
new Thread(() -> process()).start();
```


Lambda Expressions: GUI Event Handling

```
btEnlarge.setOnAction {
  new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      // Code for processing event e
    }
  }
});
```

(a) Anonymous inner class event handler

```
btEnlarge.setOnAction(e -> {
  // Code for processing event e
});
```

(b) Lambda expression event handler

Unlike anonymous inner class, using lambda expression **doesn't generate** additional .class file when the main class is compiled.

Lambda Expressions: Iterating Collection

Utilizing the default method `forEach()` in Iterable interface

```
List<String> strings = Arrays.asList(new String[] { "A", "B", "C", "D", "E" });
strings.forEach(s -> System.out.println(s));
```

```
A
B
C
D
E
```


Lambda-like expression in Switch (Java 14)

Normal switch	Lambda-like switch

`switch (var) {`
`  case const1: { statements-1; break; }`
`  case const2: { statements-2; break; }`
`  ...`
`  default: def-statements;`
`}`
	`switch (var) {`
	`  case const1 -> { statements-1 }`
	`  case const2 -> { statements-2 }`
	`  ...`
	`  default: def-statements;`
	`}`

Types of Functional Interface in Java

- `Predicate<T> { boolean test(T t); }`
  - Evaluates a predicate, returns true or false
- `Function<T, R> { R apply(T t); }`
  - Accepts an argument and returns a result
- `Consumer<T> { void accept(T t); }`
  - Accepts an input, consumes or modifies the input, no output
- `Supplier<T> { T get(); }`
  - Does not accept any input, only returns a result

Package java.util.function

Functional interfaces provide target types for lambda expressions and method references.

See: Description

Interface Summary

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

List of Functional Interface in Java

References

- Oracle Java tutorial: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

- Lambda Expressions in Java 8 by Dr. Ionut Cardei: http://wisenet.fau.edu/class/cop4331/notes/java-lambda-expr.pdf

- Liang, Introduction to Java Programming, 11th Edition, Ch. 15