Lambda Expression

AUGUST 2023

expleo

Anonymous Classes

Anonymous Classes

Introduction

- A class that have **no name** is known as **anonymous class** in Java.
- It is a **inner class**.
- There are **two Ways we can create Anonymous Inner class**:
  - Class (may be abstract or concrete).
  - Interface
- We may use either **extend an existing class** or **implement an interface**.
- Define a class **in place** instead of **in a separate file.**

Needs

- Logically **group code** in one place
- Increase **Encapsulation**
- Make code **more readable**
- Anonymous inner classes are useful in writing implementation classes for listener interfaces in **graphics programming**.

Example #1

```
abstract class Person{
  abstract void eat();
}
class AnonymousExample {
        public static void main(String args[]){
                Person obj=new Person(){
                        void eat(){
                                System.out.println("Nice Fruits");
                        }
                };
                obj.eat();
        }
}
```

Internal class generated by the compiler

```
static class TestAnonymousInner$1 extends Person
{
  TestAnonymousInner$1(){}
  void eat()
   {
      System.out.println("Nice Fruits");
   }
}
```

Note: A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method. An object of Anonymous class is created that is referred by **obj** reference variable of Person type.

Output:
Nice Fruits

Example #2

```
/*** This example demonstrates anonymous class*/
interface A
{
   void print();
}
class AnonymousExample {
      public static void main(String [] args) {
            A obj= new A(){       //Anonymous inner class
                  public void print(){
                        System.out.println("Happy Learning");
                  }
            };
            obj.print();
      }
}
```

Output:
Happy Learning

Issues

- The syntax of anonymous classes may seem unwieldy and unclear.
- Even an anonymous class seems a bit excessive and cumbersome.

Lambda Expression

Introduction

- **Lambda expressions** are a **new and important Java feature** introduced in Java SE 8.
- It provides a **clear and concise way** to represent **one method interface** using an expression.
- Lambda expressions basically express instances of **functional interfaces**.
  - An interface which has **only one abstract method** is called functional interface.
- Lambda expressions implement the **only abstract function** and therefore **implement functional interfaces.**

Anonymous Class Vs Lambda Expression

- As we discussed, The syntax of anonymous classes may seem **unwieldy** and **unclear**. But incase of lambda the syntax is **clear** and **look** like a normal expression.
- A lambda expression is a **short form for writing an anonymous class**. By using a lambda expression, we can declare **methods without any name**.
- Anonymous classes can be used in case of more than one **abstract method** while a lambda expression specifically used for **functional interfaces**.

Syntax

Expression Type	Example
No Arguments: ()-> Expression;	()->System.out.println("Hello, world!");
One Argument: parameter -> Expression;	x->System.out.println(x);
Two Arguments: (parameter, parameter) -> Expression;	(x, y)->x + y;
With Explicit Argument Types: (int parameter1, int parameter 2) -> Expression;	(int x, int y)-> x + y;

Multiple Statements:
(int parameter1, int parameter 2)-> {
Statements;
}
	(x, y) -> {
	System.out.println(x);
	System.out.println(x);
	return (x+y);
	}

Example #1

```
interface Welcome{
   public String welcomeMessage();
}
public class LambdaExpressionExample{
        public static void main(String[] args) {
                Welcome obj=()->{
                        return "Welcome to Learning!";
                };
                System.out.println(obj.welcomeMessage());
        }
}
```

Output:
Welcome to Learning!

Example #2

```
interface GreetingService {
    void sayMessage(String message);
  }
public class LambdaExpressionExample{
  public static void main(String args[]) {
    //without parenthesis
    GreetingService greetService1 = message -> System.out.println("Hello " + message);
    //with parenthesis
    GreetingService greetService2 = (message) -> System.out.println("Hello " + message);
    greetService1.sayMessage("Mahesh");
    greetService2.sayMessage("Suresh");
  }
}
```

Output:
Hello Mahesh
Hello Suresh

Example #3

```
interface Addable{
   int add(int a,int b);
}
public class LambdaExpressionExample{
   public static void main(String[] args) {
      // Multiple parameters in lambda expression
       Addable ad1=(a,b)->(a+b);
       System.out.println(ad1.add(10,20));

       // Multiple parameters with data type in lambda expression
       Addable ad2=(int a,int b)->(a+b);
       System.out.println(ad2.add(100,200));
```

Example #3

```
        // Lambda expression without return keyword.
        Addable ad3=(a,b)->(a+b);
        System.out.println(ad3.add(10,20));

        // Lambda expression with return keyword.
        Addable ad4=(int a,int b)->{
                        System.out.println("a="+a);
                        System.out.println("b="+b);
                        return (a+b);
        };
        System.out.println("Addition of a&b is="+ad4.add(100,200));
    }
}
```

Output:

```
30
300
30
a=100
b=200
Addition of a&b is=300
```

Example #4

```
interface MathOperation {
    int operation(int a, int b);
  }
public class LambdaExpressionExample {
  public static void main(String args[]) {
    //with type declaration
    MathOperation addition = (int a, int b) ->a + b;

    //with out type declaration
    MathOperation subtraction = (a, b) ->a - b;

    //with return statement along with curly braces
    MathOperation multiplication = (int a, int b) -> { return a * b; };
```

Example #4

```
    //without return statement and without curly braces
    MathOperation division = (int a, int b) -> a / b;

    System.out.println("10 + 5 = " + addition.operation(10, 5));
    System.out.println("10 - 5 = " + subtraction.operation(10, 5));
    System.out.println("10 x 5 = " + multiplication.operation(10, 5));
    System.out.println("10 / 5 = " + division.operation(10, 5));
  }
}
```

Output:
10 + 5 = 15
10 - 5 = 5
10 x 5 = 50
10 / 5 = 2

Lambda Expressions in GUI Applications

- To process events in a **graphical user interface (GUI) application**, like
  - Keyboard actions,
  - Mouse actions,
  - Scroll actions,
- Typically need to create event handlers, which usually involves **implementing a particular interface**.
- Often, **event handler interfaces are functional interfaces**, they tend to have only one method.