Java SE 8 Programming Language

Lab Guides

Document Code	25e-BM/HR/HDCV/FSOFT
Version	1.1
Effective Date	20/11/2012

Hanoi, 06/2019

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1	01/Oct/2018	Add the new labs	Create new	DieuNT1	VinhNV
2	01/Jun/2019	Update template	Fsoft template	DieuNT1	VinhNV

Contents

FRESHER ACADEMY	CODE:	JPL.S.L601
	TYPE:	SHORT
	LOC:	
	DURATION:	45 MINUTES

Unit 10: Lambda Expressions and Functional Interfaces

Knowledge Summary

Java Lambda Expressions

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Java lambda expression is treated as a function, so compiler does not create .class file.

Functional Interface

Lambda expression provides implementation of functional interface. An interface which has only one abstract method is called functional interface. Java provides an anotation @FunctionalInterface, which is used to declare an interface as functional interface.

Why use Lambda Expression

- To provide the implementation of Functional interface.
- Less coding.

Java Lambda Expression Syntax

(argument-list) -> {body}

Java lambda expression is consisted of three components.

1) Argument-list: It can be empty or non-empty as well.

2) Arrow-token: It is used to link arguments-list and body of expression.

3) Body: It contains expressions and statements for lambda expression.

Lab Guide 1: Use Lambda Expressions

Objectives:

This lab guide helps trainees know how to use Lambda Expressions and define functional interface.

Problem Descriptions:

Create a Java Project named **JPL.S.L601** in Eclipse.

Create package **fa.training.model** that contains:

- User class

Create package **fa.training.lambdademo** that contains:

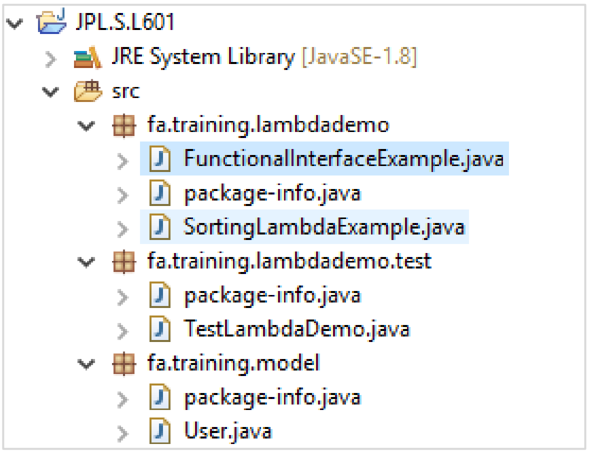
- SortingLambdaExample class
- FunctionalInterfaceExample class

Create package **fa.training.lambdademo.test** that contains:

- TestLambdaDemo class

Guidelines:

Step 1: Create a project struture like this:



Step 2: Create **User** class:

```
package fa.training.model;

/**
 * @author hoabt2
 *
 */
public class User {
        private String name;
        private float salary;
        private int age;

/**
 * A constructor with three parameters.
 */
public User(String name, float salary, int age) {
        super();
        this.name = name;
        this.salary = salary;
        this.age = age;
}

/**
 * getter and setter methods
```

```java
 */

@Override
public String toString() {
        return "User [name=" + name + ", salary=" + salary +
                                                 ", age=" + age + "]";
}
}

```

Step 3: Create **SortingLambdaExample** class

```java
package fa.training.lambdademo;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import fa.training.model.User;

/**
 * Examples of how to use Lambda Expressions
 *
 * @author hoabt2
 *
 */
public class SortingLambdaExample {

/**
 * Get users
 *
 * @return a list of user
 */
private static List<User> getUsers() {

        List<User> users = new ArrayList<User>();
        users.add(new User("Peter", 5000, 33));
        users.add(new User("Mary", 2000, 20));
        users.add(new User("John", 1500, 10));
        users.add(new User("Ivan", 2300, 55));
        return users;
}

/**
 * Sort data and display user info
 *
 */
public void showUserInfoSortByName() {
        List<User> users = getUsers();
        System.out.println("Before Sort");
        for (User user : users) {
                System.out.println(user);
        }

        System.out.println("After Sort");
        //sort by age
        //users.sort((User user1, User user2) -> user1.getAge() - user2.getAge());
        //sort by name
        System.out.println("Sort by name");
        users.sort((user1, user2) -> user1.getName().compareTo(user2.getName()));
        //display data
        users.forEach((user) -> System.out.println(user));
}

/**
 * Sort data by salary
 *
 */
public void showUserInfoSortBySalary() {
        List<User> users = getUsers();
        System.out.println("Before Sort");
        for (User user : users) {
                System.out.println(user);
        }

        System.out.println("After Sort");
        //Sort by salary
        Comparator<User> salaryComparator = (o1, o2)->
                                        o1.getSalary().compareTo(o2.getSalary());
        users.sort(salaryComparator);
        //display data
        System.out.println("Sort by salary");
        users.forEach((user) -> System.out.println(user));
}

/**
 * Sort data by salary and reverse order
 *
 */
public void showUserInfoSortBySalaryReverse() {
        List<User> users = getUsers();
        System.out.println("Before Sort");
        for (User user : users) {
                System.out.println(user);
        }

        System.out.println("After Sort");
        //Sort by salary
        Comparator<User> salaryComparator = (o1, o2)->
                                        o1.getSalary().compareTo(o2.getSalary());
        users.sort(salaryComparator.reversed());
        //display data
        System.out.println("Sort by salary, reverse order");
        users.forEach((user) -> System.out.println(user));
}
}
```

Step 4: Create **FunctionalInterfaceExample** class

```
package fa.training.lambdademo;

/**
 * Examples of how to use functional interface
 *
 * @author hoabt2
 *
 */
public class FunctionalInterfaceExample {

/**
 *
 */
public void calculateSquare() {
        System.out.println("Calculate Square, Functional Interface !!!");
        int a = 10;
         // lambda expression to define the calculate method
         Square s = (int x)->x * x;

         // parameter passed and return type must be
         // same as defined in the prototype
         int result = s.calculate(a);
         System.out.println(result);
}
}
```

```java
//define a functional interface
@FunctionalInterface
interface Square {
        int calculate(int x);
}

```

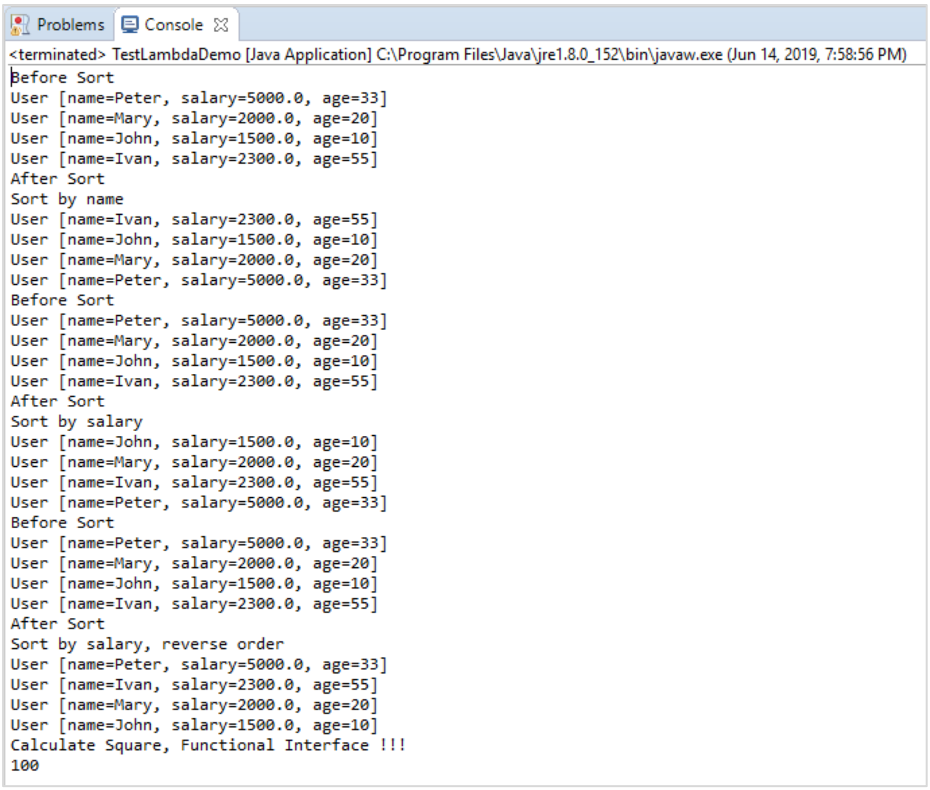
Step 5: Create **TestLambdaDemo** class

```java
package fa.training.lambdademo.test;

import fa.training.lambdademo.FunctionalInterfaceExample;
import fa.training.lambdademo.SortingLambdaExample;

/**
 * @author hoabt2
 *
 */
public class TestLambdaDemo {

/**
 * @param args
 */
public static void main(String[] args) {
        SortingLambdaExample sortLambda = new SortingLambdaExample();
        FunctionalInterfaceExample funcInterface = new FunctionalInterfaceExample();
        sortLambda.showUserInfoSortByName();
        sortLambda.showUserInfoSortBySalary();
        sortLambda.showUserInfoSortBySalaryReverse();
        funcInterface.calculateSquare();
}
}
```

Step 6: Run **TestLambdaDemo** to see the result

You can call corresponding methods separatedly in order to test the result clearly.

Result:



```
<terminated> TestLambdaDemo [Java Application] C:\Program Files\Java\jre1.8.0_152\bin\javaw.exe (Jun 14, 2019, 7:58:56 PM)
Before Sort
User [name=Peter, salary=5000.0, age=33]
User [name=Mary, salary=2000.0, age=20]
User [name=John, salary=1500.0, age=10]
User [name=Ivan, salary=2300.0, age=55]
After Sort
Sort by name
User [name=Ivan, salary=2300.0, age=55]
User [name=John, salary=1500.0, age=10]
User [name=Mary, salary=2000.0, age=20]
User [name=Peter, salary=5000.0, age=33]
Before Sort
User [name=Peter, salary=5000.0, age=33]
User [name=Mary, salary=2000.0, age=20]
User [name=John, salary=1500.0, age=10]
User [name=Ivan, salary=2300.0, age=55]
After Sort
Sort by salary
User [name=John, salary=1500.0, age=10]
User [name=Mary, salary=2000.0, age=20]
User [name=Ivan, salary=2300.0, age=55]
User [name=Peter, salary=5000.0, age=33]
Before Sort
User [name=Peter, salary=5000.0, age=33]
User [name=Mary, salary=2000.0, age=20]
User [name=John, salary=1500.0, age=10]
User [name=Ivan, salary=2300.0, age=55]
After Sort
Sort by salary, reverse order
User [name=Peter, salary=5000.0, age=33]
User [name=Ivan, salary=2300.0, age=55]
User [name=Mary, salary=2000.0, age=20]
User [name=John, salary=1500.0, age=10]
Calculate Square, Functional Interface !!!
100
```

-- THE END --