*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

| | |
|---|---|
| CODE: | JPL.S.L403 |
| TYPE: | SHORT |
| LOC: | |
| DURATION: | 60 MINUTES |

# Unit 8: Generics and Collections

## Knowledge Summary

### Arrays.sort() vs Collections.sort()

Arrays.sort works for arrays which can be of primitive data type also. Collections.sort() works for objects Collections like ArrayList, LinkedList, etc.

We can use Collections.sort() to sort an array after creating a ArrayList of given array items.

### Comparable and Comparator

Comparable and Comparator both are interfaces and can be used to sort collection elements.

| Comparabe | Compator |
|---|---|
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

# Lab Guide 1: Sort data with Comparator and Comparable

## Objective:

This lab guide helps trainees know how to sort data using Comparator and Comparable.

- Sort an Array
- Sort an ArrayList
- Use Comparable to sort an object
- Use Comparator to sort an object

## Problem Description:

Create a Java Project named **JPL.S.L403** in Eclipse.

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

- Fruit class

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

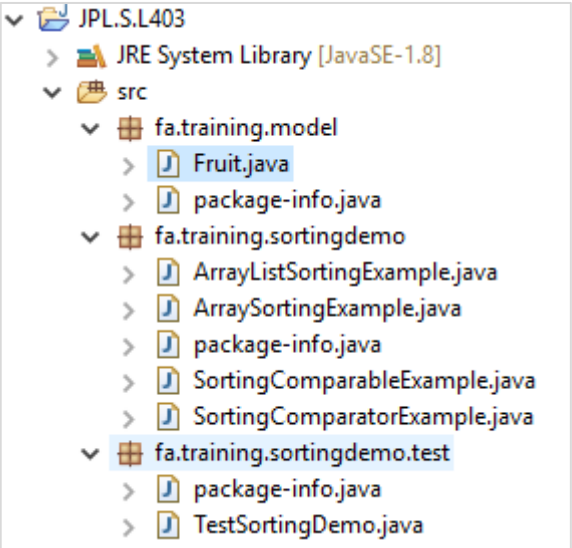
- ArraySortingExample class
- ArrayListSortingExample class
- SortingComparableExample class
- SortingComparatorExample class

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

- TestSortingDemo class

## Guidelines:

**Step 1**: Create a project struture like this:



**Step 2**: Create **Fruit** class that implements **Comparable** interface and override **compareTo()** method

```
1.  package fa.training.model;
2.
3.  /**
4.   * @author hoabt2
5.   *
6.   */
7.  public class Fruit implements Comparable<Fruit> {
8.
9.  private String fruitName;
10. private String fruitDesc;
11. private int quantity;
12.
13. /**
14.  * @param fruitName
15.  * @param fruitDesc
16.  * @param quantity
17.  */
18. public Fruit(String fruitName, String fruitDesc, int quantity) {
19.         super();
20.         this.fruitName = fruitName;
21.         this.fruitDesc = fruitDesc;
22.         this.quantity = quantity;
23. }
24.
25. /**
26.  * @return the fruitName
27.  */
28. public String getFruitName() {
29.         return fruitName;
30. }
31. /**
32.  * @param fruitName the fruitName to set
33.  */
34. public void setFruitName(String fruitName) {
35.         this.fruitName = fruitName;
36. }
37. /**
38.  * @return the fruitDesc
39.  */
40. public String getFruitDesc() {
41.         return fruitDesc;
42. }
43. /**
44.  * @param fruitDesc the fruitDesc to set
45.  */
46. public void setFruitDesc(String fruitDesc) {
47.         this.fruitDesc = fruitDesc;
48. }
49. /**
50.  * @return the quantity
51.  */
52. public int getQuantity() {
53.         return quantity;
54. }
55. /**
56.  * @param quantity the quantity to set
57.  */
58. public void setQuantity(int quantity) {
59.         this.quantity = quantity;
60. }
61.
62. @Override
63. public int compareTo(Fruit compareFruit) {
64.
65.         int compareQuantity = ((Fruit) compareFruit).getQuantity();
66.         //ascending order
67.         return this.quantity - compareQuantity;
```

```
68.         //descending order
69.         //return compareQuantity - this.quantity;
70. }
71.
72. @Override
73. public String toString() {
74.         return "Fruit [fruitName=" + fruitName + ", fruitDesc=" +
75.                                 fruitDesc + ", quantity=" + quantity + "]";
76. }
77. }
78.
```

**Step 3**: Create **ArraySortingExample** class

```
1.  package fa.training.sortingdemo;
2.
3.  import java.util.Arrays;
4.
5.  /**
6.   * Examples of how to sort an array.
7.   *
8.   * @author hoabt2
9.   *
10.  */
11. public class ArraySortingExample {
12.
13. /**
14.  * Sort an array using Arrays.sort()
15.  *
16.  */
17. public void sortArray() {
18.
19.         System.out.println("sortArray() !!!");
20.
21.         String[] fruits = new String[] { "Pineapple", "Apple", "Orange", "Banana" };
22.
23.         Arrays.sort(fruits);
24.
25.         int i = 0;
26.         for (String temp : fruits) {
27.                 System.out.println("fruits " + ++i + " : " + temp);
28.         }
29. }
30. }
```

**Step 4**: Create **ArrayListSortingExample** class

```
1.  package fa.training.sortingdemo;
2.
3.  import java.util.ArrayList;
4.  import java.util.Collections;
5.  import java.util.List;
6.
7.  /**
8.   * Examples of how to sort an ArrayList.
9.   *
10.  * @author hoabt2
11.  *
12.  */
13. public class ArrayListSortingExample {
14.
15. /**
16.  * Sort an ArrayList using Collections.sort()
17.  *
18.  */
19. public void sortArrayList() {
```

```
20.         System.out.println("sortArrayList() !!!");
21.
22.         List<String> fruits = new ArrayList<String>();
23.
24.         fruits.add("Pineapple");
25.         fruits.add("Apple");
26.         fruits.add("Orange");
27.         fruits.add("Banana");
28.
29.         Collections.sort(fruits);
30.
31.         int i = 0;
32.         for (String temp : fruits) {
33.                 System.out.println("fruits " + ++i + " : " + temp);
34.         }
35. }
36. }
37.
```

**Step 5**: Create **SortingComparableExample** class

```
1.  package fa.training.sortingdemo;
2.
3.  import java.util.Arrays;
4.
5.  import fa.training.model.Fruit;
6.
7.  /**
8.   * Examples of how to sort object using Comparable
9.   *
10.  * @author hoabt2
11.  *
12.  */
13. public class SortingComparableExample {
14.
15. /**
16.  * Sort data using Comparable
17.  *
18.  */
19. public void sortElementWithComparable() {
20.
21.         System.out.println("sortElementWithComparable() !!!");
22.
23.         Fruit[] fruits = new Fruit[4];
24.
25.         Fruit pineappale = new Fruit("Pineapple", "Pineapple description", 70);
26.         Fruit apple = new Fruit("Apple", "Apple description", 100);
27.         Fruit orange = new Fruit("Orange", "Orange description", 80);
28.         Fruit banana = new Fruit("Banana", "Banana description", 90);
29.
30.         fruits[0] = pineappale;
31.         fruits[1] = apple;
32.         fruits[2] = orange;
33.         fruits[3] = banana;
34.
35.         Arrays.sort(fruits);
36.
37.         int i = 0;
38.         for (Fruit temp : fruits) {
39.                 System.out.println("fruits " + ++i + " : " +
40.                         temp.getFruitName() + ", Quantity : " + temp.getQuantity());
41.         }
42. }
43. }
```

**Step 6**: Create **SortingComparatorExample** class

```
1.  package fa.training.sortingdemo;
2.
3.  import java.util.ArrayList;
4.  import java.util.Collections;
5.  import java.util.Comparator;
6.  import java.util.List;
7.
8.  import fa.training.model.Fruit;
9.
10. /**
11.  * Examples of how to sort data using Comparator
12.  *
13.  * @author hoabt2
14.  *
15.  */
16. public class SortingComparatorExample {
17.
18. /**
19.  * Sort data with Comparator
20.  *
21.  */
22. public void sortElementComparator() {
23.         System.out.println("sortElementComparator() !!!");
24.
25.         List<Fruit> fruitList = new ArrayList<>();
26.         Fruit pineappale = new Fruit("Pineapple", "Pineapple description", 70);
27.         Fruit apple = new Fruit("Apple", "Apple description", 100);
28.         Fruit orange = new Fruit("Orange", "Orange description", 80);
29.         Fruit banana = new Fruit("Banana", "Banana description", 90);
30.         fruitList.add(pineappale);
31.         fruitList.add(apple);
32.         fruitList.add(orange);
33.         fruitList.add(banana);
34.
35.         System.out.println("Fruits : " + fruitList);
36.         // Sort fruits by fruit name
37.         Comparator<Fruit> fruitNameComparator = new Comparator<Fruit>() {
38.
39.                 @Override
40.                 public int compare(Fruit obj1, Fruit obj2) {
41.                         String fruitName1 = obj1.getFruitName().toUpperCase();
42.                         String fruitName2 = obj2.getFruitName().toUpperCase();
43.
44.                         // ascending order
45.                         return fruitName1.compareTo(fruitName2);
46.                 }
47.         };
48.
49.         Collections.sort(fruitList, fruitNameComparator);
50.         System.out.println("\nFruits (Sorted by fruit name) : " + fruitList);
51.
52.         // Sort fruits by quantity
53.         Comparator<Fruit> quantiyComparator = new Comparator<Fruit>() {
54.
55.                 @Override
56.                 public int compare(Fruit obj1, Fruit obj2) {
57.                         if (obj1.getQuantity() < obj2.getQuantity()) {
58.                                 return -1;
59.                         } else if (obj1.getQuantity() > obj2.getQuantity()) {
60.                                 return 1;
61.                         } else {
62.                                 return 0;
63.                         }
64.                 }
65.         };
66.         Collections.sort(fruitList, quantiyComparator);
67.         System.out.println("\nFruits (Sorted by quantiy) : " + fruitList);
```

```
68.
69. }
70. }
```

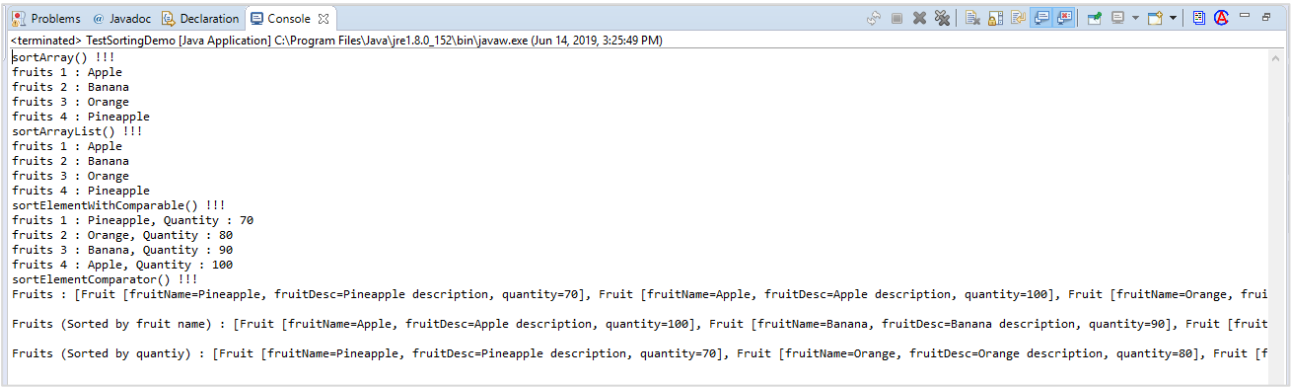
**Step 7**: Create **TestSortingDemo** class

```
1.  package fa.training.sortingdemo.test;
2.
3.  import fa.training.sortingdemo.ArrayListSortingExample;
4.  import fa.training.sortingdemo.ArraySortingExample;
5.  import fa.training.sortingdemo.SortingComparableExample;
6.  import fa.training.sortingdemo.SortingComparatorExample;
7.
8.  /**
9.   * @author hoabt2
10.  *
11.  */
12. public class TestSortingDemo {
13.
14. /**
15.  * @param args
16.  */
17. public static void main(String[] args) {
18.         ArraySortingExample arraySort = new ArraySortingExample();
19.         ArrayListSortingExample arrayListSort = new ArrayListSortingExample();
20.         SortingComparableExample comparableSort = new SortingComparableExample();
21.         SortingComparatorExample comparatorSort = new SortingComparatorExample();
22.         arraySort.sortArray();
23.         arrayListSort.sortArrayList();
24.         comparableSort.sortElementWithComparable();
25.         comparatorSort.sortElementComparator();
26. }
27. }
28.
```

**Step 8**: Run **TestSortingDemo** to see the result

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

Result:




**-- THE END --**