Practices for Lesson 7: Generics and Collections

Chapter 7

Practices for Lesson 7: Overview

Practices Overview

In these practices, use generics and collections to practice the concepts covered in the lecture. For each practice, a NetBeans project is provided for you. Complete the project as indicated in the instructions.

Practice 7-1: Summary Level: Counting Part Numbers by Using HashMaps

Overview

In this practice, use the HashMap collection to count a list of part numbers.

Assumptions

You have reviewed the collections section of this lesson.

Summary

You have been asked to create a simple program to count a list of part numbers that are of an arbitrary length. Given the following mapping of part numbers to descriptions, count the number of each part. Produce a report that shows the count of each part sorted by the part's product description. The part-number-to-description mapping is as follows:

Part Number	Description
1S01	Blue Polo Shirt
1S02	Black Polo Shirt
1H01	Red Ball Cap
1M02	Duke Mug

Once complete, your report should look like this:

```
=== Product Report ===
Name: Black Polo Shirt    Count: 6
Name: Blue Polo Shirt     Count: 7
Name: Duke Mug            Count: 3
Name: Red Ball Cap        Count: 5
```

Tasks

1. In NetBeans, open the `GenericsHashMap07-01Prac` project
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/07-Generics_Collections/practices/practice1`
   c. Select `GenericsHashMap07-01Prac` and click Open Project.
2. Expand the project directories.
3. Open `ProductCounter.java` in the editor and make the following changes:
   a. For the `ProductCounter` class, add two private map fields. The first map counts part numbers. The order of the keys does not matter. The second map stores the mapping of product description to part number. The keys should be sorted alphabetically by description for the second map.
   b. Create a one argument constructor that accepts a Map as a parameter. The map that stores the description-to-part-number mapping should be passed in here.

c. Create a `processList()` method to process a list of String part numbers. Use a HashMap to store the current count based on the part number.

```
public void processList(String[] list){ }
```

d. Create a `printReport()` method to print out the results.

```
public void printReport(){ }
```

e. Add code to the `main` method to create the `ProductCounter` object and process the same.

4. Run the `ProductCounter.java` class to ensure that your program produces the desired output.

Practice 7-1: Detailed Level: Counting Part Numbers by Using HashMaps

Overview

In this practice, use the HashMap collection to count a list of part numbers.

Assumptions

You have reviewed the collections section of this lesson.

Summary

You have been asked to create a simple program to count a list of part numbers that are of an arbitrary length. Given the following mapping of part numbers to descriptions, count the number of each part. Produce a report that shows the count of each part sorted by the part's product description. The part number to description mapping is as follows:

Part Number	Description
1S01	Blue Polo Shirt
1S02	Black Polo Shirt
1H01	Red Ball Cap
1M02	Duke Mug

Once complete, your report should look like this:

```
=== Product Report ===
Name: Black Polo Shirt    Count: 6
Name: Blue Polo Shirt     Count: 7
Name: Duke Mug            Count: 3
Name: Red Ball Cap        Count: 5
```

Tasks

1. In NetBeans, open the `GenericsHashMap07-01Prac` project.
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/07-Generics_Collections/practices/practice1`
   c. Select `GenericsHashMap07-01Prac` and click Open Project.
2. Expand the project directories.
3. Open `ProductCounter.java` in the editor and make the following changes:
   a. Add two private map fields- `productCountMap` and `productNames`. The first map counts part numbers. The order of the keys does not matter. The second map stores the mapping of product description to part number. The keys should be sorted alphabetically by description for the second map.

```
private Map<String, Long> productCountMap = new HashMap<>();
private Map<String, String> productNames = new TreeMap<>();
```

b. Create a one argument constructor that accepts a `Map` as a parameter.

```
    public ProductCounter(Map productNames){
        this.productNames = productNames;
    }
```

c. Create a `processList()` method to process a list of `String` part numbers. Use a `HashMap` to store the current count based on the part number.

```
    public void processList(String[] list){
        long curVal = 0;
        for(String itemNumber:list){
            if (productCountMap.containsKey(itemNumber)){
                curVal = productCountMap.get(itemNumber);
                curVal++;
                productCountMap.put(itemNumber, new
Long(curVal));
            } else {
                productCountMap.put(itemNumber,new Long(1));
            }
        }
    }
```

d. Create a `printReport()` method to print out the results.

```
    public void printReport(){
        System.out.println("=== Product Report ===");
        for (String key:productNames.keySet()){
            System.out.print("Name: " + key);
            System.out.println("\t\tCount: " +
productCountMap.get(productNames.get(key)));
        }
    }
```

4. Add the following code to the `main` method to create the `ProductCounter` object and process the same.

```
        ProductCounter pc1 = new ProductCounter (productNames);
        pc1.processList(parts);
        pc1.printReport();
```

5. Run the `ProductCounter.java` and verify the output.

```
run:
=== Product Report ===
Name: Black Polo Shirt          Count: 6
Name: Blue Polo Shirt           Count: 7
Name: Duke Mug                  Count: 3
Name: Red Ball Cap              Count: 5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practice 7-2: Summary Level: Implementing Stack using a `Deque`

Overview

In this practice, you use the `Deque` object to implement a Stack.

Assumptions

You have reviewed all the content in this lesson.

Summary

Use the `Deque` data structure to implement a stack to support `push`, `pop` and `peek` operations.

Tasks

1. In NetBeans, open the `Stack07-02Prac` project
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/07-Generics_Collections/practices/practice2`
   c. Select `Stack07-02Prac` and click Open Project.
2. Expand the project directories.
3. Open `IntegerStack.java` in the editor and make the following changes:
   a. Implement the `push()` method to add an Integer to the stack:
      Use the method `addFirst(element)` from the `Deque` API.
   b. Implement the `pop()` method that deletes an Integer from the top of the stack:
      Use the `removeFirst()` method from the `Deque` API, also check for stackunderflow condition before deleting the element by using `isEmpty()` method from the `Deque` API.
   c. Implement `peek()` method which returns the element at the top of the stack:
      Use the method `peekFirst()` from the `Deque` API.
   d. Override the `toString()` method.
4. Add a `main` method the class and perform the following steps:
   a. Create an instance of the Stack Class:

```
IntegerStack stack = new IntegerStack();
```

   b. Perform various operations on the `stack` by invoking various methods: `push()`, `pop()` and `peek()`.
5. Run the `IntegerStack.java` class to ensure that your program produces the desired output.

Practice 7-2: Detailed Level: Implementing Stack Using a `Deque`

Overview

In this practice, you use the `Deque` object to implement a Stack.

Assumptions

You have reviewed all the content in this lesson.

Summary

Use the `Deque` data structure to implement a stack to support `push, pop` and `peek` operations.

Tasks

1. In NetBeans, open the `Stack07-02Prac` project.
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/07-Generics_Collections/practices/practice2`
   c. Select `Stack07-02Prac` and click Open Project.
2. Expand the project directories.
3. Open `IntegerStack.java` in the editor and make the following changes:
4. Implement the `push()` method to add an Integer to the stack:

```
public void push(Integer element) {
        data.addFirst(element);
    }
```

5. Implement the `pop()` method that deletes an Integer from the top of the stack:

```
public Integer pop() {
      if(data.isEmpty())
      {
        System.out.print("Stack is empty");

    }
      return data.removeFirst();
    }
```

6. Implement the `peek` method():

```
public Integer peek() {
        return data.peekFirst();
    }
```

7. Override the `toString()` method:

```
public String toString() {
        return data.toString();
    }
```

8. Add a `main` method the class.
   a. Create an instance of the Stack Class:

```
IntegerStack stack = new IntegerStack();
```

   b. Perform various operations on the `stack` by invoking various methods: `push()`,`pop()` and `peek()`.

```
public static void main(String[] args) {
        IntegerStack stack = new IntegerStack();
        for (int i = 0; i < 5; i++) {
            stack.push(i);
        }
        System.out.println("After pushing 5 elements: " +
stack);

        int element = stack.pop();
        System.out.println("Popped element = " + element);

        System.out.println("After popping 1 element : " +
stack);


        int top = stack.peek();
        System.out.println("Peeked element = " + top);
        System.out.println("After peeking 1 element : " +
stack);
    }
```

9. Run the `IntegerStack.java` class to ensure that your program produces the desired output.

```
run:
After pushing 5 elements: [4, 3, 2, 1, 0]
Popped element = 4
After popping 1 element : [3, 2, 1, 0]
Peeked element = 3
After peeking 1 element : [3, 2, 1, 0]
BUILD SUCCESSFUL (total time: 2 seconds)
```