Practices for Lesson 6

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 6-1: 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

Open the `Generics-Practice01` project and make the following changes.

1. 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.
2. 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.
3. 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){ }
```

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

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

5. Add code to the `main` method to print out the results.
6. Run the `ProductCounter.java` class to ensure that your program produces the desired output.

Practice 6-1: 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

Open the `Generics-Practice01` project and make the following changes.

1. 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.

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

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

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

3. 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));
            }
        }
    }
```

4. 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)));
        }
    }
```

5. Add the following code to the `main` method to print out the results.

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

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

Practice 6-2: Summary Level: Matching Parentheses by Using a Deque

Overview

In this practice, you use the `Deque` object to match parentheses in a programming statement.

Assumptions

You have reviewed the collections section of this lesson.

Summary

Use the `Deque` data structure as a stack to match parentheses in a programming statement. You will be given several sample lines containing logical statements. Test the lines to ensure that the parentheses match, return true if they do, false if they do not.

For example, the output from the program might look like the following.

```
Line 0 is valid
Line 1 is invalid
Line 2 is invalid
Line 3 is valid
```

Tasks

Open the `Generics-Practice02` project and make the following changes.

1. Modify the `processLine()` method in `ParenMatcher.java` to read a line in and convert the string into a character array.
2. Loop through the array. Push "(" onto the stack. When a ")" is encountered, pop a "(" from the stack. Two conditions should return `false`.
    a. If you need to call a pop operation and the stack is empty, the number of parentheses do not match, return `false`.
    b. If after completing the loop a "(" is left on the stack return `false`. The number of parentheses does not match.
3. Run the `ParanMatcher.java` class to ensure that your program produces the desired output.

Practice 6-2: Detailed Level: Matching Parentheses by Using a Deque

Overview

In this practice, you use the `Deque` object to match parentheses in a programming statement.

Assumptions

You have reviewed the collections section of this lesson.

Summary

Use the `Deque` data structure as a stack to match parentheses in a programming statement. You will be given several sample lines containing logical statements. Test the lines to ensure that the parentheses match, return `true` if they do, `false` if they do not.

For example, the output from the program might look like the following.

```
Line 0 is valid
Line 1 is invalid
Line 2 is invalid
Line 3 is valid
```

Tasks

Open the `Generics-Practice02` project and make the following changes.

1. Modify the `processLine()` method in `ParenMatcher.java` to read a line in and convert the string into an array of characters. Clear the stack and convert the line to a character array.

```
        stack.clear();
        curLine = line.toCharArray();
```

2. In the same method, loop through the array. Push “(” onto the stack. When a “)” is encountered, pop a “(” from the stack. If “(” is left on the stack, or you attempt to perform a pop operation on an empty stack, the parentheses do not match, return `false`. Otherwise, return `true`. To do this, add the following code to the `processLine` method (replace the `return true;` statement).

```
        for (char c:curLine){
            switch (c){
                case '(':stack.push(c);break;
                case ')':{
                    if (stack.size() > 0){
                        stack.pop();
                    } else {
                        return false;
                    }
                    break;
                }
            }
        }
        if (stack.size()> 0){
           return false; // Missing match invalid expression
        } else {
           return true; //
        }
```

3. Run the `ParanMatcher.java` class to ensure that your program produces the desired output.

Practice 6-3: Summary Level: Counting Inventory and Sorting by Using Comparators

Overview

In this practice, you process shirt-related transactions for a Duke's Choice store. Compute the inventory level for a number of shirts. Then print out the shirt data sorted by description and by inventory count.

Assumptions

You have reviewed all the content in this lesson.

Summary

Any Duke's Choice stores carry a number of products including shirts. In this practice, process the shirt-related transactions and calculate the inventory levels. After the levels have been calculated, print a report sorted by description and a report sorted by inventory count. You will create two classes that implement the Comperator interface to allow sorting shirts by count and by description.

For example, the output from the program might look like the following.

```
=== Inventory Report - Description ===
Shirt ID: P002
Description: Black Polo Shirt
Color: Black
Size: M
Inventory: 15

Shirt ID: P001
Description: Blue Polo Shirt
Color: Blue
Size: L
Inventory: 24

Shirt ID: P003
Description: Maroon Polo Shirt
Color: Maroon
Size: XL
Inventory: 20
```

```
Shirt ID: P004
Description: Tan Polo Shirt
Color: Tan
Size: S
Inventory: 19

=== Inventory Report - Count ===
Shirt ID: P002
Description: Black Polo Shirt
Color: Black
Size: M
Inventory: 15

Shirt ID: P004
Description: Tan Polo Shirt
Color: Tan
Size: S
Inventory: 19

Shirt ID: P003
Description: Maroon Polo Shirt
Color: Maroon
Size: XL
Inventory: 20

Shirt ID: P001
Description: Blue Polo Shirt
Color: Blue
Size: L
Inventory: 24
```

Tasks

Open the `Generics-Practice03` project and make the following changes.

1. Review the `Shirt` class and `InventoryCount` interface to see how the `Shirt` class has changed to support inventory features.
2. Review the `DukeTransaction` class to see how transactions are defined for this program.
3. Update the `SortShirtByCount Comparator` class to sort shirts by count.
4. Update the `SortShirtByDesc Comparator` class to sort shirts by description.

5. Update the `TestItemCounter` class to process the shirts and transactions and produce the desired report.
   - Loop through the transactions and update the appropriate shirt object contained in the `polos map`.
   - Each `Shirt` class implements the `InventoryCount` interface. Use those methods and the `count` field to increment and decrement the inventory levels.
   - Print the list of shirts by description.
   - Print the list of shirts by count
6. Run the `TestItemCounter.java` class to ensure that your program produces the desired output.

Practice 6-3: Detailed Level: Counting Inventory and Sorting by Using Comparators

Overview

In this practice, you process shirt-related transactions for a Duke's Choice store. Compute the inventory level for a number of shirts. Then print out the shirt data sorted by description and by inventory count.

Assumptions

You have reviewed all the content in this lesson.

Summary

Any Duke's Choice stores carry a number of products including shirts. In this practice, process the shirt-related transactions and calculate the inventory levels. Once the levels have been calculated print a report sorted by description and a report sorted by inventory count. You will create two classes that implement the Comperator interface to allow sorting shirts by count and by description.

For example, the output from the program might look like the following.

```
=== Inventory Report - Description ===
Shirt ID: P002
Description: Black Polo Shirt
Color: Black
Size: M
Inventory: 15

Shirt ID: P001
Description: Blue Polo Shirt
Color: Blue
Size: L
Inventory: 24

Shirt ID: P003
Description: Maroon Polo Shirt
Color: Maroon
Size: XL
Inventory: 20
```

```
Shirt ID: P004
Description: Tan Polo Shirt
Color: Tan
Size: S
Inventory: 19

=== Inventory Report - Count ===
Shirt ID: P002
Description: Black Polo Shirt
Color: Black
Size: M
Inventory: 15

Shirt ID: P004
Description: Tan Polo Shirt
Color: Tan
Size: S
Inventory: 19

Shirt ID: P003
Description: Maroon Polo Shirt
Color: Maroon
Size: XL
Inventory: 20

Shirt ID: P001
Description: Blue Polo Shirt
Color: Blue
Size: L
Inventory: 24
```

Tasks

Open the `Generics-Practice03` project and make the following changes.

1. Review the `Shirt` class and `InventoryCount` interface to see how the `Shirt` class has changed to support inventory features.
2. Review the `DukeTransaction` class to see how transactions are defined for this program.

3. Update the `SortShirtByCount Comparator` class to sort shirts by count.

```
public class SortShirtByCount implements Comparator<Shirt>{
    public int compare(Shirt s1, Shirt s2){
        Long c1 = new Long(s1.getCount());
        Long c2 = new Long(s2.getCount());

        return c1.compareTo(c2);
    }
}
```

4. Update the `SortShirtByDesc Comparator` class to sort shirts by description.

```
public class SortShirtByDesc implements Comparator<Shirt>{
    public int compare(Shirt s1, Shirt s2){
        return
s1.getDescription().compareTo(s2.getDescription());
    }
}
```

5. Update the `TestItemCounter` class to process the shirts and transactions and produce the desired report.
   - Loop through the transactions and update the appropriate shirt object contained in the `polos Map`. This will produce an inventory count for each product.

```
// Count the shirts
for (DukeTransaction transaction:transactions){
    if (polos.containsKey(transaction.getProductID())){
        currentShirt = polos.get(transaction.getProductID());
    } else {
         System.out.println("Error: Invalid part number");
    }

    switch (transaction.getTransactionType()) {
        case "Purchase":currentShirt.
addItems(transaction.getCount()); break;

        case "Sale":currentShirt.
removeItems(transaction.getCount()); break;

       default: System.out.println("Error: Invalid Transaction
Type"); continue;
    }

}
```

- Print the list of shirts by description.

```
// Convert to List
List<Shirt> poloList = new ArrayList<>(polos.values());

// Init Comparators
Comparator sortDescription = new SortShirtByDesc();
Comparator sortCount = new SortShirtByCount();

// Print Results - Sort by Description
Collections.sort(poloList, sortDescription);
System.out.println("=== Inventory Report - Description ===");

for(Shirt shirt:poloList){
    System.out.println(shirt.toString());
}
```

- Print the list of shirts by count.

```
// Print Results - Sort by Count
Collections.sort(poloList, sortCount);
System.out.println("=== Inventory Report - Count ===");

for(Shirt shirt:poloList){
    System.out.println(shirt.toString());
}
```

6. Run the `TestItemCounter.java` class to ensure that your program produces the desired output.