MARKING REPORT

**Group number: 14**

|  |  |
| --- | --- |
| Name | Student ID |
| 1. Chong Kai Wei | 2002318 |
| 2. Yoon Wei Kai | 2003219 |
| 3. Lee Boon Hao | 2106860 |
| 4. E Zhong Lin | 2001777 |

**Marks breakdown**

Part A: Test Plan (10 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Test objective, scope and test basis | 5 |  |  |
| Test condition, entry and exit criteria | 5 |  |  |

Part B: Test Design (20 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Decision table | 5 |  |  |
| Appropriateness of test cases | 15 |  |  |

Part C: Java Program (application code and test code) (70 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Setup jar file location to C:\ jar\_files | 2 |  |  |
| Source directories | 3 |  |  |
| Appropriate used of assertsXXX methods. | 10 |  |  |
| Using parameterised tests correctly | 10 |  |  |
| Invalid values are checked for in implemented code, and tests for invalid values are performed. | 10 |  |  |
| Use of mocks or stubs for testing. | 10 |  |  |
| Combining test cases into test suites | 5 |  |  |
| Setting up some tests so that test values are read from a text file instead of hardcoding into test code | 10 |  |  |
| Perform integration testing after unit tests have been completed | 10 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A: | B: | C: | Total:  /100 | /20 |

Table of Contents

[Document Control 6](#_Toc133229094)

[Version History 7](#_Toc133229095)

[Part A: Test Plan 8](#_Toc133229096)

[1.0 Introduction 8](#_Toc133229097)

[1.1 Objective 8](#_Toc133229098)

[1.2 Scope 8](#_Toc133229099)

[1.3 Reference 8](#_Toc133229100)

[2.0 Test Plans 9](#_Toc133229101)

[2.1 Test Items 9](#_Toc133229102)

[2.2 Features to be Tested 10](#_Toc133229103)

[2.3 Features not to be Tested 12](#_Toc133229104)

[2.4 Test Basis 13](#_Toc133229105)

[2.4.1 Requirements 13](#_Toc133229106)

[R1- Order Module 13](#_Toc133229107)

[R2- Charge Module 14](#_Toc133229108)

[R3- Printing Module 15](#_Toc133229109)

[2.4.2 Test Case 16](#_Toc133229110)

[2.4.3 Decision Table 16](#_Toc133229111)

[2.5 Test Condition 17](#_Toc133229112)

[Ordering process 17](#_Toc133229113)

[Charging process: 17](#_Toc133229114)

[Printing process: 19](#_Toc133229115)

[2.6 Entry Criteria 20](#_Toc133229116)

[2.7 Exit Criteria 20](#_Toc133229117)

[Part B: Test Design 21](#_Toc133229118)

[B1. Decision Table 21](#_Toc133229119)

[B2. Test Case 28](#_Toc133229140)

[Appendix 29](#_Toc133229141)

[Class Diagram 29](#_Toc133229142)

[Application Code 30](#_Toc133229143)

[JUnit Test Code 50](#_Toc133229144)

## **Document Control**

|  |  |
| --- | --- |
| **Document Name** | Test Plan |
| **Reference Number** | - |
| **Version** | V1.3 |
| **Project Code** | UECS2354 Software Testing Assignment |
| **Status** | [to be updated] |
| **Date Released** | [to be updated] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Position** | **Signature** |
| Prepared By: | **Chong Kai Wei** | **System Developer** |  |
| Reviewed By: | **Yoon Wei Kai** | **Software Tester** |  |
| Approved By: | **Lee Boon Hao** | **Software Tester** | **A close-up of a signature  Description automatically generated with medium confidence** |
| Approved By: | **E Zhong Lin** | **Software Tester** | **A close-up of a black text  Description automatically generated with low confidence** |

# 

## **Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Release Date** | **Section** | **Amendments** |
| V1.0 | 6/4/2023 | All | Original Document |
| V1.1 | 9/4/2023 | Charge, MainApp, Order | improve the structure of the system |
| V1.2 | 15/4/2023 | Charge, Printing | errors found in printing request is fixed |
| V1.3 | 15/4/2023 | MainApp | fixed error validation issues fond in getCustomerOrder() |

## **Part A: Test Plan**

### 1.0 Introduction

#### 1.1 Objective

The test objectives are mainly to verify the functionality of Printing Kiosk System which will allow the user to make orders by choosing the type, option, quantity, and additional option, and calculate the total price of the order before proceeding to printing process.

The project should focus on the two main modules in the system which are:

1. Ordering operation – perform ordering features such as choosing the order type, order option, order quantity, and other additional option.
2. Calculate order operation – calculate the total prize of the orders.
3. \* Printing operation – perform printing for the order after the system has perform the calculate order operation

Assumption:

The printing operation will only perform as a mock to proceed with the printing process, the system will connect to a third-party developed printing module in the real scenario.

#### 1.2 Scope

The scope of testing only covers black box-functional testing for features developed in this Printing Kiosk System. This test plan only covers unit test, integration test, and regression test, excluding acceptance test, confirmation test as well as other type of test that are not mentioned in this Test Plan. It would focus on meeting the requirement of the system and making sure that each of the function in the system would function correctly.

#### 1.3 Reference

1. 202301\_UECS2354\_Assignment\_Details.pdf

### 2.0 Test Plans

#### 2.1 Test Items

The following table contains the test items. Listed together are the modules and its descriptions.

|  |  |
| --- | --- |
| Module | Description |
| Order Module | The main validation in this module is to make sure that the order made by the user is valid. Besides that, in this module, we will also test that the information pass to the charge module is correct. |
| Charge Module | The main validation in this module is to make sure that the calculation for order is correct and valid as well as make sure that the printing requests pass to the printing module is correct. |
| Printing Module | The main validation in this module is to make sure that the printing requests received from the charge module is correct. |

#### 2.2 Features to be Tested

The following table contains the features to be tested. Listed together are the function id and its corresponding functions, corresponding modules, and estimated risk level.

|  |  |  |
| --- | --- | --- |
| Module | Feature | Description |
| Order | Set Type | The main validation is to make sure that the type set in an order is valid. Exception will be thrown if invalid values is entered by the user. |
| Set Option | The main validation is to make sure that the option set in an order is valid. Exception will be thrown if invalid values is entered by the user. |
| Set Quantity | The main validation is to make sure that the quantity set in an order is valid. Exception will be thrown if invalid values is entered by the user. |
| Set Add Option 1 | The main validation is to make sure that the additional option 1 (high design paper) set in an order is valid. Exception will be thrown if invalid values is entered by the user. |
| Set Add Option 2 | The main validation is to make sure that the additional option 2 (design effect) set in an order is valid. Exception will be thrown if invalid values is entered by the user. |
| Add Customer Order | The main validation is to make sure that the order that has been created is correct and valid before adding it to the array list. |
| Charge | Calculate Document Charge | The main validation is to verify that the calculation for document type order (if the user has only one order) is correct. |
| Calculate Photo Charge | The main validation is to verify that the calculation for photo type order (if the user has only one order) is correct. |
| Get Total Charge | The main validation is to verify that the calculation for multiple orders (different type) is correct. |
| Pass Printing Requests | The main validation is to make sure that the printing requests will be arranged according to the rules\*. |
| Printing | Queue Request | The main validation is to make sure that the printing requests will be received and it is correct. |

Assumption:

1. Rule for passing the printing requests is as follow: If the user has made multiple order requests with the same type, option, additional option 1 and additional option 2, the system will pass the order requests as one printing requests. For example: if the user has made orders as follow:
   1. 5 documents in black & white.
   2. 7 documents in colour.
   3. 10 documents in black & white.
2. When the charge module passes the information to the printing module, it will pass the printing requests as follow:
   1. 7 documents in colour.
   2. 15 document is black & white.

#### 2.3 Features not to be Tested

Aside from the features listed in Section 2.2 Features to be tested, other aspects of the application are not covered in this test plan. This includes:

1. Hardware compatibility

It might not be necessary to test the printing kiosk system's compatibility with other hardware if it is built to function with particular hardware parts, like printers.

1. Operating system compatibility

It might not be essential to test the printing kiosk system's compatibility with various operating systems if it is built to work with a certain set of operating systems.

1. Network connectivity

It might not be essential to test network connectivity capabilities if the printing kiosk system is not meant to be used on a network.

1. Third-party developed printing module

The actual interactions between the charge module and the printing module will not be tested since the printing module is developed by the third party. A printing class named will act as mock when we test the system to make sure that the printing requests can be passed to the printing module.

#### 2.4 Test Basis

The test basis of a kiosk printing system may include the software requirements, test cases and decision table.

##### 2.4.1 Requirements

###### R1- Order Module

1.1 The system shall allow user to choose the order type. (Document/ Photo)

1.1.1 If user enter invalid input, error message is displayed, and user is prompted to perform

input again.

1.1.2 If user enter valid input, system shall allow user to proceed to step 1.2.

1.2 The system shall allow user to choose option (between 1 to 2) for printing.

1.2.1 If user enter invalid input, error message is displayed, and user is prompted to perform

input again.

1.2.2 If user enter valid input, system shall allow user to proceed to step 1.3.

1.3. The system shall allow user to enter quantity (between 1 to 50) for printing.

1.3.1. If user enter invalid input error message is displayed and user is prompted to perform

input again.

1.3.2. System shall allow user to proceed to 1.4 if the input for 1.1 is “P” (“Photo”)

1.3.3. System shall proceed to step 1.6 if the input for 1.1 is “D” (“Document”)\*\*.

1.4 The system shall allow user whether want to choose high quality paper.

1.4.1 If user enter invalid input error message is displayed and user is prompted to perform

input again.

1.4.2 If user enter valid input, system shall allow user to proceed to 1.5.

1.5 The system shall allow user to choose whether want to have design effect.

1.5.1 If user enter invalid input, error message is displayed, and user is prompted to perform

input again.

1.5.2 If the user enters valid input, the system will proceed to Step 1.6.

1.6 The system shall add the order has been made to an order list, then the system will proceed to

Step 1.7.

1.7 The system shall allow the user to choose whether want to make another order.

1.7.1 If the user enters invalid input, error message is displayed, and the user is prompted to

perform input again.

* + 1. If the user wants to add another order, the system shall repeat the Step 1.1 to 1.7.
    2. If the user does not want to add another order, the system will proceed to Step 1.8.

1.8 System will pass the orders received to the Charge Module.

\*\*Extra information for 1.3.3, the system will automatically set the additional option 1 (for high design paper effect) and the additional option 2 (for design effect) to “N” when the user chooses to print document as the document order will not have this option to choose.

###### R2- Charge Module

2.1 The system shall calculate the Charge of a set of order requests.

2.1.1 if the input for 1.1 is “D” (for Document) and 1.2 is 1 (for Black & White).

2.1.1.1 if the quantity input is 1-4, the charge will be multiplied by 0.50.

2.1.1.2 if the quantity input is 5-10, the charge will be multiplied by 0.40.

2.1.1.3 if the quantity input is 11-20, the charge will be multiplied by 0.30.

2.1.1.4 if the quantity input is 21-50, the charge will be multiplied by 0.20.

2.1.2 if the input for 1.1 is “D” (for Document) and 1.2 is 2 (for Color).

2.1.2.1 if the quantity input is 1-4, the charge will be multiplied by 1.00.

2.1.2.2 if the quantity input is 5-10, the charge will be multiplied by 0.90.

2.1.2.3 if the quantity input is 11-20, the charge will be multiplied by 0.80.

2.1.2.4 if the quantity input is 21-50, the charge will be multiplied by 0.70.

2.1.4 if the input for 1.1 is “P” (for Photo) and 1.2 is 1 (for Normal (4R)).

2.1.3.1 if the quantity input is 1-4, the charge will be multiplied by 1.00.

2.1.3.2 if the quantity input is 5-10, the charge will be multiplied by 0.90.

2.1.3.3 if the quantity input is 11-20, the charge will be multiplied by 0.75.

2.1.3.4 if the quantity input is 21-50, the charge will be multiplied by 0.50.

2.1.4 if the input for 1.1 is “P” (for Photo) and 1.2 is 2 (for Passport).

2.1.4.1 if the quantity input is 1-4, the charge will be multiplied by 1.20.

2.1.4.2 if the quantity input is 5-10, the charge will be multiplied by 0.95.

2.1.4.3 if the quantity input is 11-20, the charge will be multiplied by 0.85.

2.1.4.4 if the quantity input is 21-50, the charge will be multiplied by 0.75.

2.1.6 If the input for 1.1 is “P” (for Photo) and 1.4 is “Y” (for High Design Paper effect),

add RM0.10 per piece for (2.1.3/2.1.4).

2.1.7 If the input for 1.1 is “P” (for Photo) and 1.5 is “Y” (for Design effect), add RM0.15

per piece for (2.1.3/2.1.4).

2.2 The system shall repeat 2.1 until all order is completely calculated.

2.3 The system shall show the orders that have been made in a table (include sub charge for each

order).

2.4 The system will show the total charge of the orders (in RM).

2.5 The system will pass the orders (in the form of array list) to the printing module.

2.5.1 System will copy the order list that has been made to another order list.

2.5.2 System will compare each of the orders that has been made. If there is two order that

has same type, option, additional option 1, and additional option 2. The system

will delete the two orders in the order list and add another order. This order will have

the same type, option, additional option 1, and additional option 2 as the two orders.

However, the quantity of the order will be the sum of the two orders before.

2.5.3 System will compare each of the orders until each of the orders do not have the same

type, option, additional option 1, and additional option 2.

###### R3- Printing Module

* 1. System shall receive the printing requests (in the form of array list) from the charge module.
  2. System will show the printing requests in the form of table. (without including the sub charge of each order).
  3. System will also show the total number of printing requests after it show the table.

Assumptions

1. The printing kiosk system will accept payment through cash, credit card, or any other payment method supported by the system.
2. The printing kiosk system will have a database to store clients’ orders and payment information, which can be retrieved and managed by authorized users.
3. The printing kiosk system will be able to notify clients of any errors or issues that may occur during the printing process, such as low ink or paper jams.
4. Invalid values entered by the user will be handled in the ordering process, and exception will be thrown in the order module. Since charge module will receive the order information from the order module and it will pass the printing request information to the printing module, so charge module and printing module no need to handle error that may be occurred during the order process.

##### 2.4.2 Test Case

Kiah Ordering System test case is a set of actions executed to verify a feature or functionality of a software application primarily based on the requirements.

*(may refer to Part B – Test Design)*

##### 2.4.3 Decision Table

Decision tables are a concise visual representation for specifying which actions to perform depending on given conditions.

(may refer to Part B – Test Design)

#### 2.5 Test Condition

##### Ordering process

1. When user chooses order type, the order type must be “d” or “D” for document and “p” or “P” for photo.
2. When the user chooses the order option:
   1. When the order type is “Document”, the option must be 1(Black & White) or 2(Color).
   2. When the order type is “Photo”, the option must be 1(Normal(4R)) or 2(Passport)
3. When the user chooses the quantity to be printed: quantity for every request can only be 1 to 50.
4. When the user chooses “p” or “P” as the order type:
   1. The system shall let the user choose for a high-quality paper and the option available can only be “Y”, “y” or “N”, “n”.
   2. The system shall let the user choose for a design effect and the option available can only be “Y”, “y” or “N”, “n”.
5. When the user chooses “d” or “D” as the order type:
   1. The system will automatically set the option for high-quality paper and design effect to “N”.
6. When the process of 1 to 5 is finished, the system will create an order with the option chose by the user and add the order to a list, and the list will be passed to the next process.

##### Charging process:

1. List of orders will be received and will be calculated separately.
   1. If the order type is “D” (Document):
      1. If the option is 1 (for Black & White)
         1. If the quantity input is 1-4, the charge will be multiplied by 0.50 per piece.
         2. If the quantity input is 5-10, the charge will be multiplied by 0.40 per piece.
         3. If the quantity input is 11-20, the charge will be multiplied by 0.30 per piece.
         4. If the quantity input is 21-50, the charge will be multiplied by 0.20 per piece.
      2. If the option is 2 (for Color)
         1. If the quantity input is 1-4, the charge will be multiplied by 1.00 per piece.
         2. If the quantity input is 5-10, the charge will be multiplied by 0.90 per piece.
         3. If the quantity input is 11-20, the charge will be multiplied by 0.80 per piece.
         4. If the quantity input is 21-50, the charge will be multiplied by 0.70 per piece.
   2. If the order type is “P” (Photo)
      1. If the option is 1 (for Normal (4R)):
         1. If the quantity input is 1-4, the charge will be multiplied by 1.00 per piece.
         2. If the quantity input is 5-10, the charge will be multiplied by 0.90 per piece.
         3. If the quantity input is 11-20, the charge will be multiplied by 0.75 per piece.
         4. If the quantity input is 21-50, the charge will be multiplied by 0.50 per piece.
      2. If the option is 2 (for Passport)
         1. If the quantity input is 1-4, the charge will be multiplied by 1.20 per piece.
         2. If the quantity input is 5-10, the charge will be multiplied by 0.95 per piece.
         3. If the quantity input is 11-20, the charge will be multiplied by 0.85 per piece.
         4. If the quantity input is 21-50, the charge will be multiplied by 0.75 per piece.
      3. If the additional option 1 (for High Design Paper effect) is “Y”, add RM0.10 into charge per piece.
      4. If the additional option 2 (for Design effect) is “Y”, add RM0.15 into charge per piece.
2. List of orders will be arranged and copied to the list of printing requests.
   1. If there are two orders that have the same order type and order option (for document order), the two orders will be merge into one order and the quantity of the order will be the sum of the two orders before the two orders merged.
   2. If there are two orders that have the same order type, order option, additional option 1 and 2 (for document order), the two orders will be merge into one order and the quantity of the order will be the sum of the two orders before the two orders merged.
3. The list of printing requests will be passed to the next process.

##### Printing process:

1. The list of printing requests will be received, and it will be added to the array list in the printing module.

#### 2.6 Entry Criteria

This project must fulfil several test entry criteria before proceeding to the test phase.

The test entry criteria are as below:

1. Bug tracking and test tracking systems are in place.
2. The environment of the system test, including hardware components and subsystems, is configured and ready. (JDK 17, JUnit 4, Mockito 1.9.5, JParam 1.0.5, Eclipse).
3. All required test bases are ready and available.
4. All components are under formal, automated configuration and release management control.
5. The development teams have completed all features and bug fixes scheduled for release.
6. Test phase entry meeting is held, and the teams passed unanimous agreement to begin the test phase.

#### 2.7 Exit Criteria

Several exit criteria are to be met before the test phase can be deemed complete:

1. No changes (design/code/features), except to address System Test defects, occurred in the prior 1 weeks.
2. No panic, crash, halt, wedge, unexpected process termination, or other stoppage of processing has occurred on any server software or hardware for the previous 1 weeks.
3. The test team has executed all the planned tests.
4. The test team has checked that all issues in the bug tracking system are either closed or deferred and where appropriate, verified by regression and confirmation testing.
5. Test phase exit meeting is held, and the teams passed unanimous agreement to complete the test phase.
6. The team agrees to exit the testing phase in a Test Phase Exit Meeting.

## **Part B: Test Design**

### B1. Decision Table

#### Order Module

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Order Module | Function: Set Type | | | | |
| \*Rule | 1.1.1 | 1.1.2 | 1.1.3 | 1.1.4 | 1.1.5 |
| Type | Valid | Valid (in lowercase) | Invalid (null) | Invalid | Invalid (blank) |
|  | | | | | |
| \*Action |  | | | | |
| Throw Exception | F | F | T | T | T |
| Set the type to corresponding type | T | T | F | F | F |

|  |  |  |  |
| --- | --- | --- | --- |
| Order Module | Function: Set Option | | |
| \*Rule | 1.2.1 | 1.2.2 | 1.2.3 |
| Option | 1-2 | <1 | >2 |
|  | | | |
| \*Action |  | | |
| Throw Exception | F | T | T |
| Set the option to corresponding option | T | F | F |

|  |  |  |  |
| --- | --- | --- | --- |
| Order Module | Function: Set Quantity | | |
| \*Rule | 1.3.1 | 1.3.2 | 1.3.3 |
| Quantity | 1-50 | <1 | >50 |
|  | | | |
| \*Action |  | | |
| Throw Exception | F | T | T |
| Set the quantity to corresponding quantity | T | F | F |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Order Module | Function: Set Add Option 1 | | | | |
| \*Rule | 1.4.1 | 1.4.2 | 1.4.3 | 1.4.4 | 1.4.5 |
| Additional Option 1 | Valid | Valid (in lowercase) | Invalid (null) | Invalid | Invalid (blank) |
|  | | | | | |
| \*Action |  | | | | |
| Throw exception | F | F | T | T | T |
| Set the add option 1 to the corresponding option | T | T | F | F | F |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Order Module | Function: Set Add Option 2 | | | | |
| \*Rule | 1.5.1 | 1.5.2 | 1.5.3 | 1.5.4 | 1.5.5 |
| Additional Option 2 | Valid | Valid (in lowercase) | Invalid (null) | Invalid | Invalid (blank) |
|  | | | | | |
| \*Action |  | | | | |
| Throw exception | F | F | T | T | T |
| Set the add option 2 to the corresponding option | T | T | F | F | F |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Order Module | Function: Add Customer Order | | | | | | | | | | |
| \*Rule | 2.1.1 | 2.1.2 | 2.1.3-2.1.5 | 2.1.6 | 2.1.7 | 2.1.8 | 2.1.9 | 2.1.10-2.1.12 | 2.1.13-2.1.14 | 2.1.15 | 2.1.16 |
| Order | Valid | invalid | valid | valid | valid | valid | valid | valid | valid | valid | valid |
| Type | Valid | ‒ | invalid | valid | valid | valid | valid | valid | valid | D | D |
| Option | 1‒2 | ‒ | 1‒2 | <1 | >2 | 1‒2 | 1‒2 | 1‒2 | 1‒2 | 1‒2 | 1‒2 |
| Quantity | 1‒50 | ‒ | 1‒50 | 1‒50 | 1‒50 | <1 | >50 | 1‒50 | 1‒50 | 1‒50 | 1‒50 |
| Add option 1 | Valid | ‒ | valid | valid | valid | valid | valid | invalid | valid | Y | valid |
| Add option 2 | Valid | ‒ | valid | valid | valid | valid | valid | valid | invalid | valid | Y |
|  |  |  |  |  |  |  |  |  |  |  |  |
| \*Action |  |  |  |  |  |  |  |  |  |  |  |
| Throw Exception | F | T | T | T | T | T | T | T | T | T | T |
| Add Order To List | T | F | F | F | F | F | F | F | F | F | F |

#### Charge Module

|  |  |  |
| --- | --- | --- |
| Charge Module | Function: Calculate Document Charge (Part 1) | |
| Condition |  | |
| Option (T for Black & White, F for Color) | T | F |
|  | | |
| Action | Refer to Calculate Document Charge (Part 2) | Refer to Calculate Document Charge (Part 3) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Document Charge (Part 2) (For Black & White) | | | | | | | |
| \*Rule | 1.1.1 | 1.1.2 | 1.1.3 | 1.1.4 | 1.1.5 | 1.1.6 | 1.1.7 | 1.1.8 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*0.5 | 4\*0.5 | 5\*0.4 | 10\*0.4 | 11\*0.3 | 20\*0.3 | 21\*0.2 | 50\*0.2 |
|  | | | | | | | | |
| Expected Charge | 0.5 | 2 | 2 | 4 | 3.3 | 6 | 4.2 | 10 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Document Charge (Part 3) (For Color) | | | | | | | |
| \*Rule | 1.1.9 | 1.1.10 | 1.1.11 | 1.1.12 | 1.1.13 | 1.1.14 | 1.1.15 | 1.1.16 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1 | 4\*1 | 5\*0.9 | 10\*0.9 | 11\*0.8 | 20\*0.8 | 21\*0.7 | 50\*0.7 |
|  | | | | | | | | |
| Expected Charge | 1 | 4 | 4.5 | 9 | 8.8 | 16 | 14.7 | 35 |

|  |  |  |
| --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 1) | |
| Condition |  | |
| Option (T for Normal (4R), F for Passport) | T | F |
|  | | |
| Action | Refer to Calculate Photo Charge (Part 2 – 5) | Refer to Calculate Photo Charge (Part 6 – 9) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 2) (For Normal(4R)) | | | | | | | |
| \*Rule | 2.1.1 | 2.1.2 | 2.1.3 | 2.1.4 | 2.1.5 | 2.1.6 | 2.1.7 | 2.1.8 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | F | F | F | F | F | F | F | F |
| Additional Option 2 | F | F | F | F | F | F | F | F |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1 | 4\*1 | 5\*0.9 | 10\*0.9 | 11\*0.75 | 20\*0.75 | 21\*0.5 | 50\*0.5 |
|  | | | | | | | | |
| Expected Charge | 1 | 4 | 4.5 | 9 | 8.25 | 15 | 10.5 | 25 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 3) (For Normal (4R)) (HD) | | | | | | | |
| \*Rule | 2.1.9 | 2.1.10 | 2.1.11 | 2.1.12 | 2.1.13 | 2.1.14 | 2.1.15 | 2.1.16 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | T | T | T | T | T | T | T | T |
| Additional Option 2 | F | F | F | F | F | F | F | F |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1 | 4\*1 | 5\*0.9 | 10\*0.9 | 11\*0.75 | 20\*0.75 | 21\*0.5 | 50\*0.5 |
| Extra charge | 1\*0.1 | 4\*0.1 | 5\*0.1 | 10\*0.1 | 11\*0.1 | 20\*0.1 | 21\*0.1 | 50\*0.1 |
|  | | | | | | | | |
| Expected Charge | 1.1 | 4.4 | 5 | 10 | 9.35 | 17 | 12.6 | 30 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 4) (For Normal (4R)) (DE) | | | | | | | |
| \*Rule | 2.1.17 | 2.1.18 | 2.1.19 | 2.1.20 | 2.1.21 | 2.1.22 | 2.1.23 | 2.1.24 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | F | F | F | F | F | F | F | F |
| Additional Option 2 | T | T | T | T | T | T | T | T |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1 | 4\*1 | 5\*0.9 | 10\*0.9 | 11\*0.75 | 20\*0.75 | 21\*0.5 | 50\*0.5 |
| Extra charge | 1\*0.15 | 4\*0.15 | 5\*0.15 | 10\*0.15 | 11\*0.15 | 20\*0.15 | 21\*0.15 | 50\*0.15 |
|  | | | | | | | | |
| Expected Charge | 1.15 | 4.6 | 5.25 | 10.5 | 9.9 | 18 | 13.65 | 32.5 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 5) (For Normal (4R)) (HD + DE) | | | | | | | |
| \*Rule | 2.1.25 | 2.1.26 | 2.1.27 | 2.1.28 | 2.1.29 | 2.1.30 | 2.1.31 | 2.1.32 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | T | T | T | T | T | T | T | T |
| Additional Option 2 | T | T | T | T | T | T | T | T |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1 | 4\*1 | 5\*0.9 | 10\*0.9 | 11\*0.75 | 20\*0.75 | 21\*0.5 | 50\*0.5 |
| Extra charge | 1\*0.25 | 4\*0.25 | 5\*0.25 | 10\*0.25 | 11\*0.25 | 20\*0.25 | 21\*0.25 | 50\*0.25 |
|  | | | | | | | | |
| Expected Charge | 1.25 | 5 | 5.75 | 11.5 | 11 | 20 | 15.75 | 37.5 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 6) (For Passport) | | | | | | | |
| \*Rule | 2.2.1 | 2.2.2 | 2.2.3 | 2.2.4 | 2.2.5 | 2.2.6 | 2.2.7 | 2.2.8 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | F | F | F | F | F | F | F | F |
| Additional Option 2 | F | F | F | F | F | F | F | F |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1.2 | 4\*1.2 | 5\*0.95 | 10\*0.95 | 11\*0.85 | 20\*0.85 | 21\*0.75 | 50\*0.75 |
|  | | | | | | | | |
| Expected Charge | 1.2 | 4.8 | 4.75 | 9.5 | 9.35 | 17 | 15.75 | 37.5 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 7) (For Passport) (HD) | | | | | | | |
| \*Rule | 2.2.9 | 2.2.10 | 2.2.11 | 2.2.12 | 2.2.13 | 2.2.14 | 2.2.15 | 2.2.16 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | T | T | T | T | T | T | T | T |
| Additional Option 2 | F | F | F | F | F | F | F | F |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1.2 | 4\*1.2 | 5\*0.95 | 10\*0.95 | 11\*0.85 | 20\*0.85 | 21\*0.75 | 50\*0.75 |
| Extra charge | 1\*0.1 | 4\*0.1 | 5\*0.1 | 10\*0.1 | 11\*0.1 | 20\*0.1 | 21\*0.1 | 50\*0.1 |
|  | | | | | | | | |
| Expected Charge | 1.3 | 5.2 | 5.25 | 10.5 | 10.45 | 19 | 17.85 | 42.5 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 8) (For Passport) (DE) | | | | | | | |
| \*Rule | 2.2.17 | 2.2.18 | 2.2.19 | 2.2.20 | 2.2.21 | 2.2.22 | 2.2.23 | 2.2.24 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | F | F | F | F | F | F | F | F |
| Additional Option 2 | T | T | T | T | T | T | T | T |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1.2 | 4\*1.2 | 5\*0.95 | 10\*0.95 | 11\*0.85 | 20\*0.85 | 21\*0.75 | 50\*0.75 |
| Extra charge | 1\*0.15 | 4\*0.15 | 5\*0.15 | 10\*0.15 | 11\*0.15 | 20\*0.15 | 21\*0.15 | 50\*0.15 |
|  | | | | | | | | |
| Expected Charge | 1.35 | 5.4 | 5.5 | 11 | 11 | 20 | 18.9 | 45 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Charge Module | Function: Calculate Photo Charge (Part 9) (For Passport) (HD + DE) | | | | | | | |
| \*Rule | 2.2.25 | 2.2.26 | 2.2.27 | 2.2.28 | 2.2.29 | 2.2.30 | 2.2.31 | 2.2.32 |
| Quantity | 1 | 4 | 5 | 10 | 11 | 20 | 21 | 50 |
| Additional Option 1 | T | T | T | T | T | T | T | T |
| Additional Option 2 | T | T | T | T | T | T | T | T |
|  | | | | | | | | |
| \*Action |  | | | | | | | |
| Corresponding charge | 1\*1.2 | 4\*1.2 | 5\*0.95 | 10\*0.95 | 11\*0.85 | 20\*0.85 | 21\*0.75 | 50\*0.75 |
| Extra charge | 1\*0.25 | 4\*0.25 | 5\*0.25 | 10\*0.25 | 11\*0.25 | 20\*0.25 | 21\*0.25 | 50\*0.25 |
|  | | | | | | | | |
| Expected Charge | 1.45 | 5.8 | 6 | 12 | 12.1 | 22 | 21 | 50 |

### B2. Test Case

May refer to “Test\_Case.xlsx”

## **Appendix**

### Class Diagram

May refer to “cd222.eapx”

A picture containing text, diagram, screenshot, parallel

Description automatically generated

Assumption:

1. Display class created to show the message when throwing an IllegalArgumentException.
2. MainApp class created for the developer to test run the code.
3. NewOrder, NewCharge, NewPrinting class and OrderFunctionality, PrintingFunctionality interfaces was used for creating test doubles and for testing purposes.

### Application Code

Order class

**package** my.edu.utar;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** Order **implements** OrderFunctionality{

**private** String type;

**private** **int** option;

**private** **int** quantity;

**private** String addOption1;

**private** String addOption2;

**private** List <Order> orderList = **new** ArrayList <Order>();

**public** Order () {

}

**public** Order (String type, **int** option, **int** quantity, String addOption1, String addOption2) {

**this**.type = type;

**this**.option = option;

**this**.quantity = quantity;

**this**.addOption1 = addOption1;

**this**.addOption2 = addOption2;

}

**public** Order(Order[] orders) {

orderList = Arrays.*asList*(orders);

}

**public** String getType() {

**return** type;

}

**public** **int** getOption () {

**return** option;

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** String getAddOption1() {

**return** addOption1;

}

**public** String getAddOption2() {

**return** addOption2;

}

**public** **void** setType(String type) {

**if** (type == **null**)

**throw** **new** IllegalArgumentException("Type cannot be null\n");

type = type.toUpperCase();

**if** (!type.equals("D")&& !type.equals("P"))

**throw** **new** IllegalArgumentException("Invalid type. Please enter again.\n");

**this**.type = type;

}

**public** **void** setOption (**int** option) {

**if** (option < 1 || option > 2)

**throw** **new** IllegalArgumentException ("Invalid option. Please enter again.\n");

**this**.option = option;

}

**public** **void** setQuantity (**int** quantity) {

**if** (quantity < 1 || quantity > 50)

**throw** **new** IllegalArgumentException ("Invalid quantity. Please enter again.\n");

**this**.quantity = quantity;

}

**public** **void** setAddOption1 (String addOption1) {

**if** (addOption1 == **null**)

**throw** **new** IllegalArgumentException ("addOption1 cannot be null.\n");

addOption1 = addOption1.toUpperCase();

**if** (!addOption1.equals("Y")&& !addOption1.equals("N"))

**throw** **new** IllegalArgumentException("Invalid choice. Please enter again.\n");

**this**.addOption1 = addOption1;

}

**public** **void** setAddOption2 (String addOption2) {

**if** (addOption2 == **null**)

**throw** **new** IllegalArgumentException ("addOption2 cannot be null.\n");

addOption2 = addOption2.toUpperCase();

**if** (!addOption2.equals("Y")&& !addOption2.equals("N"))

**throw** **new** IllegalArgumentException("Invalid choice. Please enter again.\n");

**this**.addOption2 = addOption2;

}

**public** **void** addCustomerOrder(Order anOrder) {

**if** (anOrder == **null**)

**throw** **new** IllegalArgumentException ("Order cannot be null");

**if** (anOrder.getType() == **null** || (!anOrder.getType().equals("D") && !anOrder.getType().equals("P")))

**throw** **new** IllegalArgumentException ("Order type is incorrect");

**if** (anOrder.getOption() < 1 || anOrder.getOption() > 2)

**throw** **new** IllegalArgumentException ("Order option is invalid");

**if** (anOrder.getQuantity() < 1 || anOrder.getQuantity() > 50)

**throw** **new** IllegalArgumentException ("Order quantity is out of range");

**if** (anOrder.getAddOption1() == **null** || (!anOrder.getAddOption1().equals("Y") && !anOrder.getAddOption1().equals("N")))

**throw** **new** IllegalArgumentException ("Order additional option 1 is incorrect");

**if** (anOrder.getAddOption2() == **null** || (!anOrder.getAddOption2().equals("Y") && !anOrder.getAddOption2().equals("N")))

**throw** **new** IllegalArgumentException ("Order additional option 2 is incorrect");

**if** ((anOrder.getType().equals("D") && anOrder.getAddOption1().equals("Y")) || (anOrder.getType().equals("D") && anOrder.getAddOption2().equals("Y")))

**throw** **new** IllegalArgumentException("Order with Document cannot have high quality paper effect and design effect");

orderList.add(anOrder);

}

**public** Order[] getOrderList(){

Order [] orders = orderList.toArray(**new** Order[orderList.size()]);

**return** orders;

}

}

Charge class

**package** my.edu.utar;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** Charge {

Order o;

Printing print;

List <Order> orderList;

List <Order> printList;

**public** Charge() {

}

**public** Charge (Order order, Printing print) {

**this**.o = order;

**this**.print = print;

}

**public** Order[] getPrintList(){

Order [] prints = printList.toArray(**new** Order[printList.size()]);

**return** prints;

}

**public** **void** setPrintList (List <Order> printList) {

**this**.printList = printList;

}

**public** **double** getTotalCharge () {

Order [] orders = o.getOrderList();

orderList = Arrays.*asList*(orders);

**double** total = 0;

**double** price;

**for** (**int** i = 0; i < orderList.size(); i++) {

Order anOrder = orderList.get(i);

//count Total Charge part

**if** (anOrder.getType().equals("D") || anOrder.getType().equals("d"))

price = calDocCharge(anOrder);

**else**

price = calPhotoCharge(anOrder);

total += price;

price = 0;

}

total = Math.*round*(total\*100.0)/100.0;

**return** total;

}

**public** **double** calDocCharge(Order anOrder) {

**double** price = 0;

**if** (anOrder.getOption() == 1) {

**if** (anOrder.getQuantity()> 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 0.5;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.4;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.3;

**else**

price = anOrder.getQuantity() \* 0.2;

}

**else** {

**if** (anOrder.getQuantity() > 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 1;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.9;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.8;

**else**

price = anOrder.getQuantity() \* 0.7;

}

**return** price;

}

**public** **double** calPhotoCharge (Order anOrder) {

**double** price = 0;

**if** (anOrder.getOption() == 1) {

**if** (anOrder.getQuantity() > 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 1;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.9;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.75;

**else**

price = anOrder.getQuantity() \* 0.5;

}

**else** {

**if** (anOrder.getQuantity() > 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 1.2;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.95;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.85;

**else**

price = anOrder.getQuantity() \* 0.75;

}

**if** (anOrder.getAddOption1().equals("Y") || anOrder.getAddOption1().equals("y"))

price = price + anOrder.getQuantity() \* 0.1;

**if** (anOrder.getAddOption2().equals("Y") || anOrder.getAddOption2().equals("y"))

price = price + anOrder.getQuantity() \* 0.15;

price = Math.*round*(price\*100.0)/100.0;

**return** price;

}

/\*

\* Steps to arrange the orderList as printing request list

\* 1. Copy the orderList to another array list

\* 2. Compare the orders in the array list

\* 3. If there are two orders same for document, if the option is the same:

\* 3.1 Get the quantity of the two orders.

\* 3.2 Remove the two orders from the array list

\* 3.3 Add another orders to the array list, with the type, and option is the same as the two orders,

\* and the quantity of the order sum of the two orders

\* 4. If there are two orders same for photo, if the option, addOption1, and addOption2 are same:

\* 3.1 Get the quantity of the two orders.

\* 3.2 Remove the two orders from the array list

\* 3.3 Add another orders to the array list, with the type, option, addOption1, and addOption2 is the

\* same as the two orders, and the quantity of the order sum of the two orders

\*/

**public** **void** passPrintingRequest () {

Order [] orders = o.getOrderList();

orderList = Arrays.*asList*(orders);

List <Order> printingList = **new** ArrayList <Order> ();

Order order1, order2;

**int** quantity;

**for** (**int** i = 0; i < orderList.size(); i++) {

order1 = orderList.get(i);

printingList.add(order1);

}

**int** k = 0;

**while** (k < printingList.size()) {

**int** m = k + 1;

**while** (m < printingList.size()) {

order1 = printingList.get(k);

order2 = printingList.get(m);

**if** (order1.getType().equals("D") || order1.getType().equals("d")) {

**if** (order1.getType().equals(order2.getType()) && order1.getOption() == order2.getOption()) {

quantity = order1.getQuantity() + order2.getQuantity();

printingList.remove(m);

printingList.remove(k);

printingList.add(**new** Order(order1.getType(), order1.getOption(), quantity, order1.getAddOption1(), order1.getAddOption2()));

m = k + 1;

k = 0;

}

**else** {

m++;

}

}

**else** {

**if** (order1.getType().equals(order2.getType()) && order1.getOption() == order2.getOption() && order1.getAddOption1().equals(order2.getAddOption1()) && order1.getAddOption2().equals(order2.getAddOption2())) {

quantity = order1.getQuantity() + order2.getQuantity();

printingList.remove(m);

printingList.remove(k);

printingList.add(**new** Order(order1.getType(), order1.getOption(), quantity, order1.getAddOption1(), order1.getAddOption2()));

m = k + 1;

k = 0;

}

**else** {

m++;

}

}

}

k++;

}

setPrintList(printingList);

Order [] prints = printingList.toArray(**new** Order[printingList.size()]);

print.queueRequest(prints);

}

}

Printing class

**package** my.edu.utar;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** Printing **implements** PrintingFunctionality{

**private** List <Order> printList = **new** ArrayList <Order> ();

**public** Printing (){

printList = **new** ArrayList <Order> ();

}

**public** **int** getPrintingRequest () {

**return** printList.size();

}

**public** Order [] getPrintList (){

Order [] orders = printList.toArray(**new** Order[printList.size()]);

**return** orders;

}

**public** **void** queueRequest (Order[] print) {

List <Order> printingList = Arrays.*asList*(print);

**for** (**int** i = 0; i < printingList.size(); i++) {

Order anOrder = printingList.get(i);

printList.add(anOrder);

}

}

}

MainApp class

**package** my.edu.utar;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

/\*

\* Display class created at here in order to show the IllegalArgumentException message for the user if they input invalid value

\*/

**class** Display{

Scanner input = **new** Scanner(System.***in***);

**public** **void** showToScreen(String message) {

System.***out***.print(message);

}

}

**public** **class** MainApp {

**private** **static** Order *o* = **new** Order ();

**private** **static** Printing *p* = **new** Printing ();

**private** **static** Display *d* = **new** Display ();

**public** **static** **void** main(String[] args) {

System.***out***.println(" Welcome ");

System.***out***.println(" Below is the details of printing services ");

System.***out***.println("-----------------------------------------------------------------------");

System.***out***.println("| Type | Option | Quantity | Charge(RM/piece) |");

System.***out***.println("|---------------------------------------------------------------------|");

System.***out***.println("| | | <5 | 0.50 |");

System.***out***.println("| | Black & White | 5-10 | 0.40 |");

System.***out***.println("| | | 11-20 | 0.30 |");

System.***out***.println("| | | 21-50 | 0.20 |");

System.***out***.println("| Document |--------------------------------------------------------|");

System.***out***.println("| | | <5 | 1.00 |");

System.***out***.println("| | Color | 5-10 | 0.90 |");

System.***out***.println("| | | 11-20 | 0.80 |");

System.***out***.println("| | | 21-50 | 0.70 |");

System.***out***.println("|---------------------------------------------------------------------|");

System.***out***.println("| | | <5 | 1.00 |");

System.***out***.println("| | Normal(4R) | 5-10 | 0.90 |");

System.***out***.println("| | | 11-20 | 0.75 |");

System.***out***.println("| | | 21-50 | 0.50 |");

System.***out***.println("| Photo |--------------------------------------------------------|");

System.***out***.println("| | | <5 | 1.20 |");

System.***out***.println("| | Passport | 5-10 | 0.95 |");

System.***out***.println("| | | 11-20 | 0.85 |");

System.***out***.println("| | | 21-50 | 0.75 |");

System.***out***.println("|---------------------------------------------------------------------|");

System.***out***.println();

System.***out***.println(" --------------------------------------------- ");

System.***out***.println(" | Additional Option | Surcharge(RM/piece) |");

System.***out***.println(" |-------------------------------------------|");

System.***out***.println(" | High quality paper | 0.10 |");

System.***out***.println(" | Design effect | 0.15 |");

System.***out***.println(" ---------------------------------------------");

*getCustomerOrder*();

*showCharge*();

Charge c = **new** Charge(*o*, *p*);

c.passPrintingRequest();

*showPrintingRequest*();

}

**public** **static** **void** getCustomerOrder () {

String choice;

Scanner scanner = **new** Scanner (System.***in***);

String type, addOption1, addOption2;

**int** option, quantity;

**do**

{

//Get Order Type

**do** {

System.***out***.println("Enter printing type (D for document, P for Photo): ");

type = scanner.next();

type = type.toUpperCase();

**try** {

*o*.setType(type);

}

**catch**(IllegalArgumentException iae) {

*d*.showToScreen(iae.getMessage());

}

} **while** (!type.equals("D") && !type.equals("P"));

//Get Order Option

**if** (type.equals("D")) {

**do** {

System.***out***.println("1. Black & White\n2. Colour\nChoose an option(1/2): ");

option = scanner.nextInt();

**try** {

*o*.setOption(option);

}

**catch**(IllegalArgumentException iae) {

*d*.showToScreen(iae.getMessage());

}

} **while** (option != 1 && option != 2);

}

**else** {

**do** {

System.***out***.println("1. Normal(4R)\n2. Passport\nChoose an option(1/2): ");

option = scanner.nextInt();

**try** {

*o*.setOption(option);

}

**catch**(IllegalArgumentException iae) {

*d*.showToScreen(iae.getMessage());

}

} **while** (option != 1 && option != 2);

}

//Get Order Quantity

**do** {

System.***out***.println("Enter quantity (1 to 50): ");

quantity = scanner.nextInt();

**try** {

*o*.setQuantity(quantity);

}

**catch**(IllegalArgumentException iae) {

*d*.showToScreen(iae.getMessage());

}

} **while** (quantity < 1 || quantity > 50);

//Get Additional Option 1 (High quality paper)

**if** (type.equals("P")) {

**do** {

System.***out***.println("Do you want to print your photo on high quality paper (Y/N): ");

addOption1 = scanner.next();

addOption1 = addOption1.toUpperCase();

**try**{

*o*.setAddOption1(addOption1);

}

**catch**(IllegalArgumentException iae) {

*d*.showToScreen(iae.getMessage());

}

} **while** (!addOption1.equals("Y") && !addOption1.equals("N"));

}

**else**

addOption1 = "N";

//Get Additional Option 2 (Design Effect)

**if** (type.equals("P")) {

**do** {

System.***out***.println("Do you want to have design effect on your photo (Y/N): ");

addOption2 = scanner.next();

addOption2 = addOption2.toUpperCase();

**try**{

*o*.setAddOption2(addOption2);

}

**catch**(IllegalArgumentException iae) {

*d*.showToScreen(iae.getMessage());

}

} **while** (!addOption2.equals("Y") && !addOption2.equals("N"));

}

**else**

addOption2 = "N";

//create an order and add it to orderList

Order anOrder = **new** Order (type, option, quantity, addOption1, addOption2);

*o*.addCustomerOrder(anOrder);

System.***out***.println("Order added successfully");

//Any other order to add

**do** {

System.***out***.print("Do you want to add other order(Y/N): ");

choice = scanner.next();

choice = choice.toUpperCase();

**if** (!choice.equals("Y") && !choice.equals("N"))

System.***out***.println("Error.\nPlease enter again");

} **while** (!choice.equals("Y") && !choice.equals("N"));

} **while**(choice.equals("Y"));

}

**public** **static** **void** showCharge() {

Charge ch = **new** Charge(*o*, *p*);

Order [] orders = *o*.getOrderList();

List <Order> orderList = Arrays.*asList*(orders);

System.***out***.println(orderList.size());

System.***out***.println("No.\tType\t\tOption\t\tQuantity\tHigh Quality Paper\tDesign Effect\tSubCharge(RM)");

**for** (**int** i = 0; i < orderList.size(); i++) {

System.***out***.print((i + 1) + ". \t");

Order anOrder = orderList.get(i);

**if** (anOrder.getType().equals("D") || anOrder.getType().equals("d")) {

System.***out***.print("Document\t");

**if** (anOrder.getOption() == 1)

System.***out***.print("Black & White\t" + anOrder.getQuantity() + "\t\t-\t\t\t-\t\t" + Math.*round*(ch.calDocCharge(anOrder)\*100.0)/100.0 + "\n");

**else**

System.***out***.print("Colour\t\t" + anOrder.getQuantity() + "\t\t-\t\t\t-\t\t" + Math.*round*(ch.calDocCharge(anOrder)\*100.0)/100.0 + "\n");

}

**else** {

System.***out***.print("Photo\t\t");

**if** (anOrder.getOption() == 1)

System.***out***.print("Normal(4R)\t" + anOrder.getQuantity() + "\t\t" + anOrder.getAddOption1() + "\t\t\t" + anOrder.getAddOption2() + "\t\t" + Math.*round*(ch.calPhotoCharge(anOrder)\*100.0)/100.0 + "\n");

**else**

System.***out***.print("Passport\t" + anOrder.getQuantity() + "\t\t" + anOrder.getAddOption1() + "\t\t\t" + anOrder.getAddOption2() + "\t\t" + Math.*round*(ch.calPhotoCharge(anOrder)\*100.0)/100.0 + "\n");

}

}

Order [] arrayToReturn = **new** Order [orderList.size()];

arrayToReturn = orderList.toArray(arrayToReturn);

System.***out***.println("Total: RM " + Math.*round*(ch.getTotalCharge()\*100.0)/100.0);

}

**private** **static** **void** showPrintingRequest() {

System.***out***.println("\nPrinting Request");

System.***out***.println("No.\tType\t\tOption\t\tQuantity\tHigh Quality Paper\tDesign Effect");

Order [] prints= *p*.getPrintList();

List <Order> printList = Arrays.*asList*(prints);

Order order;

**for** (**int** i = 0; i < printList.size(); i++) {

System.***out***.print((i + 1) + ".\t");

order = printList.get(i);

**if** (order.getType().equals("D")) {

System.***out***.print("Document\t");

**if** (order.getOption() == 1)

System.***out***.print("Black & White\t" + order.getQuantity() + "\t\t-\t\t\t-\t\t" + "\n");

**else**

System.***out***.print("Colour\t\t" + order.getQuantity() + "\t\t-\t\t\t-\t\t" + "\n");

}

**else** {

System.***out***.print("Photo\t\t");

**if** (*o*.getOption() == 1)

System.***out***.print("Normal(4R)\t" + order.getQuantity() + "\t\t" + order.getAddOption1() + "\t\t\t" + order.getAddOption2() + "\t\t" + "\n");

**else**

System.***out***.print("Passport\t" + order.getQuantity() + "\t\t" + order.getAddOption1() + "\t\t\t" + order.getAddOption2() + "\t\t" + "\n");

}

}

System.***out***.println("Total Printing Request: " + *p*.getPrintingRequest());

}

}

OrderFunctionality interface (for creating test doubles)

**package** my.edu.utar;

**public** **interface** OrderFunctionality {

**public** Order[] getOrderList();

}

PrintingFunctionality interface (for creating test doubles)

**package** my.edu.utar;

**public** **interface** PrintingFunctionality {

**public** **void** queueRequest(Order[] print);

**public** **int** getPrintingRequest();

**public** Order[] getPrintList ();

}

NewOrder class (for creating test doubles)

**package** my.edu.utar;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** NewOrder **implements** OrderFunctionality{

**private** String type;

**private** **int** option;

**private** **int** quantity;

**private** String addOption1;

**private** String addOption2;

**private** List <Order> orderList = **new** ArrayList <Order>();

**public** NewOrder () {

}

**public** NewOrder (String type, **int** option, **int** quantity, String addOption1, String addOption2) {

**this**.type = type;

**this**.option = option;

**this**.quantity = quantity;

**this**.addOption1 = addOption1;

**this**.addOption2 = addOption2;

}

**public** NewOrder(Order[] orders) {

orderList = Arrays.*asList*(orders);

}

**public** String getType() {

**return** type;

}

**public** **int** getOption () {

**return** option;

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** String getAddOption1() {

**return** addOption1;

}

**public** String getAddOption2() {

**return** addOption2;

}

**public** **void** setType(String type) {

**if** (type == **null**)

**throw** **new** IllegalArgumentException("Type cannot be null\n");

type = type.toUpperCase();

**if** (!type.equals("D")&& !type.equals("P"))

**throw** **new** IllegalArgumentException("Invalid type. Please enter again.\n");

**this**.type = type;

}

**public** **void** setOption (**int** option) {

**if** (option < 1 || option > 2)

**throw** **new** IllegalArgumentException ("Invalid option. Please enter again.\n");

**this**.option = option;

}

**public** **void** setQuantity (**int** quantity) {

**if** (quantity < 1 || quantity > 50)

**throw** **new** IllegalArgumentException ("Invalid quantity. Please enter again.\n");

**this**.quantity = quantity;

}

**public** **void** setAddOption1 (String addOption1) {

**if** (addOption1 == **null**)

**throw** **new** IllegalArgumentException ("addOption1 cannot be null.\n");

addOption1 = addOption1.toUpperCase();

**if** (!addOption1.equals("Y")&& !addOption1.equals("N"))

**throw** **new** IllegalArgumentException("Invalid choice. Please enter again.\n");

**this**.addOption1 = addOption1;

}

**public** **void** setAddOption2 (String addOption2) {

**if** (addOption2 == **null**)

**throw** **new** IllegalArgumentException("addOption2 cannot be null.\n");

addOption2 = addOption2.toUpperCase();

**if** (!addOption2.equals("Y")&& !addOption2.equals("N"))

**throw** **new** IllegalArgumentException("Invalid choice. Please enter again.\n");

**this**.addOption2 = addOption2;

}

**public** **void** addCustomerOrder(Order anOrder) {

**if** (anOrder == **null**)

**throw** **new** IllegalArgumentException ("Order cannot be null");

**if** (anOrder.getType() == **null** || (!anOrder.getType().equals("D") && !anOrder.getType().equals("P")))

**throw** **new** IllegalArgumentException ("Order type is incorrect");

**if** (anOrder.getOption() < 1 || anOrder.getOption() > 2)

**throw** **new** IllegalArgumentException ("Order option is invalid");

**if** (anOrder.getQuantity() < 1 || anOrder.getQuantity() > 50)

**throw** **new** IllegalArgumentException ("Order quantity is out of range");

**if** (anOrder.getAddOption1() == **null** || (!anOrder.getAddOption1().equals("Y") && !anOrder.getAddOption1().equals("N")))

**throw** **new** IllegalArgumentException ("Order additional option 1 is incorrect");

**if** (anOrder.getAddOption2() == **null** || (!anOrder.getAddOption2().equals("Y") && !anOrder.getAddOption2().equals("N")))

**throw** **new** IllegalArgumentException ("Order additional option 2 is incorrect");

**if** ((anOrder.getType().equals("D") && anOrder.getAddOption1().equals("Y")) || (anOrder.getType().equals("D") && anOrder.getAddOption2().equals("Y")))

**throw** **new** IllegalArgumentException("Order with Document cannot have high quality paper effect and design effect");

orderList.add(anOrder);

}

**public** Order[] getOrderList(){

Order [] orders = orderList.toArray(**new** Order[orderList.size()]);

**return** orders;

}

}

NewCharge class (for creating test doubles)

**package** my.edu.utar;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** NewCharge {

OrderFunctionality of;

PrintingFunctionality pf;

List <Order> orderList;

List <Order> printList;

**public** NewCharge() {

}

**public** NewCharge (OrderFunctionality of, PrintingFunctionality pf) {

**this**.of = of;;

**this**.pf = pf;

}

/\*

\* Steps to arrange the orderList as printing request list:

\* 1. Copy the orderList to another array list

\* 2. Compare the orders in the array list

\* 3. If there are two orders same for document, if the option is the same:

\* 3.1 Get the quantity of the two orders.

\* 3.2 Remove the two orders from the array list

\* 3.3 Add another orders to the array list, with the type, and option is the same as the two orders,

\* and the quantity of the order sum of the two orders

\* 4. If there are two orders same for photo, if the option, addOption1, and addOption2 are same:

\* 3.1 Get the quantity of the two orders.

\* 3.2 Remove the two orders from the array list

\* 3.3 Add another orders to the array list, with the type, option, addOption1, and addOption2 is the

\* same as the two orders, and the quantity of the order sum of the two orders

\*/

**public** **void** passPrintingRequest () {

Order [] orders = of.getOrderList();

orderList = Arrays.*asList*(orders);

List <Order> printingList = **new** ArrayList <Order> ();

Order order1, order2;

**int** quantity;

**for** (**int** i = 0; i < orderList.size(); i++) {

order1 = orderList.get(i);

printingList.add(order1);

}

**int** k = 0;

**while** (k < printingList.size()) {

**int** m = k + 1;

**while** (m < printingList.size()) {

order1 = printingList.get(k);

order2 = printingList.get(m);

**if** (order1.getType().equals("D") || order1.getType().equals("d")) {

**if** (order1.getType().equals(order2.getType()) && order1.getOption() == order2.getOption()) {

quantity = order1.getQuantity() + order2.getQuantity();

printingList.remove(m);

printingList.remove(k);

printingList.add(**new** Order(order1.getType(), order1.getOption(), quantity, order1.getAddOption1(), order1.getAddOption2()));

m = k + 1;

k = 0;

}

**else** {

m++;

}

}

**else** {

**if** (order1.getType().equals(order2.getType()) && order1.getOption() == order2.getOption() && order1.getAddOption1().equals(order2.getAddOption1()) && order1.getAddOption2().equals(order2.getAddOption2())) {

quantity = order1.getQuantity() + order2.getQuantity();

printingList.remove(m);

printingList.remove(k);

printingList.add(**new** Order(order1.getType(), order1.getOption(), quantity, order1.getAddOption1(), order1.getAddOption2()));

m = k + 1;

k = 0;

}

**else** {

m++;

}

}

}

k++;

}

Order [] prints = printingList.toArray(**new** Order[printingList.size()]);

pf.queueRequest(prints);

}

**public** **double** getTotalCharge () {

Order [] orders = of.getOrderList();

orderList = Arrays.*asList*(orders);

**double** total = 0;

**double** price;

**for** (**int** i = 0; i < orderList.size(); i++) {

Order anOrder = orderList.get(i);

//count Total Charge part

**if** (anOrder.getType().equals("D") || anOrder.getType().equals("d"))

price = calDocCharge(anOrder);

**else**

price = calPhotoCharge(anOrder);

total += price;

price = 0;

}

total = Math.*round*(total\*100.0)/100.0;

**return** total;

}

**public** **double** calDocCharge(Order anOrder) {

**double** price = 0;

**if** (anOrder.getOption() == 1) {

**if** (anOrder.getQuantity()> 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 0.5;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.4;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.3;

**else**

price = anOrder.getQuantity() \* 0.2;

}

**else** {

**if** (anOrder.getQuantity() > 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 1;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.9;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.8;

**else**

price = anOrder.getQuantity() \* 0.7;

}

**return** price;

}

**public** **double** calPhotoCharge (Order anOrder) {

**double** price = 0;

**if** (anOrder.getOption() == 1) {

**if** (anOrder.getQuantity() > 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 1;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.9;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.75;

**else**

price = anOrder.getQuantity() \* 0.5;

}

**else** {

**if** (anOrder.getQuantity() > 0 && anOrder.getQuantity() < 5)

price = anOrder.getQuantity() \* 1.2;

**else** **if** (anOrder.getQuantity() < 11)

price = anOrder.getQuantity() \* 0.95;

**else** **if** (anOrder.getQuantity() < 21)

price = anOrder.getQuantity() \* 0.85;

**else**

price = anOrder.getQuantity() \* 0.75;

}

**if** (anOrder.getAddOption1().equals("Y") || anOrder.getAddOption1().equals("y"))

price = price + anOrder.getQuantity() \* 0.1;

**if** (anOrder.getAddOption2().equals("Y") || anOrder.getAddOption2().equals("y"))

price = price + anOrder.getQuantity() \* 0.15;

price = Math.*round*(price\*100.0)/100.0;

**return** price;

}

}

NewPrinting class (for creating test doubles)

**package** my.edu.utar;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** NewPrinting **implements** PrintingFunctionality {

**private** List <Order> printList = **new** ArrayList <Order> ();

**public** NewPrinting (){

printList = **new** ArrayList <Order> ();

}

**public** **int** getPrintingRequest () {

**return** printList.size();

}

**public** Order [] getPrintList (){

Order [] orders = printList.toArray(**new** Order[printList.size()]);

**return** orders;

}

**public** **void** queueRequest (Order[] print) {

List <Order> printingList = Arrays.*asList*(print);

**for** (**int** i = 0; i < printingList.size(); i++) {

Order anOrder = printingList.get(i);

printList.add(anOrder);

}

}

}

### JUnit Test Code

**O1: OrderUnitTests.java (Order module)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

**import** org.junit.runner.RunWith;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** OrderUnitTests {

Order o = **new** Order ();

//Valid test for setType()

/\*

\* Order Module

\* Test Case 1.1.1~1.1.2

\*/

**private** Object [] getParamsForSetTypeValidTest () {

**return** **new** Object [] {

**new** Object [] {"D", "D"},

**new** Object [] {"d", "D"}

};

}

@Test

@Parameters(method = "getParamsForSetTypeValidTest")

**public** **void** testSetTypeValidTest(String inputType, String expectedType) {

o.setType(inputType);

*assertEquals*(o.getType(), expectedType);

}

//Invalid test for setType()

/\*

\* Order Module

\* Test Case 1.1.3~1.1.5

\*/

**private** Object [] getParamsForSetTypeInvalidTest() {

**return** **new** Object [] {

**new** Object [] {**null**},

**new** Object [] {"do"},

**new** Object [] {""}

};

}

@Test(expected = IllegalArgumentException.**class**)

@Parameters(method = "getParamsForSetTypeInvalidTest")

**public** **void** testSetTypeInvalidTest(String inputType) {

o.setType(inputType);

}

//Valid test for setOption ()

/\*

\* Order Module

\* Test Case 1.2.1

\*/

**private** Object[] getParamsForSetOptionValidTest(){

**return** **new** Object[]{

**new** Object[] {1,1}

};

}

@Test

@Parameters(method="getParamsForSetOptionValidTest")

**public** **void** testSetOptionValidTest(**int** option,**int** expectedOption){

o.setOption(option);

*assertEquals*(o.getOption(),expectedOption);

}

//Invalid test for setOption ()

/\*

\* Order Module

\* Test Case 1.2.2~1.2.3

\*/

**private** Object[] getParamsForSetOptionInvalidTest () {

**return** **new** Object [] {

**new** Object [] {0},

**new** Object [] {3}

};

}

@Test(expected = IllegalArgumentException.**class**)

@Parameters(method = "getParamsForSetOptionInvalidTest")

**public** **void** testSetOptionInvalidTest(**int** option) {

o.setOption(option);

}

//Valid test for setQuantity ()

/\*

\* Order Module

\* Test Case 1.3.1

\*/

**private** Object[] getParamsForSetQuantityValidTest(){

**return** **new** Object[]{

**new** Object[] {1,1}

};

}

@Test

@Parameters(method="getParamsForSetQuantityValidTest")

**public** **void** testSetQuantityValidTest(**int** quantity,**int** expectedQuantity){

o.setQuantity(quantity);

*assertEquals*(o.getQuantity(),expectedQuantity);

}

//Invalid test for setQuantity ()

/\*

\* Order Module

\* Test Case 1.3.2~1.3.3

\*/

**private** Object[] getParamsForSetQuantityInvalidTest () {

**return** **new** Object [] {

**new** Object [] {0},

**new** Object [] {51}

};

}

@Test(expected = IllegalArgumentException.**class**)

@Parameters(method = "getParamsForSetQuantityInvalidTest")

**public** **void** testSetQuantityInvalidTest(**int** quantity) {

o.setQuantity(quantity);

}

//Valid test for setAddOption1()

/\*

\* Order Module

\* Test Case 1.4.1~1.4.2

\*/

**private** Object[]getParamsForSetAddOption1ValidTest(){

**return** **new** Object[]{

**new** Object[] {"Y","Y"},

**new** Object[] {"y","Y"}

};

}

@Test

@Parameters(method="getParamsForSetAddOption1ValidTest")

**public** **void** testSetAddOption1ValidTest(String addOption1,String expectedOption1){

o.setAddOption1(addOption1);

*assertEquals*(o.getAddOption1(),expectedOption1);

}

//Invalid test for setAddOption1 ()

/\*

\* Order Module

\* Test Case 1.4.3~1.4.5

\*/

**private** Object[]getParamsForSetAddOption1InvalidTest(){

**return** **new** Object[]{

**new** Object [] {**null**},

**new** Object [] {"do"},

**new** Object [] {""}

};

}

@Test(expected=IllegalArgumentException.**class**)

@Parameters(method="getParamsForSetAddOption1InvalidTest")

**public** **void** testSetAddOption1InValidTest(String addOption1){

o.setAddOption1(addOption1);

}

//Valid test for setAddOption2()

/\*

\* Order Module

\* Test Case 1.5.1~1.5.2

\*/

**private** Object[]getParamsForSetAddOption2ValidTest(){

**return** **new** Object[]{

**new** Object[] {"Y","Y"},

**new** Object[] {"y","Y"}

};

}

@Test

@Parameters(method="getParamsForSetAddOption2ValidTest")

**public** **void** testSetAddOption2ValidTest(String addOption2,String expectedOption2){

o.setAddOption2(addOption2);

*assertEquals*(o.getAddOption2(),expectedOption2);

}

//Invalid test for setAddOption2()

/\*

\* Order Module

\* Test Case 1.5.3~1.5.5

\*/

**private** Object[]getParamsForSetAddOption2InvalidTest(){

**return** **new** Object[]{

**new** Object [] {**null**},

**new** Object [] {"do"},

**new** Object [] {""}

};

}

@Test(expected=IllegalArgumentException.**class**)

@Parameters(method="getParamsForSetAddOption2InvalidTest")

**public** **void** testSetAddOption2InValidTest(String addOption2){

o.setAddOption2(addOption2);

}

}

**O2: AddCustomerOrderUnitTests.java (Order module) (Test data read from text file)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** java.util.\*;

**import** java.util.Scanner;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** org.junit.Test;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

**import** org.junit.runner.RunWith;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** AddCustomerOrderUnitTests {

Order o = **new** Order ();

**static** Scanner *inputStream* = **null**;

//Valid test for addCustomerOrder()

/\*

\* Order Module

\* Test Case: 2.1.1

\*/

**private** Object [] getParamsForAddCustomerOrderValidValue() {

String filename = "valid.txt";

String [] tokens = **null**;

**try** {

*inputStream* = **new** Scanner (**new** File(filename));

}

**catch** (FileNotFoundException e){

System.***out***.println("Error opening the file " + filename);

System.*exit*(0);

}

**while**(*inputStream*.hasNext()) {

String singleLine = *inputStream*.next();

tokens = singleLine.split(",");

}

Order inputOrder1 = **new** Order(tokens[0], Integer.*parseInt*(tokens[1]), Integer.*parseInt*(tokens[2]), tokens[3], tokens[4]);

Order inputOrder2 = **new** Order(tokens[5], Integer.*parseInt*(tokens[6]), Integer.*parseInt*(tokens[7]), tokens[8], tokens[9]);

Order [] expectedResult = **new** Order[] {inputOrder1, inputOrder2};

**return** **new** Object [] {

**new** Object [] {inputOrder1,inputOrder2, expectedResult}

};

}

@Test

@Parameters(method = "getParamsForAddCustomerOrderValidValue")

**public** **void** testAddCustomerOrderValidTest(Order inputOrder1,Order inputOrder2, Order [] expectedOrder) {

o.addCustomerOrder(inputOrder1);

o.addCustomerOrder(inputOrder2);

*assertArrayEquals*(expectedOrder, o.getOrderList());

}

//Invalid test for addCustomerOrder ()

/\*

\* Order Module

\* Test Case: 2.1.2~2.1.16

\*/

**private** Object [] getParamsForAddCustomerOrderInvalidValue() {

String filename = "invalid.txt";

String [] tokens = **null**;

**try** {

*inputStream* = **new** Scanner (**new** File(filename));

}

**catch** (FileNotFoundException e){

System.***out***.println("Error opening the file " + filename);

System.*exit*(0);

}

List<Object[]> testData = **new** ArrayList <>();

**while**(*inputStream*.hasNext()) {

String singleLine = *inputStream*.nextLine();

**if** (singleLine.trim().equals("null")) {

testData.add(**new** Order[] {**null**});

}

**else**{

tokens = singleLine.split(",");

Order order = **new** Order(tokens[0], Integer.*parseInt*(tokens[1]), Integer.*parseInt*(tokens[2]), tokens[3], tokens[4]);

testData.add(**new** Order[] {order});

}

}

**return** testData.toArray();

}

@Test(expected = IllegalArgumentException.**class**)

@Parameters(method = "getParamsForAddCustomerOrderInvalidValue")

**public** **void** testAddCustomerOrderInvalidTest (Order inputOrder) {

o.addCustomerOrder(inputOrder);

}

}

valid.txt

D,1,2,N,N,P,2,3,Y,Y

invalid.txt

null

null,1,2,N,N

do,1,2,N,N

,1,2,N,N

D,0,2,N,N

D,3,2,N,N

D,1,0,N,N

D,1,51,N,N

D,1,2,null,N

D,1,2,yes,N

D,1,2,,N

D,1,2,N,null

D,1,2,N,no

D,1,2,Y,N

D,1,2,N,Y

**C1: CalculateEachDocChargeUnitTests.java (Charge module)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** **static** org.mockito.ArgumentMatchers.\*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** CalculateEachDocChargeUnitTests {

//Test for CalculateDocumentCharge()

/\*

\* Charge Module

\* Test Case: 1.1.1~1.1.16

\* Boundary Value Analysis

\*/

**private** Object [] getParamsForTestCalculateEachDocChargeValidValues () {

**return** **new** Object [] {

**new** Object [] {**new** Order ("D", 1, 1, "N", "N"), 0.5},

**new** Object [] {**new** Order ("D", 1, 4, "N", "N"), 2},

**new** Object [] {**new** Order ("D", 1, 5, "N", "N"), 2},

**new** Object [] {**new** Order ("D", 1, 10, "N", "N"), 4},

**new** Object [] {**new** Order ("D", 1, 11, "N", "N"), 3.3},

**new** Object [] {**new** Order ("D", 1, 20, "N", "N"), 6},

**new** Object [] {**new** Order ("D", 1, 21, "N", "N"), 4.2},

**new** Object [] {**new** Order ("D", 1, 50, "N", "N"), 10},

**new** Object [] {**new** Order ("D", 2, 1, "N", "N"), 1},

**new** Object [] {**new** Order ("D", 2, 4, "N", "N"), 4},

**new** Object [] {**new** Order ("D", 2, 5, "N", "N"), 4.5},

**new** Object [] {**new** Order ("D", 2, 10, "N", "N"), 9},

**new** Object [] {**new** Order ("D", 2, 11, "N", "N"), 8.8},

**new** Object [] {**new** Order ("D", 2, 20, "N", "N"), 16},

**new** Object [] {**new** Order ("D", 2, 21, "N", "N"), 14.7},

**new** Object [] {**new** Order ("D", 2, 50, "N", "N"), 35}

};

}

@Test

@Parameters(method = "getParamsForTestCalculateEachDocChargeValidValues")

**public** **void** testCalculateEachDocChargeValidValues(Order anOrder, **double** expectedResult) {

Charge charge = **new** Charge ();

*assertEquals*(charge.calDocCharge(anOrder), expectedResult, 0);

}

//Invalid test is not needed at here, since invalid orders will be handled by the order module in order class.

}

**C2: CalculateEachPhotoChargeUnitTests.java (Charge module)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** **static** org.mockito.ArgumentMatchers.\*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** CalculateEachPhotoChargeUnitTests {

//Test for CalculatePhotoCharge()

/\*

\* Charge module

\* Test case: 2.1.1~2.1.32

\* Boundary-value analysis

\*/

**private** Object [] getParamsForTestCalculateEachPhotoChargeForNormalSizeValidValues () {

**return** **new** Object [] {

**new** Object [] {**new** Order ("P", 1, 1, "N", "N"), 1},

**new** Object [] {**new** Order ("P", 1, 4, "N", "N"), 4},

**new** Object [] {**new** Order ("P", 1, 5, "N", "N"), 4.5},

**new** Object [] {**new** Order ("P", 1, 10, "N", "N"), 9},

**new** Object [] {**new** Order ("P", 1, 11, "N", "N"), 8.25},

**new** Object [] {**new** Order ("P", 1, 20, "N", "N"), 15},

**new** Object [] {**new** Order ("P", 1, 21, "N", "N"), 10.5},

**new** Object [] {**new** Order ("P", 1, 50, "N", "N"), 25},

**new** Object [] {**new** Order ("P", 1, 1, "Y", "N"), 1.1},

**new** Object [] {**new** Order ("P", 1, 4, "Y", "N"), 4.4},

**new** Object [] {**new** Order ("P", 1, 5, "Y", "N"), 5},

**new** Object [] {**new** Order ("P", 1, 10, "Y", "N"), 10},

**new** Object [] {**new** Order ("P", 1, 11, "Y", "N"), 9.35},

**new** Object [] {**new** Order ("P", 1, 20, "Y", "N"), 17},

**new** Object [] {**new** Order ("P", 1, 21, "Y", "N"), 12.6},

**new** Object [] {**new** Order ("P", 1, 50, "Y", "N"), 30},

**new** Object [] {**new** Order ("P", 1, 1, "N", "Y"), 1.15},

**new** Object [] {**new** Order ("P", 1, 4, "N", "Y"), 4.6},

**new** Object [] {**new** Order ("P", 1, 5, "N", "Y"), 5.25},

**new** Object [] {**new** Order ("P", 1, 10, "N", "Y"), 10.5},

**new** Object [] {**new** Order ("P", 1, 11, "N", "Y"), 9.9},

**new** Object [] {**new** Order ("P", 1, 20, "N", "Y"), 18},

**new** Object [] {**new** Order ("P", 1, 21, "N", "Y"), 13.65},

**new** Object [] {**new** Order ("P", 1, 50, "N", "Y"), 32.5},

**new** Object [] {**new** Order ("P", 1, 1, "Y", "Y"), 1.25},

**new** Object [] {**new** Order ("P", 1, 4, "Y", "Y"), 5},

**new** Object [] {**new** Order ("P", 1, 5, "Y", "Y"), 5.75},

**new** Object [] {**new** Order ("P", 1, 10, "Y", "Y"), 11.5},

**new** Object [] {**new** Order ("P", 1, 11, "Y", "Y"), 11},

**new** Object [] {**new** Order ("P", 1, 20, "Y", "Y"), 20},

**new** Object [] {**new** Order ("P", 1, 21, "Y", "Y"), 15.75},

**new** Object [] {**new** Order ("P", 1, 50, "Y", "Y"), 37.5},

};

}

@Test

@Parameters(method = "getParamsForTestCalculateEachPhotoChargeForNormalSizeValidValues")

**public** **void** testCalculateEachPhotoChargeForNormalSizeValidValues(Order anOrder, **double** expectedResult) {

Charge charge = **new** Charge ();

*assertEquals*(charge.calPhotoCharge(anOrder), expectedResult, 0);

}

/\*

\* Charge module

\* Test case: 2.2.1~2.2.32

\* Boundary-value analysis

\*/

**private** Object [] getParamsForTestCalculateEachPhotoChargeForPassportSizeValidValues () {

**return** **new** Object [] {

**new** Object [] {**new** Order ("P", 2, 1, "N", "N"), 1.2},

**new** Object [] {**new** Order ("P", 2, 4, "N", "N"), 4.8},

**new** Object [] {**new** Order ("P", 2, 5, "N", "N"), 4.75},

**new** Object [] {**new** Order ("P", 2, 10, "N", "N"), 9.5},

**new** Object [] {**new** Order ("P", 2, 11, "N", "N"), 9.35},

**new** Object [] {**new** Order ("P", 2, 20, "N", "N"), 17},

**new** Object [] {**new** Order ("P", 2, 21, "N", "N"), 15.75},

**new** Object [] {**new** Order ("P", 2, 50, "N", "N"), 37.5},

**new** Object [] {**new** Order ("P", 2, 1, "Y", "N"), 1.3},

**new** Object [] {**new** Order ("P", 2, 4, "Y", "N"), 5.2},

**new** Object [] {**new** Order ("P", 2, 5, "Y", "N"), 5.25},

**new** Object [] {**new** Order ("P", 2, 10, "Y", "N"), 10.5},

**new** Object [] {**new** Order ("P", 2, 11, "Y", "N"), 10.45},

**new** Object [] {**new** Order ("P", 2, 20, "Y", "N"), 19},

**new** Object [] {**new** Order ("P", 2, 21, "Y", "N"), 17.85},

**new** Object [] {**new** Order ("P", 2, 50, "Y", "N"), 42.5},

**new** Object [] {**new** Order ("P", 2, 1, "N", "Y"), 1.35},

**new** Object [] {**new** Order ("P", 2, 4, "N", "Y"), 5.4},

**new** Object [] {**new** Order ("P", 2, 5, "N", "Y"), 5.5},

**new** Object [] {**new** Order ("P", 2, 10, "N", "Y"), 11},

**new** Object [] {**new** Order ("P", 2, 11, "N", "Y"), 11},

**new** Object [] {**new** Order ("P", 2, 20, "N", "Y"), 20},

**new** Object [] {**new** Order ("P", 2, 21, "N", "Y"), 18.9},

**new** Object [] {**new** Order ("P", 2, 50, "N", "Y"), 45},

**new** Object [] {**new** Order ("P", 2, 1, "Y", "Y"), 1.45},

**new** Object [] {**new** Order ("P", 2, 4, "Y", "Y"), 5.8},

**new** Object [] {**new** Order ("P", 2, 5, "Y", "Y"), 6},

**new** Object [] {**new** Order ("P", 2, 10, "Y", "Y"), 12},

**new** Object [] {**new** Order ("P", 2, 11, "Y", "Y"), 12.1},

**new** Object [] {**new** Order ("P", 2, 20, "Y", "Y"), 22},

**new** Object [] {**new** Order ("P", 2, 21, "Y", "Y"), 21},

**new** Object [] {**new** Order ("P", 2, 50, "Y", "Y"), 50},

};

}

@Test

@Parameters(method = "getParamsForTestCalculateEachPhotoChargeForPassportSizeValidValues")

**public** **void** testCalculateEachPhotoChargeForPassportSizeValidValues(Order anOrder, **double** expectedResult) {

Charge charge = **new** Charge ();

*assertEquals*(charge.calPhotoCharge(anOrder), expectedResult, 0);

}

//Invalid test is not needed at here, since invalid orders will be handled by the order module in order class.

}

**C3: GetTotalChargeUnitTests.java (Charge module)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** **static** org.mockito.ArgumentMatchers.\*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** GetTotalChargeUnitTests {

//Test for GetTotalCharge()

/\*

\* Charge module

\* Test case: 3.1.1~3.1.11

\*/

**private** Object [] getParamsForGetTotalChargeValidValues () {

**return** **new** Object [] {

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 3, "N", "N"), **new** Order("D", 2, 18, "N", "N")}, 15.9},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 7, "N", "N"), **new** Order("P", 1, 6, "Y", "N")}, 8.8},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 12, "N", "N"), **new** Order ("P", 2, 3, "N", "N")}, 7.2},

**new** Object [] {**new** Order[] {**new** Order ("D", 2, 30, "N", "N"), **new** Order("P", 1, 15, "Y", "Y")}, 36},

**new** Object [] {**new** Order[] {**new** Order ("D", 2, 4, "N", "N"), **new** Order("P", 2, 30, "N", "Y")}, 31.0},

**new** Object [] {**new** Order[] {**new** Order ("P", 1, 40, "N", "N"), **new** Order("P", 2, 17, "Y", "Y")}, 38.7},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 38, "N", "N"), **new** Order ("D", 2, 8, "N", "N"), **new** Order("P", 1, 3, "N", "Y")}, 18.25},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 23, "N", "N"), **new** Order ("D", 2, 3, "N", "N"), **new** Order("P", 2, 9, "Y", "N")}, 17.05},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 14, "N", "N"), **new** Order ("P", 1, 9, "N", "Y"), **new** Order("P", 2, 13, "N", "N")}, 24.7},

**new** Object [] {**new** Order[] {**new** Order ("D", 2, 12, "N", "N"), **new** Order ("P", 1, 18, "Y", "N"), **new** Order("P", 2, 3, "N", "Y")}, 28.95},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 50, "N", "N"), **new** Order("D", 2, 50, "N", "N"), **new** Order("P", 1, 50, "Y", "Y"), **new** Order("P", 2, 50, "Y", "Y")}, 132.5}

};

}

@Test

@Parameters(method = "getParamsForGetTotalChargeValidValues")

**public** **void** testGetTotalChargeValidValues(Order[] orderList, **double** expectedResult) {

OrderFunctionality of = **new** Order(orderList);

PrintingFunctionality pf = **new** Printing();

NewCharge nc = **new** NewCharge (of, pf);

*assertEquals*(nc.getTotalCharge(), expectedResult, 0);

}

//Invalid test is not needed at here, since invalid orders will be handled by the order module in order class.

}

**C4: PassPrintingRequestsUnitTests.java (Charge module)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** **static** org.mockito.Mockito.*mock*;

**import** **static** org.mockito.Mockito.*times*;

**import** **static** org.mockito.Mockito.*verify*;

**import** **static** org.mockito.Mockito.*when*;

**import** java.util.Arrays;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** PassPrintingRequestsUnitTests {

//Test for PassPrintingRequest()

/\*

\* Charge Module

\* Test Case: 4.1.1~4.1.5

\* Test Steps:

\* 1. Test the length of the expectedPrintList and the length of the printList that is passed to printing module

\* 2. For each of the orders, test each of the components of the orders to check whether the value of the components in

\* the expectedPrintList is same with the value of the components in the printList that is passed to the printing module.

\*/

**private** Object [] getParamsForPassPrintingRequestsValidValues() {

//1. Two order that has the same type, option, addOption1, addOption2

Order order1\_1 = **new** Order("D", 1, 2, "N", "N");

Order order1\_2 = **new** Order("D", 1, 3, "N", "N");

Order order1\_expected = **new** Order("D", 1, 5, "N", "N");

Order [] orderList1 = **new** Order[]{order1\_1, order1\_2};

Order [] expectedList1 = **new** Order[] {order1\_expected};

//2. Two order with different type, or different option, or different addOption1, or different addOption2

Order order2\_1 = **new** Order("D", 1, 2, "N", "N");

Order order2\_2 = **new** Order("P", 1, 2, "N", "N");

Order [] orderList2 = **new** Order[] {order2\_1, order2\_2};

Order [] expectedList2 = **new** Order[] {order2\_1, order2\_2};

//3. Three order: two same type, option, addOption1, and addOption2, one different

Order order3\_1 = **new** Order("D", 1, 2, "N", "N");

Order order3\_2 = **new** Order("D", 1, 3, "N", "N");

Order order3\_3 = **new** Order("D", 2, 10, "N", "N");

Order order3\_add = **new** Order("D", 1, 5, "N", "N");

Order [] orderList3 = **new** Order[] {order3\_1, order3\_2, order3\_3};

Order [] expectedList3 = **new** Order [] {order3\_3, order3\_add};

//4. Four order: two same type, option, addOption1, and addOption2, another two also has same type, option, addOption1, and addOption2

Order order4\_1 = **new** Order("D", 1, 2, "N", "N");

Order order4\_2 = **new** Order("P", 2, 3, "Y", "Y");

Order order4\_3 = **new** Order("P", 2, 5, "Y", "Y");

Order order4\_4 = **new** Order("D", 1, 3, "N", "N");

Order order4\_add1 = **new** Order("D",1 , 5, "N", "N");

Order order4\_add2 = **new** Order("P", 2, 8, "Y", "Y");

Order [] orderList4 = **new** Order[] {order4\_1, order4\_2, order4\_3, order4\_4};

Order [] expectedList4 = **new** Order [] {order4\_add1, order4\_add2};

//5. Four order: three same type, option, addOption1, and addOption2, other one is different

Order order5\_1 = **new** Order ("D", 1, 3, "N", "N");

Order order5\_2 = **new** Order ("D", 1, 9, "N", "N");

Order order5\_3 = **new** Order ("P", 1, 3, "Y", "N");

Order order5\_4 = **new** Order ("D", 1, 12, "N", "N");

Order order5\_add = **new** Order ("D", 1, 24, "N", "N");

Order[] orderList5 = **new** Order[] {order5\_1, order5\_2, order5\_3, order5\_4};

Order[] expectedList5 = **new** Order[] {order5\_3, order5\_add};

**return** **new** Object [] {

**new** Object [] {orderList1, expectedList1},

**new** Object [] {orderList2, expectedList2},

**new** Object [] {orderList3, expectedList3},

**new** Object [] {orderList4, expectedList4},

**new** Object [] {orderList5, expectedList5}

};

}

@Test

@Parameters(method = "getParamsForPassPrintingRequestsValidValues")

**public** **void** testWriteRequestsValidValues(Order[] orderList, Order[] expectedList) {

OrderFunctionality of = **new** Order(orderList);

PrintingFunctionality pf = **new** Printing ();

NewCharge nc = **new** NewCharge (of, pf);

nc.passPrintingRequest();

*assertEquals*(expectedList.length, pf.getPrintingRequest());

**for** (**int** i = 0; i < expectedList.length; i++) {

*assertEquals*(expectedList[i].getType(), pf.getPrintList()[i].getType());

*assertEquals*(expectedList[i].getOption(), pf.getPrintList()[i].getOption());

*assertEquals*(expectedList[i].getQuantity(), pf.getPrintList()[i].getQuantity());

*assertEquals*(expectedList[i].getAddOption1(), pf.getPrintList()[i].getAddOption1());

*assertEquals*(expectedList[i].getAddOption2(), pf.getPrintList()[i].getAddOption2());

}

}

//Invalid test is not needed at here, since invalid orders will be handled by the order module in order class.

}

**C5: ChargeModuleIntegrationTests.java (Charge module)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** ChargeModuleIntegrationTest {

//Test for getTotalCharge()

/\*

\* Charge module

\* Test Case: 5.1.1~5.1.11

\*/

**private** Object[] getParamsForGetTotalChargeValidValues() {

**return** **new** Object[] {

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 3, "N", "N"), **new** Order("D", 2, 18, "N", "N")}, 15.9},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 7, "N", "N"), **new** Order("P", 1, 6, "Y", "N")}, 8.8},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 12, "N", "N"), **new** Order ("P", 2, 3, "N", "N")}, 7.2},

**new** Object [] {**new** Order[] {**new** Order ("D", 2, 30, "N", "N"), **new** Order("P", 1, 15, "Y", "Y")}, 36},

**new** Object [] {**new** Order[] {**new** Order ("D", 2, 4, "N", "N"), **new** Order("P", 2, 30, "N", "Y")}, 31.0},

**new** Object [] {**new** Order[] {**new** Order ("P", 1, 40, "N", "N"), **new** Order("P", 2, 17, "Y", "Y")}, 38.7},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 38, "N", "N"), **new** Order ("D", 2, 8, "N", "N"), **new** Order("P", 1, 3, "N", "Y")}, 18.25},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 23, "N", "N"), **new** Order ("D", 2, 3, "N", "N"), **new** Order("P", 2, 9, "Y", "N")}, 17.05},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 14, "N", "N"), **new** Order ("P", 1, 9, "N", "Y"), **new** Order("P", 2, 13, "N", "N")}, 24.7},

**new** Object [] {**new** Order[] {**new** Order ("D", 2, 12, "N", "N"), **new** Order ("P", 1, 18, "Y", "N"), **new** Order("P", 2, 3, "N", "Y")}, 28.95},

**new** Object [] {**new** Order[] {**new** Order ("D", 1, 50, "N", "N"), **new** Order("D", 2, 50, "N", "N"), **new** Order("P", 1, 50, "Y", "Y"), **new** Order("P", 2, 50, "Y", "Y")}, 132.5}

};

}

@Test

@Parameters(method = "getParamsForGetTotalChargeValidValues")

**public** **void** testGetTotalChargeValidValues(Order[] orderList, **double** expectedResult) {

Order order = **new** Order (orderList);

Printing print = **new** Printing ();

Charge charge = **new** Charge (order, print);

*assertEquals*(charge.getTotalCharge(), expectedResult, 0);

}

//Invalid test is not needed at here, since invalid orders will be handled by the order module in order class.

//Test for passPrintingRequest()

/\*

\* Charge module

\* Test Case: 5.2.1~5.2.5

\* \* Test Steps:

\* 1. Test the length of the expectedPrintList and the length of the printList that is passed to printing module

\* 2. For each of the orders, test each of the components of the orders to check whether the value of the components in

\* the expectedPrintList is same with the value of the components in the printList that is passed to the printing module.

\*/

**private** Object [] getParamsForPassPrintingRequestsValidValues() {

//1. Two order that has the same type, option, addOption1, addOption2

Order order1\_1 = **new** Order("D", 1, 2, "N", "N");

Order order1\_2 = **new** Order("D", 1, 3, "N", "N");

Order order1\_expected = **new** Order("D", 1, 5, "N", "N");

Order [] orderList1 = **new** Order[]{order1\_1, order1\_2};

Order [] expectedList1 = **new** Order[] {order1\_expected};

//2. Two order with different type, or different option, or different addOption1, or different addOption2

Order order2\_1 = **new** Order("D", 1, 2, "N", "N");

Order order2\_2 = **new** Order("P", 1, 2, "N", "N");

Order [] orderList2 = **new** Order[] {order2\_1, order2\_2};

Order [] expectedList2 = **new** Order[] {order2\_1, order2\_2};

//3. Three order: two same type, option, addOption1, and addOption2, one different

Order order3\_1 = **new** Order("D", 1, 2, "N", "N");

Order order3\_2 = **new** Order("D", 1, 3, "N", "N");

Order order3\_3 = **new** Order("D", 2, 10, "N", "N");

Order order3\_add = **new** Order("D", 1, 5, "N", "N");

Order [] orderList3 = **new** Order[] {order3\_1, order3\_2, order3\_3};

Order [] expectedList3 = **new** Order [] {order3\_3, order3\_add};

//4. Four order: two same type, option, addOption1, and addOption2, another two also has same type, option, addOption1, and addOption2

Order order4\_1 = **new** Order("D", 1, 2, "N", "N");

Order order4\_2 = **new** Order("P", 2, 3, "Y", "Y");

Order order4\_3 = **new** Order("P", 2, 5, "Y", "Y");

Order order4\_4 = **new** Order("D", 1, 3, "N", "N");

Order order4\_add1 = **new** Order("D",1 , 5, "N", "N");

Order order4\_add2 = **new** Order("P", 2, 8, "Y", "Y");

Order [] orderList4 = **new** Order[] {order4\_1, order4\_2, order4\_3, order4\_4};

Order [] expectedList4 = **new** Order [] {order4\_add1, order4\_add2};

//5. Four order: three same type, option, addOption1, and addOption2, other one is different

Order order5\_1 = **new** Order ("D", 1, 3, "N", "N");

Order order5\_2 = **new** Order ("D", 1, 9, "N", "N");

Order order5\_3 = **new** Order ("P", 1, 3, "Y", "N");

Order order5\_4 = **new** Order ("D", 1, 12, "N", "N");

Order order5\_add = **new** Order ("D", 1, 24, "N", "N");

Order[] orderList5 = **new** Order[] {order5\_1, order5\_2, order5\_3, order5\_4};

Order[] expectedList5 = **new** Order[] {order5\_3, order5\_add};

**return** **new** Object [] {

**new** Object [] {orderList1, expectedList1},

**new** Object [] {orderList2, expectedList2},

**new** Object [] {orderList3, expectedList3},

**new** Object [] {orderList4, expectedList4},

**new** Object [] {orderList5, expectedList5}

};

}

@Test

@Parameters (method = "getParamsForPassPrintingRequestsValidValues")

**public** **void** testPassPrintingRequests (Order[] orders, Order[] expectedPrintRequests) {

Order order = **new** Order (orders);

Printing print = **new** Printing ();

Charge charge = **new** Charge (order, print);

charge.passPrintingRequest();

Order [] actualPrintRequests = charge.getPrintList();

*assertEquals*(expectedPrintRequests.length, actualPrintRequests.length);

**for** (**int** i = 0; i < expectedPrintRequests.length; i++) {

*assertEquals*(expectedPrintRequests[i].getType(), actualPrintRequests[i].getType());

*assertEquals*(expectedPrintRequests[i].getOption(), actualPrintRequests[i].getOption());

*assertEquals*(expectedPrintRequests[i].getQuantity(), actualPrintRequests[i].getQuantity());

*assertEquals*(expectedPrintRequests[i].getAddOption1(), actualPrintRequests[i].getAddOption1());

*assertEquals*(expectedPrintRequests[i].getAddOption2(), actualPrintRequests[i].getAddOption2());

}

}

//Invalid test is not needed at here, since invalid orders will be handled by the order module in order class.

}

**P1: PrintingUnitTests.java (Printing module)**

**package** my.edu.utar;

**import** **static** org.junit.Assert.\*;

**import** **static** org.junit.Assert.*assertArrayEquals*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** junitparams.JUnitParamsRunner;

**import** junitparams.Parameters;

@RunWith(JUnitParamsRunner.**class**)

**public** **class** PrintingUnitTests {

Printing print = **new** Printing ();

//Test for queueRequest() in printing module

/\*

\* Printing module

\* Test case: 1.1.1

\*/

**private** Object getParamsForGetAndWrite() {

Order order1 = **new** Order("D", 1, 2, "N","N");

Order [] orderList = **new** Order[] {order1};

Order [] expectedOrder = **new** Order[]{order1};

**return** **new** Object [] {

**new** Object [] {orderList, expectedOrder}

};

}

@Test

@Parameters (method = "getParamsForGetAndWrite")

**public** **void** testGetandWrite(Order[] order, Order[] expectedOrder) {

print.queueRequest(order);

*assertArrayEquals*(print.getPrintList(), expectedOrder);

}

//Invalid test is not needed at here, since invalid orders will be handled by the order module in order class.

}

**RegressionUnitTestSuite.java**

**package** my.edu.utar;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Suite;

**import** org.junit.runners.Suite.SuiteClasses;

@RunWith(value = Suite.**class**)

@SuiteClasses(value = {AddCustomerOrderUnitTests.**class**,

CalculateEachDocChargeUnitTests.**class**,

CalculateEachPhotoChargeUnitTests.**class**,

GetTotalChargeUnitTests.**class**,

OrderUnitTests.**class**,

PassPrintingRequestsUnitTests.**class**,

PrintingUnitTests.**class**

})

**public** **class** RegressionUnitTestSuite {

}