# Msc Programme:

**F21M:**

**Advanced Internet Applications**

# Course F21AS:

# Advanced Software Engineering

Group Report Stage 1

Miss Sakura Komiya

Mr Victor Godayer

Mr Dimitrios Tsoumanis



1. Duties

The work during the process of program implementation was approximately equally distributed. More specifically, Sakura provide the team with a frame to move in, as well as what type of items we are going to use, which are books in our case. On the other hand, it was Victor’s idea the implementation of the Abstract class and the Interface. Dimitrios did the discount calculation and the output files method. Victor worked for the Manager class, as well as for the Worker class, in order process part. Sakura did the JUnit part. In the report process Victor was mainly consecrate in the technical report. Dimitrios handled diagram, summary of who did what and the summary on decisions for items- format of ID’s. Sakura did most of the development report, and some part of the technical report. The bottom line is that no one did nothing on its own. Because we used google code: [*https://aes-dsv.googlecode.com*](https://aes-dsv.googlecode.com) ,everything that someone did, was instantly available to the other parts, giving the possibility for a member of the group, to make any changes to improve the functionality of the program. One difficult part (time consuming) of the coursework, was the handling of exceptions, in order to cover all possible cases, that could lead to a ‘risky method’.

1. Specifications

This program meets the specification completely.

1. Summary

We use books as items. The format of ID’s for books goes like this:

* ID;[unit price];[quantity]

1. ID: is declared as a string originally, converted to an integer, consisting by a 3 digits number, making it a unique identification number (e.g. 402, 503,…)
2. [unit price]: is declared as a string originally, converted to a double
3. [quantity]: is declared as a string originally, converted to an integer

In the case of the orders, we use the following format:

* ID;[customerID ]; [itemID]; [quantity]

1. ID: is an integer starting from value 1
2. [customerID]: the ID of a Customer is composed by a KEY string and 3 digits separated with a hyphen. In our case we deal with two types of customers: VIP and STD (stands for standard). How the price is calculated in each case we will describe that below.
3. [quantity]: is declared as a string originally, converted to an integer

In both txt files: Items and Orders, details are separated with semicolon which will be used as tokens in browsing procession.

### Price Calculation

The basic logic will be based on the type of the customer and the quantity ordered. For a normal customer (e.g. ID = STD-001), if the quantity is over 50 there will be a discount of 10% whereas a VIP customer (e.g. ID = VIP-001) there is a basic discount of 10% even below 50 items and an additional discount of 5% for each range of 50. Here are some possible cases of pricing for a book with ₤5.00 as unit price:

* Orders with q = 49 (case 1 : quantity <50)
  + 1. VIP customers

*Total Price = (5.00 \* 49) \* 0.9 =* ***₤220.50*** *→ 10% discount*

* + 1. STD customers

*Total Price = 5.00 \* 49 =* ***₤245.00*** *→ no discount*

* Orders with q = 100 (case 2 : 100 ≥ quantity ≥ 50)
  + 1. VIP customers

*Total Price = (5.00 \* 100) \* 0.9 – (5.00 \* 100) \* 0.05 =* ***₤425.00*** *→ 10% discount + 5%*

* + 1. STD customers

*Total Price = (5.00 \* 100) \* 0.9 =* ***₤450.00*** *→ 10% discount*

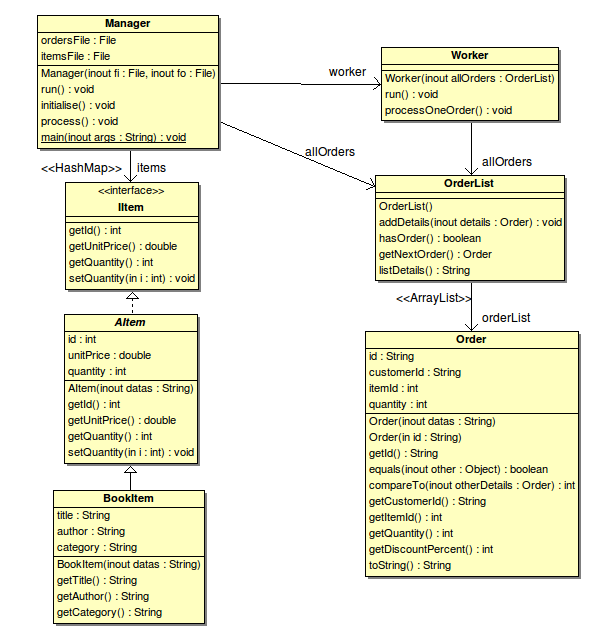
* Orders with q = 200 (case 3 : quantity over 100)
  + 1. VIP customers

*Total Price = (5.00 \* 200) \* 0.9 – [(5.00 \* 200) \* 0.05] \* 3 =* ***₤750.00*** *→ 10% discount + three times 5% since quantity is three range of 50 over 50*

* + 1. STD customers

*Total Price = (5.00 \* 200) \* 0.9 =* ***₤900.00*** *→ 10% discount*

1. UML diagram



1. Technical Report

Project is composed by 5 classes, 1 Abstract class and one interface.

Classes:

Order, Worker, BookItem, OrderList and the Manager

Abstract Class:

AItem

Interface:

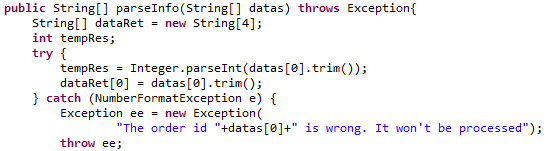
IItem

Mision of the classes:

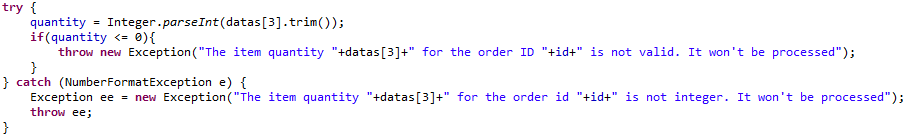
Order class: In that class, price calculation is taking place. Depending on the type of the client(VIP or STD) and the amount of Items, we make the calculation.

In this class we have several exceptions:

For instance the



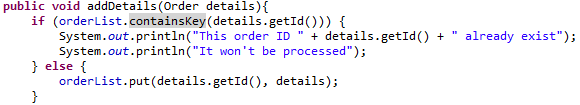
Another example is the following:



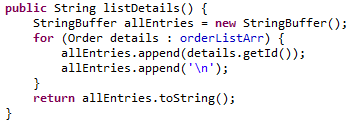
where we guarantee that the value contained in the quantity variable, would be have a non zero or negative value. Moreover, in the case that the risky thing in the ‘try’ part works, we have the ‘catch’ , for what to do if the exceptional situation happens, in the “risky method”.

BookItem class: This class inherits from the superclass Abstract ‘AItem’. The ‘BookItem’ automatically overrides the instance variables and methods of ‘AItem’ . On the other hand, it creates new varibles and methods for its own use. Main duties of this class are, to get the title, the author and the category of the book.

OrderList class: In this class, we create an ArrayList, based on and the HashMap, which provides an integer for the ID and a string for the order.



Method addDetails, is …..

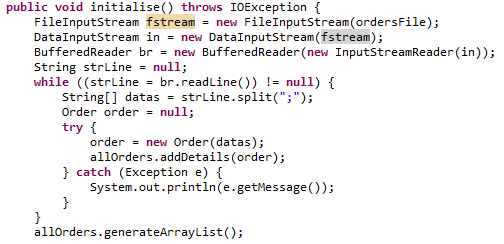


The above part of code, returns all the details of each order. Firstly we declare a reference variable and then we create an object. The for loop, is an enhanced for loop; for each element in orderListArr, assign the element to the details variable.

Manager class: Is the class that handles the 2 text files: Items & Orders.

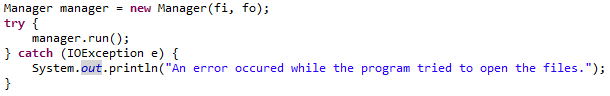
For this class we used several exceptions, in order to handle several cases.

For instance:



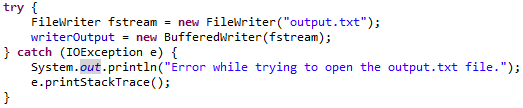
We try to create 3 new objects, and a catch in order to handle the risky method.

Another case of try/catch is the following:



when we run the reference variable manager and use catch for case occur reading txt files.

Worker class: We use this class in order to get the output.txt, with the results, according to the specifications.



We try to create the output.txt folder, and in case it fails it will display the message with the error in trying.

Interface describes the functionality of an object. We could also use it in order to avoid “multiple Inheritance” in future cases . Interface will provide a role that other classes can play, regardless of where those classes are in the inheritance tree.

We have created an abstract class in order to define a template for a group of subclasses and use at least some implementation code that all subclasses can use. With abstract class we can guarantee that nobody can make objects of that type.

**Data Structures**

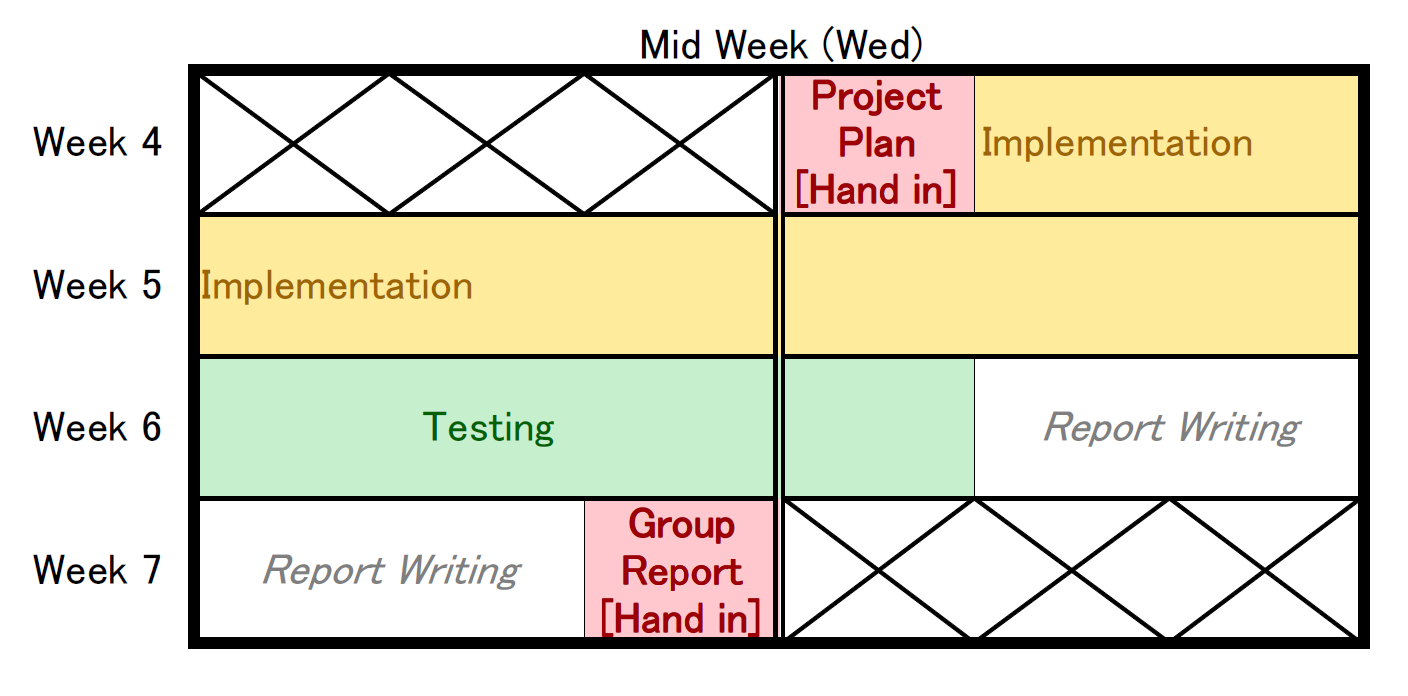
We use HashMap collection, because we want to make something like property list, where you provide a key and it gives back a value. In our case we use : “HashMap <Integer, IItem>”.

Integer is the value and IItem the key. We also use : “HashMap <Integer, Order>”,

There is also the ArrayList data structure,which comes as a default in the OrderList class

1. Development Report

Initial development schedule:



* Implementation
  + 1. Abstract Item class and Item Interface – Victor Godayer
    2. Updating warehouse function – Sakura Komiya
    3. Modifying the Worker class for order processing – Victor Godayer
    4. Discount calculating method – Dimitris Tsoumanis
    5. Adding output files writing method – Dimitris Tsoumanis
* Testing
  + 1. JUnit testing – Sakura Komiya and Victor Godayer
    2. Any modifications and correction to the code according to the testing results – each member of the team who has implemented the code.

Generally we abide the initial project, as much as possible. We had arranged meetings, in person, once per week, in order to solve the encountered problems, as well as to propose each one, ideas about the implementation of several frames of the program. What was quite “painful”, was the exceptions handling, from the point that all possible possibilities should be covered.

For the test process, we have used two input txt files, the Items and the Orders,

Items:

401;20.4;200; Oxford Dictionary;Oxford; Dictionary

402;12.34;46; Larousse Dictionary; Larousse; Dictionary

503;5;34; Internet for All; J Smith ; Textbook

200;5;34; Conquering the galaxy; D Vader ; Literature

Orders:

1; STD-001;402;60

2; VIP-002;503;10

3; STD-003;402;3

4; STD-002;401;51

5; VIP-002;401;1

The output result is as shown:

ORDER ID : 1 ITEM ID : 402 QUANTITY ORDERED: 60

CUSTOMER TYPE : STD

ORIGINAL COST : 740.4

DISCOUNT COST : 666.36

PROCESS STATUS : Unprocessed

CAUSE : Stock running low

---------------------------------------

ORDER ID : 2 ITEM ID : 503 QUANTITY ORDERED: 10

CUSTOMER TYPE : VIP

ORIGINAL COST : 50

DISCOUNT COST : 47.5

PROCESS STATUS : Processed

---------------------------------------

ORDER ID : 3 ITEM ID : 402 QUANTITY ORDERED: 3

CUSTOMER TYPE : STD

ORIGINAL COST : 37.02

DISCOUNT COST : 37.02

PROCESS STATUS : Processed

---------------------------------------

ORDER ID : 4 ITEM ID : 401 QUANTITY ORDERED: 51

CUSTOMER TYPE : STD

ORIGINAL COST : 1040.4

DISCOUNT COST : 936.36

PROCESS STATUS : Processed

---------------------------------------

ORDER ID : 5 ITEM ID : 401 QUANTITY ORDERED: 1

CUSTOMER TYPE : VIP

ORIGINAL COST : 20.4

DISCOUNT COST : 19.38

PROCESS STATUS : Processed

---------------------------------------

SUMMARY

Total income: 1706.62

Total item sold: 65

Total customer: 4