F21AS ADVANCED SOFTWARE ENGINEERING COURSEWORK

Project Initial Plan

Students : Dimitris Tsoumanis / Sakura Komiya / Victor Godayer

STAGE 1

1. ITEMS AND ORDERS FOR THE WAREHOUSE

We have decided to use books as item and here are the IDs and other description each book has:

- ID;[unit price];[quantity];[Title of the Book];[Author];[Category]
    a. ID: it is composed by one letter for the category of the book and two digits for the item (e.g. A01 for a dictionary and B01 for a foreign literature)
    b. [unit price] and [quantity] are numbers where prices will be transformed into Double after reading in the text file and the quantity into Integer designing the amount of the item remaining in the warehouse
    c. other fields are in String and are here to describe the item's specs

The orders will be input in the following format:

- ID;[ID customer];[ID item];[quantity]
    a. ID: an unique Integer starting at 1
    b. [ID customer]: the ID of a customer is composed by a KEY and three digits separated with a hyphen. In actual stage we have only came up with two types of customers: VIP and STD (stands for standard). How these keys affect the price will be described in the calculation part below.

In both text files, details will be separated by a semi-column which will be used as tokens while browsing for processing.

2. 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)
  - *a.* <u>VIP customers</u>

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

  - *b.* <u>STD customers</u>

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

- Orders with q = 100 (case 2 : 100 ≥ quantity ≥ 50)
  - a. <u>VIP customers</u>

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

  - b. <u>STD customers</u>

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

- Orders with q = 200 (case 3 : quantity over 100)
  - a. <u>VIP customers</u>

*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*

  - b. <u>STD customers</u>

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

3. Development Plan

- Implementation of Items and Workers

a. Items

We are planning to implement an interface and an abstract class for the Items because of their unique type (only books). This part of the code will be mainly taken care of by one student and if any trouble occurs with a help of the others.

The items will be identified by its id using the *HashMap* functionality (ID in Integer related to a unique Item)

Another class specifying the Book Items will also be implemented. It extends the abstract Item class and will only deal with the title, author and category. In this way, it would be easier when/if the warehouse decides to store new types of items in the future.

In the constructor of an Item there must be an exception handler.

b. Workers

At this point, a worker only receives the order details. In this way, the worker has no way to know the warehouse availability. We are thinking of passing into parameters the items list when an order is processed in the *processOnOrder* method.

- Price Calculation Function

The function will be added into the Order class. Based on the quantity and the customer ID, it will return the discount percentage for the final calculation when a worker processes the order.

A single student will be assigned to this task.

- Updating warehouse

It consists in updating the Items' text file with the correct quantity stored in the warehouse after an order has been processed. A function at the end of the Order class will be added which will subtract the amount of items sold from the initial quantity in the text file at the corresponding line.

- Output files writing

Two text files will be created, one for processed orders and another for those which has not been able to proceed due to a lack of amount in the warehouse. One of two students will be assigned to this task; they will be modifying the Order class.

- Testing

We have been thinking of adding JUnit to the real case testing. Functionalities that must be working for this stage are:

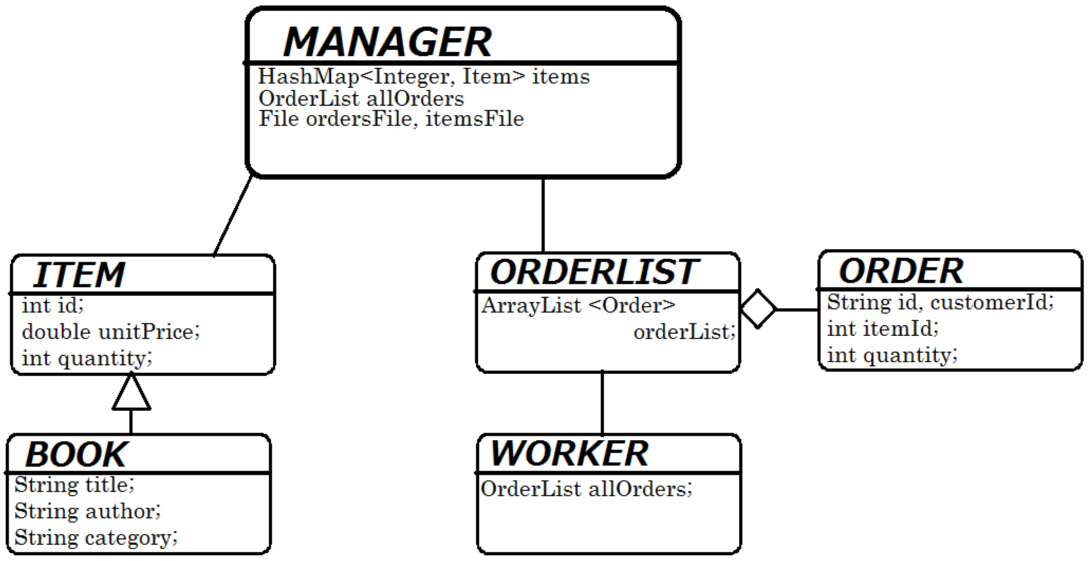
a. Orders can be processed without any problem
b. Correctness of the item processed
c. Creation and update of the output files (text)

The JUnit code should not be very long since there are small amount of functionalities and will be assigned to one or two students. Several input files for the items and orders will also be created for different cases.

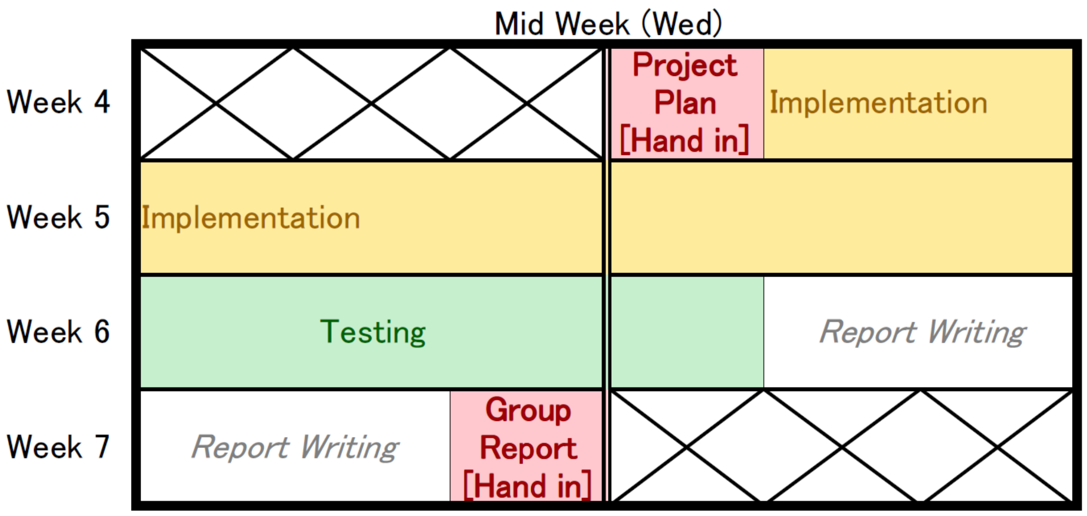
- Report writing (due by 22nd February 2012)

Bugs, problems and difficulties will be noted as the project goes on. All the speculations and decisions made for the programs should be put together and compared to this report plan and also to the requirements of Stage 2 for a smooth transition to the latter. We have already created our Google Code Hosting Page (*https://aes-dsv.googlecode.com*) so that anyone from our team can modify the code. Diagrams and other graphical representation of the code will be added into this report. Based on how we have advanced in the project, writing the report should take at least four to five days.

4. Initial Class Diagram (Simplified)



5. Initial Project Plan for Stage 1



			Mid Week (Wed)	
Week 4	✕	✕	Project Plan [Hand in]	Implementation
Week 5	Implementation			
Week 6	Testing			*Report Writing*
Week 7	*Report Writing*	Group Report [Hand in]	✕	✕

Choices and decision for the student assignation may be modified:

- Implementation
  a. Abstract Item class and Item Interface – Victor Godayer
  b. Updating warehouse function – Sakura Komiya
  c. Modifying the Worker class for order processing – Victor Godayer
  d. Discount calculating method – Dimitris Tsoumanis
  e. Adding output files writing method – Dimitris Tsoumanis

- Testing
    a. JUnit testing – Sakura Komiya and Victor Godayer
    b. Any modifications and correction to the code according to the testing results – each member of the team who has implemented the code.

STAGE 2

For this stage, new functionalities must be added according to the requirements:

- Change the workers into threads
    - ✧ Modification on the Worker class should not take long but we will have to think of how orders are dispatched between each worker and how each time an order is processed the worker reads the correct line of the order text file.
    - ✧ Since many workers will be modifying the warehouse storage data at the same time, lock functionality must be considered. We have noticed that an order is a Comparable in the original code so we are currently thinking of comparing the type of customer (VIP or STD) and adding a priority comparison method.
- Add a graphical user interface for the workers
    - ✧ This part will probably be implemented at a later stage of the project, when all the functionalities have been set up.
- Output text files at the end of the day with statistics
- Time control

Stage 2 requires an agile programming method and not a waterfall implementation as it consists on adding new functionalities to an already existing project. Still, by Monday 12th March 2012, all the implementation and testing must be over so that we have time to write the final deliverable.