Jordi Alfonso Poza
Eric Navarro López

Object Oriented Programming

Report Lab4:

The aim of this seminar was to design a system to manage the Online Store that we started in the previous seminar. From the previous session we reused the Item class hierarchy and the User class hierarchy that we designed. Overall, the task was to define a set of classes that could store data and answer queries related to the store, and to draw the class diagrams showing the relations between classes.Keeping in mind that the class diagram had to be a connected graph.

Overall, this lab has not been a positive experience for the team owing to the fact that it was over the top extensive as the last one or more and that we were almost not able to grasp a better idea of what we needed to implement for the code. Nonetheless, it still has presented quite an unforgiving challenge because of some parts of the code and the fact that lately we were required to hand in a considerably large amount of deliverables aside from the term exams and find the time to do so.

We started working on the Store class that we designed during Seminar 3.
The store needed to manage users, items and packages It also needed to keep a record of the sales for bookkeeping. We tried to be faithful to the asked approach by making each sale consist of an item, a user (who bought the item), a date at which the sale took place, the date at which the package was sent and the package used.
In addition, we had to include a ”Date” class that represented the date of a sale or the deadline of an auction item, instead of using strings to represent dates.
We then added the necessary attributes and relations to be able to perform multiple tasks such as:

-Buy operation of the Buyer user.
-Sell operation of the Seller user.
-Inventory operation of Admin user(consisting in computing the price and profit of the items that had not been sold yet.).
-Expel user. (Updating the class diagram to reflect the changes of the new classes and relations).

Alongside to this we also had to design and add The respective implementation of objects that could be sold and that were taxable as well.
Said interface does include the tax percentage (as a static variable).
In addition, we also made it possible to access the real price of the object (without IVA), the IVA tax percentage, and the total price of the object (including IVA).
The interface also includes a Sum operation that takes an object of the interface as argument and returns an object of the same type whose prices are the sums of the prices of the original objects. It must be noted that this operation is able to compute the total price of a collection of objects, including the IVA tax of the sales.
We also made use of the interface “Comparable” to be able to order sales by date and items by profit(it contains a single abstract method public int compareTo(Object o).

It must be noted that we experienced some problems when it came to work with ManageAuction owing to the fact that we had to take into account the function Sell so that they worked together properly. We decided to make sellers an attribute of ManageAuction (which was not) so that ManageAuction was able to execute Sell properly.

In addition, we also realized that it was needed to take into account and create an implementation for every one of the different possibilities when it came to what could happen in the auction e.g: if the buyer bought the auctioned product or if actually no one bought said product.

Aside from this , we also realized a mistake that we made regarding the process of buying products or selling them. Previously, when a user bought an item this item was assumed to exist from a seller that was not implemented which was wrong.
Currently, when an user buys an item there is a seller that actually has that good register as sold and has it subtracted from him. Same opposite thing would happen if it were given the inverse situation.

Moreover, we changed the class Date to LocalDate since we realized that OnlineStore needed the attribute CurrentDay and we needed to be able to change CurrentDate using a method. Yet, the class Date was not adequate for this said method this being the reason we decided to use this new and more advanced , with more functions; class that allowed to increment the date day number by using the function AddDays that increments the number of days via an input parameter. This made it also possible to execute ManageAuctions method with the new date.

Later, we reached the conclusion that OnlineStore (the main class) needed an instance of OnlineStore in order to be able to adapt to the statics from attributes and methods except main and initialize the attributes on the newly created constructor method instead of in the main. That also required the parameters of the methods to be changed.

In a Nutshell, the team has worked really hard pushing itself and its capabilities to the limit to try to come up with a reliable solution on time, nonetheless; it has been tough (especially because a minor change conditioned almost the entire structure of the code). Yet it has been useful to put to the test all of our resources and try to come up on top of the situation.