CS 2212 B

Introduction to Software Engineering

Project Description

Project 2

Kostas Kontogiannis

1. Introduction

This document presents information on your second deliverable.

For the second deliverable (Project 2), you will:

a) enhance the Software Requirements Specification document you have submitted for Project 1 by i) adding activity diagrams for each use case, and ii) providing a unified domain model that stems from the three use cases;
b) provide implementations for the i) state of an item using the State Design Pattern; ii) restock notifications using the Observer Design Pattern and; iii) factories using the Factory Design Pattern.

Please refer to the system description provided in Project 1 (Section 2) “Overall System Structure” for a documentation of the system’s packages and classes.

2. Use Cases

For the second deliverable you will consider the same use cases as in Project 1. These are provided below:
UC1: A user sends a placeOrder request to the system (i.e. the Welcome Proxy). The user needs to be authenticated. If the user is authenticated then his/her request is handled by the back-end system. Consider that there is a database with user credentials. The users can be authenticated using user name and password or just a PIN. In special cases can be authenticated by calling an agent on the phone.
UC2: High quantity orders (i.e. > 10 items) and Low quantity orders (i.e. less or equal to 10 items) should be handled in different jurisdictions (i.e. by different components). It has to be transparent to the user which component handles the request. In order to service a placeOrder request the system has a) create a new Order, b) reduce appropriately the stock for this item, c) create an invoice, and d) receive payment. You can consider that there is an external banking system on the back-end and corresponding data bases that hold the orders and keep received payment details. In exceptional cases, payment should be done in advance before the item is sent to the buyer.
UC3: Once the final price of an order is ready to be processed there may be special discounts for specific users so that the total price may not be just the sum of the prices of the individual items in the order. There are three types of buyers, the Gold status (they get a 10% discount for all orders more than 100 dollars), the Silver status (they get a 2% discount for all orders more than 500 dollars), and the Bronze status buyers (they

get a discount of 2% for discount for all orders more than 1000 dollars). In exceptional cases, there may discounts of 10% for buyers approved by the manager (management decision process). Consider that there is a database that keeps the status of the buyers.

3. Tasks for Project 2

Part A.

1. For each use case above, provide a domain model. Work in collaboration with your team members so that all domain model classes from the individual use cases are compatible.
2. For all three domain models compiled, create one unified model which is the one you will submit. For completeness, you are allowed to extend the Domain Model with additional classes beyond the ones stem from the three use cases above.
3. For each use case above, provide an activity diagram. Make sure that the activity diagrams compiled from each use case are all compatible between them (i.e. refer to the same processes, use common terminology etc.)

Part B.

Implement the following:

I1. The appropriate use of the State Design Pattern. An Item (see `item.java` in the `model` package) is associated with a state through its `state` attribute. Note that it is the state object of an Item which is the one that is finally handling a deplete operation. The deplete operation in the state object will be the one to notifyViewers of the Item (see `model.item.states` package). As noted above, states are created by a factory which is used by the Item to create and assign to itself the appropriate state (i.e. In Stock State, Out Of Stock State and Low Stock State). The different states implement the `ItemState` interface (see `item.states` package) that is the `deplete` and the `replenish` operations. Also note that a `deplete` or a `replenish` operation on a state may bring the item into a new state (e.g. a deplete operation on the state of an item can make the item to go from “Low Stock” to “Out Of Stock” state). Also, note that the three possible states In Stock State, Out Of Stock State and Low Stock State are to be created by the appropriate Factory (see point I2 below for a factory for the *ItemState* interface).

I2. The appropriate factories. Make sure you implement the Factory Method design pattern (more specifically a variation as we have discussed in class). See the `AggregatePricingStrategyFactory.java` as an example given to you already on how to use the Factory Method Design Pattern. You will need factories for the classes implementing the *ItemState*, IndividualPricingStrategy and, RestockStrategy interfaces.

I3. The appropriate notification mechanism. Make sure you implement the Observer Design Pattern. When a deplete operation is called on an Item, this is handled by the deplete operation of the associated state attached to the Item (see I1 above). Then, the deplete operation of the state calls the `notifyViewers` of the item which notifies all the attached to the Item viewers (see the `viewer` package) by invoking their inform method. An Item has a list of viewers but it is the State that initiates the notification process. Currently there are two classes implementing the Viewer interface (i.e. StockManager and Messenger – see the `viewer` package). So the sequence of invocations is:

1. deplete operation on an Item → 2. deplete operation on the state → 3. State invokes the notifyViewers on the Item → 4. The notifyViewers invokes the inform method of each attached to the Item Viewer to do its work.

4. Deliverables for Project 2

Work on, and enhance, the Software Requirements Specification (SRS) document you have submitted for Project 1. Submit with this enhanced document a complete report with the activity diagrams (one for each use case), the unified domain model, and the description of the classes in the domain model. Please revisit your use case descriptions, your use case diagrams, and your sequence diagrams you have submitted on Project 1 so that you take into account the comments you have received, and make sure they are all consistent with each other (i.e. common use of processes, actors, messages etc.). The idea is that in each deliverable cycle to enhance and perfect the SRS.

Submit the following:

1. An enhanced SRS with the use case diagrams and the sequence diagrams for UC1, UC2, and UC3 from project 1, which enhanced SRS will now contain also:
    a. The activity diagrams for UC1, UC2, UC3.
    b. The unified domain model stemming from UC1, UC2, UC3, along with a short description (one or two sentences) of each class in the model.

    The report has to have a cover page with your group number, your names, and your emails. There should be short but clear explanations of the diagrams. Please take into account the comments you received in Project 1.
2. An archive file with your project in a form that can be imported in Eclipse and tested using the input data files you should also include in the archive (i.e. the driver file, buyer_file, etc.).

Your new SRS will be an enhanced resubmission of your previous one.

Submit your deliverables at OWL by Wednesday March 18, 23:59:00.

Please submit also a hard copy as you did for Deliverable 1.

The implementation of Project 2 will be demonstrated along with the implementation of Project 1 on a date which will be announced.

APPENDIX

In order for your system to work with these new extensions you need

a) to change the `order` method in the `StockManager` by the one below

```
public void order() {
        System.out.println("restocked with " + restockDetails);
        Supplier supplier = new Supplier();
        supplier.supply(restockDetails);
        restockDetails.clear(); //was commented
}
```

b) to change the constructor of the StockManager class to the one below:

```
private StockManager() {
        super();
        // restockDetails.put("apple", 50);
        // restockDetails.put("pear", 50);
        // restockDetails.put("mango", 50);
        // restockDetails.put("onions", 50);
        Thread t = new Thread(this);
        t.start();
}
```

and;

c) to complete the inform method of the StockManager class. Here your inform method will be updating the restockDetails map by a selected by you quantity e.g. `restockDetails.put("apple", 50);`
In Project 3 this will be replaced by a strategy.