Metropolitan State University, St. Paul, Minnesota
ICS 372 Object-Oriented Design and Implementation
Group Project 1

1 Due Dates and Goals

Due:

11:59 PM on April 4, 2020

1.1 Goals:

1. Perform use-case analysis techniques to discover and specify the conceptual classes.
2. Use design principles to translate conceptual class design into an appropriate set of abstract and concrete classes and interfaces
3. Efficiently develop systems using design patterns, including Facade and Singleton
4. Use principles of the agile methodology by following the Unified Process
5. Use the Unified Modeling Language to document work
6. Implement a design utilizing structures such as classes and interfaces,
7. Work in small groups
8. Employ Java coding standards.

2 The Problem

You have to analyze, design, and implement a system using object-oriented principles. The system concerns an application for a company that sells cloth washers, cloth dryers, kitchen ranges, dish washers, refrigerators, and furnaces. Each appliance has a brand name and a model name. The company stocks multiple brands of different models and keeps track of them. It also keeps track of its customers (name and phone number).

Customers can order appliances. If the required quantity is not on stock, the request is put on back order and when enough quantity is on stock, the sale is completed. (Back order is not possible for furnaces.)

The company remembers the total amount from all sales for all types of appliances. To help you better understand what the organization maintains **conceptually**, here is an example.

Brand Name	Model	Stock	Price
b1	m1	4	300.0
b1	m2	0	400.0
b2	m3	1	500.0

Suppose a customer C1 orders 1 unit of the machine with brand b2 and model m3. That can be immediately delivered. The company now has $500 from sales. in total sales. The stock for this appliance is now 0.

Now assume a second customer C2 orders 2 units of model m2 of b1. Then, a third customer C3 orders 1 unit of the same model. Neither can be delivered because there is none in stock. These are now on back order.

At a later stage, assume that 2 units of (b1, m2) comes to stock. Since C2 ordered first, C2's back order is satisfied to bring down the stock to 0, and C3 will have to wait. The total sale would now be $1300.

Here are exceptions related to some of the appliances.

1. For furnaces, the company requires that the model be in stock at the time of order. All other appliances use the concept of backorders.
2. Refrigerators have an extra attribute, capacity, which is in liters. A furnace has the extra attribute Maximum Heating Output in British Thermal Units (BTU).
3. The company allows repair plans for cloth washers and cloth dryers it sells. The monthly payment amount is an extra attribute for these two appliances.
4. It is expected that, in the future, different appliance types may end up adding more unique attributes.

The amount charged to the customer for all purchases and repair plans are kept track of separately.

3 The Business Processes

The business processes are:

1. Add a model. The model could be any of the types of appliances. The extra attributes needed for the appliance (such as the monthly repair plan cost for washers and dryers, capacity of refrigerators, heat output for furnaces) are also entered at the time the model is added. The system creates an appliance id, which it remembers. This use case does NOT add to the inventory.
2. Add a customer. The company keeps track of the name and phone number. The system creates a customer id, which it remembers.
3. Add to Inventory. The user may add any of the different types of appliances. The user enters the quantity, which gets added to the total in stock. Any back orders are processed in the chronological order and the total from sales is updated. The user refers to an appliance using the appliance id.
4. Purchase: The actor identifies the appliance by its id and the customer by the customer id. The actor enters the quantity as well. If there is enough on stock, the purchase is immediate. Otherwise, this goes on back order. There is no concept of a backorder for a furnace.
5. Enroll in a repair plan. The user id and the eligible appliance id are input. The system should do nothing if the appliance does not have a repair plan.
6. Withdraw from a repair plan. The user id and the eligible appliance id are input.

7. Charge repair plans. Done when the company desires, probably on a monthly basis. The customers' accounts are updated. (You need not worry how often this gets invoked.)
8. Print revenue from all sales and repair plans. (That means two numbers, properly annotated.)
9. List appliances. The user can list all appliances or a specific type of appliance. Each item with its id, brand name, model name, price, and items on stock are listed.
10. List all users in repair plans. The user name, phone, id, their account balances, and the appliances (brand and model) for which they have enrolled.
11. List customers. The list should print the details of all customers, indicating whether the customer is enrolled in a service plan.
12. List all backorders: appliance brand, model, and quantity.
13. Save data to disk.

4. The User Interface and Other Aspects

For the purposes of ensuring uniformity in grading, I ask the following.

1. The interface must be non-GUI, but command driven, just like the library system. I should be able to invoke the business processes by typing in a number as listed under the business processes above. Also use 0 for exit and 14 for help.
2. When the program starts, it should give an option to look for and load existing data on stable storage. If the user answers in the affirmative, that data should be loaded and used.
3. In general (that is unless specified elsewhere), the feel of the interface should be similar to that of the library system. (Obviously, the functionality is different.) This includes the nature of inputting commands and information, displays, customer id, etc.

When the user interface starts, it should give an option to programmatically set up a test bed, by prompting as follows. (This happens if and only if the user does NOT wish to use a saved file.)

Do you want to generate a test bed and invoke the functionality using asserts?

If the user answers **y** or **yes**, the program

1) creates and enrolls a number of customers (at least 5) and appliances (at least 20). Every type of appliance should have at least two instances. You could generate these by employing randomly-generated names, phone numbers, stock in hand, and so on, or get them from pre-defined arrays. You can use a loop to invoke the methods in the façade.
2) invokes the façade methods for business processes numbered 1 through 6. The façade methods for each of these should have an appropriate return type other than void. Using assert statements, it checks the return values and ensure that the returned values are correct.

After completing Step 2 above, the user interface enters the loop to interactively receive and process the commands from the user.

4 Deliverables

You need to submit two files: one for analysis and design, and a second one for implementation.

4.1 Analysis

Submit a single PDF document that contains all of the following.

1. A use case for business processes 1, 3, 4, 5, 7, 8, and 10. There is no need to submit use cases for the others.

2. The conceptual class diagram.

The use cases must be typed and the class diagram drawn using an appropriate software tool. I will ignore all parts of the document that are not typed. I will also ignore any document that is in any other format (Word, GIF, JPEG, etc.) If you submit multiple documents, I will choose one of the PDF files and ignore everything else. Be sure to ensure that the information is all in the portrait mode wherever possible. Do not have any part of the information cut off or unviewable in any way.

You can talk to me to get any clarifications you need, especially with respect to the requirements.

4.2 Design

1. A sequence diagram for each of the seven use cases you were asked to write for analysis.
2. The physical class diagram.

The sequence and class diagrams must be drawn using an appropriate software tool. I will ignore all parts of the document that are hand drawn. I will also ignore any document that is in any other format (Word, GIF, JPEG, etc.) If you submit multiple documents, I will choose one of the PDF files and ignore everything else. Be sure to ensure that the information is all in the portrait mode wherever possible. Do not have any part of the information cut off or unviewable in any way.

Draw the diagrams clearly. The following are common mistakes and improprieties I have observed over the years.

1. Placing the classes awkwardly: Class A extends class B, but A is not placed below B.
2. Not using proper lines for class extension, composition, etc. (I suggest that you use Visio, because it labels the icons for interface implementation, class extension, composition, etc; that gives you better clues as to which stencil to use for what.)
3. Not identifying the relationships properly.
4. Not drawing the lines (for method calls) in the sequence diagrams horizontally.
5. Not using the proper notations (**+, -, #**) for public, private, and protected members.

Such mistakes will result in loss of credit.

4.3 Implementation

Submit the entire application as an Eclipse project. Do not create a module. All code must be properly formatted and documented. You are welcome to use the library code provided on D2L and adapt the functionality. The documentation, naming conventions, and code formatting must follow the coding conventions we have discussed and documented before. Refer to the library code for more examples.

Document and lay out your code properly as specified under ICS 372 Coding Standards.

5 Grading

Your assignment will be graded as written in this section.

While doing the project ensure that

1. The use cases reflect the business processes.
2. The sequence diagrams implement the use cases.
3. The class diagrams are based on the information gathered from the sequence diagrams.
4. The Java code is based on the class diagrams.

5.1 Analysis (30 points)

The grade will be based on the use cases and the conceptual class diagram.

Component	Number of Items	Points	
		per Item	Total
Use cases	7	3	21
Overall	1	3	3
Conceptual class diagram	1	6	6

To get full credit for a use case:

1. It should reflect the business process completely.
2. It should be written properly.

The miscellaneous component is to take care of the situation when several use cases have some (possibly different) minor error (usually the way it is written as opposed to being faithful to the business process), that were not individually penalized because that would be unfair. For example, a single misspelling in one use case will not result in any loss of credit: I think that would be too harsh. But repeated instance of such errors are penalized through this component.

Part of the credit for use cases is for presenting it properly. Unless the use case reflects the business process, there will be no credit for presentation.

Be sure to properly label the conceptual class diagram. You must have all conceptual classes in the diagram, the relationships must be properly labeled and the correct notations should be used.

5.2 Design (33 points)

The grade will be based on the sequence and class diagrams.

Component	Number of Items	Points	
		per Item	Total
Sequence diagrams	7	3	21
Design quality	1	6	6
Class diagram	1	6	6

The sequence and class diagrams will be graded based on

- Quality of design
- Correctness with respect to the requirements
- Quality of the presentation

Each sequence diagram should correctly implement the functionality of the respective use case and should be properly drawn. If a sequence diagram is meaningless (w.r.t the use case), no credit will be given for it, even if it is well drawn.

Design quality will depend on how well you implemented the use cases and how well the classes are properly structured. As long as the design is understandable, I will not consider the presentation (how well a sequence diagram is drawn) itself into account.

5.3 Implementation (37 points)

This will be based on accuracy, program structure, coding and documentation, etc.

Criterion	Points
Class and method documentation	6
Correctness (My testing)	15
Automated testing	10
Coding conventions and structure	6

Correctness will be based on the following.

1. Are all use cases realized?
2. Is the user interface as specified?
3. Are classes designed as per design?
4. Are results displayed properly?
5. Is the interface similar in feel to the library system? (For example, is the customer id relatively simple to key in?)
6. Is there an automated testing facility? Does it test all the specified functions? Does the program pass those tests?

If you don't submit an Eclipse project as a zip file that opens correctly or is not executable directly, you will lose 10 percent of the credit for implementation.

6 Agile Development

I suggest you analyze, design, and implement the code using the agile development approach. Perhaps you can write two use cases: to add customers and appliances, draw the sequence diagrams quickly (perhaps hand drawn), and implement the code (maybe working in a pair), and test that out. A (quick) creation of the user interface and the facade will be immensely useful for getting off to a great start.

One team member can play the role of the end-user.

After the above two are completed, you can add a few more use cases and eventually complete the process in three or four iterations.

The Wikipedia article on agile development has some interesting and useful sections.