Metropolitan State University
ICS 372 Object-Oriented Design and Implementation
Spring 2021

Group Project 1

Due: 11:59 PM on April 2, 2021
Points: 100

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 the Unified Modeling Language to document work
5. Implement a design utilizing structures such as classes and interfaces,
6. Work in small groups
7. Employ Java coding standards

The Problem

Create, test, and document a system for an application with business processes and other requirements described below.

A small co-op grocery store works by having members join it by paying a certain fee. The business processes (12 in number) /system functionalities are:

1) Enroll a member: the system enrolls a member and remembers him/her; it keeps track of the name, address, phone number, date joined, and the fee paid. Also, the system creates and maintains a unique id for each member. Only one member is added when this functionality is invoked.

2) **Remove a member**: If a valid id is received, the corresponding member is removed; the system would need the member's id for this purpose. Only one member is removed when this functionality is invoked.

3) **Add a product**: From time to time, the store decides to carry a new set of products. Relevant attributes input by the user are: product name, id, stock in hand, current price, and a reorder level for the product. Once the quantity in stock reaches the reorder level or below, the product will be automatically reordered. No two products have the same name. So, if the store carries coconut milk, there could be products such as "Low-value coconut milk" and "Unsafeway coconut milk." Only one product is added when this functionality is invoked. Immediately after a new product is created, the system generates an order for the product for twice the reorder level.

As an example, suppose we have a new product “1 lb packets of Milano Pasta” with a reorder level of 100 lb, When the product is created, 200 lbs. of the product are ordered.

4) **Check out a member’ items**: The member comes to the check-out counter with a cart of grocery items. The cashier inputs the product id and quantity of each item in the cart. The system computes the price and, of course, computes the total price. All members pay by cash. The system must display the individual items, the number of units, unit price, and price for the item. For example, if the user purchases three gallons of milk, the display might be

   Milk 3 $4.50 $13.50

   The total price must also be displayed.

   When a customer is checked out, reorder twice the reorder level for any product whose level reaches the reorder level (or below that value). The system must display a message saying that the item will be reordered, how much was reordered, and what the order number is.

5) **Process shipment**: Whenever there is delivery of items from a supplier, the stocks must be updated. The system needs the order number. We assume that the quantity ordered has been delivered. Multiple products may be processed, one by one. Display the product id, name, and the new stock.

6) Change price: Changes the price of a product given its id. The system displays the product name and the new price.

7) Retrieve product info: Given a string, list all products with name that starts with this string. List the product name, product id, price, stock in hand, and reorder level.

8) **Retrieve member info**: Given a string, the system displays the member’s address, fee paid, and id of all members whose name begins with the given string. In general, there will be multiple members listed.

9) **Print transactions**: Given the id of a member and two dates (input in the mm/dd/yyyy format), the system prints all transactions for the corresponding member between the two dates (including both dates). The first date input by the actor must be validated to ensure that it equal to or earlier than the second date.

10) List all outstanding (not yet fulfilled) orders: Display the order id, corresponding product name, date of order, and quantity ordered.

11) List all members. Display the name, date joined, address, and phone number.

12) List all products. Display the product name, id, stock in hand, current price, and a reorder level for the product

The following could be considered business processes, perhaps, but let us not treat them that way. They form part of the functionality of the system.

13) Save: Saves all data to disk.

14) Help: Displays a list of commands.

0) Exit: Quits the application.

Make sure that wherever there is reasonable common functionality, factor out code fragments from similar classes and move them to/create super classes. Use generics where applicable.

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: 1 for adding members, 2 for removing members, etc. and 13 for saving data to disk. 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. Do not assume any specific directory structure on my system: use the current directory.
3. In general (that is unless specified elsewhere), the look and 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, help screen, etc.
4. Error messages must be as specific as possible.
5. The façade and user interface should communicate via the data transfer pattern, as implemented in the library system.
6. When the user interface starts, if the user does NOT wish to use a saved file, the program should give an option to programmatically set up a test bed, by prompting as follows.
7. The system should be organized into packages in a very clean manner.

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

If the user answers **y** or **yes**:
1) The program first creates a number of products (at least 20) and members (at least 5).
2) After completing Step 1, the program invokes the façade methods for business processes numbered 1 through 6. Using assert statements, it checks the return values and ensure that the returned values are correct. This step should be a thorough test of business processes numbered 1-6.

3) After the automated testing processing is complete, there should be at least 20 products and 5 members. There should have been at least a couple of outstanding orders, multiple products (at least 3 or 4) starting with the same sequence of 2-4 characters.
4) Finally, the program presents the command line interface, so the user can enter more test data using the interactive approach. The test bed from Step (3) must be available for testing in Step 4.

You may have to put the automated tester class in the business package, if you have provided restricted view of business objects in the ui package.

Things to be Submitted

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

Analysis and Design

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

1. A use case for business processes 2, 3, 4, 5, 8 and 9. There is no need to submit use cases for the others.
2. The conceptual class diagram.
3. Sequence diagrams for use cases corresponding to business processes 2, 3, 4, 5, 8 and 9.
4. The physical class diagram.

The use cases must be typed.

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

The sequence and class diagrams must be drawn using an appropriate software tool. I will ignore all parts of the document that are hand-written or 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.

Implementation

Submit the entire application as an Eclipse project. *Do not create a module*. All code (classes, interfaces, constructors, and methods) 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 before. Document and lay out your code properly as specified under the CodingStandards.pdf document. Refer to the library code for more examples.

Do not use hyphens in any identifiers, except final variables. For final variables, use capital letters and full words separated by hyphens.

I will provide no debugging support. You must resolve such issues within the group.

Responsibilities

Every student must write at least one use case and draw at least one sequence diagram. Label each sequence diagram and use case with the name of the person who did the work.

If a submission does not indicate that every student in the group has written a use case (or drew one sequence diagram), that student will not receive any points for analysis (design).

Grading

Your assignment will be graded as written in this section.

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 design.

Analysis (21 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	6	2	12
Miscellaneous	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 errors (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 more than one instance of such errors or several single instances of different minor 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 almost completely, 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.

Remember that conceptual class diagrams are not the same as physical class diagrams. This has been a confusion with a few groups in past semesters.

Design (36 points)

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

Component	**Number of Items**	**Points**	
		per Item	**Total**
Sequence diagrams	6	4	24
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.

Implementation (43 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	12
Organization into packages	4
Coding conventions and structure	6

Grading of correctness and automated testing 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. Does the implementation adhere to the principles of cohesion and coupling?
6. Is the user interface similar in feel to the library system?
7. 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 Java project as a zip file that opens correctly or is not executable directly, you will lose 10 percent of the credit for implementation.

Grading Issues Related to Group Work

Usually, all members of a group get the same grade. But I have had multiple groups disputes, which have often made grading a somewhat difficult process. I am forced to reserve the right to give students within a group different grades. This will not be done capriciously; I will be meeting with every group every week, so will have at least weekly updates on how things are going on within groups, which might obviate the need to assign different grades. I will inform the student involved before I give him/her a different grade.