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

1 Due Dates and Goals

1.1 Analysis and Design Due:

11:59 PM on October 10, 2017

1.2 Implementation Due:

11:59 PM on October 19, 2017

1.3 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. To 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 an organization (imagine a non-profit) that gets funding from individuals (donors).

The donors pay using credit cards. A donor may have multiple credit cards and pay a certain amount using each credit card. For example, Smith may have two credit cards, say CC_1 and CC_2 and may be paying 20 dollars using CC_1 and 25 using CC_2. All that information is stored by the organization.

To help you better understand what the organization maintains **conceptually**, here is another example.

Donor			Credit Card Info
Name	Id	Phone	
Smith	D1	123-45678	(num1, $20), (num2, $10)
Hardy	D2	123-45679	(num3, $25)
Laurel	D2	223-45678	

The organization has three donors: Smith, Hardy, and Laurel. The organization has assigned them ids (D1, D2, and D3) and stores their phone numbers. In addition to remembering these three pieces of information for every donor, the organization also stores the credit card number and the amount to be deducted from each donor every "donation cycle."

Suppose donations are processed once a month. At that time, Smith will have `$20` deducted from credit card with number `num1` and `$10` deducted from credit card with number `num2`. The organization will get `$25` from Hardy who has a single credit card with number `num3`. As it stands Laurel does not have an associated credit card and does not pay. All these transactions are stored by the organization.

Notice that donor may have zero or more credit cards. Two donors never share a credit card.

To avoid too much complexity, the problem statement will be silent on how frequently the donations occur. We shall see how they occur when we talk about the business processes.

3 The Business Processes

The business processes are:

1. Add a donor: the information to be supplied by the actor are the donor name and phone number. The system generates a unique id.

2. Add a credit card: the information to be supplied by the actor are the donor id (the corresponding donor is the credit card holder), the credit card number, and the amount to be donated each time a donation occurs.

3. Process donations: the organization uses this process to get donations from each credit card, the amount stored along with the credit card. (As stated earlier, we don't worry about how often this process is invoked.) The system must generate and store transactions for each donation: the credit card information (card number, amount) and the date (month, day, and year) are stored for each donation. It displays the total amount collected.

4. List all transactions: the system displays the card number, amount, and the date (month, day, and year) for every transaction for every donor.

5. List all donors: the system displays the name, id, and phone for each donor. No credit card info is displayed.

6. List a specific donor. The actor will provide the donor id. The system must display the donor name, phone and card number and amount of all credit cards associated with the donor. No transactions should be output.

7. Remove a specific donor: The actor will provide the donor id. If the donor with this donor id exists, all information associated with the donor including transactions are removed.

8. Remove a credit card: The actor will provide the donor id and the credit card number. If the entriesare valid, the information for that credit card (card number and amount) will be removed. The transactions associated with the card are not removed.

9. Save the data. The data (basically, the donor info, credit card info, transactions) is saved for long-term use.

3.1 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 (and other stuff) by typing in a number as specified below.

Command	**Process**	**Result; Not all prompts and interactions shown**
0	Exit the application	A prompt to save (if and only if the data is "dirty")
1	Add a donor	The information (donor data) added
2	Add a credit Card	The information (card data) added
3	Process donations	Total amount received as donation
4	List Transactions	One transaction per line (see business processes)
5	List Donors	One donor per line (see business processes)
6	List Donor	Donor information on one line All credit cards, one card per line (see business processes)
7	Remove donor	Success or failure info
8	Remove credit card	Success or failure info
9	Save	Success or failure info
10	Help	A helpful list of commands (See the Library system behavior)

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, donor id, etc.

4 Deliverables

You need to submit three files: one for analysis, one for design, and a third one for implementation.

4.1 Analysis

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

1. A use case for each of the eight business processes. There is no need to submit use cases for save, exit, and help.

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 requirements.

4.2 Design

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

1. A sequence diagram for each use case.

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 improperieties I have observed over the years.

1. Placing the classes awkwardly: Class A extends class B, but Ais 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 naming the relationships.

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. 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.

5 Grading

Your assignment will be graded as written in this section and given in more detail under the rubrics attached to the submission folder.

While doing the project ensure that

1. The use cases reflect the business proecsses.
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 (32 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	8	3	24
Overall	1	3	3
Conceptual class diagram	1	5	5

To get full credit for a use case:

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

Take a look at the rubrics for details.

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 (36 points)

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

Component	**Number of Items**	**Points**	
		per Item	**Total**
Sequence diagrams	8	2	16
Design quality	1	12	12
Class diagram	1	8	8

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.

Look at the rubrics for more details on grading.

5.3 Implementation (32 points)

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

Criterion	**Points**
Documentation	6
Correctness	20
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 donor id relatively simple to key in?)

Be sure to take a look at the rubrics for this part of the project.

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 that you analyze, design, and implement the code using the agile development approach. Pethaps you can write one use case: to add donors and credit cards, draw the sequence diagrams quickly (perhaps hand darwn) 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.