Lab 7

Demo Points

Story 28 (5 marks)

1. AreaComperableTest for area() succeeds and prints “area() is not defined.” (1 mark)
2. AreaComperableTest for areaLT() succeeds and prints “area() is not defined.” (1 mark)
3. AreaComperableTest for areaGT() succeeds and prints “area() is not defined.” (1 mark)
4. AreaComperableTest for areaEquals() succeeds and prints “area() is not defined.” (1 mark)
5. All JUnit tests run (1 mark)

Stories 29-31

For each story (5 Marks):

6. The test for area() succeeds (1 mark)
7. The test for areaLT() succeeds and prints “area() is not defined.” (1 mark)
8. The test for areaGT() succeeds and prints “area() is not defined.” (1 mark)
9. The test for areaEquals() succeeds and prints “area() is not defined.” (1 mark)
10. All JUnit tests run (1 mark)

Next Week

Next week will be a review of what we have done so far. We will be starting a new project that will contain a library of books. It is an application that you could use to keep track of things you lend to your friends.

In the following weeks we will move back and forth between the Shapes project and the Library project. We will use Shapes to explore new content: next week's lab will be adding exceptions to shapes when they cannot be formed rather than returning a special shape.

The Library program will take you through the process of building a program from the initial idea to the final delivery. Since we only have a few more weeks to work on it, and a lot left to learn, the process will be simplified. However, I do hope to touch on all aspects of developing a program so you will know what all of those other people who are working at your company do when you start work and so you will be know what to do if you decide to start your own company.

The Requirements

But this week we will start the library. Programs from an idea. Here the idea is: wouldn't it be nice to have a program that could keep track of all of the things we lend out. It is a simle idea and it needs a lot more work to be fully fleshed out. The first step is to determin what the program needs to do. This is called *requirements analysis*.

Requirements may be gather at the beginning of the project, the traditional approach, or they may be added to as the project progresses, the agile approach. Even in the agile approach, we still collect as many requirements as we can; we just acknolwedge that we may not know all of them at the beginning.

Requirements are always specified from the user's point of view: it is what the program does in response to the user. When we think of requirements we think of what the user does and how the system responds. At this stage, we do not think about what the system does to prepare the responses, only what that response is. Here is a brief synopsis of what the program does:

> MyLender is a program that keeps track of items I lend out to my friends. It keeps a list of friends I lend things to and the things I lend to them. It keeps contact information about the friends I lend things to and descriptive information about the things I lend them. It also keeps a list of the lendings tracking when I lent the items, who I lent it to, when I lent it, and when it was returned. I am able to pull up a list of items lent to a particular friend, a list of all items lent and the person to whom it was lent, and a list of friends. I can also call up an item and find whether it is lent and, if so, to whom. I am able to add friends to my list of friends and to my list of items to lend. I can add a record of lending an item to a friend.

From the synopsis, we can develop *scenarios*. A scenario is a sequence of actions performed by an *actor*. An actor is a person or system that is involved in a scenario. A scenario is written as a numbered list of sentences. The subject of the sentence is an actor. The sentences are always written in active voice. Here we have seven scenarios: add an item, add a user, lend an item, retrieve an item, list items lent, list items lent to friend, and find item. There are two actors: the user and the system. The user initiates actions and the system responds.

Scenario 1: Add an item

1. User starts the program
2. System presents a list of actions including “add an item,” “add a user,” “lend an item,” “retrieve an item,” “list items lent,” “list items lent to friend,” and “find item.”
3. User selects “add an item”
4. System requests an item description
5. User fills in requested information and indicates he or she is finished
6. System stores the requested information as a new user

Scenario 2: Add a friend

1. User starts the program
2. System presents a list of of actions including “add an item,” “add a user,” “lend an item,” “retrieve an item,” “list items lent,” “list items lent to friend,” and “find item.”
3. User selects “add an friend”
4. System requests information including name, address, phone number, and email
5. User fills in requested information and indicates he or she is finished
6. System stores the requested information as a new item

Scenario 3: Lend an item

1. User starts the program
2. System presents a list actions of including “add an item,” “add a user,” “lend an item,” “retrieve an item,” “list items lent,” “list items lent to friend,” and “find item.”
3. User selects “lend an item”
4. System presents a list of items.
5. User selects an item from a list of items
6. System presents a list of friends.
7. User selects a friend from a list of friends
8. User indicates he or she is finished
9. System stores a record of the loan.

Scenario 4: Retrieve an item

1. User starts the program
2. System presents a list of actions including “add an item,” “add a user,” “lend an item,” “retrieve an item,” “list items lent,” “list items lent to friend,” and “find item.”
3. User selects “retrieve an item”
4. System presents a list of loans.
5. User selects an loan from a list of loans
6. User indicates he or she is finished
7. System removes the record of the loan from the store.

Scenario 5: List items lent

1. User starts the program
2. System presents a list of actions including “add an item,” “add a user,” “lend an item,” “retrieve an item,” “list items lent,” “list items lent to friend,” and “find item.”
3. System displays a list of all items lent with the name of the user to whom they have been lent

Scenario 6: List items lent to friend

1. User starts the program
2. System presents a list of actions including “add an item,” “add a user,” “lend an item,” “retrieve an item,” “list items lent,” “list items lent to friend,” and “find item.”
3. User selects “list items lent to friend”
4. System presents a list of friends.
5. User selects a friend from a list of friends.
6. System presents a list of items lent to that friend.

Scenario 7: List items lent to friend

1. User starts the program
2. System presents a list of actions including “add an item,” “add a user,” “lend an item,” “retrieve an item,” “list items lent,” “list items lent to friend,” and “find item.”
3. User selects “find item”
4. System presents a list of items.
5. User selects an item from the list of items.
6. System presents information on that item including the borrower if it has been lent.

System stores a record of the loan.The Design

Once we have enough requirement to satisfy the initial synopsys, we can begin designing the program. Program design involves thinking about the pieces of the program we will need to write and arranging them in such a way that we can attack the problem rationally. Like requirements analysis, design may be done at the beginning of the project or as it is needed. If one chooses to do agile requirements analysis, one must do agile design because as the requirements change the design must change to reflect the changes.

To create the design, we first create a class diagram from the scenarios. We do this by looking for nouns in the use cases. These are our classes. Next we look for verbs. These suggest methods. Data is

suggested by objects.

Program, friends, items, and loans are the three nouns that are most important from the use cases. We will create classes from them. In addition, we have lists of friends, lists of items and list of records of loans. We could create classes for the lists, but instead we will encode them as ArrayLists in the program. So, the classes we will create are MyLibrary for the program, Friend for a friend, Item for an item, and Loan for a loan. MyLibrary will contain a list of Friends, a list of Items, and list of Loans. We can create a class diagram for this design.

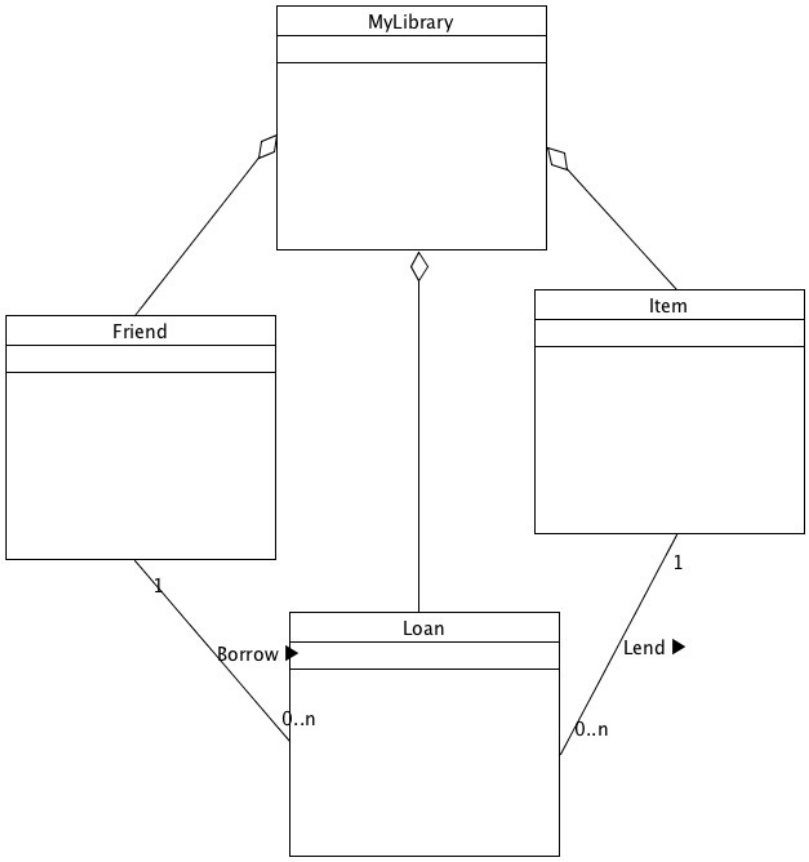

Figure 1: MyLibrary Initial Class Diagram

Figure 1 shows an initial class diagram. It shows the four classes and their relationship with each other. MyLibrary is a composition of Friends, Loans, and Items. Each Friend may have many loans, but each loan is to a single Friend. Similarly, each item may participage in many loans, but each loan involves only a single item. From this, we can elaborate the design by adding fields and methods.

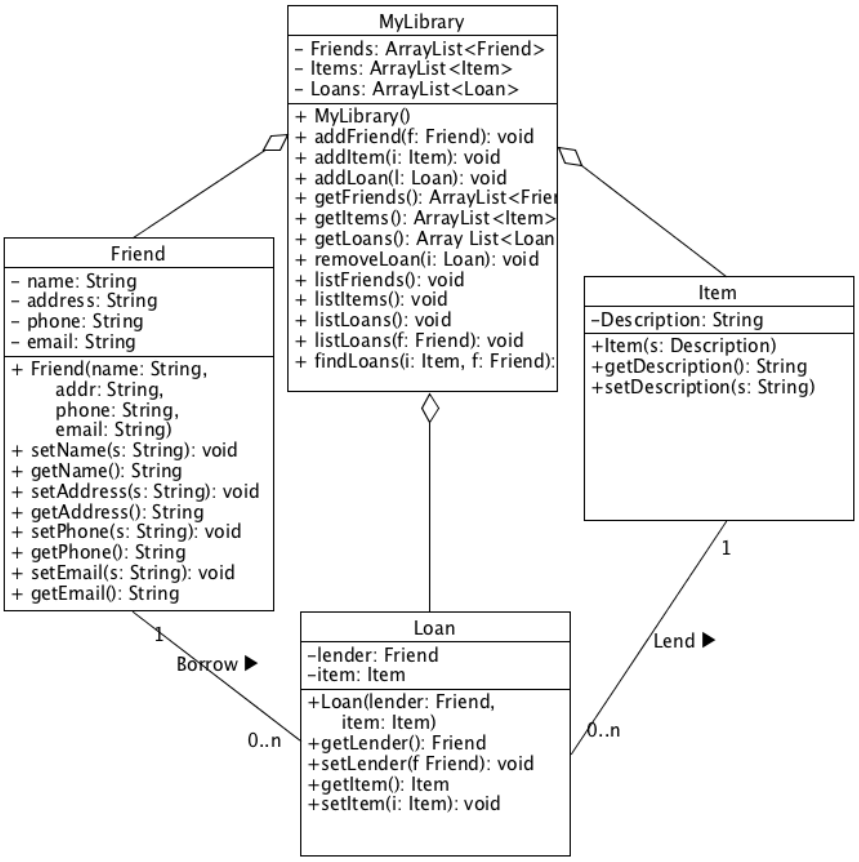

Figure 2: Elaborated MyLibrary Class Diagram

When we understand the relationship between the classes we can fill in the fields and methods. MyLibrary will have three fields: a list for friends, a list for items, and a list for loans. The constructor will just make a MyLibrary class. There are methods for adding items to the lists, and a method for removing an item from a loan list. There are also methods for showing the contents of the list. Finally, there is a method for showing a list of loans given to a particular friend, and for finding the loan of a particular item.

The other classes: Friend, Item, and Loan each have information about the friend, item or loan and getter and setter methods.

Development

This is sufficient design to get started. The methods so far are simple enough to do without analyzine their behavior in depth. They just add elements to the class and return those elements. One removes an element from a list. From this, we can build a backlog (libraryBacklog.pdf) which is available from the main page for this week. Work using test driven development. Write the test first, then use Eclipse to guide you through the writing of the code. It will be much quicker this way, and you will save a lot of typoes.

Next Weeks Demo Marks

1. Story 1 (1 Mark)
2. Strory 2 (2 Marks)
3. Story 3 (2 Marks)
4. Story 4 (3 Marks)
5. Story 5 (4 Marks)
6. Story 6 (3 Marks)
7. Story 7 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
8. Story 8 (3 Marks)
9. Story 9 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
10. Story 10 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
11. Story 11 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
12. Story 12 (4 Marks) The tests must actually test the methods; not just succeed.
    1. Tests compile (2 marks)
    2. Tests run successfully (2 marks)
13. Story 13 (2 Marks)