Object-Oriented Design and Implementation
Assignment 2

Important *This is a group assignment, but you may work alone, if you wish. If you are working in a group, you must let me know by March 21 the list of your group members. You cannot change the group afterwards.*

Due Date 11:59 PM on April 4, 2016

Goals

- To learn to apply design patterns
- To refactor a design and implmentation as modifications come in

Problem Modify the library implementation in the following ways.

Compute Fines Consider the situation where the library decides to cut down on truancy by imposing fines. When an overdue book is returned, the librarian would like to know the amount of fine and send out a notice to the user regarding the fine payable. The system should therefore compute the fines and display the relevant information. The resulting changes in the business process are captured in the use case in the following table.

Actions performed by the actor	Responses from the system
1. The member arrives at the return counter with a set of books and gives the clerk the books.	
2. The clerk issues a request to return books.	
	3. The system asks for the identifier of the book.
4. The clerk enters the book identifier.	
	5. If the identifier is valid, the system marks that the book has been returned and informs the clerk if there is a hold placed on the book; otherwise (that is, in case of an invalid id), it notifies the clerk that the identifier is not valid. If there is a fine involved, the system computes the amount of fine using *Rule for Fines* and adds it to the user's account and information about the member is displayed. It then asks if the clerk wants to process the return of another book.
6. If there is a hold on the book, the clerk sets it aside. He/she then informs the system if there are more books to be returned.	
	7. If the answer is in the affirmative, the system goes to Step 3. Otherwise, it exits.

Use-case Book Return with Fines

This use case for Book Return with Fines is similar to what we had earlier, with one addition - the amount of fine owed is computed whenever a book is returned. Obviously, the `Member` class needs to be changed to track the amount of fine owed.

We have the following formula for computing the fine:

Rule for Fines: New books (less than 90 days old) are charged 25 bani for the first day and 10 bani for every subsequent day. Older books are charged 15 bani for the first day and 5 bani for every subsequent day. If a book has a hold on it, the amount of fine is doubled.

Other Loanable Items Consider a more sophisticated version of the `Library` system that we created in the implementation posted. With the advent of technology, our clients now wish to expand their collection to include non-print media. Thus we now have laptops, cameras, CDs (music, books), and DVDs in addition to printed books. Also, the library wants to include some periodicals, which are to be handled differently from the other books in that, recent periodicals cannot be checked out. For CDs and DVDs, we wish to keep track of the duration; they are checked out just like books. Use the factory pattern to encapsulate the possibly varying functionality.

Pay Fine Create functionality that allows a user to pay all or some of his/her dues.

Factory Apply the Factory pattern in a simple way to isolate the creation of the various loanable items.

New Books and Costly Books All books that are less than 90 days old are considered "new" books. They cannot be renewed. Some books are considered "costly" books: the user determines whether a book is "costly" at the time ofadding the book. Thus all books are "new" at the time of adding them to the catalog and some of them could be "costly." A "costly" book cannot be issued to a member who has unpaid fines. A "costly" book goes out of that state after one year, but then the library system has a command (which you should implement) to make the book "costly" again. Use the decorator pattern for controlling the behavior.

Preparing the Library System for New Functionality Implement the visitor pattern for the hierarchy of loanable items. Then use it to implement a *Visitor* for printing the number of items checked out (at the moment the command is issued) for each category: books, CDs, music CDs, DVDs, laptops, and cameras.

Lay out your code properly.

Add an extra command for testing purposes. The numeric value of the command should be 99. It should allow the user to add fine (positive or negative) to a member. For example,

```
99
Enter member id
M1
Enter fine
3.0
```

adds 3,0 lei to the fine.

Program Submission

Draw sequence diagrams for each of the new functionalities. All of the sequence diagrams should be in one PDF document. Zip the source files and the PDF (with all the sequence diagrams) and submit to the dropbox. Gross violations such as missing some files or not submitting a zip file will incur penalties. If the program has syntax errors, the grade will be 0: no exceptions.

Program Grading Criteria

Sequence diagrams: 40

Correctness of the Java code: 60 points; Approximate division of points follows.

- Hierarchy: 30
- Fines: 10
- New books and costly books (decorator): 10
- Visitor: 10

Program structure: 40 points

- Proper access for methods and fields; logic: 10
- Proper organization of methods: 10
- Proper organization of classes and interfaces: 20

Coding standards: 10 points

- Proper indentation and line breaks, etc.: 6
- Properly named identifiers: 4

Coding Standards It is important that you properly format your program. There are several reasons why it is important to follow coding conventions. There is the high cost of software maintenance. It is rare that the original author would maintain a piece of software for its entire life. It is quite difficult for most people to remember their code if they haven't seen it for a long period of time. Programmers should create code that they are proud of.

In your programs, you must obey the following subset of conventions. As we write more complicated programs, I may ask you to follow more requirements.

- Do not type long lines. Try to limit them to 80 characters
- If an expression does not fit on a single line, break it according to these general principles:
  - Break after a comma or before other operators.
  - Prefer higher-level breaks to lower-level breaks: For example, if the following expression

    ```
    (a + b * (c + d)) * (e + f)
    ```

    runs into multiple lines, you should try to break it as

    ```
    (a + b * (c + d))
    * (e + f)
    ```

    as opposed to

    ```
    (a + b *
    (c + d)) * (e + f)
    ```

  - Attempt to align the new line with the beginning of the expression at the same level on the previous line.
  - If the application of the above rules results in ugly-looking code, just indent with enough tabs that make the code look reasonable.
- Variable Names: Variable names must begin with a lower-case letter and be in full words. Words after the one must have their first letter capitalized.

  Here are examples of acceptable variable names:

  ```
  count costOfItem gallonsPumped testSucceeded
  ```

  Here are examples of unacceptable variable names:

  ```
  Count itmCst gallonspumped tstSucceeded
  ```

- Indentation and Alignment: If you are using Eclipse, use `Window->Preferences, Java, Editor, Save Actions` and check the `Format source code` option. Eclipse will format your source code when you save the file.

  - Indent code within a class by one tab.
  - Indent code within a method by one tab.
  - Align code as below.

```
int thatValue;
int thisValue = 10;
int count;
thatValue = 0;
count = Math.min(thatValue, thisValue);
```

- There are more requirements: like putting a space before and after a binary operator other than comma, separating tokens properly with a space, writing version number, your name, etc. at the beginning, writing information about methods, etc. As we write more involved programs, we will learn more and more standards.

Refer to the code below for an example of a program with acceptable formatting.

```
import java.util.LinkedList;
public class LinkedSet implements Cloneable, Set {
    LinkedList list = new LinkedList();
    public boolean insert(int value) {
        if (isMember(value)) {
           return false;
        }
        return list.add(new Integer(value));
    }

    public boolean remove(int value) {
        return list.remove(new Integer(value));
    }

    public Object clone() {
       LinkedSet copy;
       try {
          copy = (LinkedSet) super.clone();
       } catch (CloneNotSupportedException cnse) {
         cnse.printStackTrace();
         return null;
       }
       copy.list = new LinkedList();
       for (int i = 0; i < list.size(); i++) {
           copy.list.add(new Integer(((Integer) list.get(i)).intValue()));
       }
       return copy;
    }
}
```