**CSE 1325 – Summer 2013**

**Object-Oriented and Event-driven Programming in Java**

**Lab 3**

Design Document Due Date: **Thursday, July 18, 2013 at 11:59pm**

Lab Due Date: **Thursday, July 25, 2013 at 11:59pm**

(see instructions on website for how to turn this in - “lab submission info”)

Grade value: 11% out of 100% for all grades

**Objective:**

The theme of this project is to continue to design and update classes and methods for managing items (books and media) in the library (textbook, magazine, journal, newspaper, novels, DVD, audioCD, other) using object-oriented design (OOD) principles. Updating existing code is a very common task in the software industry. Feel free to re-code your classes. You will have the freedom to implement the internal functionality of your objects but the class name and signature of each method will be defined in a standard Javadoc.

Please make sure you adhere to OO principles throughout the design and implementation. The projects are designed to give you an opportunity to exercise the principles taught in the course. Remember, Java is only a vehicle for supporting OOD. Make sure you fully leverage the features available in Java for your benefit as long as we’ve touched on them in class. Ask before jumping into totally new concepts for your lab.

In this project, you will be designing the data abstractions and required operations on those abstractions for managing items in the library.

Be sure to check the DEDUCTIONS section at the end of this assignment to avoid penalties

You are also required to design your program in advance before you begin writing code. You will document your design and turn in the design document at least a week before the lab assignment is due. The goal of the design document is to assist you in developing the actual program.

**Description:**

Now that you have experienced structuring your own program and designing your own tests, you will now you adhere to these testing specifications as you would in the real world by learning to refactor your code (by changing method names, parameters, etc) to meet the requirements. Specifically, the code that will be most modified will be:

* Method names
* Method parameters
* Specific, testable return types such as Boolean
* Failing conditions

Design classes, fields (attributes) and methods for the problem description below. Make sure you understand and differentiate between public/private/protected fields and methods. Use static fields/methods where appropriate. Make sure you have at least one constructor per class. Each class should have appropriate get (accessor) and set (mutator) methods. Each class should have a toString() method (and other formatting methods as needed).

Files accompanying this project:

• Javadocs describing the minimum methods that need to be implemented

• info.txt a sample file

• Constants.java

• Java files for enumerated types and most of the LibraryDate class

• This document

(Note: that was a Java joke)

**LibraryTest**

As in the previous lab, a “test” class will drive the application and will now be named, ‘LibraryTest’ (if it wasn’t already named that). The test class should open and process the info.txt, print the menu seen below, accept user input, and execute methods to perform tasks, and resulting understandable output. All inputs (as appropriate for the menu) should be accepted from the terminal window and should prompt the user for the input with appropriate messages. If bounds are appropriate, please indicate it in the prompt and check for input errors. Expect the user to enter the proper data type. As seen in the class descriptions, nearly every setter method must return a boolean type, and the LibraryTest must act on it with the possibility that the method might fail; even if it only displays a message such as, ‘Unable to Check Out Desired Item"). In other words don’t let methods fail silently. Test your source code—be sure every method works correctly.

The tasks for your library are now separated into subcategories of Item tasks, patron tasks, and employee tasks as follows. If desired, you may build a two-level menu that asks the user if they wish to handle item, patron, or employee tasks and then based on that input, only give them the tasks for that subcategory. You may also keep all tasks at the top level if desired. You may use any numbering or lettering scheme you prefer for the tasks below. For all classes, the methods listed below are IN ADDITION to the normal constructor, getter, and setter functions that are required.

Item tasks:

1) Add a new item to the library

2) Remove an item from the library

3) Check out an item (input ISBN, the due date, the patron ID)

4) Check in an item (input ISBN, the patron ID) <should return a fee amount if the item is overdue>

5) Display all item details (along with their specialization)

6) Display all available item details

7) Display all checked-out items (including due date and how many days are left before the due date or how many days the item is overdue)

8) Display all items in a given specialization

9) Search for ISBN (input title, publisher, and publication year)

Patron tasks:

20) Find a patron ID from the first and last name

21) Find the first and last name from a patron ID

22) Display all items checked out by a specific patron (input Patron ID)

Employee tasks:

30) Clock in an employee

31) Clock out an employee

32) Hire an employee

33) Pay an employee

34) Remove (fire) an employee

35) Display all employee details (along with their specialization)

0) Exit program

The “Test” class has a “Library” class to manage items.

I. The Library class

Has data elements:

A. libraryContents - Use ArrayList and Item class

B. patronList – Use any collection class (such as ArrayList) and the Patron class

C. employeeList – Use any collection class (such as ArrayList) and the Employee class

Has methods:

1) Add a new item to the library

2) Remove an item from the library

3) Check out an item (input ISBN, the due date, the patron ID)

4) Check in an item (input ISBN, the patron ID) <should return a fee amount if the item is overdue>

5) Display one item with all item details (along with their specialization)

6) Display all available item details

7) Display all checked-out items (including due date and how many days are left before the due date or how many days the item is overdue)

8) Display all items in a given specialization (enumeration type)

9) Search for ISBN (input title, publisher, and publication year)

20) Find a patron ID from the first and last name

21) Find the first and last name from a patron ID

22) Display all items checked out by a specific patron (input Patron ID)

30) Clock in an employee

31) Clock out an employee

32) Hire an employee

33) Calculate pay for an employee <hourly employees are rate times hours worked; salary get monthly amt>

34) Remove (fire) an employee

35) Display all employee details (along with their specialization)

II. The Item class (this was the Book class in the previous lab)

Has data elements: EXAMPLES

A. ISBN (10 digits): 1001234567

B. title (string): "Java Programming Language"

C. type (enum): textbook, magazine, journal,

newspaper, novel, DVD, AudioCD,

and other

D. publisher (string): "UT Arlington Press"

E. page information (how many pages) (integer): 506

F. price (float): 20.00

G. publication year (integer): 2013

H. checked out status (boolean): false=available, true=checked-out

I. Patron ID who checked out the item : 295603

J. due date (use LibraryDate class)

Has methods:

1. Calculate how many days are left before the due date (or how many days overdue)

III. The LibraryDate class (Don’t use Calendar or Date class that Java provides)

Has data elements:

A. day (integer): 1

B. month (integer): 1

C. year (integer): 2013

Has methods:

1. Default constructor with no inputs that gets the real current date from the OS by using one of the built in date types (e.g., Date, Calendar, GregorianCalendar, …) The Java built-in type should ONLY be used to get the current date to put in the LibraryDate object.

2. Constructor that accepts an input day, month, and year, error checks these values and stores them in a new LibraryDate object. For Lab #3, dates must be REAL dates. You may not use the "30-day" simplification.

3. Constructor that accepts an input day and month only, assumes the current year, error checks these values and stores them in a new LibraryDate object. For Lab #3, dates must be REAL dates. You may not use the "30-day" simplification.

4. Calculate the date difference (implementing “-“ function) between two dates.

IV. Person

Each person has the following data elements /attributes EXAMPLE

A. First name (string) Jay

B. Last name (string) Smith

C. Address <an Address object – see below>

D. Gender (enumerated type) male / female / no data

V. Employee – inherits from class Person

Each employee has the following data elements EXAMPLE

A. Identification (EmployeeID) number 006 <a value between 1 and 999>

B. Type of employee salary/hourly/contract

C. Pay rate (monthly salary or hourly rate) $2500.00

D. Area of specialization (use enum) reference, special collections, research,

desk, media, general

E. Currently working? (Clocked in) (boolean) true/false

F. Hours worked in current month (integer) 42

Has method:

1) Calculate pay for current month <hourly employees are rate times hours worked; salary get monthly amt; contract per job>

VI. Patron – inherits from class Person

Each patron has the following attributes EXAMPLE

A. Identification (PatronID) number 122002 <a value between 100000 and 999999>

B. The last date any item was checked out <a LibraryDate object>

VII. Address

Each address object has the following attributes EXAMPLE

A. Street address (string) "123 Sesame St."

B. City (string) "Baltiyork"

C. State "TN" <two letter postal code>

D. ZipCode 43564 <five digit zip code greater than 10000>

Java provides mechanisms for comments and input for Javadoc. Please make sure your code contains both. Also, submit the files generated by the Javadoc for your program. Look up the convention for specifying parameters (@param) and user information for javadoc. Every method should have its meaningful description for use by someone else. Use the relevant Javadoc tags specified in -http://en.wikipedia.org/wiki/Javadoc or <http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#javadoctags> . In NetBeans you can work with JavaDoc comments in a couple of ways as described in the NetBeans quicktour <https://edu.netbeans.org/quicktour/javadoc.html> .

**Standard Java conventions and coding style:**

Be sure to observe standard Java naming conventions and style. These will be observed across all projects for this course, hence it is necessary that you understand and follow them correctly. They include:

* Class names begin with an upper-case letter, as do any subsequent words in the class name.
* Method names begin with a lower-case letter, and any subsequent words in the method name begin with an upper-case letter. This is called "camelCase".
* Class, instance and local variables begin with a lower-case letter, and any subsequent words in the name of that variable begin with an upper-case letter.
* Each set of opening/closing curly braces must be in the same column.
* All user prompts must be clear and understandable
* Give meaningful names for classes, methods, and variables even if they seem to be long. The point is that the names should be easy to understand for a person looking at your code
* All static final variables will be upper case identifiers (i.e.constants)

In addition, ensure that your code is properly documented in terms of comments and other forms of documentation wherever necessary.

**What and How to Submit**

For Lab #3 WRITE A DESIGN DOCUMENT FIRST. The design must include

a) all the classes you expect to write,

b) brief (one line) descriptions of each data member of the class

c) brief (one line) descriptions of each method (member function) of the class (including describing any

error checking to be done), and

d) some indication of the relationships between the classes.

The design document may be written as lines of text or as diagrams (such as a diagram that start with the main function at the top and all others below it, a tree, UML, etc.) or as some combination of those, but it must include the information listed for a), b), c), and d) above. Each class should encapsulate one thing and each piece of data at a lower level should have a more specific purpose than the class that owns it. Be sure to include all the classes that are described in this lab assignment. See the website or this assignment for the DESIGN DOCUMENT due date. It is usually ONE WEEK PRIOR to the lab due date.

A Lab #3 Design Document must be turned in in order for your Lab #3 assignment to be graded.

For this lab project itself, please submit using the Blackboard system at elearn.uta.edu. Please submit a .zip file that contains:

* Each .java file used in development with the appropriate name for the included class
* A document listing each test case that is in your test class and what the purpose is of each test. This is your test plan.
* A script file of a test session on omega
* A development notes file (standard text or word file) that describes at least three logical errors that you found while developing your code, how you solved it, and the total amount of time you spent on this project in terms of hours.
* The JavaDocs created from your code

The .zip file containing your project must have the name format of: **proj3\_100number\_firstname\_lastname**

Upload this code to Blackboard at elearn.uta.edu

**A Few Helpful Tips:**

* Netbeans by default recompiles your class every time you save your file. If you want to force everything to be recompiled, press the ‘clean and build’ button.
* Ctrl + space in most IDE’s will show a list of available members (or variables) and methods (or functions).

**Grading Rubric:**

**Code: (43%)**

Class header and method headers for all classes as JavaDoc (5 points)

Comments (line comments and block comments) (3 points)

Style (following Java conventions, indentation, consistency, meaningful identifiers, lateral separation of code from line comments, etc.) (3 points)

Modularity (division of the problem into small parts) (3 points)

Correct REVISED declaration of the required classes with data and methods and any other needed classes

Library (4 points)

Item (4 points)

LibraryDate (3 points)

Employee (3 points)

Patron (2 points)

Person (3 points)

Address (2 points)

Test plan includes at least two tests for each of the required

functions (8 points)

**Output: (57%)**

User clearly understands what is being requested for input (3 points)

Required tasks performs correctly

Item tasks (11 points)

Patron tasks (5 points)

Employee tasks (10 points)

Set functions handle inputs correctly including error checking (6 points)

Get functions provide values safely (4 points)

Output gives clear information to explain the values to the user (3 points)

Output contains all the given test data (if any) and at least two additional data sets, i.e at least one other library, two add’l items, and two add’l dates up to the maximum of 5 (10 points)

Output demonstrates all elements of test plan (5 points)

**Grading Deductions:**

Labs which have errors in them and do not terminate normally will receive an overall grade of 0 (zero) [-100 deduction]

Late submission of softcopy of design document, code and/or script file or other documents to Blackboard will result in an overall grade of 0 (zero) UNLESS student has obtained prior instructor approval [-100 deduction]

Use of Java language elements not yet discussed in class by the lab due date will result in potential deduction of points - discuss with instructor before using.

Use of Java public variables (C equivalent to global variables) will result in an overall grade of 0 (zero) [-100 deduction]

Use of unstructured control elements, like the break (except in the case of switch statements) or continue keywords, will result in an overall grade of 0 (zero) [-100 deduction]

See lab re-grading policy to understand how to gain back some points on a lab.

Miscellaneous: If you have questions, e-mail Dr. T ([tiernan@uta.edu](mailto:tiernan@uta.edu)) and the TA ([mingon.kang@mavs.uta.edu](mailto:seanp@uta.edu) )