**CSE 1325 – Summer 2013**

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

**Lab 4 (final lab)**

Design Document Due Date: **Tuesday, August 6, 2013 at 11:59pm**

Lab Due Date: **Tuesday, August 13, 2013 at 11:59pm**

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

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

**Objective:**

The theme of this project is to continue designing and updating classes and methods for managing a library using object-oriented design (OOD) principles. When it comes to an application’s usability, Graphical User Interfaces (GUI’s) offer obvious benefits and drastically improve the user experience (UX). When programming GUIs, code can quickly become large and complex but in this lab we will limit the use of the GUI windows to certain parts of the program. We'll also make a few other changes to test out exception handling and, for a bonus, to try out the automated JUnit testing system. You will have the freedom to implement the internal functionality of your objects but the class name and signature of each method should match those defined in the accompanying 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.

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:**

This lab will obviously have you experimenting with building a GUI but it will also enforce handling errors gracefully with exceptions. In addition you can choose to try out testing your code with an automated process.

Specifically, this lab differs from that last in these major ways:

* Add exception handling to user input for numeric data and for data with specified values (like state abbreviations).
* The Person class will implement the Comparable interface on the last name
* Make simple GUI windows for some aspects of Item information as described below
* Bonus: Automate object testing via JUnit testing

Reminders: Remember that in good object oriented design all your input and output should be in one class. 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 ( no changes from Lab 3)
* *This* document   
  (Note: that was a <very small> 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 menu inputs should be accepted from the 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. Use try-catch (exception handling) to ensure that the user enters the correct 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.

Unless explicitly stated otherwise, assume that all required data and methods from previous labs are still required.

**The Graphic User Interfaces**

The GUI(s) design and layout are completely within your control.. The minimum amount of functionality is that a window must be displayed with all the relevant and related information per menu option. For example, menu options 5 through 8 must result in a window display of the appropriate data to the user. If you create a window of more complexity than a JOptionPane it is suggested that you create your own classes that extend a JFrame.

**The Classes**

I. The Library Class

As stated earlier you may implement more methods than those listed in the accompanying JavaDoc. However those found in the Javadoc are the minimum and must function properly in order to receive full credit.

Note: The typical structure of a main function would be a do while loop containing the menu and switch statement. It is suggested that in each of these cases you place a try/catch blocks to catch any exceptions thrown by the user entering bad input.

II. The Item Class

There should be no limit to the number of Items in the Library. Add the following data elements to the Item class:

1. author (should be a new class that is a subclass of Person)
2. keywords (a group of strings separated by commas)

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

1. Modify the method to calculate the date difference (implementing “-“ function) between two dates. Make sure this can be any two dates, not just the current date and an input date. This will require some modification of the comparison function.

IV. Person

1. Modify the Person class to be an abstract class (if it is not already)
2. Implement the Comparable interface on the last name

V. Employee – inherits from class Person

VI. Patron – inherits from class Person

Add the following data elements to the Item class:

1. Add the data that the Patron first joined the library
2. Add a record of donations that the Patron has made in the last year to the library.

1. Create an output display window to display all Patron information and use this to show Patrons

VII. Address

**The Constants Interface**

The Constants.java file is given with this assignment. Use the keyword, ‘implement’ to use it in your classes. Put any new constants in this file such as WINDOW\_HEIGHT.

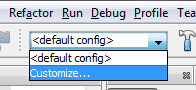
**Bonus: The JUnit tests**

This would be the code version of a very detailed test document. You will write a JUnit 3.x or 4.x test class for each of the classes you will have in this lab. You will test every method of sufficient complexity (no need to test all the getter methods or GUI’s). For a basic intro to JUnit testing in netbeans see <https://netbeans.org/kb/docs/java/junit-intro.html>

**The sample info.txt file.**

The info.txt file is also given as a template to test your file input method. The best way to effectively test your code is to make your own info.txt file; in this lab you **must** make your own test file and submit it with your lab. Each object will be preceded by its name in the text file. For each instance of an ‘Item’, ‘Patron’, or ‘Employee’, the strings ‘Item’, ‘Patron’, or ‘Employee’ will precede the correct amount of information respectively. All lines starting with ‘//’ should be ignored.

Note: When reading in a file you have to change the ‘Working Directory’ to the location of your info.txt file. This configures Netbeans to execute your code at that location in the filesystem. This allows your java program to find your data file with a relative file and path name like, ‘info.txt’ as opposed to an absolute path name such as, ‘C:\Users\etc\...\info.txt’.



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> .

If you are having difficulty generating Javadocs (Run -> Generate Javadoc), then you might need to create a new package then move your classes into that package.

If you are missing classes or methods in your Javadocs make sure the keyword public is being used.

**What and How to Submit**

For Lab #4 WRITE A DESIGN DOCUMENT FIRST. The design must include

a) all the classes you expect to write, (Bonus points if you use UML to describe your design)

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 #4 Design Document that is different from your previous design documents must be turned in in order for your Lab #4 assignment to be graded.

**Testing Plan**

For Lab #4 you must also write and turn in a TESTING PLAN along with your program and test data. The testing plan should include at least the required sets of test data values (at least 3 items, 2 employees, and 2 patrons) that will test at least the following elements of your code:

Two tests of the add an item function

Two tests of the remove an item function

Two tests of the check in an item function

Two tests of the check out function

Two tests of the due date checking function

Demonstration of the GUI to display each kind of result data

For each “test” above, describe what data values you will use for this test (from the data in your test file of input) and indicate what the correct result of the test should be and how to demonstrate that this result has occurred, i.e. will the user have to ask for the data to be printed out or will that happen automatically?

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
* Your testing plan as described above
* The test file of input data that you have created and used to test your program. This cannot be the given info.txt file, it cannot include the same data as the info.txt file given, and it must include data for at least 3 items, 2 employees, and 2 patrons.
* 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 them,
  + the total amount of time you spent on this project in terms of hours, and
  + the parts of your program that are not functioning
* The JavaDocs created from your code
* You are not required to run this project on omega – only on Netbeans

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

**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.

**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: (37%)**

Documentation and coding style including: (6 points total)

Class header and method headers for all classes

Good Javadoc content

Comments (line comments and block comments)

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

Modularity (division of the problem into small parts)

Correct changes/ additions to the required classes (5 points)

Correct implementation of file input and test file (6 points)

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

functions of add, remove, check in, and check out and uses the

display functions to verify test results (6 points)

All input/output is in the Test class and GUI’s (6 points)

Other code requirements (8 points)

**Output: (63%)**

Required tasks performs correctly and all specified modifications are made to data and methods in required classes (16 points)

Output demonstrates all elements of test plan including exception handling (13 points)

GUI’s (13 points)

Output quality (8 points)

Set functions handle inputs correctly including error checking and get functions provide values safely (5 points)

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

User clearly understands what is being requested for input and output gives clear information to explain the values to the user (3 points)

**Bonus: ()**

JUnit testing demonstrated and documented (10 points)

UML used for documenting design (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 any required file or other document 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) )