**CSE 1325 – Summer 2014**

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

**Lab 3**

Design Document Due Date: **Tuesday, July 29, 2014 at 11:59pm**

 **(one minute before midnight 12am)**

Lab Due Date: **Tuesday, August 5, 2014 at 11:59pm**

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

Grade value: 14% out of 100% for all grades

Modified on Aug 2 – removed references to Contract object in Lab #3 and put all into Lab #4

**Objective:**

The theme of this project is to design classes and methods for managing a performance hall using object-oriented design (OOD) principles. This project builds on the previous labs as will the rest of the projects in the semester. 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 a performance hall.

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:**

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

Use a “test” class to run the application (to test/call each method defined in classes). The test class should ask for and accept user input, print the following menu, and execute the menu to provide understandable output. All inputs (as appropriate for the menu) should be accepted from the user input (input file or keyboard input or window input) as specified and should prompt for the input with appropriate messages. If bounds are appropriate, please indicate it in the prompt and check for input errors. Test your source code—be sure every method works correctly.

The test class main routine should have menu choices for all of the following. For Lab #3 you will see additional requirements below. This means that each element (choice) below must be executable (in a complete implementation). Each choice should be run with multiple sets of input data. Each set of input data for one menu choice is considered a test case for your classes. See further requirements for test plan document which includes test data that you have used. Menu choices for the overall performance hall:

1) Contract for one or more performances by a group ("Group" can be one person performing or multiple people, i.e. a band, a symphony, a theater troupe, a singer, a comedian, a play, etc.) [i.e. "Add" a performance to the list of contracted performances and, if needed, add the performance group to the group list]

2) Buy advertising for a performance group [ The hall buys the ads]

3) Host the performance (including calculating expenses; attendance ticket count) [i.e. "List" all the information for a completed performance]

4) Contract for catering for performers [The hall contracts for catering]

5) Sell tickets/seats (One ticket = one seat) for one given performance – make sure to credit the correct account with the ticket sale amount and the ticket count for the performance.

6) Check current account balances for the hall and for any other entity with accounts. Print balances for selected entity and indicate if account is at zero or is overdrawn (negative).

7) List all performances under contract (including dates, times, and days; performance group; number of seats; minimum seats guaranteed; min. advertising budget)

 a) list by date

 b) list alphabetically by name of performance group

 c) list by number of seats guaranteed

8) List all performing groups under contract (including contracting agent; performance group; number of performances contracted; catering requirements)

9) Find a specific contracted performance and print its information in a GUI window for a potential ticket purchaser.

 a) by performer

 b) by date

 c) by ticket price

10) Print all data about a specified performance group in a GUI window

11) List all contracted companies that are not performing groups.

12) List all contracted events on a specific date.

13) Define the "current date" for purposes of the system so that past/completed performances and future performances can be determined.

14) Find a specific contracted performance and print its information in a GUI window for the performance hall manager. Include name of performance group(s), date and time, ticket price(s), number of tickets sold, number of seats/tickets still available, amount spent on advertising, amount spent on catering, amount guaranteed to performance group, and any other info you feel pertinent.

15) Cancel a contracted performance [Remove it from the list]

16) Pay a group/company or a person

17) List budgeted amounts and actual account balances for the performance hall

18) End the program

[NOTE: at this point, you should be thinking about what classes will be needed to implement this performance hall model. There is no one right way to do this. However, you are required to meet the following criteria:]

I.  Interface for CalculatePay as

double CalculatePay(double Amt1, double Amt2);

A. The performance hall class should contain the following data and methods:

Data members should include all items from Lab #2 with the following changes and additions:

2. Has a current performance with information about the performance group. This is the performance happening on the current date.

8. Has a list of contracts with companies that are not performance groups for the next 12 months

9. Has a hall manager (in a Person Object – see Person class below)

10. Has a list of hall staff members (ushers, ticket sellers, show crew, other) (Persons)

11. Has a hall contracting agent (Person)

12. Has a list of donors (in Donor Objects – see Donor class below)

Methods: (in addition to the standard methods that should be included)

Methods should include all methods from Lab #2 with the following changes and additions:

16. Pays contracted groups (companies) at the end of each week, adjusting the operating budget as necessary

18. Keeps track of monetary donations

19. Adds new donors

20. Pays all hall employees weekly – manager, staff members, and agent – implementing the interface CalculatePay

The method for CalculatePay for a performance hall staff member is to pass in the pay scale per hour and the hours worked for one week. The theatre staff member's weekly pay is :

pay per hour \* hours per week

The method for CalculatePay for a performance hall manager is to pass in the weekly salary and the hours worked for a week. The hall manager's weekly pay is :

If hours per week >= 40

 Pay = weekly salary

Else

 Pay = weekly salary \* (hours per week/40)

The method for CalculatePay for a performance hall agent is to pass in the weekly contract amount and the hours worked for a week. The hall agent's weekly pay is :

If hours per week >= 10

 Pay = weekly contracted amount

Else

 Pay = 0

B. The performance class should contain the following data and methods:

Data members:

should include all items from Lab #2 with the following changes and additions:

1. Performance group or groups that are contracted for the selected performance

4. List advertising and catering companies connected with these groups or this performance

5. Performance date, time, and day (in a PDate object – see PDate class below)

8. Total seats/tickets sold

 a) Ticket price per seat per category

 b) Available seat categories (festival, box, orchestra, special)

Methods:

should include all methods from Lab #2 with the following changes and additions:

11. Calculate ticket sales including new category information

12. Calculate profit or loss (ticket sales – expenses = profit or loss)

C. The performance group/company class should contain the following data and methods. You may call this the PerformanceGroupCompany class or you may choose another class name such as PCGroup or similar. This class should be an abstract superclass for later classes.

Data members:

should include all items from Lab #2 with the following changes and additions:

7. Most recent contract signing date

8. List of all members/employees represented with Person objects – see Person class

Methods:

should include all methods from Lab #2 with the following changes and additions:

10. Sign a contract including

 a. performance date

 b. amount

C.i. Each performance group also has the following attributes and methods. (This class should be a sub class of the Performance Group / Company class.)

Data members:

should include all items from Lab #2 with the following changes and additions:

1., 2., 3., Performer/Leader (in a Person Object – see Person class below)

4., 5., 6., 7. Agent (in a Person Object – see Person class below)

9. Type – this value must be an enumerated type of

 { BAND, SYMPHONY, SINGER, COMEDIAN, THEATER\_TROUPE, OTHER\_PERFORMANCE}

Methods:

should include all methods from Lab #2 with the following changes and additions:

C.ii. Each company also has the following attributes and methods. (This class should be a sub class of the Performance Group / Company class.)

Data members:

should include all items from Lab #2 with the following changes and additions:

1.,6. Company Address (in an Address Object – see Address class below)

2. List of services/items available

3.,4.,5., Contact person (in a Person Object – see Person class below)

7. Type – this value must be an enumerated type of

 { ADVERTISING\_CO, CATERER, RENTAL\_CO, OTHER\_CO}

Methods:

should include all methods from Lab #2 with the following changes and additions:

7. Print all company info

8. Print company address

D. Each Donor has the following attributes and methods.

Data members:

1. Donor personal info (in a Person Object – see Person class below)

2. Donor address info (in an Address Object – see Address class below)

3. Total donations amount given

4. Date of first donation of $1000 or more

5. Date of most recent donation of $1000 or more

6. Amount of most recent donation of $1000 or more

E. Each Person has the following attributes EXAMPLE

1. First name Jaya

2. Last name Smith

3. Identification number 006 – can be left blank

4. Address (in an Address Object – see Address class below)

5.Date of Birth (in a PDate object – see PDate class below)

6. Job Title Manager or Agent or …

7. Pay Type enumerated as SALARY, HOURLY, CONTRACTED, OTHER

8. Hours worked per week

9. Pay rate per hour

10. Weekly salary

11. Weekly contract amount

F. Each Address has the following attributes

1. Street Number

2. Street Name

3. Street Designator (St., Ave., Blvd., etc.)

4. Additional info (Box number, Suite, Apt., etc.)

5. City

6. State/Province

7. Country

8. Postal Code (Zip+4)

9. Preferred phone number

10. Preferred Email Address

For Lab #3, the PDate class should be modified to inherit from the GregorianCalendar class in Java.

<http://docs.oracle.com/javase/7/docs/api/java/util/GregorianCalendar.html>

<http://www.tutorialspoint.com/java/util/java_util_gregoriancalendar.htm>

Use the GregorianCalendar fields and methods to provide the functionality of PDate, to be able to get the current date, and to be able to compare dates. Note that the default values for a GregorianCalendar object are different from the PDate class. So use the GregorianCalendar defaults for PDate now. Make sure that PDate still provides the same output methods. Note that the GregorianCalendar object has a variety of constructors for creating new objects with dates and times.

The PDate class should still contain the following methods:

3. Print date as named month string: January 16, 2015

4. Print date as integers only in month, day, year format: 1-16-2015 or 1/16/2015

5. Print time as 12 hour clock with am or pm – 5:43PM

6. Print time as 24 hour time – 17:43

7. Print date and time (you may select formats)

**Program Requirements**

For Lab #3 you should implement all the requirements for Lab #2 and :

1. use inheritance to create at least three of the classes,
2. read initial data in from a input file (using a Scanner object connected to the file)
3. output all information to the user in GUI windows
4. Use Javadoc commands to comment your code and generate the Javadoc documentation to submit with the assignment

Java provides mechanisms for creating good documentation using Javadoc. Please make sure your code contains comments, input descriptions, method headers, etc. Use the relevant Javadoc tags specified in -http://en.wikipedia.org/wiki/Javadoc or <http://java.sun.com/j2se/javadoc/writingdoccomments/>

**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 or used consistently at end of line ( { open} and on a line alone ( } close).
* 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 and enumerated types 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, and

 d) some indication of the relationships between the classes.

The design document may be created from your Javadoc commands, 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 the following material using the Blackboard system at elearn.uta.edu. Please submit ONE .zip file that contains:

* Each .java file used in development with the appropriate name for the included class, enum, etc.
* A document, called your "test plan", listing each test case that is in your test class and what the purpose is of each test
* 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 .zip file containing your project must have the name format of: **proj2\_1000number\_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: (46%)**

 Headers, comments, style and modularity (10 points)

 Class header and method headers for all classes, 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 declaration of the required classes with data and methods and any other needed classes including all modifications and new classes (20 points)

 Correct interface definition (8 points)

 Test plan includes at least two tests for each of the required functions and uses the list and display functions to verify test results (8 points)

**Output: (54%)**

 File input works correctly (6 pts)

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

 All required tasks perform correctly (12 points)

 Set functions handle inputs correctly including error checking (3 points)

 Get functions provide values safely (3 points)

 toString functions implemented correctly (3 pts)

 Graphical user interfaces implemented as required (8 pts)

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

 Output contains at least two data sets, (6 points)

 Output demonstrates all elements of test plan (7 points)

**Grading Deductions:**

 Labs which do not compile in NetBeans will receive an overall grade of 0 (zero) [-100 deduction]

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

 Late submission of softcopy of design document, code and/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]

Miscellaneous: If you have questions, e-mail Dr. T (tiernan@uta.edu) and the TA.