CSE 1325 – Spring 2014
Object-Oriented and Event-driven Programming (Using Java)
Instructor: Sharma Chakravarthy
Design of Simple collaborating classes
Project II

Assigned On: 1/30/2014
Due on: 2/20/2014 (before 11:55pm)
Submit by: Blackboard (1 folder containing all the files/sub-folders)
https://elearn.uta.edu/
Weight: 5% of total
Total Points: 100

Overview of all projects: In a sequence of four (4) projects over the semester, you will develop an application that captures the operations of an enterprise. The scope of the project will include designing the components of the enterprise as described in here, choosing appropriate data structures, algorithms, and other aspects of managing information. You will develop a GUI along the way in one of the projects.

Problem (project 2)

The theme of this project is the design of classes, attributes (both static and instance), constructors, and methods (without using inheritance and polymorphism which will be exercised in project 3) along with error checks for managing information of an eBay-like enterprise (mavBay). The personnel for managing this system, such as employees (monitors, web designers, and maintenance people) will also be modeled. Information about putting items for fixed price sale/auction, bidding by users will be used to analyze revenue generation, popular items, commissions, and high and low bidders. We will use a simplified version of rules to keep the problem manageable. Please take a look at http://ebay.about.com/od/gettingstarted/a/gs_bids.htm for more information about understanding eBay rules. We will not use all of them.

This design will be based on the object-oriented design (OOD) principles. This project will form the basis for the rest of the projects in the semester. Please make sure that you adhere to the OO principles (taught in the lectures and the book) throughout the design and implementation phases. The projects, though small and tailored to the available timeframe, are designed to give you an opportunity to understand and exercise specific OO principles, taught in the course, for each project. Remember, Java is only a vehicle for supporting OOD. Make sure you fully and properly understand and leverage the features available in Java for your design. You will also use UML for your design and incorporate documentation as part of the code to facilitate meaningful javadoc output for reuse by someone else. Meaningful comments should also be included from a developer's perspective.

In this project, you will be designing the data abstractions (classes and their attributes) and required operations (constructors and methods) on those abstractions for processing auction and bidding information. Please note that this document is a requirements specification and not a template for the classes you will come up with. You should think about the problem and come up with as many meaningful classes as appropriate. First do the design using UML, get it approved before proceeding to implement it. This will save a lot of backtracking and wasted effort. Please note that this project does not require **and we do not want you** to use inheritance, polymorphism, or interfaces (except for the use of the constants file given). So make sure that you stick to the specification and expectations for this project. You will be exercising other OO features in subsequent projects.

You will be using the Date, Time, and DateTime classes (all provided; please study them for coding guidelines, javadoc comments, correct use and encapsulation of classes, attribute and method specifications) as utility in this project for dealing with date of birth (dob), start date, end date, time of bidding, total duration etc. You will be using constructors, and other methods including the compareTo() method from each class. Please study them.

General description of the problem (applicable to all 4 projects)

A. The enterprise (you will implement only a part of this in this project and others in subsequent projects. See section B onwards for what you need to implement in this project)

1. Has a name (e.g., mavBay)
2. Has employees: monitors, shippers, web-designers
3. Has a number of customers/users
4. Items are put up for sale/auction at a specific time
5. Biddings are done for an item on auction/sale
6. Keeps track of how many employees are currently working
7. Pays employees' salaries/wages on a monthly basis
8. Keeps track of items sold, their quantities, and other relevant details
9. Keeps track of customers – both sellers and bidders (one can be both)
10. Keeps track of revenue generated by sales/auction for each sale and its aggregation
11. Displays employees, sellers, bidders, and other information (in a easy to read tabular or graphic form) as specified with appropriate details

Specifically for this project, you will design and implement

B. Employees with	EXAMPLE
1. employee id	006 (system generated starting at 1)
2. First name	"John"

3.	Last name	"Smith"
4.	Dob	7-4-1978
5.	Gender (enum)	Male/Female
6.	Hire Date (Date type)	4-1-2008
7.	Release date (Date type)	1-13-2014 or null if still employed
8.	Monthly base salary	$1000.00
9.	Employee type (enum) (Web designer, accountant, customer support)	WD, ACCT, CUST_SUPPORT

Note that Employee id is generated and managed by the application (and is **not** input)

C. Auction/Fixed price/both items

		EXAMPLE
1.	Item Id (int)	1006 (given)
2.	Item category (String or enum)	antiques/books/consumer electronics/motors
3.	Item name (String)	Camera
4.	Item description (String)	Cannon EOS rebel T3i 18MP DSLR With 18-55mm
5.	Sale type (enum)	FIX_PRICE/AUCTION/BOTH
6.	Quantity (int)	5
7.	Item condition (enum)	NEW/USED
8.	Minimum starting bid	500.00
9.	Increment	20.00
10.	Reserve amount (or buy it now for FIX_PRICE or BOTH)	920.00
11.	Start date and time in CST	1-29-2014,10:00:00
12.	Number of days	3
13.	Seller id	18
14.	Seller feedback score	4800

Fixed price items can only be bought at the reserved price. We will assume they are also available only for the duration listed. They can be relisted again. Auction items need to start at the minimum starting bid or higher. Bids have to be higher than the last bid by at least 1 increment of amount specified. For both kind of items, bidding takes place as above and at any time someone can pay the reserve amount (by clicking on a separate buy it now button than the place bid button) and buy it right away. The auction will stop if the number of items bought is the same as the number of items for sale. Otherwise, the auction will continue for the rest of the items. If at the end of the auction, the quantity left is less than that requested in auction, the quantity left will be sold. If multiple quantities are shipped, the shipping charge will increase incrementally by 10% of the single shipping charge for each additional item. For example, if 3 items are shipped, the shipping charge will be 29.95 + 2.99 + 2.99. The bidder is allowed to change

the quantity from one bid to another. Reserve amount of an item is not shown to the bidder (unless it is both auction). The seller reserves the right not to sell if the bid amount at the end of the auction time is lower than the reserved amount. **For this project, we will assume that the seller will not sell if the bid amount at the end of the auction is less than 95% of the reserve amount.**

We will assume our bids in listed in chronological order of DateTime and each bid value respects the bidding rule described above. We will also assume a flat shipping charge of $29.95 is applied for each item (irrespective of the domestic destination).
Two kinds of fees are charged by mavBay for the seller: a) an insertion fee – a flat 1% of the minimum starting bid amount whether the item gets sold or not and b) a final value fee – 10% of the value of the item sold for + the shipping amount. This is applied **only if** the item is sold. We will not use the automatic bidding system used by eBay.

D. CUSTOMERS with	EXAMPLE
1. First Name (String)	David
2. Last Name (String)	Copperfield
3. User Id (int)	1 (given)
4. Date of birth (Date)	2-19-1978 or null if not known
5. Address (String)	500 UTA Blvd
6. State (String)	Texas
7. Zipcode (String)	76019
8. Status of Customer (enum)	SILVER/GOLD
9. Customer classification (enum)	BUYER/SELLER/BOTH

Customers are those who buy and/or sell items on mavBay. Customer classification is derived based on the data provided. Any customer who has transacted (sold or bought) Less than or equal to 2K per calendar year is considered silver and those more than 2K per calendar year as gold. The status is as of the end of 2013 (last year).

E. Bidding with	EXAMPLE
1. User id	10
2. Item id	1025
3. DateTime the bid was placed	1-26-2914,9:40:22
4. Bid amount	524.00
5. Quantity	2

Of course, the same user can bid any number of times for the same item. A number of error checks need to be performed by your software. For example, the DateTime of each bid needs to be chronological. Otherwise, discard that bid entry with a msg and ignore that bid value. Also, make sure the bid amount follows the at least 1 increment rule.

F. Input will be provided as well as the code for reading input

We will finalize the input for this project (to be read from an ASCII file) and provide code for reading it as MavBayTest class. You will also be given a sample input file. You will use this as the driver class for reading input, showing the menu, and executing actions as needed. You will be adding your code into this class. Please do not hardwire your design to this input file. It should work for any other input file in terms of the number of employees, facilities as well as other details. You will add code to this driver class to implement the menu and the output. Also, study the input file and the code provided to read it.

You will first read the name of the enterprise followed by the number of employees. Do not assume limits on numbers unless explicitly specified. Then you will read details of employees (all attributes). Then you will read details of items to be listed followed by customer details. Number of items to be listed or customers is not specified. Should be computed from the input (the code given does it; please study it)

The input data file (given to you) will consist of the following:
//comment line
Name of the enterprise
`#employees`
//Employee details: one employee per line
`emp type!fname!lname, dob!gender!hire date!release date!monthly base salary`
//Items listed: one item per line
`Itemid!item category!item name!sale type!qty!Item condition!Minimum starting bid!bid Increment!Reserve amount!Start date and time in CST!Number of days!Seller id!seller feedback score!item description`
End
//Customers of mavBay: one customer per line
`User id!First Name!Last Name!Date of birth!Address!State!Zipcode`
End
//Bidding information: one bid per line
//for this project, we will give bidding for each item one after another
//we may jumble it up for later projects
`User id!Item id!dateTime!amount!quantity`
End
You may want to convert all String inputs into either lower or upper case to make comparison easier and avoid problems with input. This can be done by using the following methods:

String upper = myString.toUpperCase() or
String lower = myString.toLowerCase()
where myString is an attribute/local variable of String type.

At the end of processing the input, you should display, on the command prompt, the menu described below, accept input from user, and process commands as specified.

What you need to accomplish in this project (and will be graded on):

A. Design classes (both public, package-private, and private as needed), attributes (class and instance, as well as their visibility/accessibility), constructors, and methods (private, public, protected) for the above problem description. Make sure you understand and differentiate between public/private/protected/package-private attributes and methods. Use static fields/methods where appropriate. Make sure you have at least one constructor (more as needed) with arguments in each class (except may be in the driver class). Use appropriate constructors to initialize your attributes appropriately. Each class should have appropriate `get` and `set` methods (as needed, not for the sake of it). Each class should have a `toString`() method (and other formatting methods for display as needed). Keep all classes as a single package. Make sure all the constants used in your program is grouped into an interface file. We will supply a Proj2Constants file. You can add additional constants, if needed.

B. Use the driver class (the MavBayTest class given to you) to run and test the application. The main driver class should read in the data, create objects and use other data structures as appropriate, and print the menu indicated below, accept user input and execute the commands to provide understandable formatted output. All inputs (as appropriate for the menu) should be accepted from the command prompt window and should prompt for the input with appropriate messages. If bounds are meaningful, please indicate it in the prompt and check for input errors. Please do unit testing (i.e., testing of each class using a main method of that class) to make sure each class works as expected. You can do this by including a main method in each class for testing the correctness of that class.

C. We will be using command prompt input only for this project. From next project, we will take commands also from a file and process them.

D. In addition to the design and implementation of classes (i.e., attributes, constructors, and methods), your program should display the following menu list (as indicated below, after reading all input and creating appropriate objects) on the screen to the user and accept a number (e.g., 3 for listing customers) associated with

the menu from the command prompt window (along with needed inputs) and perform the functions as indicated. Input is indicated in the parenthesis:

1) List all employees with details including specialization (no input)
2) List items to be sold with details (* means all, amount means list items whose reserve amount is above that amount specified) including their min starting bid, duration etc.
3) List all the customers with details (no input)
4) Display items sold ("*", "fixed price" or "auction" or "both" to list all items or items sold in that category)
5) Display total fee collection (optionally by category)
6) Display
7) Release an employee (input: employee id)
0) Exit program

Make sure that proper error checks, error messages as appropriate are given for the above. For example, you cannot bid on an item you have listed for sale; you cannot bid for more than the quantity listed for auction, etc. You can discard the input and give meaningful error messages along with the offending input values. Please note that this project does not require you to create any GUI.

D. 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://java.sun.com/j2se/javadoc/writingdoccomments/

E. In addition, you will report a few (3 to 5) significant logical (not syntactic) errors (that were difficult to debug) that you encountered while doing this project. Furthermore, you will briefly provide an explanation on how you debugged and fixed each of these errors.

F. For a bonus of 10 points, you can submit a UML diagram for the project and get it approved by the TA or me during the first 10 days of the project. It should have, at the least, each class description in detail along with their relationship with the other classes. You can do this using .doc or .ppt tool. If you use visio or any other tool, convert it into .pdf format and submit it. It does not have to be fancy, but contain proper description of the class. Further information about modeling classes/methods/etc. in UML can be found at http://en.wikipedia.org/wiki/Unified_Modeling_Language. You will not get bonus points if you submit it along with the code!

G. You will submit a script file as part of the project submission as specified in the other document (see the other document for generating the script file)

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:

a. Class names begin with an upper-case letter, as do any subsequent words in the class name.
b. Method names begin with a lower-case letter, and any subsequent words in the method name begin with an upper-case letter.
c. 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.
d. No hardwiring of constants. Constants should be declared using all upper case identifiers with _ as separators. And preferable as an interface.
e. Each set of opening/closing curly braces must be in the same column. All user prompts must be clear and understandable
f. 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
g. Your program is properly indented to make it understandable. Proper matching of if … then … else and other control structures is important and should be easily understandable
h. Do not put multiple statements in a single line
i. All *static final* variables will be upper case identifiers
j. There will **no hardwired constants** used in the programs. We have provided a constants file to be used for the project (proj2Constants.java)
k. Please use a main method in **each** class to test that class

In addition, ensure that your code is properly documented (apart from javadoc commands) in terms of comments and other forms of documentation wherever necessary.

Grading Scheme:

The project will be graded using the following scheme (total)	**100**
1. A UML diagram depicting the classes, methods, and relationships	5
2. Consistency of implementation with UML diagram	10
3. Correctness of the design, implementation of classes, attributes, and methods:	45
4. The ability to accept appropriate input commands and generate correct output:	20
5. Reporting and fixing 3 to 5 logical errors in the program:	5

6. Documentation (javadoc) and commenting of the code: 5
7. Following the coding style listed above: 5
8. Inclusion of the script from Omega as specified: 5
9. Bonus for UML submission and getting it approved earlier 10

Correct execution of the program constitutes a significant percentage of the grade. Please make sure it works correctly before you submit it. We reserve the right to question you about the details of your implementation and testing to make sure you have indeed designed and implemented the project.

What and How to Submit:

For this project, you will submit the following using the naming convention suggested below:

1. Each .java file will contain one or more related classes
2. A text file listing 3 logical errors encountered and the process used to debug and fix each of them. *Include the total number of hours spent on the project as well.* The file should be named as – 'proj2_errors_firstname_lastname.txt'
3. All the javadoc and html files generated by the javadoc command should be in a folder named javadoc where the .java files are. Use the command `javadoc –d javadoc *.java` to create java documentation.
4. The UML diagram named as 'proj2_uml_firstname_lastname.xxx' (Use appropriate extension in place of 'xxx' depending on the tool used; only doc, ppt, and pdf are accepted)
5. A script file as described in a separate document on the data that will be supplied for submission.

All of the above files should be placed in a single zipped folder named as - 'proj2_firstname_lastname_final_setion.zip'. **The folder should then be uploaded using blackboard using your login. Note that blackboard will not permit you to submit after the deadline [11:55 pm] has passed. Please login and familiarize yourself in using blackboard. Please make sure you press the SUBMIT button to make sure the project is submitted.**

Your java program must compile and execute (without any modification by us) from the command prompt on UTA's Omega server. So, please test it on Omega before submitting it. However, the source code files can be created and/or edited on any editor that produces an ASCII text file. As I mentioned in the class, an IDE is not necessary for these projects. If you decide to use it, please learn on your own and make sure your code compiles and executes without need for any modification after submission (you may have to remove a few things the IDE may insert by default).