Rohit Tandon
ICS372 Object Oriented Design and Implementation
Group Project One – Part Two

Project Goals

- Form groups and plan work for successful project submissions
- Follow the specifications given and write compliant code.
- Refactor you part one of the project.
- Apply principles learned in class to part two.
- Document and lay out your code properly as specified under ICS 372 Coding Standards.

Project Description

In this project, you will do some refactoring and extend some of the functionality of a theater. In this iteration, you will implement the following:

I. Refactor the collection classes.

II. Fix any bugs you have from Iteration 1.

III. Implement the functionality related to issuing tickets. There are three types of tickets: ones that can be purchased on the day of the show (called regular tickets), the ones that are purchased at least a day in advance (called advance tickets), and tickets purchased for students (called student advance) . An advance ticket costs 70% of the price of a regular ticket for the same play. Student advance tickets cost 50% of the price of an advance ticket. Every ticket has a serial number, the date of the show, the type of ticket (regular, advance, student advance), and the ticket price; for student advance tickets, there is an extra message that says "Must show valid student id."

Two different plays may have different ticket prices. When a show is added (command 9), you must now ask for the price of the regular ticket. The cost of the advance ticket (and student advance) must be automatically adjusted.

A customer may purchase any number of tickets. A customer may purchase, for example, 2 advance tickets, and 3 student advance tickets for the same show. Then he/she may purchase some more at the gate. When they arrive for the show, however, there must be three individuals who can show valid student ids.

The revenue from ticket sales for a show is divided equally between the theater and the client (the show's producer). Thus, whenever a ticket is sold, you must update the balance for that client.

<u>Implement the following new commands.</u>

13. (This was "Help" in Iteration 1.) Sell regular tickets. Accept the quantity, customer id, credit card number, and the date of the show. Update the customer object and the client object appropriately.

14. Sell advance tickets. Accept the quantity, customer id, credit card number, and the date of the show. Update the customer object and the client object appropriately.
15. Sell student advance tickets. Accept the quantity, customer id, credit card number, and the date of the show. Update the customer object and the client object appropriately.
16. Pay client. Accept the client id and display the balance. Ask for the amount to be paid to the client (which must be verified to be no more than the balance) and update the client balance.
17. Print all tickets for a certain day. All fields of all tickets must be displayed.
18. Help. This should display all commands.

The following commands must be amended to include the underlined functionality.

3. List all Clients. Print information about every client. This should now show the balance correctly. 8. List all Customers. Print information about every customer, including credit card information, and all tickets ever purchased by the customer.
9. Add a Show/Play. Add a new show for a client. The values accepted are the name of the show, the client id, the period for which the client wants the theater for this play, and the regular ticket price. The entire range of dates should be available, or the process fails.

You must have a command line interface (not GUI) that uses numbers for issuing commands. It should be possible to invoke each functionality by typing the number associated with it as in the above list. For example, 0 must exit the application and 13 should display the help menu; 8 should list all customers.

The three types of tickets must be implemented using a class hierarchy. You cannot have a single type that caters to all forms of tickets.

Program Submission

1. A use case for every one of the new business processes (13-17) written in the two-column format.
2. The complete conceptual class diagram.
3. Class diagram for every new class and/or interface with their relationships. This includes any that are added or modified for refactoring the collection classes.
4. Sequence diagram for every new use case.
5. Java code for every class, including the user interface.
6. Do not worry about use cases, class diagrams, and sequence diagrams related to the modifications to commands 3, 8, and 9.
7. Items 1, 2, 3, and 4 must all be presented in a single Word/PDF file. Every entry must be clearly labeled. For example, before every use case, write its name. Obviously, class diagrams are self identifying. Use meaningful names for all classes and methods.

Zip the files and submit in the D2L dropbox. *If the program has syntax errors, the grade will be 0: no exceptions.*