Aderonke Abidoye

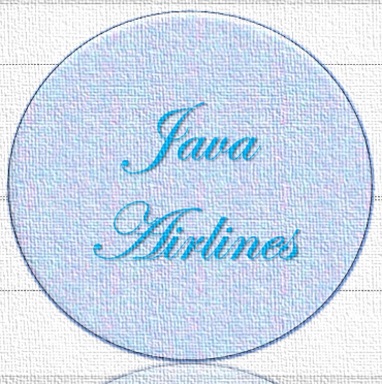
John Lee

Dr. Arash Akhlaghi

May 1, 2016

Java Airlines: CIS 3270 Project

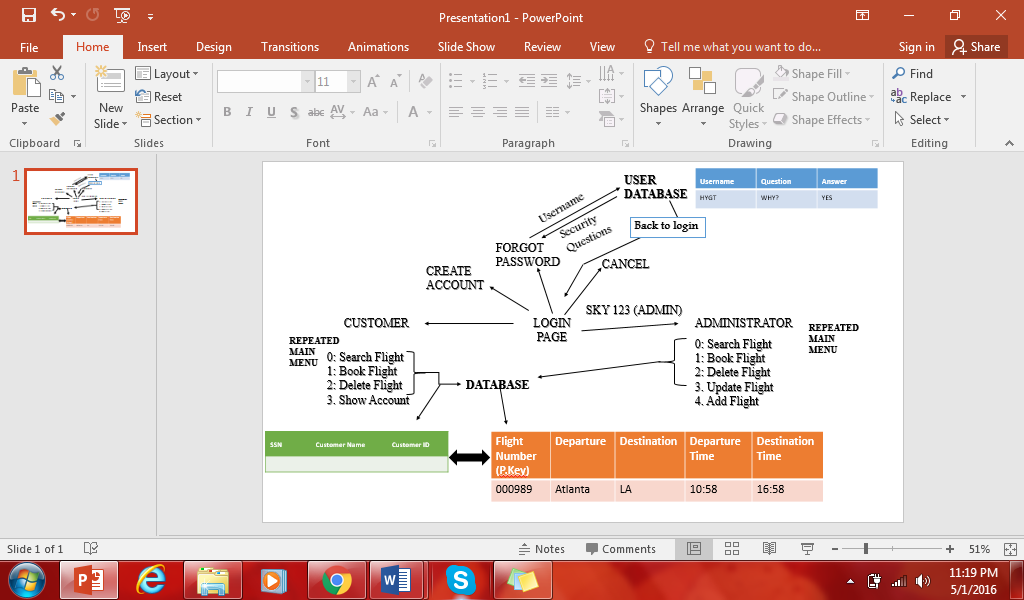
For our Java Airlines project, the basic requirement was to create a functional airline reservation system like *Expedia*. Database knowledge, graphical user interfaces, and knowledge of class inheritance were very important for this project. As outlined in the project instructions, the application should display a splash screen upon the start, which indicates the loading of the application (like Microsoft Word).



How our splash screen looks like:

Then, the application should allow users to input already existing username and password or to create an entirely new account. With this account, customers should be able to search, book, and delete flight. The customers should also be able to login and logout as they please, and there should be a main menu option, in case the customer ever needs to return to the main menu. A forgot-password option must also exist, so that returning customers may access their password with their username and security questions.

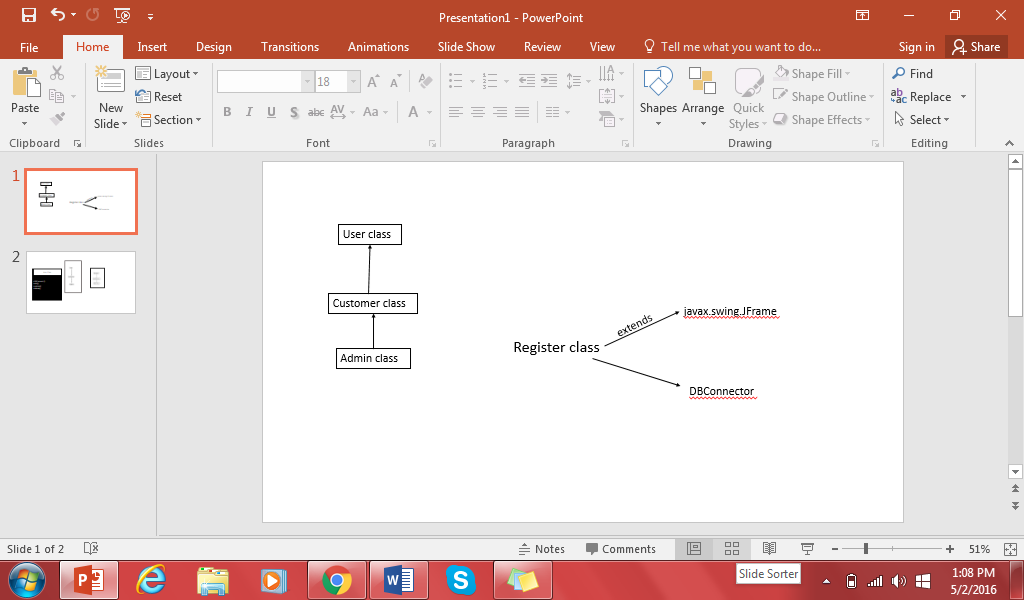
In designing this project, we came up with this layout:

As outlined above, there is another user called the administrator. This user has admin privileges and can update, delete or add a flight to the already existing database. While the customer can interact with the existing database and add flights from the general database into his own account (which is also another form of database), the customer does not have access to manipulate the general database.

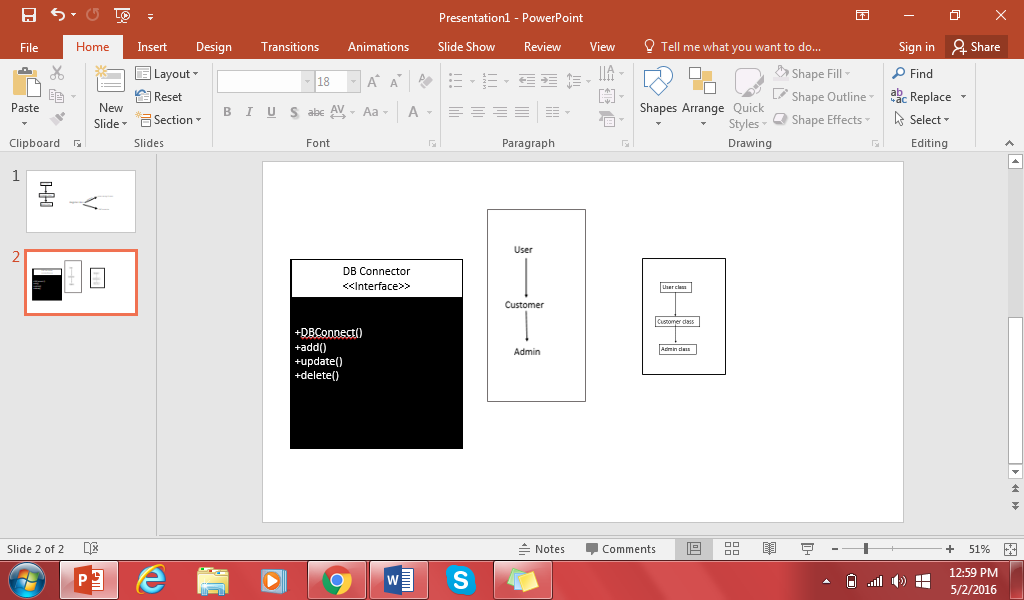
**CONCEPTS USED IN THE PROJECT**

***Inheritance***

For our project, we used a lot of instances of inheritance. In fact, almost every class extended from another class. We used both custom classes of our own, like our User class for the logic. We created a User class, from which we extended Customer, and from Customer we extended Admin—since Admin had more privileges.



We also created our own DBConnector interface, which contained the basic privileges that a user (whether admin or customer) has. The DBConnector, like the name says also connects the program to the database to create/ new information



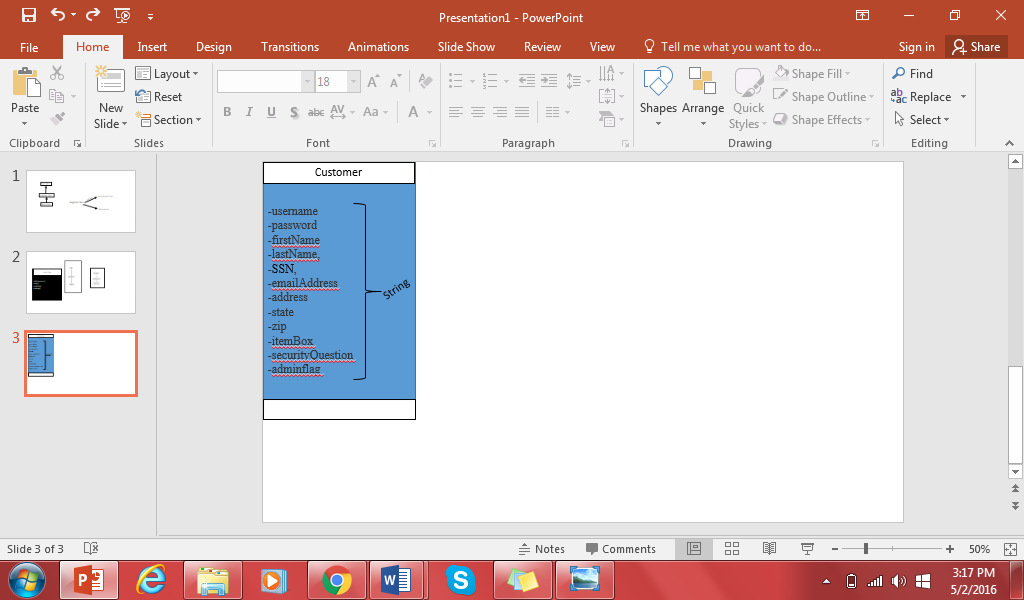
We also used already existing Java classes like JPanel and JFrame for the GUI aspect.

***Polymorphism***

There were also instances of polymorphism in the logic of our paper. For instance, a customer *is* a user and an admin *is* also a user.

***Encapsulation***

Even though all of our project was saved in one package, we utilized encapsulation to keep the properties locked into their classes, by using a private access modifier. We also had methods (setters and getters) to make the properties accessible, but not directly modifiable.

**DATABASE MODEL MANIPULATION**

The diagram below illustrates the data models for the airline reservation database. The above tables were created in MySQL. The User table contains the parameters for all users—customer or admin. The Ticket info contains information about each customer’s individual flight pattern. The Flight table contains information about all the flights available, and lastly the Plane table contains plane information.



**CODE: CLASSES AND IMPLEMENTATION**

All the classes in the project were saved in one package called *project.* Hence, there was no need to import the custom classes, just Java classes. Upon executing the program from the Start\_Program class, a splash screen, as detailed above, would show and then there is a transition to the Login page. The Login page gives the user the choice to login with their username and password, or to click “register” to create a new user account. If an existing user forgot their password, they can also click the “forgot password” button to retrieve password. Once the user clicks this button, they are asked to provide a username and answer some security questions and then updates the password. On the other hand, if the users logs in with their username and password, the program will look into the database to see if the credentials match, if not it returns an *invalid* username/password. If the user decides to create and account, the program asks the user a couple of questions and then enters the received data into the database. The user is then taken back to the login page, where they login with their username and password. When the credentials match, the user is sent into the MainMenu page where the user is presented with different selections to see their own flights, search flights, book flight or delete a flight from their account. The user can also decide to logout. Doing this will result in a “Goodbye” message and then a new login screen pops up again awaiting user input.

If the user decides to click the “List Flight” button, it goes to a new class which contains the flight information (*Flight\_Info)*. The flight list, which is dispalyed in a multidimensional array of objects, is produced. The program prompts the user to remember FlightID will be important for booking. After the user has seen the flight they want, they go back to the MainMenu, and click on “Add Flight.” The program takes them to a page where they would type in the FlightID and then add the flight to their account. The user can then go back to the “Main Menu” and click on “My Flights.” The new flight will already be there, along with any other flights that have been previously added. It also works the same way for dropping flights.

Another button that is vis only visible to the administrator, but not the customer, is the button that allows the admin to delete/update/add flights. The administrator, with this button, can access the database to delete a row or a column from the flight table. The admin can also update the flight information and even add an entire row or column.

**Important Methods**

*Login Class*

* run(): displays the login screen and enables users to click on preferred action

*MainMenu Class*

* customerRun(): Displays main menu for the customer
* adminRun(): Displays main menu for the customer

*MyFlight Class*

* run(): displays the customer’s added flights

**Problems We Faced and Lessons Learned**

There were a few obstacles in the implementation of the application and in the creation of this project. First, we had some trouble navigating Github. We were not aware that Github was a necessary requirement for the project, hence we had already begun using another code-sharing application—*Codeshare*. Hence, we had to bulk upload the project and learn how to work Github in just a few hours, so as to not to get behind schedule. It is important to note that after a few days, it was easier to work with. Another thing we struggled with was implementing admin privileges—especially since the Update function in MySQL does not work for rows, just columns.

Time was also a big problem. Being such a big project with a lot of code, it would have been more efficient if we had more time—like a few more weeks.

As a result, there were a few aspects we were not able to finish. We could not implement the checking if the flight has a time conflict. We also did not implement if the canceling of the flight if the flight was full.

The project was a big task to accomplish, but it was worthwhile. Lessons learned from the project for subsequent is to gain access to the project earlier in the semester. This will give more time for proper planning and implementation, and a better program all around.