Alex Reyes

COMP 296

Capstone Project Functions/Methods

**FISER Application Method/Functions Demonstrations & Examples (Java Swing/Window Builder)**

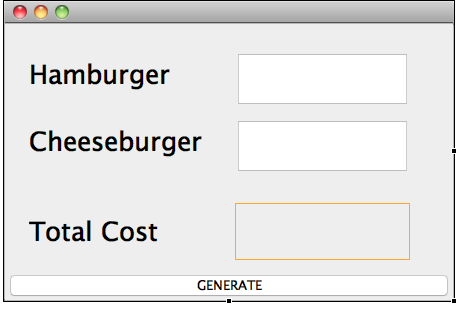
I have created a separate package that demonstrates each function. Our calculations package was a blueprint on which our main GUI (Pizza\_UI.class) will be utilized. First, we must understand the logic and structure of each method. Once we’ve established that, we can build and expand on it from there. *Note: We have since edited and added more into the original blueprint onto our Pizza\_UI.class.*

We will list the following functions of our original blueprint for:

**Table Of Contents:**

1. Adding menu items method() (Menu\_Items.java GUI).
   1. Structure of method.
   2. Logic of method.
   3. Test performed.
2. Adding “Jcombobox” items method() (Drop\_Items.java GUI).
   1. Structure of method.
   2. Logic of method.
   3. Test performed.
3. SQLite Connection to database examples (login\_S.java GUI).
   1. Structure of method.
   2. Logic of method.
   3. Test performed.

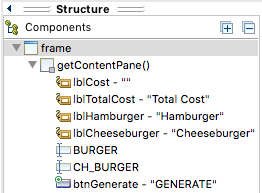
1. **Adding Menu Items Method().**



The end user will input the x amount of quantity that they desire. For our example, we added hamburger and cheeseburger. Once end user hits the “GENERATE” button, the action performed will take the x amount of input entered and calculate it with the set determined price of the hamburger or cheeseburger.

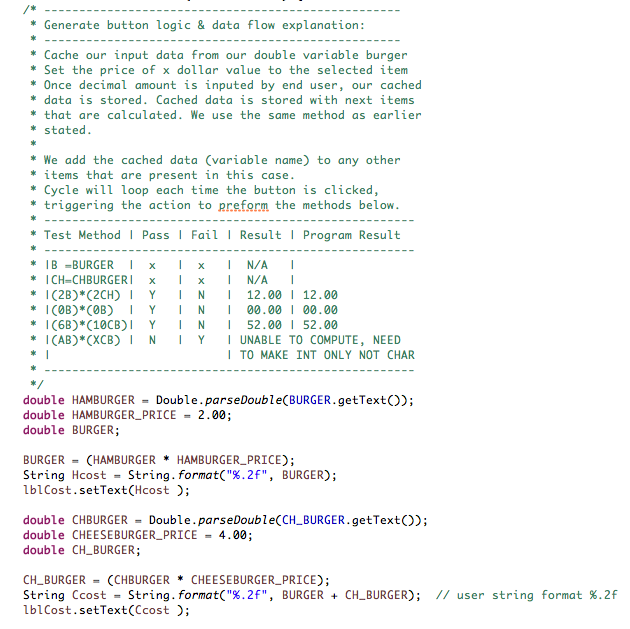
***Logic: “((Price of Item)) • ((x amount entered)) = Total Cost”***

**1a. Structure of Adding Menu Items Method()**

****

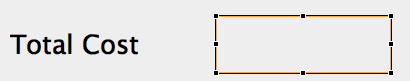
* **Four “JLabels” = (lblCost, lblTotalCost, lblHamburger,lblCheeseburger).**
* **Two “Textfields” = (BURGER and CH\_Burger).**
* **One button “GENERATE” = Triggers Action Performed (to give us total calculation amount).**
* **Custom string called “Hcost”, which will output the value and set it to the lblCost.**

1b. **The calculation for our logic:**



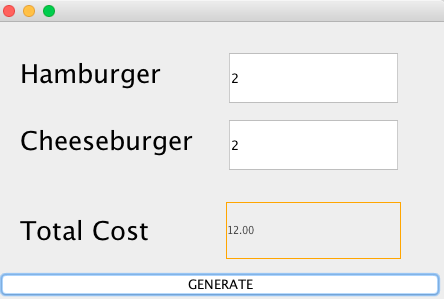
**Double HAMBURGER** is the name of the variable that is **doubleparsed()**. We use the **doubleparsed method** because this method returns the double value represented by the string argument.

**Double HAMBURGER\_PRICE** variable is the set price of x dollar amount. The **BURGER** variable has no data because this will cache our calculation. **(BURGER = (HAMBURGER \* HAMBURGER\_PRICE**). Once the end user inputs the quantity and hits the **GENERATE** button, it will trigger the *action performed* and output the calculation that is set to our **JTextLabel** to the variable name **lblCost.** *Note: The same calculations repeats for cheeseburger.*

**<--- Where the final output will occur (inside the box).**

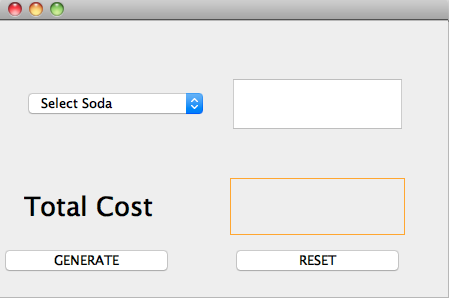
**1c. Test Performed:**

For our test we will enter *2 Hamburger and 2 Cheeseburgers.* Given the price of Hamburger = 2.00 & Cheeseburger = 4.00 The calculation will be (2H) \* (2CH) = 12.00. Which, it calculated correctly. *Note: We have to use String.format(“%.2f”, variables) to make the output number have decimal digits.*



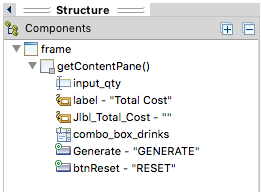
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Adding “Jcombobox” items method():



The end user will select an item from our JComboBox with predetermined items (set in an array). Once end user selects which item they want, they will input the x amount of quantity that they desire. For our example, We have an array of sodas [Select Soda, Coke, Diet Coke, Sprite, Setlzer]. Each soda has a set price that is already cached in a variable. After end user has entered the amount, it will calculate the item, based upon their selection.

2a. **Structure of JComboBox Items:**



**Two “JLabels” = (label - “Total Cost” and Jlbl\_Total\_Cost).**

**One “Textfields” = (input\_qty)**

**One “JComboBox” = (combo\_box\_drinks)**

**One button “GENERATE” = Triggers Action Performed (to give us total calculation amount).**

**One button “RESET” = Triggers our fields to reset all values to default. *We need this to ensure our reset button works in our Pizza\_UI.java GUI.***

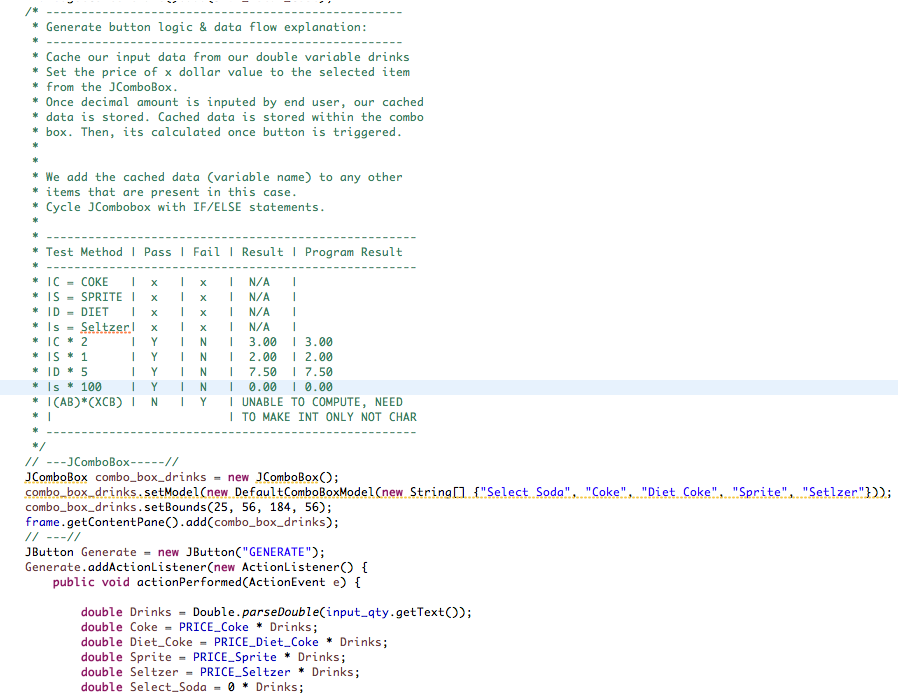
**2b. The Calculation of our logic:**

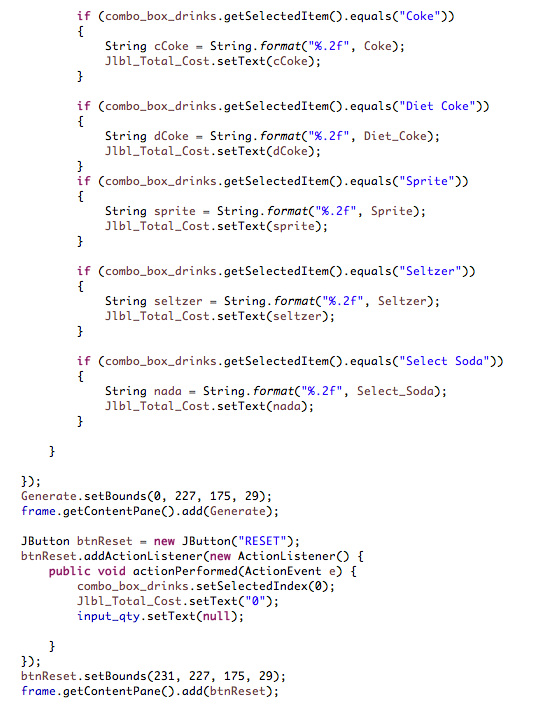
We can set our determined prices by calling ***“final double variable name = determined price;”***

Example:

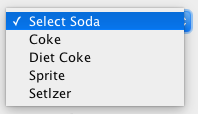


Then we start constructing our method and functions:





First, we need to declare our JComboBox model (names) in an array. We have chose four drinks (Coke, Diet Coke, Sprite, and Seltzer).



Once have our JComboBox filled out, we will then need to add our logic and data calculations. We have **6 double variables**:

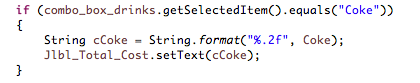
1. **Double Drinks** = The cached amount that the end user has entered via **“JTextField”** named **“input\_qty”**.
2. **Double Coke** = The cached amount that we have predetermined from our “*final double PRICE\_COKE = 1.50;*”
3. **Double Diet Coke** = The cached amount that we have predetermined from our “*final double PRICE\_DIET\_COKE = 1.50;”*
4. **Double Sprite** = The cached amount that we have predetermined from our “*final double PRICE\_SPRITE = 1.50;*”
5. **Double Seltzer** = The cached amount that we have predetermined from our “*final double PRICE\_SPRITE = 0.50;*”
6. **Double Select\_Soda** = The cached amount that we have predetermined. This is, in case the end user doesn’t select a drink on accident. So, in this case, we set it to 0.

Also, we have string “Drinks” calculated to the input of what the end user has entered:

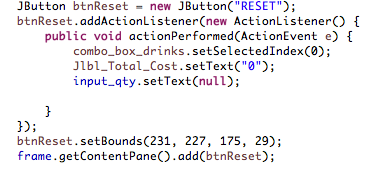


*It is “doubleparsed” because this method returns the double value represented by the string argument. In this case, we are “getting” the text for our input\_qty JTextField.*

Then, we need to use **if statements** each time the end user selects the item from the **JComboBox.** We create a **string cCoke** that stores that value from Coke and then set it the to **JLabel (Jlbl\_Total\_Cost)** after the calculation and if statements end.

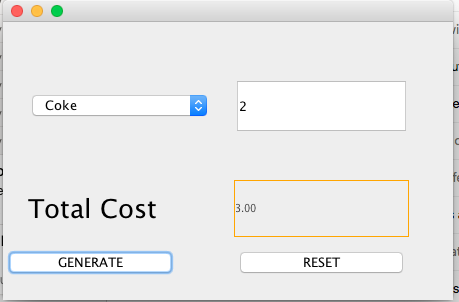


The **“RESET” button** will clear the values entered, once end user hits it. Simply to start over again. This feature is crucial in our Pizza\_UI.java GUI since we need a reset button to begin each new transactios.

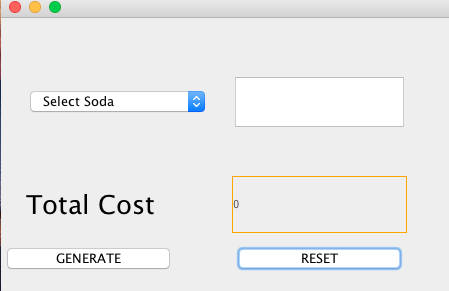


**2c. Test Performed:**

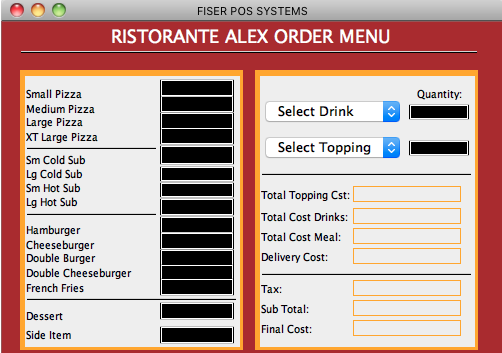
For our test we will enter *2 as our amount and select the item “Coke”.* Given the price of Coke = 1.50, the calculation will be (Coke \* (Input)) = Total Cost. Which, it calculated correctly. *Note: We have to use String.format(“%.2f”, variables) to make the output number have decimal digits.*

**

The **RESET method** works correctly. It resets the values back to the beginning, or null.

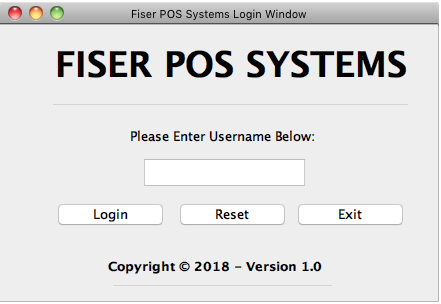


We have taken both methods and functions above, and implemented them all into our Pizza\_UI.class. I have since then edited and created more functions into our final product, based upon above blueprints.



*This is the final version of what we have created based upon our two calculation examples above. I have since expanded and implemented the functions and methods above.*

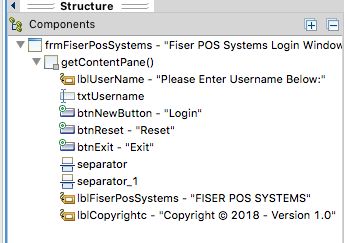
3. SQLite Connection to database examples (login\_S.java GUI).



In order for the end user to access our Pizza\_UI.java GUI interface, we must first make sure that the end user can access the database, with the given username *(provided from our Employee\_ID table in our SQLite Database).*

If the username is entered correctly, it will allow the end user to login, else it will not allow the user to login. It would throw the end user a warning message box. Once the value is entered correctly, it will dispose the login\_S.java frame.

3a. Structure of login\_S.java GUI



**Three JLabels = (lblUsername, lblFiserPosSystems, and lblCopyright)**

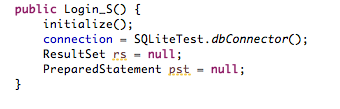
**Three Buttons = (btnLogin, btnReset, btnExit)**

**Two Separators = (separator, seperator\_1)**

**One JTextField = (txtUsername)**

3b. Logic of Login\_S.java

First we must make a method to ensure that our database has a connection:



We create two variables “**rs” for ResultSet & PreparedStatement as “pst”.** These two variables need to be used in order to utilize our JDBC plugin (Java Database Connection). By default, we set them to null. The connection is set to our class that uses the SQLite database connection. In this case, it’s our SQLiteTest.class.

Once we’ve established connection, we will run a simple SQLite query to open our Pizza\_UI.java. If the end user inputs their username correctly inside the **txtUsername JtextField**, it will open our Pizza\_UI.java GUI.



Once the end user inputs the correct username, it will allow them to proceed. Else, it will throw a warning message box.

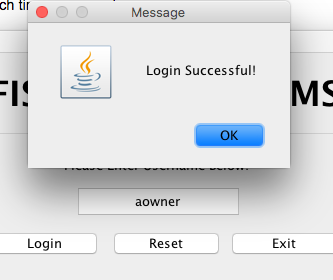
To communicate with our SQLite Database, we have to use the try/catch method to embed a custom SQLite query.

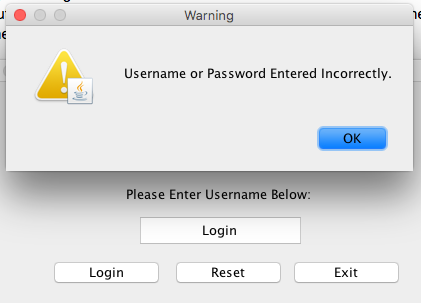
The query is as follows: **“SELECT \* FROM Employee\_ID WHERE Login=?”;**

We need to select \* (all) from our table “Employee\_ID” where the Login column is ? (meaning we don’t know yet) because we didn’t define it yet.

The  **txtUsername.getText()); method** gets the contents from the **txtUsername JTextField.** The prepared statement executes our command that syncs with our database. We must use the if/else statements for our connection each time we execute a query.

Once it can confirm that the connection and table exists in our database and the username matches our table username, our login\_S.java frame disposes with a “Login Successful” popup message. Else, a warning will pop up with “Username or Password Entered Incorrectly.” Afterwards, we need to close the connection to the database.

 <---- When the username is entered correctly.

 <---- When the username is entered incorrectly.