Unit Testing

Praveen Chekuri, Pradnya Pisal, Kartik Bhatnagar, Bill Fung, Zac Brown

Unit Testing

There are three methods that need to be tested independently to make sure Manage Menu is functioning correctly.

One of them is setMenuItems where the method seeks to store item data information correctly based on item name, item food type, item description, item price. It is important to note that in the actual scenario, itemName, itemFoodType, itemPrice must be entered in by the user in order for setMenuItems to function as needed. In other words, these fields cannot be left blank.

Below, the values are set for itemID, itemName, itemFoodType, itemDescription, itemPrice for testing purposes. This will add item “Test Menu” correctly into the database. If this test was performed twice in row, it will result in an error the second time around for two reasons: 1) contains a duplicate itemID key 2) contains a duplicate itemName

```
statement.executeUpdate ("INSERT INTO MenuItem (itemID, itemName, itemFoodType,
itemDescription, itemPrice)" + "VALUES (1000007, 'Cheeseburger', 'Burgers', 'Test
Menu Description', '10.50')");
```

Suppose, itemID was now set to 100006. This will result in a bug since there are already six items listed under ‘Burgers’ and thus counter for itemID needs to be incremented each time a item is added under that specific food type.

```
statement.executeUpdate ("INSERT INTO MenuItem (itemID, itemName, itemFoodType,
itemDescription, itemPrice)" + "VALUES (1000006, 'Cheeseburger', ‘Burgers', 'Test
Menu Description', '10.50')");
```

Error will result if itemFoodType is spelled incorrectly. After performing this unit test, we arrived at the conclusion that it would be better to create a drop down list (JComboBox) of all the food types for the user to select from.

```
statement.executeUpdate ("INSERT INTO MenuItem (itemID, itemName, itemFoodType,
itemDescription, itemPrice)" + "VALUES (1000007, 'Cheeseburger', ‘Borgers', 'Test
Menu Description', '10.50')");
```

Error will result if itemPrice is entered incorrectly.

```
statement.executeUpdate ("INSERT INTO MenuItem (itemID, itemName, itemFoodType,
itemDescription, itemPrice)" + "VALUES (1000007, 'Cheeseburger', ‘Burgers', 'Test
Menu Description', '10.5.0')");
```

Another method is removeMenuItems in which its the task is to remove the desired menu item from database. An error will display only if item is not contained in the database. I

Lastly, a separate method for getMaxMenuID was designed to generate an itemID for each food item based on its given itemFoodType. While “Food Type” is not shown in the add menu item interface yet, we were still able to test this function but initializing food type to “Burgers” or practically with any other food type stored in the database.

We focused on the individual components of the Add-Item interface in order to put together a fully functional working design. The Add-Item module can be divided into the following components for feasibility of tests and implementation:

- Choose Food Type
- Display Menu Item
- Add Item/Comments
- Remove Item
- Confirm Order

Choose Food Type

This main function of this component is to list the items under each food type. So depending on what food type is selected this function is supposed to query the database and return a list of food items. The key code for this component exists in Foodtypebuttons.jpg.

```
((MenuTest2Activity) MenuScreen).setMenu( 'Flatbreads' );
```

The above is supposed to generate a list of flatbreads from the database.

```
((MenuTest2Activity) MenuScreen).setMenu( 'Appetizers' );
```

While the above is supposed to generate a list of appetizers after querying the database.

Display Menu Item

The main function of this component is to display the description of the menu item selected.They keycode for this component exists in Menuitem.java.

```
public MenuItem(String s,String d,double p){
            }
```

The above input is supposed to print out the details of the menu item.

```
Name = s;
Description=d;
price=p;
So for example we can test with
Name = “Wings”;
Description = “Spicy Garlic.”
Price = 45.5
```

Confirm Order

The main function of this component is to delete an item that has been added to the linked list. This component is also supposed to add up the total for the order as well. The keycode for this component exists in confirmorder.java and orderlist.java.

```
private void getOrder()
```

The above function is supposed to read the linked list. The output of this function is to obtain the elements of the overall order for other order processing functions.

```
public void calculateTotal()
```

The above function adds up all the items on stored on the linked list. This is supposed to output the total of the order when called.