OBJECT ORIENTED DESIGN PATTERNS - ASSIGNMENT 2

Submitted in part fulfilment for the degree of
B.Sc. in Science in Computing
Institute of Technology Blanchardstown
Dublin, Ireland

By
Aaron Ward B00079288

2017

Declaration

I herby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of B.Sc. in Science in Computing in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed:_____ Dated: _____/_____/_____

Table of Contents

1

Introduction

The aims of this report is to highlight and explain the design and development of software using four object oriented design patterns. This project consists of a cake maker shop program written in java that can register users, store credentials in a database, allow users to make cakes, view registered users and retrieve their information. Initially the system requirements and specifications needed will be discussed to describe the functional goals of the program. This is followed by a section on system design, which will explain the design choices that have been considered and implemented. Furthermore, a section on the implementation is provided in order to explain the chosen design patterns and snippets of code in the program used to attain the goal of the project. Lastly, a conclusion section is given clarifying the achievements of the project and also further work that may be carried out.

2

System Requirements and Specification

2.1 Java Development Kit (JDK)

The Java development kit, which contains the Java Runtime Environment and the Java Virtual Machine, are required for the development of this project as it allows the code to be compiled and executed.

2.2 MySQL

MySQL is a relational database management system used in order to save information in a database. In this case, should a user register with the program a user ID, username and some other additional information may need to be stored. This persistent storage is ideal as it allows a user to retrieve important items created by a user should the program be shut down. SQL is a computer query language used to store, update and obtain data from the database. Data is stored in tables which consist of 2 attributes, table rows and table columns. When an SQL query command is executed, the RDMS determines what the most efficient way to carry out the command would be and the SQL engines determines how to interpret the task that must be carried out. SQL commands have a simple English syntax for querying data (see example below):

```
SELECT * FROM table WHERE id = '123'
```

The command above retrieves all the data from the table where the data in the id column is equal to '123'. This approach of storing and retrieving data in a database proves beneficial during the

implementation stage as it allows for clear and intuitive ways of carrying out a task that is needed.

2.2.1 Java Database Connectivity

The Java Database Connectivity (JDBC) is an application program interface that allows java programs to connect to database. The JDBC contains the `java.sql` package which supports SQL statements. The `Statement` interface allows static SQL statement to be executed during runtime.

```
Statement stmt = null;
stmt = conn.createStatement( );
```

The `PreparedStatment` interface provides a means to execute dynamic SQL queries at runtime. This provides more flexibility for specific querying of a database.

```
String SQL = "SELECT * FROM users WHERE name = 'John'";
PreparedStatement statment = conn.prepareStatement(SQL);
```

The `ResultSet` object is a table of data retrieved from a database. This collection of data is made by execution of an SQL statement. The data contained in the `ResultSet` object is access by iterating through the given table in a while loop. (See code below):

```
ResultSet resultSet = stmt.executeQuery();

//Loop through the result set
while (resultSet.next())
{
    String userID = res.getString("id");
}
```

Considering that the JDBC supports programmatical querying of a database, it is an fitting to incorporate into the system as it satisfied the proposed system requirement of storing data.

2.3 User Interface Specifications

In terms of a graphical user interface, from the proposed system suggested it is ideal to provide a clean and intuitive user interface to allow ease of use to a user. The minimal use of buttons and lack of visual noise proves beneficial during use of a program as it doesn't confuse a user on how the program should work. Furthermore, the Java swing libraries shall be utilised as they offer fast and efficient GUI development. The GUI should have a section for a user to login, register, show registered users, make a cake and view cakes that the user has already made. This is achieved by providing a navigation bar to change the view accordingly. Additionally, the user interface should use multiple layout manager interfaces to achieve a smooth and clean user interface.

3

System Design

3.1 Graphical User Interface

The design of the graphical user interface plays a vital role in the achieving the goals for this program. A simple and easy to use GUI should be implemented in order to help users to instinctively use the program without experience or confusion. To achieve this, many of the graphical views share the same basic structure. Additionally, Every view uses a consistant nagivation bar to provide easy navigation throughout the program.

3.1.1 Home

The home view consists of a basic 3 panel structure that provides some information about the shop. On the left and bottom right hand, panels are used to give brief information about the shop. A section is also allocated for a picture slide show, which displays a series of cakes continously. The home view is unique in its design as it does not share a common structure to any other view.

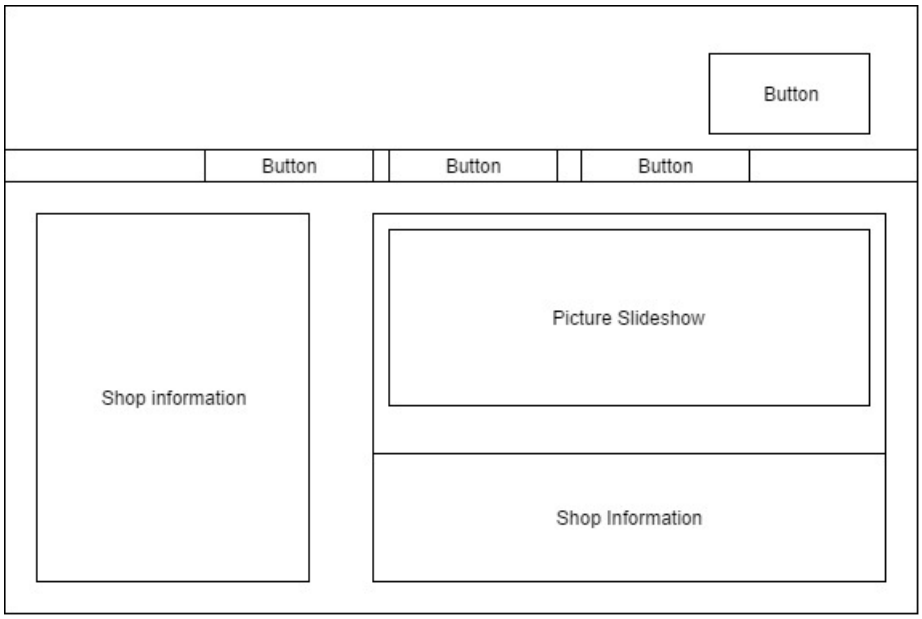

Figure 3.1: Home Panel

3.1.2 Login

The login page consists of very simple interface design, with two input fields for a user to enter their user ID and password.

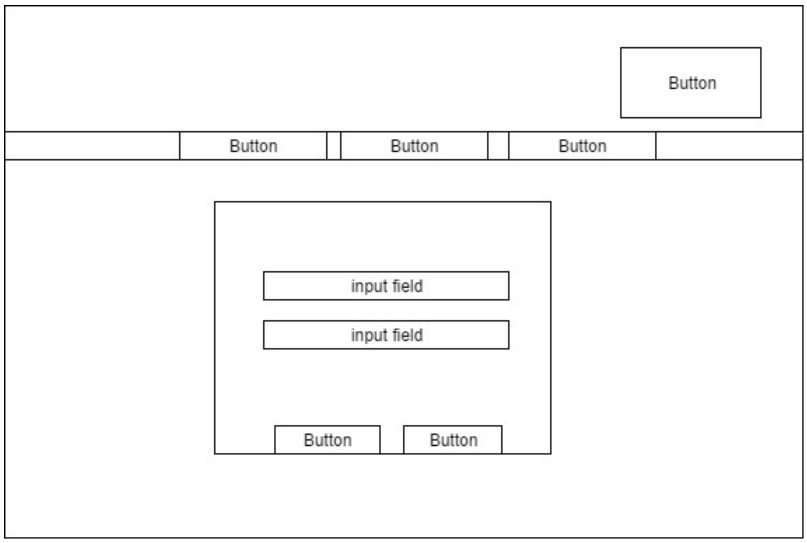

Figure 3.2: Login Panel

3.1.3 Register

The register page shares a similar structure to the login page, in terms of a simple input field section in the middle of the view for minimal visual intake. The input fields should take a users name, a chosen password and a county which they are from.

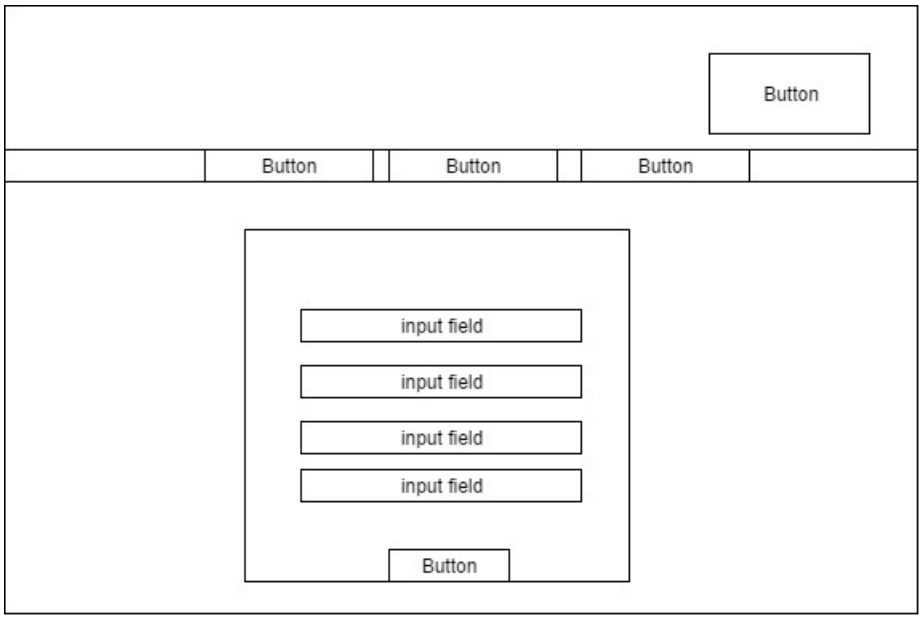

Figure 3.3: Register Panel

3.1.4 Make A Cake

This panel possesses a slightly more complex design choice. Both sections for user interaction and information displayal are kept seperate to enforce simplicity. On the left side, a section is devoted to allowing users to decide the recipe for their cake. Dropdown bars are given to provide a means to the user choose their cake type, shape, toppings and size. During interaction with this section, the right section (for information) is updated accordingly. Once a user has finished making a cake, buttons are provided to allow a user to save their cake recipe. Or, should a user wish to restart, a button allows a the user reset the form and start again.

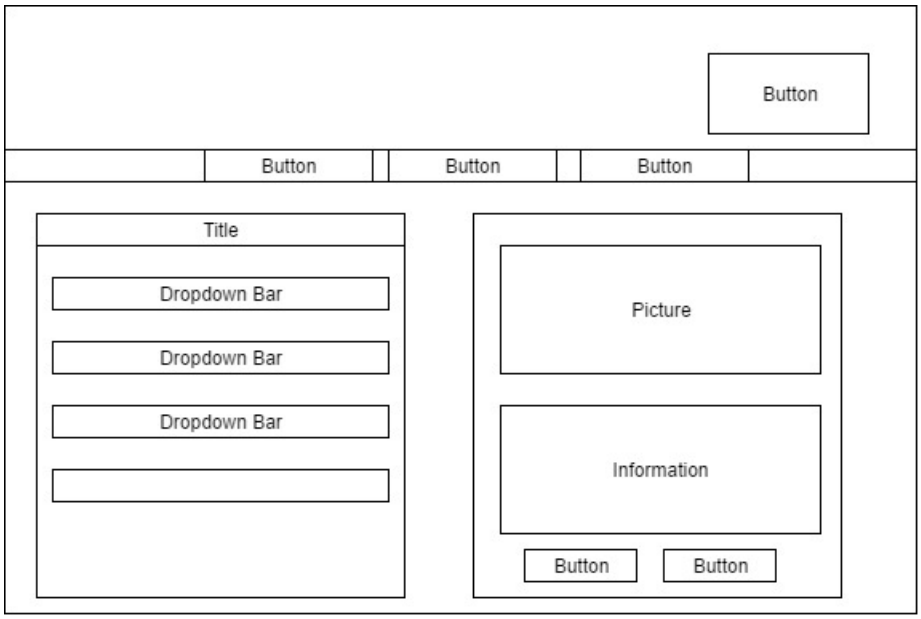

Figure 3.4: Make A Cake Panel

3.1.5 My Cakes

This panel shares similarities with the 'Make A Cake' panel as it segregates user interaction from information. Although, this section is only viewable if a user is logged into the system. The left section displays a list of cakes a user has previously saved and displays them in order. When a user clicks on the cake name, the right panel is updated with the cake information and the recipe is displayed.

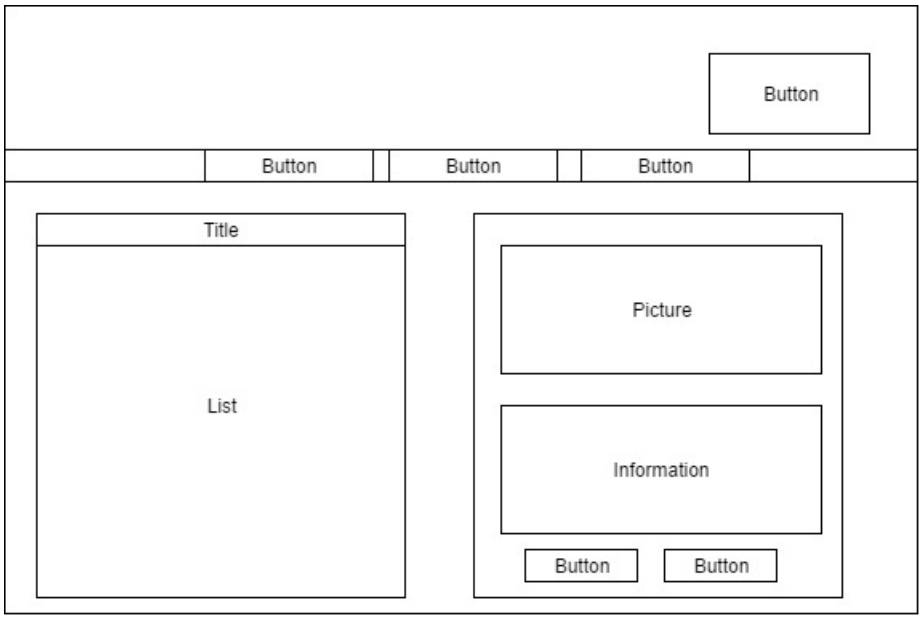

Figure 3.5: My Cakes Panel

3.1.6 View Users

In addition to the 'My Cakes' and 'Make A Cake' panels, this section shares the same information and interaction seperation structure. The view consists of a left panel with three buttons, each used for updating the list of users groups by their county. In which case, if a user clicked on the `Dublin` button, a list of user's names is displayed. If a user is to click on one of them user's name, their information is displayed in the informational panel.

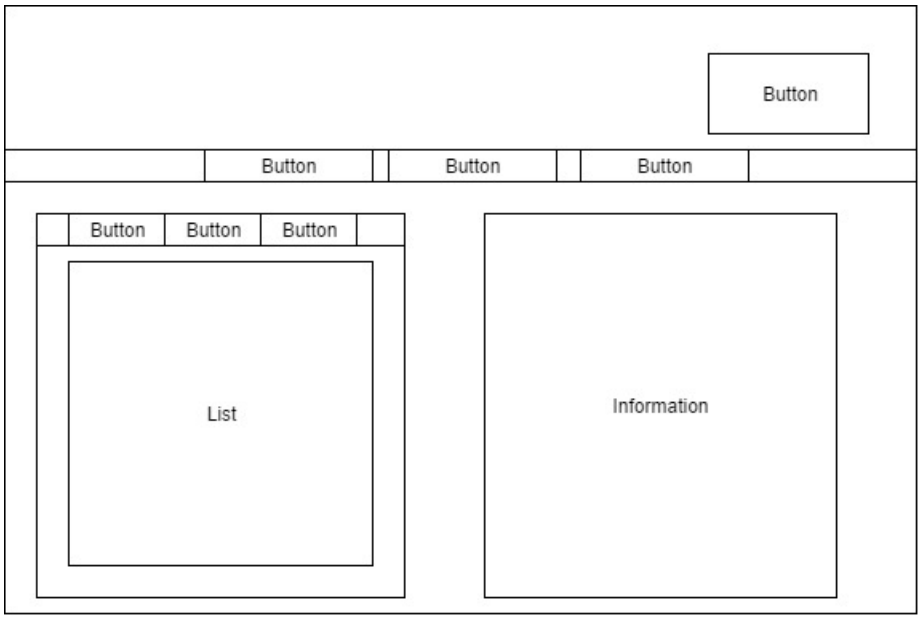

Figure 3.6: View Panel

3.2 Design Patterns

Design patterns are desribed as reusable solution to a programming problem that is commonly encountered or a template of how to solve a specific problem (Bautista, 2017). Design patterns are categorized into three different groups: Creation, Structural and Behavioral. Creation patterns provide mechanisms for instantiation which, in turn, makes for easier object creation. Structural patterns dwell upon relationships between seperate enitites and allows work between those enities simpler and more efficient. Lastly, Behavioral patterns are used for the communication across entities. Many design patterns have been considered during the process of planning and design for this project. Although, only four are utilised in the case of this program as they meet the requirements for what the system needs.

3.2.1 The Singleton Pattern

The singleton design pattern is a creational pattern that limits the creation objects as it only allows one instance of an object to be created at any given time. This can be achieved by creating a static variable (which can only be instantiated once) or to use exception handling. In the case of a user login in the system proposed, a singleton is essential as it will only allow one user to be logged into the system at any given time.

3.2.2 The Builder Pattern

The builder pattern is a creational pattern which aims to seperate the construction of complex objects from the representation. A builder class constructs several objects into one and returns a final product. The builder pattern also improves modularity of code and hides details. In this case, the builder pattern is suitable because cake objects made up of multiple other objects can be instantiated efficiently at runtime when needed.

3.2.3 The Composite Pattern

The composite pattern is ideal for cases in which objects needed to be associated into groups. The composite pattern composes objects in a hierarchial tree order: The composite object is knows as a group, and an object within that group is known as a leaf. In regards to the program specifications, the composite pattern is applicable as we can have user objects to be displayed, and also organise those user objects into groups of counties accordingly, depending on the user data obtained from the database.

3.2.4 The Abstract Factory Pattern

The abstract factory pattern is a creational design pattern that can return a factory object that can return several groups of classes. This is achieved by creating an interface for creating related objects without specifying the concrete class. This is also beneficial as it encapsulates data from the client. As the view of the graphical user interface needs to change, the abstract factory pattern can be used to update the view by returning a new panel object.

4

Implementation

4.1 Updating GUI Using The Abstract Factory Pattern

In order to update the graphical user interface within the program the abstract factory pattern is used. An abstract factory class called `PanelFactory` is made to act as an interface for the panels that will be used. This class has one method called get panel. A centre panel of type `FactoryPanel` is initialized and added to the JFrame. Initially, it is set to the `HomePanel` as it is the first page that the user will see when they start the program.

```
public void setCenterPanel(){
    panel = new HomePanel();
    centrePanel = panel.getPanel();
    mainPanel.add(centrePanel, BorderLayout.CENTER);
}
```

The navigation bar buttons and the login button have customer action listener classes which update the gui accordingly by disposing of the current centre panel, and creating a suitable panel object to replace and is then set to the JFrame. This serves as a much more efficient approach to updating the graphical user interface rather than using the `dispose()` method as it does not use as much memory. For example: The code below displays an action listener for the "Make A Cake" button.

```
public class MakeACakeListener implements ActionListener{

    @Override
    public void actionPerformed(ActionEvent e) {
```

```
            //Hide the current panel
            centrePanel.setVisible(false);

            //Create new panel object
            panel = new MakeACakePanel();

            //get the panel and set it as the current panel
            centrePanel = panel.getPanel();

            //Add to the view
            mainPanel.add(centrePanel, BorderLayout.CENTER);
        }
    }
```

4.2 Login Using The Singleton Pattern

To implement a user login, The singleton pattern is essential to enforce the rule of one user to be logged in at any given time, as it only allows one instance of `LoginSingleton` to be created. Should a user enter their user ID and password, the information is then compared to the data stored within the database. If the user ID exist and the password matches to that of the one assigned with their ID, then the user is granted access to their account.

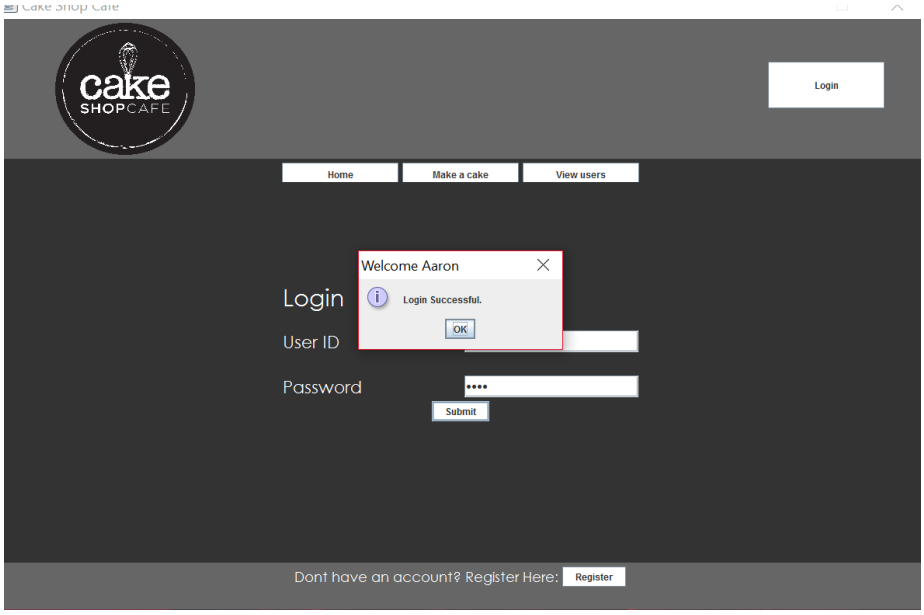

Figure 4.1: Login Successful

When the user is granted access, a instance of that user login is created with then parameters of

that user.

```
//Set the instance of user
LoginSingleton.getLoginInstance(userID, userName);
```

The `LoginSingleton` class contains a constructor that sets the ID and username of that which is passed to the current instance. Additionally, the constructor is declared as static, therefore only allowing one instance of the `LoginSingleton` to be created.

```
public static LoginSingleton getLoginInstance(String id, String name){
    //Set the ID of the user
    LoginSingleton.id = id;
    LoginSingleton.name = name;

    //Check if an instance has already been created
    if(loginInstance == null){
            loginInstance = new LoginSingleton();
    }
    return loginInstance;
}
```

Ideally, the login in page should be hidden succeeding a user login. In this case the login functionality is still available in order to display the affectiveness of the `LoginSingleton`. The screen shot below exhibits the case in which a user tries to log into the system if a login instance has already been created.

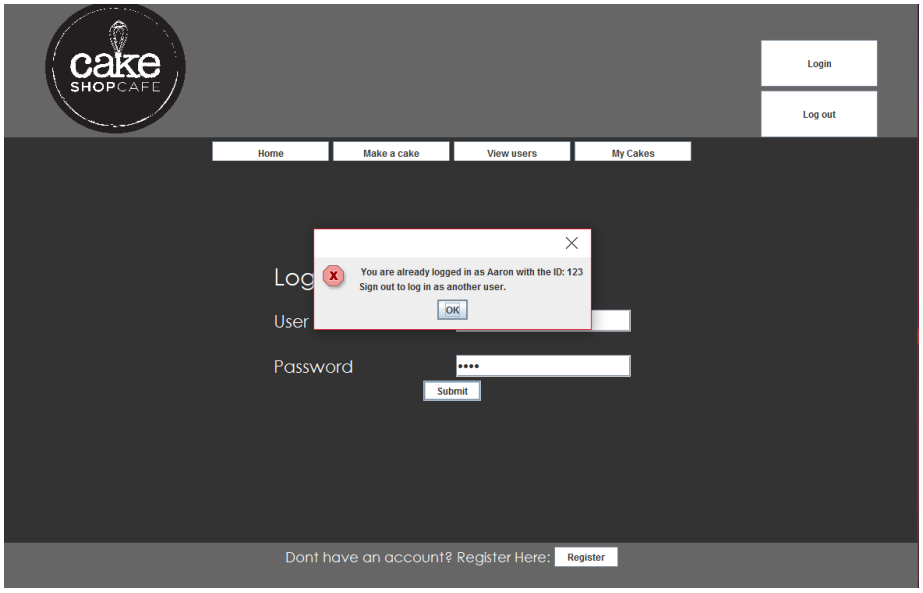

Figure 4.2: Refused Login

This is achieved by checking if there is a login ID already created. If the ID has a value then the error

message is shown and the user is not allowed to log in. See the code below of the `SubmitListener` Action Listener:

```
//Check for a value
if(LoginSingleton.getId() != null){

    ///display error message
    JOptionPane.showMessageDialog(null," You are already logged in as " +
    LoginSingleton.getName() +" with the ID: " + LoginSingleton.getId() +
    "\nSign out to log in as another user.", "", JOptionPane.ERROR_MESSAGE);
}
```

4.3 Making A Cake

In order for a user to make a cake, the user is provided with a form in which they can choose multiple different attributes. The options for cake type, size, shape and toppings are provided in the form. The user must complete the form in its entirety to save their cake and save it to the database.

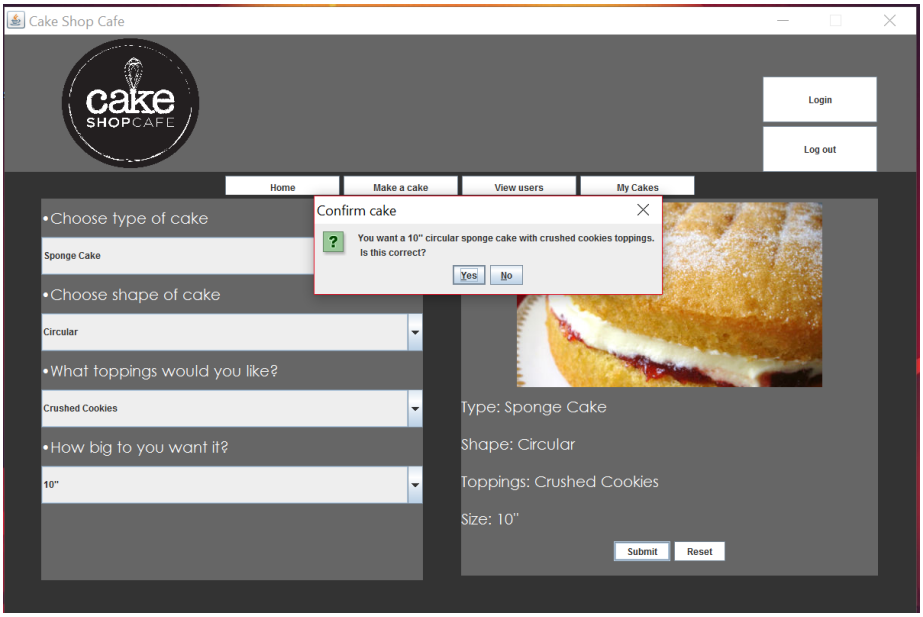

Figure 4.3: Confirm Cake

Once the user confirms the cake they want to make, the database is then updated by inserting the new cake data.

```
//Create connection to database
Connection dbConnection = DriverManager.getConnection(url, user, password);
```

```
//create a statement
Statement myStatement = dbConnection.createStatement();

//SQL statement for inserting data
String sqlStatement =  "INSERT INTO cake"
        + "(id, type, size, toppings, shape)"
        + "VALUES ('" + LoginSingleton.getId() + "','" +
                                            type +"','"+
                                            size + "','" +
                                            topping + "','" +
                                            shape + "')";
//Execute the statement
myStatement.executeUpdate(sqlStatement);
```

As seen in the code above, the SQL statement inserts the ID of the `LoginSingleton` instance (the user currently logged on), the cake type, size of the cake, the toppings on the cake and the shape of the cake.

4.4 Viewing Cakes Using The Builder Pattern

After a user is logged into their account, the they are then allowed to view the cakes that they have previously made. To achieve this, a query is sent to the database and retrieves all the data from the cake table. After the SQL data is obtained, a new `Cake` object is created using the builder pattern and added to an array of cake objects.

```
int i = 0;
while (res.next())
{
    //variables are set to compare with the user's input
    type = res.getString("type");
    shape = res.getString("shape");
    size = res.getString("size");
    toppings = res.getString("toppings");

    //Creat a new cake object using the
    //builder pattern taken from the SQL data.
    Cake cake = new Cake.Builder().type(type)
                                  .shape(shape)
                                  .size(size)
                                  .toppings(toppings)
                                  .build();
```

```
        //Add cake to the array
        cakesArray[i] = cake;

        i++;
        ...
```

The cake names are also added to a `JList` to allow users to view the list of cakes previously made. Once a user clicks on a list item,the index of the list selected is is used to retrieve the corresponding index of the `cakesArray`. The user interface is then updated by retrieving the types, shape, size and toppings of the cake object. As seen in the code below, the labels for displaying information are updated with the correct data for that specific cake.

```
public void valueChanged(ListSelectionEvent e) {
    //Check the index of the list selected
    JList<String> list = (JList<String>) e.getSource();
    int listSelected = list.getSelectedIndex();

    //Set the labels to the attributes in the cake object.
    //taken from the Cakes array
    Cake thisCake = cakesArray[listSelected];
    pickedTypeLabel.setText("Type: " + thisCake.getType());
    pickedShapeLabel.setText("Shape: "  + thisCake.getShape());
    pickedToppingsLabel.setText("Toppings: " + thisCake.getToppings());
    pickedSizeLabel.setText("Size: " + thisCake.getSize());

    ...
```

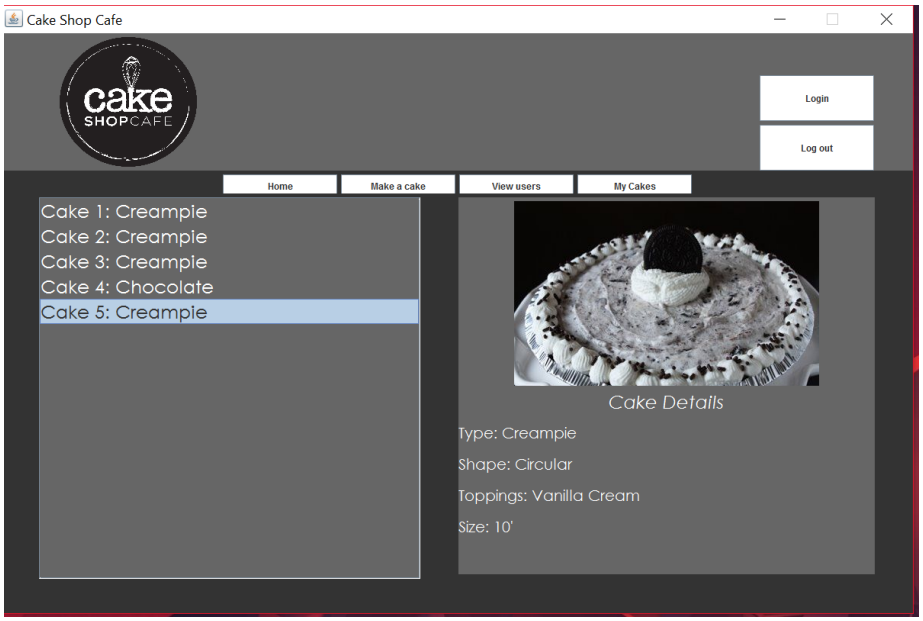

Figure 4.4: Viewing cakes made by a user

4.5 Viewing Registered Users With The Composite pattern

The composite pattern is used in the "View Users" page in order to aggregate collections of `User` objects into groups. In this case, counties are use as groups. During registration the user has the option of registering from Cork, Dublin and Galway. Although it is not realistic in a real world application to only have three counties, for this case only three are used to exhibit the use of the composite pattern. When viewing the users, the users are grouped by the county that they have registered in. See the screen capture below that displays the users that have been registered in

Galway. 

As seen above, the user with the username "Daire" is selected, and the users information is displayed on the right panel. This is achieved by querying the database for data by by county. The result set is traversed through and the data is extracted from each row. From this data, a new `User` object is created and stored in a User object array.

```
ResultSet dublinRes = getDublin.executeQuery();
int i = 0;
while(dublinRes.next()){

    //Extract data
    userName = dublinRes.getString("name");
    userCounty = dublinRes.getString("county");
    id = dublinRes.getString("id");
    //Add to user array
    usersArrayDublin[i] = new User(userName, userCounty, id);
    i++;
}
```

The users are also added to a `UserGroup` according to their county. When the a user's name is clicked in the list, a `UserListener` class is invoked to display the user object's attributes. The list index which is clicked is used to create a new user from the array of user objects.

```
public class UserListener implements ListSelectionListener{
    @Override
    public void valueChanged(ListSelectionEvent e) {

        JList<String> list = (JList<String>) e.getSource();
```

```
        int listSelected = list.getSelectedIndex();

        //Checks the current list is of the right county
        if(currentUsers.equals("Dublin")){

            //Create a new user Object of the User object in the list index
            User thisUser = usersArrayDublin[listSelected];

            //Update labels
            userNameLabel.setText("Name: " + thisUser.getUserName());
            userCountyLabel.setText("County: "  + thisUser.getCountyName());
            userIDLabel.setText("User ID: " + thisUser.getUserID());

        ...
}
```

5

Conclusions and Further Work

5.1 Conclusion

To conclude, the project was successful in implementation. The system utilises four object oriented design patterns as stated in the specification. The Singleton pattern enforced the rule of only one user to be allowed log into the system at any one time. An Abstract Factory was used for creating new panel objects and updating the graphical user interface when needed. Cake objects where dynamically created using the Builder pattern, specific to each user. A Composite pattern aided in the grouping of users for each county they have registered for and lastly, database implementation was successful and used suitably to apply persistent storage of data. Furthermore, the program was provided with a smooth and easy to user graphical user interface that allows ease of use to a user. With the utilisation of inuitive design choices a user may be able to use the program with instruction. Therefore, in conclusion, the program succeeded and met the requirements for the deliverables needed.

5.2 Furtherwork

Should the chance of designing and creating this application arose again, it would be highly considered to apply more detailed GUI components to the view. A user could choose upload a display photo when their account when registering and be displayed in the "View Users" page. Furthermore, it is desirable that more counties for regristration to given. As it is unrealistic in a real

world application for a program to only allow three counties, it might be more suitable to allow all counties or even countries choices to be implemented.

References

Bautista, Nikko. “A Beginner’S Guide To Design Patterns”. Code Envato Tuts+. N.p., 2017. Web. 22 Mar. 2017.

Code Github repository: https://github.com/AaronWard/DesignPatternAssignment

Additionally, this report was written in markdown using pandoc. See the following repository link for report implementation: https://github.com/AaronWard/Assignment2_Design_Pattern_Report_Pandoc.git