Long Methods

Throughout many classes in the view package, the main methods were all around 100+ lines long. Now we have broken up the methods into smaller methods, making it easier to read and easier to trace any errors that could arise.

Primitive Obsession

Before:

```
 catch(Exception e){
        String message = e.getMessage().split("\n", 2)[0].toLowerCase().replaceAll(" ", "");
        if(message.equals("communicationslinkfailure")) {
                new ErrorPrompt("<html>No Local Server found. Please start a<br>local MySQL
Server to run this app.</html>", true).setVisible(true);;
                        }
                }
```

After:

```
catch(SQLException e){
        new ErrorPrompt("<html>No Local Server found. Please start a<br>local MySQL Server to
run this app.</html>", true).setVisible(true);
} catch (Exception e) {
        new ErrorPrompt("<html>Unknown error has occurred</html>", true).setVisible(true);;
 }
```

In src/model/ManagerMain() in the get connection method the old way would only catch a general java Exception. A secondary check of the error string message was then used to determine whether the error would display or not. Now it's catching an SQLExecption error and the string check is removed.

If-Statements

In our DbInventory.java class there was a long if-else tree that would check the ingredient name from the SQL query, and then construct a new ingredient object to be used in the sandwich.

Original code in the getIngredient() method:

```
if (ingredientName.equals("Beef")) {
                return new Beef(ingredientName, price, "Meat");
        } else if (ingredientName.equals("American")) {
                return new AmericanCheese(ingredientName, price, "Cheese");
        } else if (ingredientName.equals("Bread")) {
                return new Bread(ingredientName, price, "Bread");
        } else if (ingredientName.equals("Cheddar")) {
                return new Cheddar(ingredientName, price, "Cheese");
        } else if (ingredientName.equals("Chicken")) {
                return new Chicken(ingredientName, price, "Meat");
        } else if (ingredientName.equals("Ketchup")) {
                return new Ketchup(ingredientName, price, "Sauce");
        } else if (ingredientName.equals("Lettuce")) {
                return new Lettuce(ingredientName, price, "Vegetable");
        }..... // it keeps going for every ingredient class
```

We removed the long if-else statement and replaced it with a map. When the SQL database returns the result, we input the string into a map as a key and it'll construct the ingredient to return.

Changed code:

```
Class<? extends Ingredient> ingredientClass = INGREDIENT_MAP.get(ingredientName);
```

Followed by a check to see if the ingredient is null.