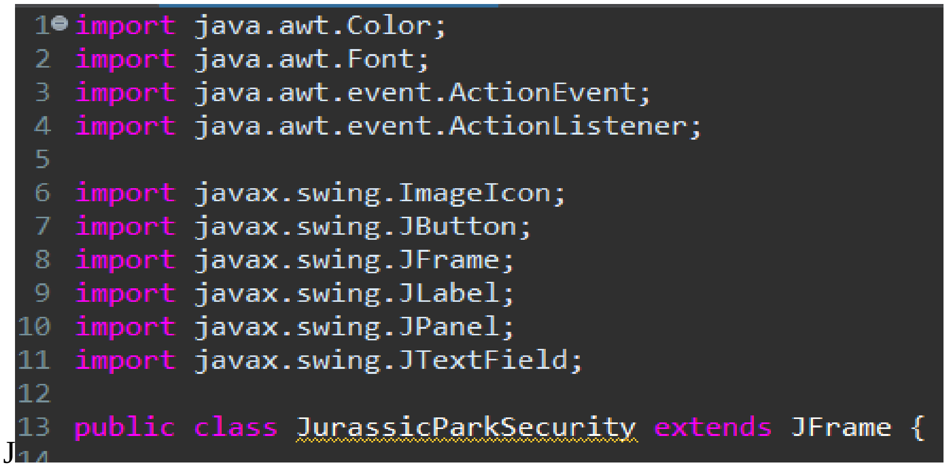
Criterion C: Development

Jurassic Park Security is a program designed for a specific need and can only be used for that. Mr. Hammond has asked for software replicating his park, so it is easy for the employees to understand how the system works. Multiple techniques used to create the program are listed below.

Development Techniques
Imported Packages Used
Google Sheets as a User Database
JSON Array and JSON Objects
ArrayLists
Object Oriented Programming
Parsing JSON Data
Registering Users inside the Database
Resetting the Password Locally and Google Sheets
LinkedHashMap
Java Swing GUI
Nested Classes
Encapsulation
Nested If Statements
Searching and Testing

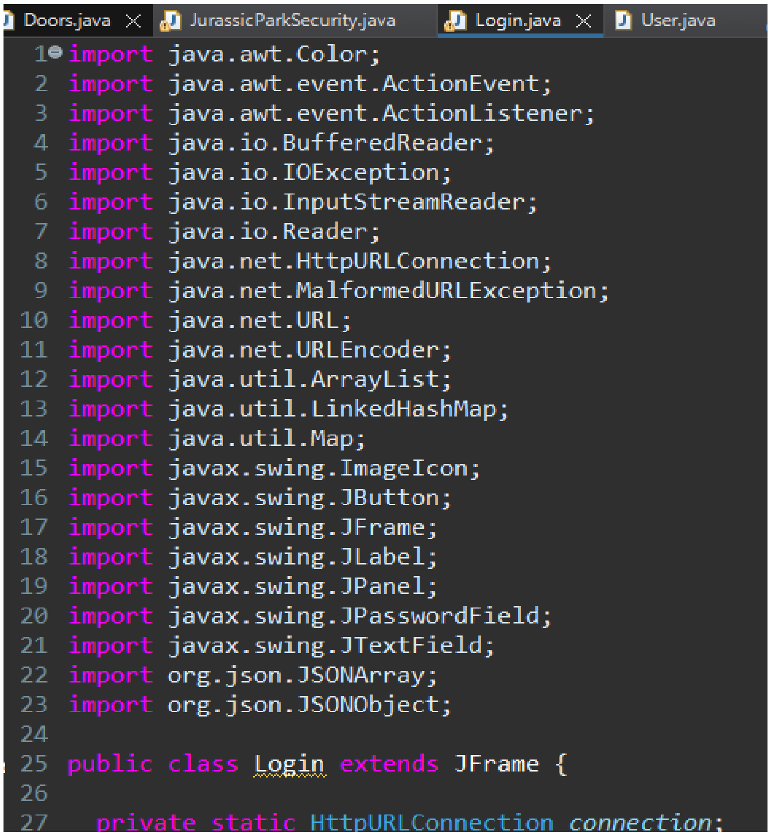
Jurassic Park Class imports



```
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class JurassicParkSecurity extends JFrame {
```

J

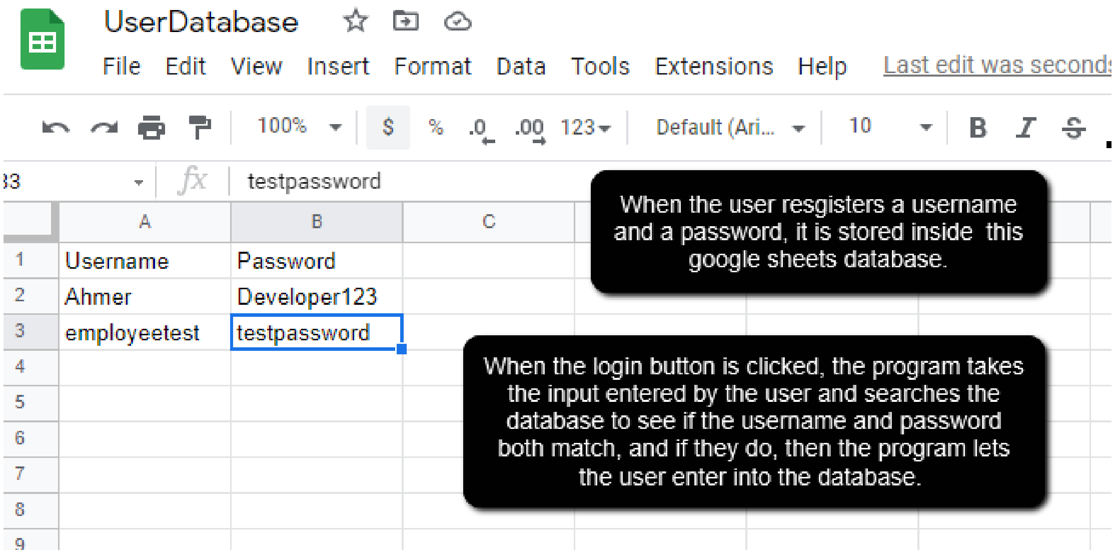
: Login Class Imports



```
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import org.json.JSONArray;
import org.json.JSONObject;

public class Login extends JFrame {

  private static HttpURLConnection connection;
```

Using Google Sheets as a database to store the users:

The program required persistent storage to store the users, therefore the need for a database occurred. I decided to go with google sheets because it is a cloud-based web application that had a wide range of compatibility making it possible for a lot of devices to be used.



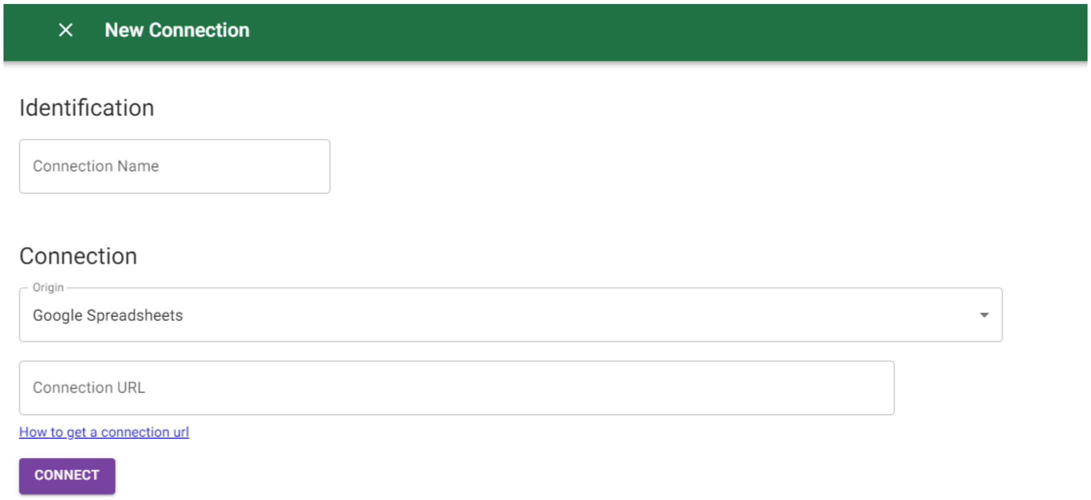
B3 testpassword

	A	B	C
1	Username	Password	
2	Ahmer	Developer123	
3	employeetest	testpassword	
4			
5			
6			
7			
8			
9			

When the user resgisters a username and a password, it is stored inside this google sheets database.

When the login button is clicked, the program takes the input entered by the user and searches the database to see if the username and password both match, and if they do, then the program lets the user enter into the database.

For Java to read the database, it required an API endpoint. Therefore, I created an API endpoint using this website called sheet.best. API endpoints are how different applications can communicate with each other. In my case, the Jurassic Park Security program can communicate with the google sheets database.



Identification

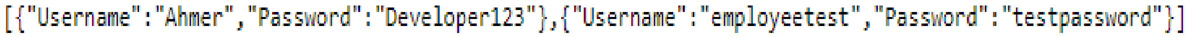
Connection Name

Connection

Origin: Google Spreadsheets

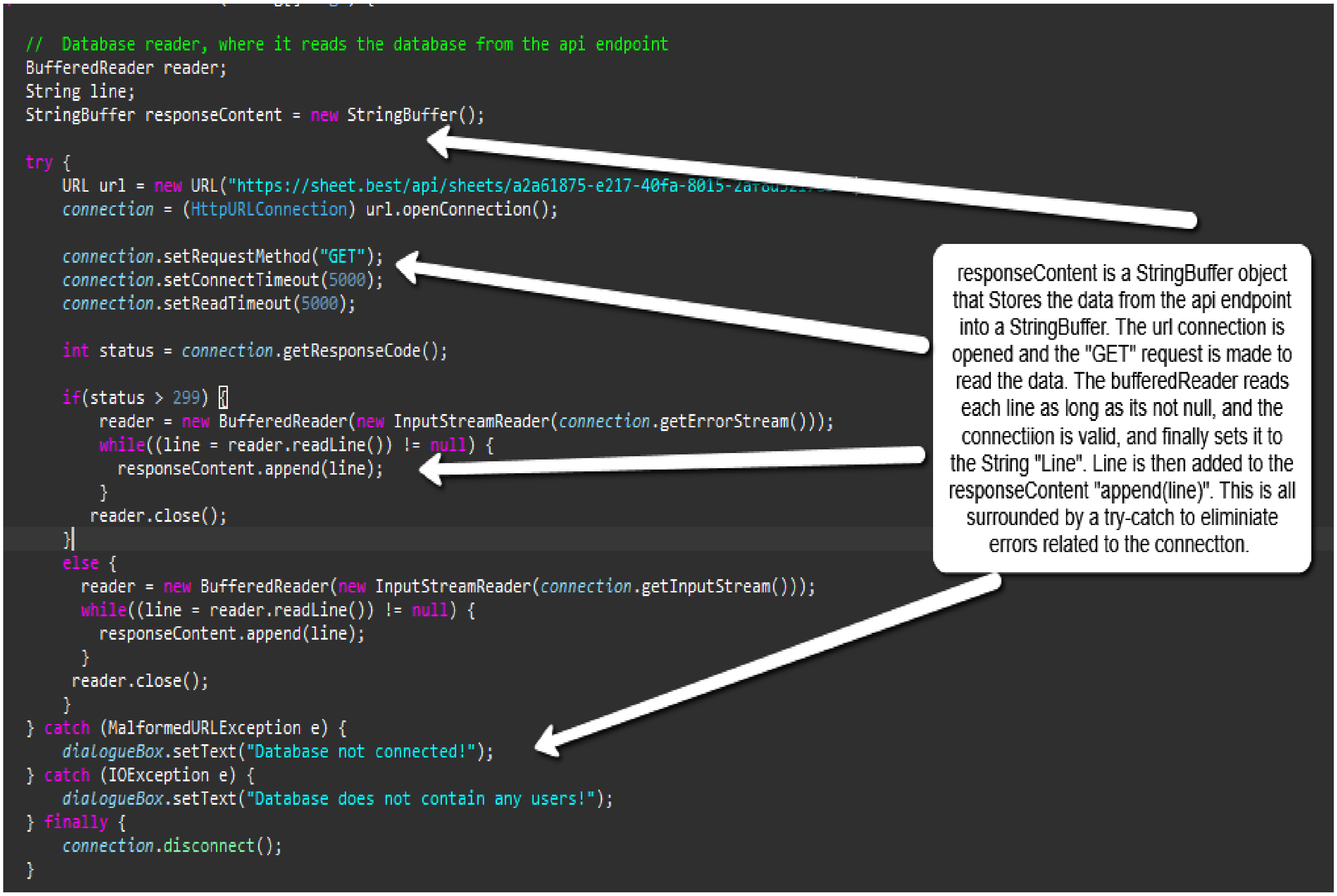
Connection URL

Sheet.best takes the URL and makes a new endpoint connection that java can read and write to using the “GET” and “POST” requests. The API endpoint, however, stored the data as a JSON Array of JSON Objects as seen below. The JSON format is used to transfer data between applications. The format is like an Array in java.



```
[{"Username":"Ahmer","Password":"Developer123"},{"Username":"employeetest","Password":"testpassword"}]
```

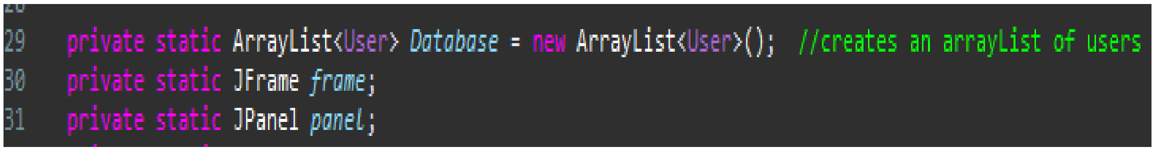
For Java to be able to read the data, I needed to open an HTTP Connection and send a “GET” request to read the data like shown below.



```
//  Database reader, where it reads the database from the api endpoint
BufferedReader reader;
String line;
StringBuffer responseContent = new StringBuffer();

try {
    URL url = new URL("https://sheet.best/api/sheets/a2a61875-e217-40fa-8015-2a
    connection = (HttpURLConnection) url.openConnection();

    connection.setRequestMethod("GET");
    connection.setConnectTimeout(5000);
    connection.setReadTimeout(5000);

    int status = connection.getResponseCode();

    if(status > 299) {
        reader = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
        while((line = reader.readLine()) != null) {
          responseContent.append(line);
        }
       reader.close();
    }
    else {
      reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
      while((line = reader.readLine()) != null) {
        responseContent.append(line);
      }
     reader.close();
    }
} catch (MalformedURLException e) {
    dialogueBox.setText("Database not connected!");
} catch (IOException e) {
    dialogueBox.setText("Database does not contain any users!");
} finally {
    connection.disconnect();
}
```

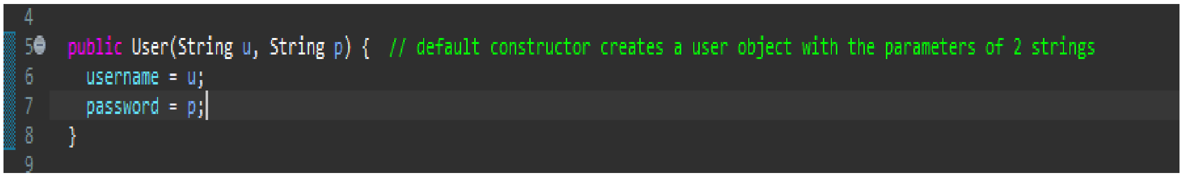
responseContent is a StringBuffer object that Stores the data from the api endpoint into a StringBuffer. The url connection is opened and the "GET" request is made to read the data. The bufferedReader reads each line as long as its not null, and the connectiion is valid, and finally sets it to the String "Line". Line is then added to the responseContent "append(line)". This is all surrounded by a try-catch to eliminiate errors related to the connectton.

After reading the google sheets from the API endpoint, I needed to store it in a local database so I can use it for my program and run algorithms like searching and comparing to see if they match the entered input by the user. So, I decided to use an ArrayList of Users. I chose an ArrayList because it is scalable and does not have a set length like arrays.



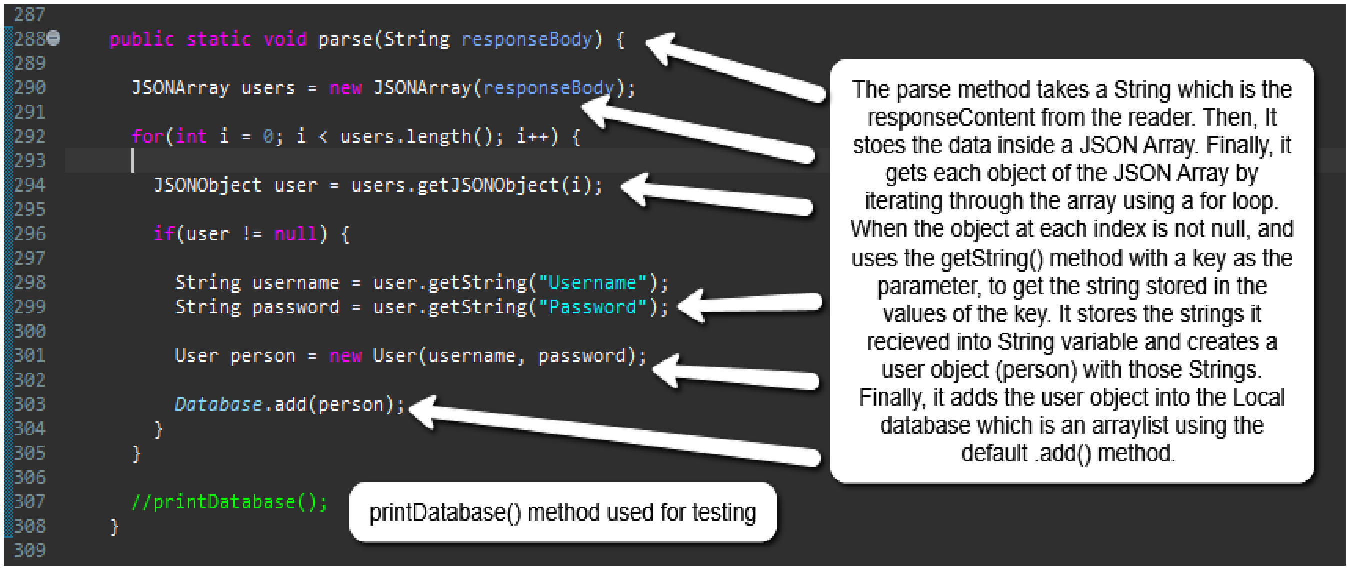
```
private static ArrayList<User> Database = new ArrayList<User>();  //creates an arrayList of users
private static JFrame frame;
private static JPanel panel;
```

The data type of the ArrayList is a custom-made user object. The user can be created using the default constructor User () which takes 2 strings and parameters which are username and password. The constructor is part of the User class which is some more object-oriented programming.



```
public User(String u, String p) {  // default constructor creates a user object with the parameters of 2 strings
  username = u;
  password = p;
}
```

Before storing the data locally, it needs to be parsed because it is a String that does not give access to the data encoded within. Therefore, I created a parse method to find the specific data and make a user object out of that data. Finally, I added the user object to the array list which is the local database.

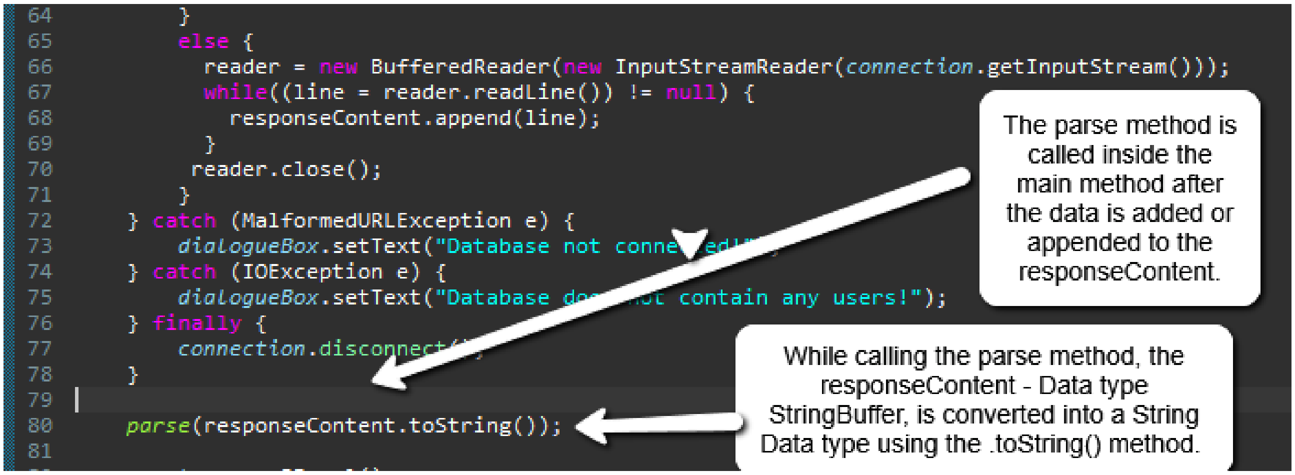


```
public static void parse(String responseBody) {

  JSONArray users = new JSONArray(responseBody);

  for(int i = 0; i < users.length(); i++) {

    JSONObject user = users.getJSONObject(i);

    if(user != null) {

      String username = user.getString("Username");
      String password = user.getString("Password");

      User person = new User(username, password);

      Database.add(person);
    }
  }

  //printDatabase();
}
```

The parse method takes a String which is the responseContent from the reader. Then, It stoes the data inside a JSON Array. Finally, it gets each object of the JSON Array by iterating through the array using a for loop. When the object at each index is not null, and uses the getString() method with a key as the parameter, to get the string stored in the values of the key. It stores the strings it recieved into String variable and creates a user object (person) with those Strings. Finally, it adds the user object into the Local database which is an arraylist using the default .add() method.

printDatabase() method used for testing

The parse method is called inside the main method of the Login Class.



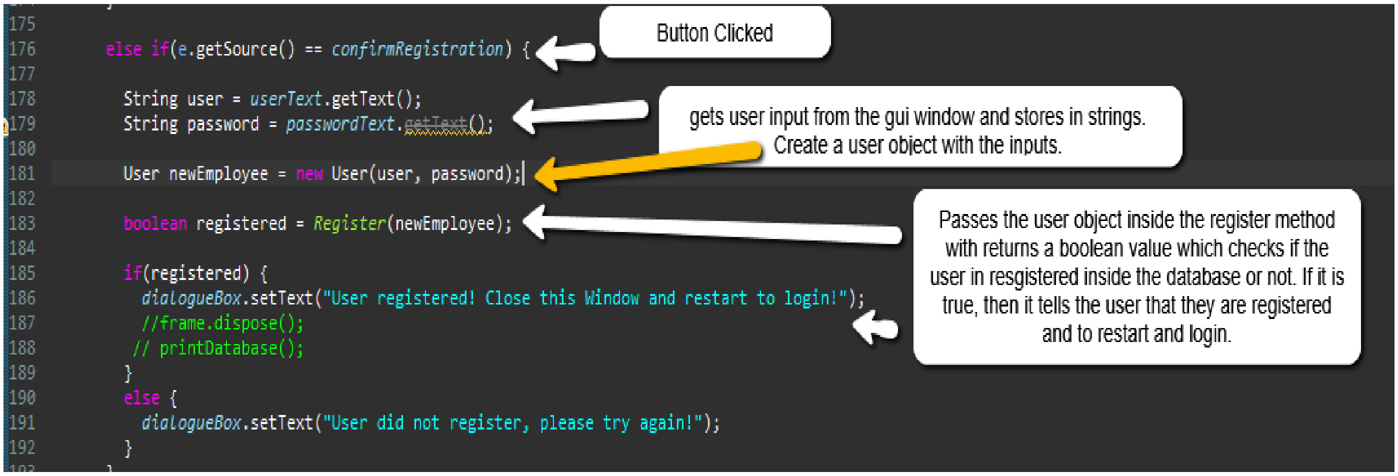
```
        }
        else {
          reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
          while((line = reader.readLine()) != null) {
            responseContent.append(line);
          }
         reader.close();
        }
    } catch (MalformedURLException e) {
        dialogueBox.setText("Database not connected!");
    } catch (IOException e) {
        dialogueBox.setText("Database does not contain any users!");
    } finally {
        connection.disconnect();
    }

    parse(responseContent.toString());

```

The parse method is called inside the main method after the data is added or appended to the responseContent.

While calling the parse method, the responseContent - Data type StringBuffer, is converted into a String Data type using the .toString() method.

Registering Users into the Google Sheets Database:

To register a user in the google sheets, I created a post () method. The method takes in a user object as a parameter. It is a multi-method programming technique. Inside the Login Handler class which implements ActionListener, when the “confirm registration” button is clicked in the register GUI window, the program gets the entered username and password by the user and creates a user object with it. It passed the user object inside the register method as a parameter.



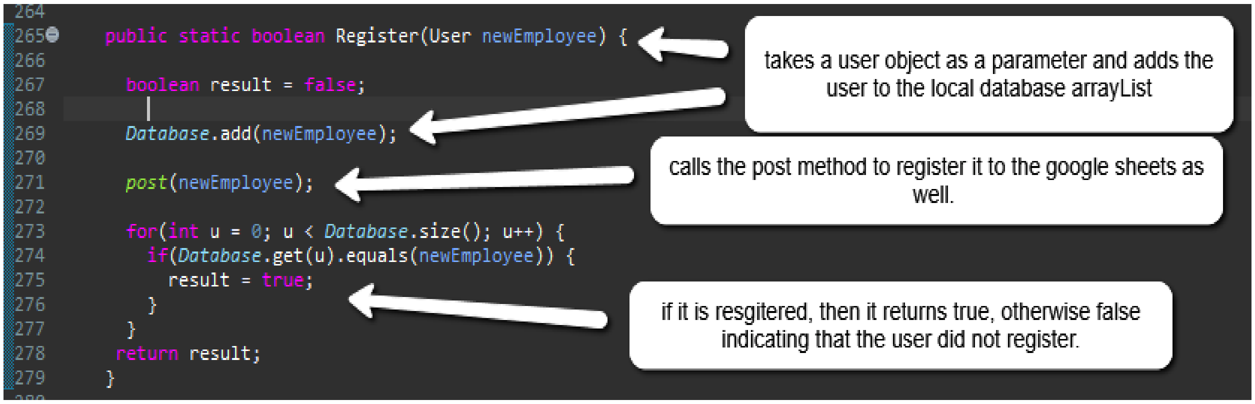
```
else if(e.getSource() == confirmRegistration) {

  String user = userText.getText();
  String password = passwordText.getText();

  User newEmployee = new User(user, password);

  boolean registered = Register(newEmployee);

  if(registered) {
    dialogueBox.setText("User registered! Close this Window and restart to login!");
    //frame.dispose();
   // printDatabase();
  }
  else {
    dialogueBox.setText("User did not register, please try again!");
  }
```

Button Clicked

gets user input from the gui window and stores in strings. Create a user object with the inputs.

Passes the user object inside the register method with returns a boolean value which checks if the user in resgistered inside the database or not. If it is true, then it tells the user that they are registered and to restart and login.

Inside the register method, the user is added to the local database which is the arrayList and then the post method is called to store the user into google sheets.



```
public static boolean Register(User newEmployee) {

  boolean result = false;

  Database.add(newEmployee);

  post(newEmployee);

  for(int u = 0; u < Database.size(); u++) {
    if(Database.get(u).equals(newEmployee)) {
      result = true;
    }
  }
 return result;
}
```

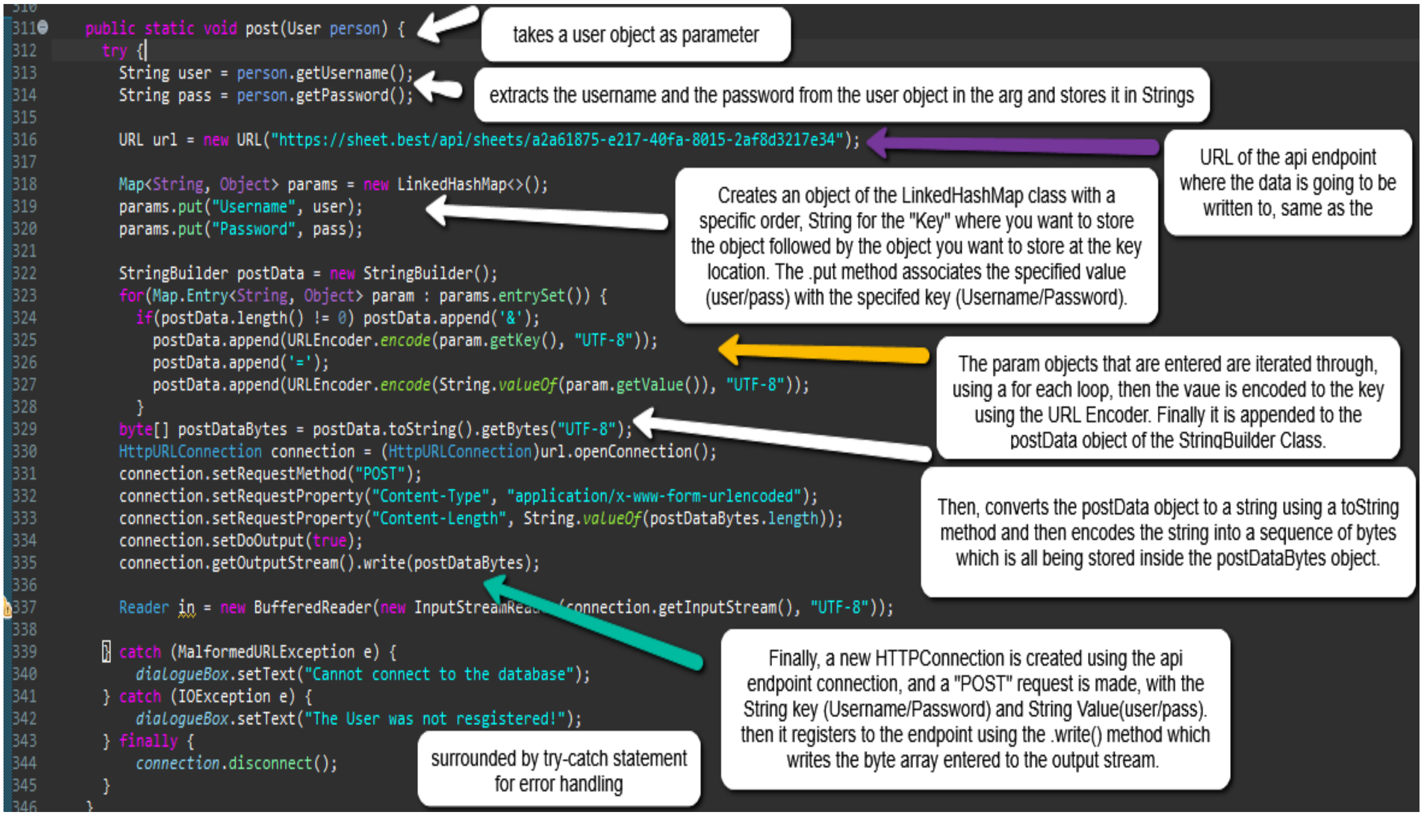
takes a user object as a parameter and adds the user to the local database arrayList

calls the post method to register it to the google sheets as well.

if it is resgitered, then it returns true, otherwise false indicating that the user did not register.

Post Method: LinkedHashMap

The LinkedHashMap class extends the HashMap class. It is used to maintain a linked list of entries in the map, preserving the order they are entered in, which is very important in this scenario because we do not want the username and the password to get mixed up.



```java
    public static void post(User person) {
      try {
        String user = person.getUsername();
        String pass = person.getPassword();

        URL url = new URL("https://sheet.best/api/sheets/a2a61875-e217-40fa-8015-2af8d3217e34");

        Map<String, Object> params = new LinkedHashMap<>();
        params.put("Username", user);
        params.put("Password", pass);

        StringBuilder postData = new StringBuilder();
        for(Map.Entry<String, Object> param : params.entrySet()) {
          if(postData.length() != 0) postData.append('&');
            postData.append(URLEncoder.encode(param.getKey(), "UTF-8"));
            postData.append('=');
            postData.append(URLEncoder.encode(String.valueOf(param.getValue()), "UTF-8"));
          }
        byte[] postDataBytes = postData.toString().getBytes("UTF-8");
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();
        connection.setRequestMethod("POST");
        connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        connection.setRequestProperty("Content-Length", String.valueOf(postDataBytes.length));
        connection.setDoOutput(true);
        connection.getOutputStream().write(postDataBytes);

        Reader in = new BufferedReader(new InputStreamReader(connection.getInputStream(), "UTF-8"));

      } catch (MalformedURLException e) {
        dialogueBox.setText("Cannot connect to the database");
      } catch (IOException e) {
        dialogueBox.setText("The User was not resgistered!");
      } finally {
        connection.disconnect();
      }
    }
```

takes a user object as parameter

extracts the username and the password from the user object in the arg and stores it in Strings

URL of the api endpoint where the data is going to be written to, same as the

Creates an object of the LinkedHashMap class with a specific order, String for the "Key" where you want to store the object followed by the object you want to store at the key location. The .put method associates the specified value (user/pass) with the specifed key (Username/Password).

The param objects that are entered are iterated through, using a for each loop, then the vaue is encoded to the key using the URL Encoder. Finally it is appended to the postData object of the StringBuilder Class.

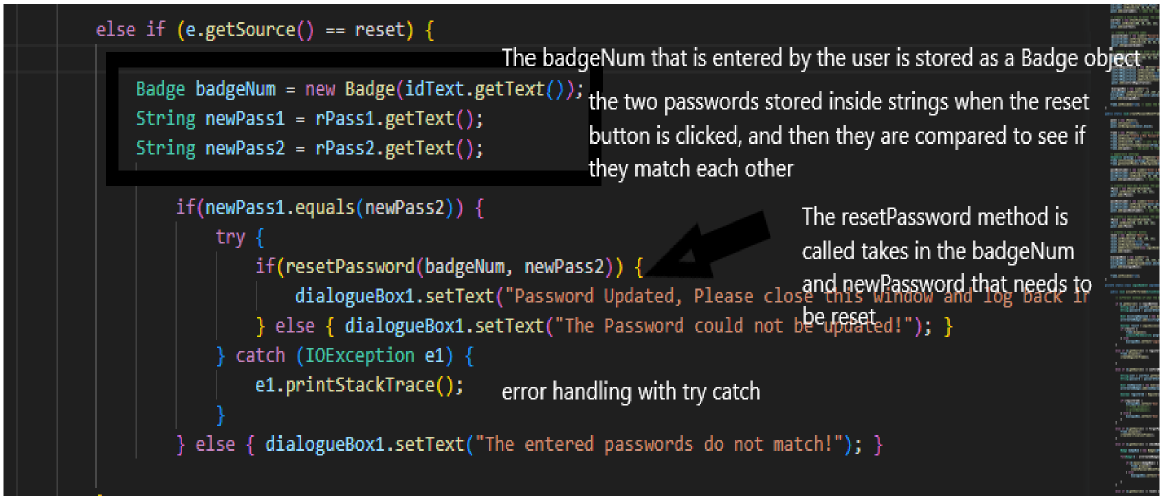
Then, converts the postData object to a string using a toString method and then encodes the string into a sequence of bytes which is all being stored inside the postDataBytes object.

Finally, a new HTTPConnection is created using the api endpoint connection, and a "POST" request is made, with the String key (Username/Password) and String Value(user/pass). then it registers to the endpoint using the .write() method which writes the byte array entered to the output stream.

surrounded by try-catch statement for error handling

Resetting Password:

Reset Button:



```
else if (e.getSource() == reset) {
    Badge badgeNum = new Badge(idText.getText());
    String newPass1 = rPass1.getText();
    String newPass2 = rPass2.getText();

        if(newPass1.equals(newPass2)) {
            try {
                if(resetPassword(badgeNum, newPass2)) {
                    dialogueBox1.setText("Password Updated, Please close this window and log back i
                } else { dialogueBox1.setText("The Password could not be updated!"); }
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        } else { dialogueBox1.setText("The entered passwords do not match!"); }
```

The badgeNum that is entered by the user is stored as a Badge object

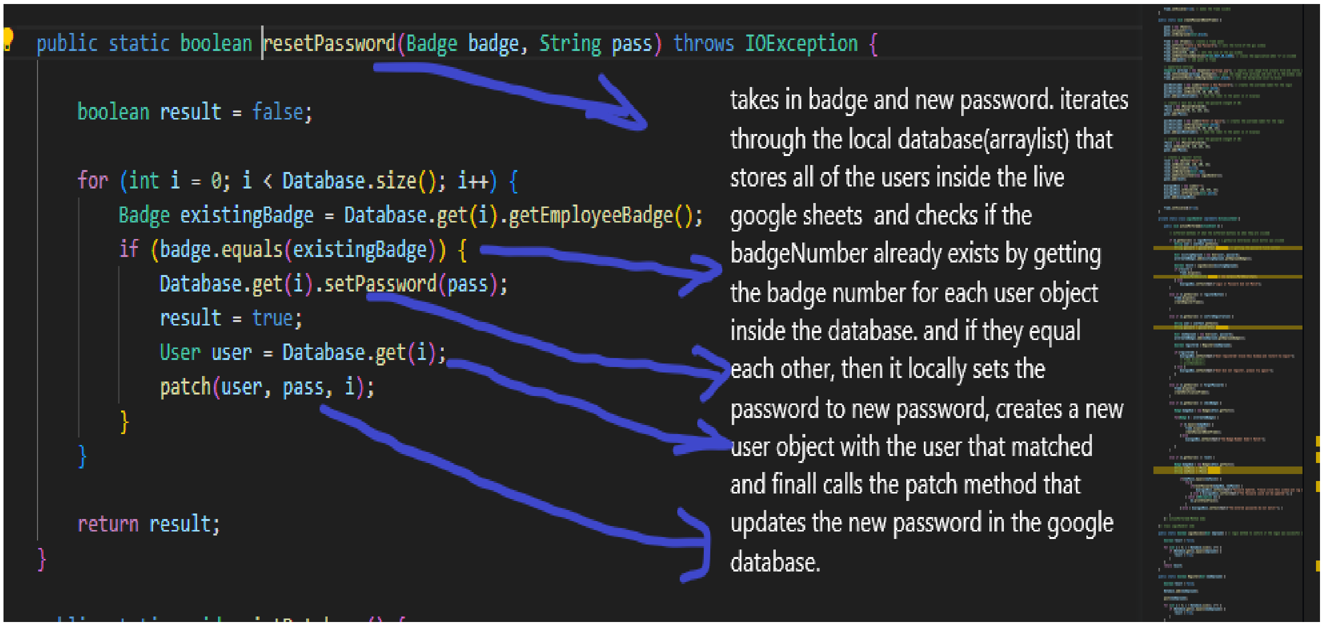
the two passwords stored inside strings when the reset button is clicked, and then they are compared to see if they match each other

The resetPassword method is called takes in the badgeNum and newPassword that needs to be reset

error handling with try catch

Inside the reset frame, when the reset button is clicked, it gets the entered in the prior frame and the 2 new passwords entered. Then it checks to see if both entered passwords are the same and if not then it sends an error message. If they are the same, then it calls the resetPassword method passing in the badge number and the new password they want to reset to.

ResetPassword method:

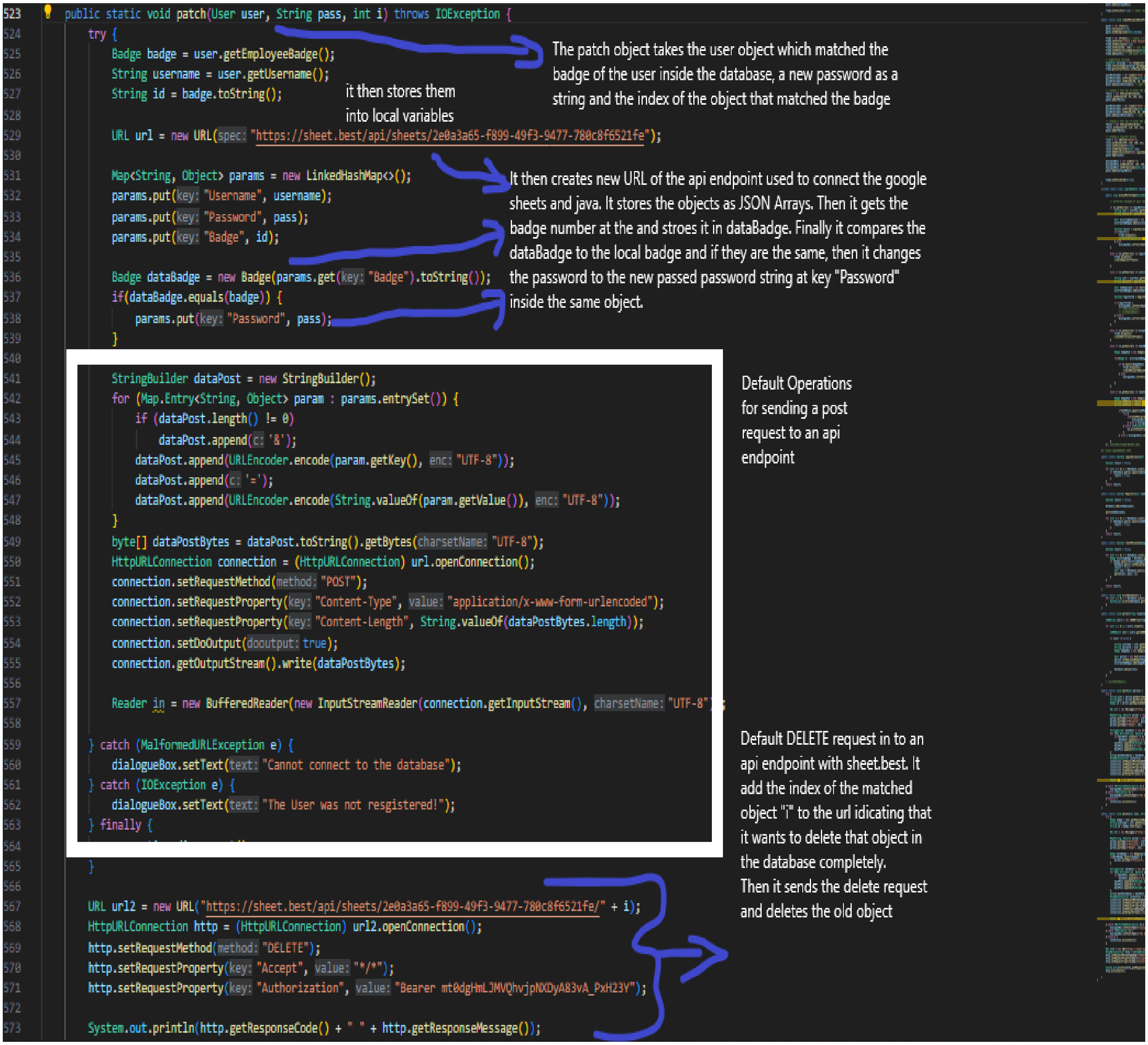


```
public static boolean resetPassword(Badge badge, String pass) throws IOException {

    boolean result = false;

    for (int i = 0; i < Database.size(); i++) {
        Badge existingBadge = Database.get(i).getEmployeeBadge();
        if (badge.equals(existingBadge)) {
            Database.get(i).setPassword(pass);
            result = true;
            User user = Database.get(i);
            patch(user, pass, i);
        }
    }

    return result;
}
```

takes in badge and new password. iterates through the local database(arraylist) that stores all of the users inside the live google sheets and checks if the badgeNumber already exists by getting the badge number for each user object inside the database. and if they equal each other, then it locally sets the password to new password, creates a new user object with the user that matched and finall calls the patch method that updates the new password in the google database.

The resetPassword method takes in the badge and the new password. First it iterates through the local database and stores the badge of each user inside a local variable called existingBadge. Then it checks if the parameter passed into the method is the same and the badge inside the user database to make sure that the user already exists in the database and has a company badge. If the badges match, it means the user already exists. Then it sets the password of the matched user object to the passed-in new Password. Finally it creates a user object with the object at matched index, and calls the patch method passing in the user object created, the new password string, and the index of where the badges match.

Patch() method:



```java
public static void patch(User user, String pass, int i) throws IOException {
    try {
        Badge badge = user.getEmployeeBadge();
        String username = user.getUsername();
        String id = badge.toString();

        URL url = new URL(spec: "https://sheet.best/api/sheets/2e0a3a65-f899-49f3-9477-780c8f6521fe");

        Map<String, Object> params = new LinkedHashMap<>();
        params.put(key: "Username", username);
        params.put(key: "Password", pass);
        params.put(key: "Badge", id);

        Badge dataBadge = new Badge(params.get(key: "Badge").toString());
        if(dataBadge.equals(badge)) {
            params.put(key: "Password", pass);
        }

        StringBuilder dataPost = new StringBuilder();
        for (Map.Entry<String, Object> param : params.entrySet()) {
            if (dataPost.length() != 0)
                dataPost.append(c: '&');
            dataPost.append(URLEncoder.encode(param.getKey(), enc: "UTF-8"));
            dataPost.append(c: '=');
            dataPost.append(URLEncoder.encode(String.valueOf(param.getValue()), enc: "UTF-8"));
        }
        byte[] dataPostBytes = dataPost.toString().getBytes(charsetName: "UTF-8");
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod(method: "POST");
        connection.setRequestProperty(key: "Content-Type", value: "application/x-www-form-urlencoded");
        connection.setRequestProperty(key: "Content-Length", String.valueOf(dataPostBytes.length));
        connection.setDoOutput(dooutput: true);
        connection.getOutputStream().write(dataPostBytes);

        Reader in = new BufferedReader(new InputStreamReader(connection.getInputStream(), charsetName: "UTF-8"));

    } catch (MalformedURLException e) {
        dialogueBox.setText(text: "Cannot connect to the database");
    } catch (IOException e) {
        dialogueBox.setText(text: "The User was not resgistered!");
    } finally {

    }

    URL url2 = new URL("https://sheet.best/api/sheets/2e0a3a65-f899-49f3-9477-780c8f6521fe/" + i);
    HttpURLConnection http = (HttpURLConnection) url2.openConnection();
    http.setRequestMethod(method: "DELETE");
    http.setRequestProperty(key: "Accept", value: "*/*");
    http.setRequestProperty(key: "Authorization", value: "Bearer mt0dgHmLJMVQhvjpNXDyA83vA_PxH23Y");

    System.out.println(http.getResponseCode() + " " + http.getResponseMessage());
```

The patch object takes the user object which matched the badge of the user inside the database, a new password as a string and the index of the object that matched the badge

it then stores them into local variables

It then creates new URL of the api endpoint used to connect the google sheets and java. It stores the objects as JSON Arrays. Then it gets the badge number at the and stroes it in dataBadge. Finally it compares the dataBadge to the local badge and if they are the same, then it changes the password to the new passed password string at key "Password" inside the same object.

Default Operations for sending a post request to an api endpoint

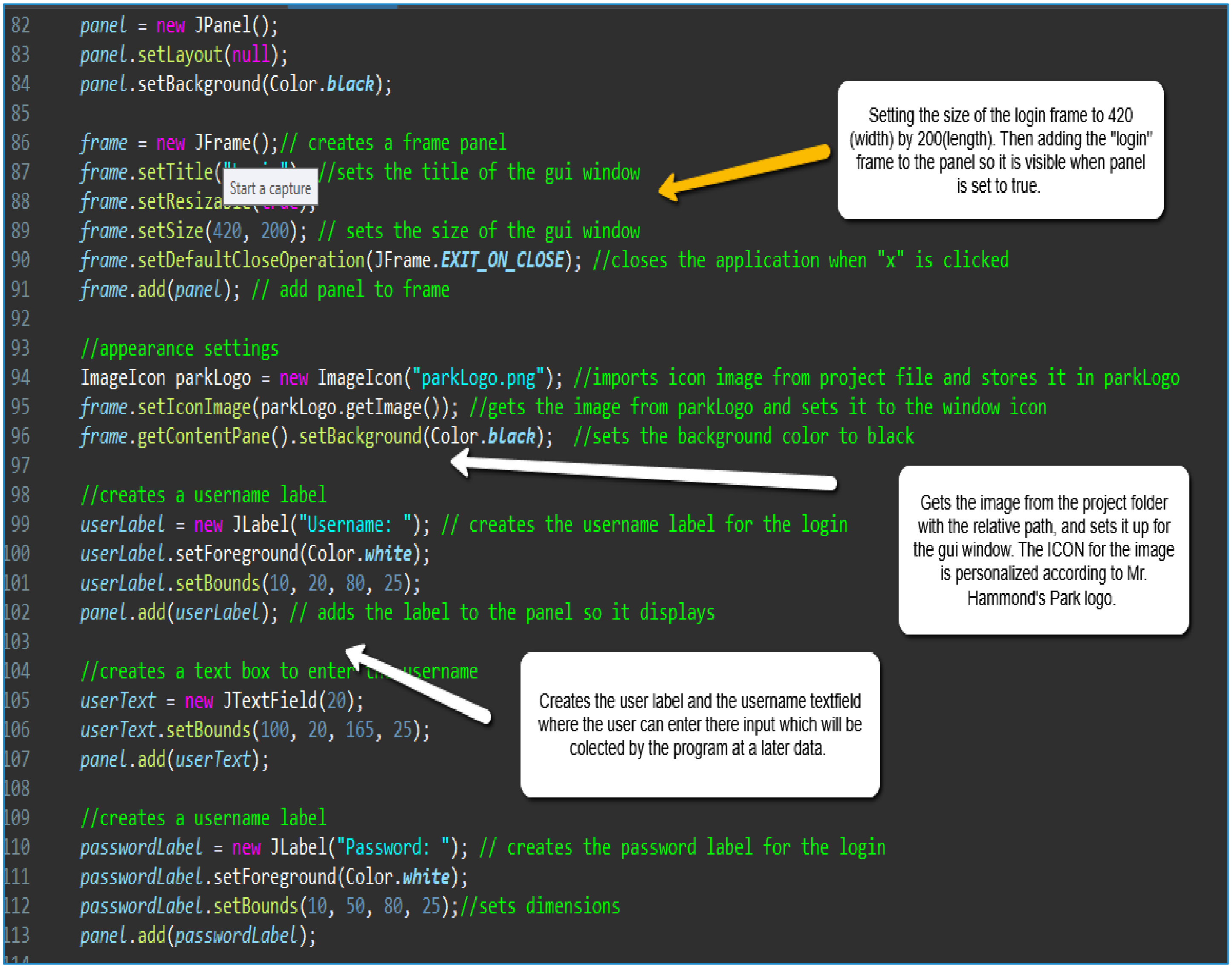
Default DELETE request in to an api endpoint with sheet.best. It add the index of the matched object "i" to the url idicating that it wants to delete that object in the database completely. Then it sends the delete request and deletes the old object

Essentially the patch method uses the post method to post a new object inside the google database. However, it checks to see that the badge of the object in the database is the same as the matched badge and then it posts and re-writes that specific object. However, this causes the database to have duplicates of the same username and badge with different passwords, which makes it less secure. Therefore, a delete request is sent to the database to delete that specific copy of the object that matched previously inside the resetPassword method. We do this because we only want to delete that specific copy of the object with the old password and the same badge in order to only keep the new password..

We do this by adding “\” + i; to the url which tells it what row we want to delete. “i” is the passed parameter inside the method and we can recall that it is the same index of the object where the badges matched. Therefore, that index is equal to the row number. And like that we can delete the row object at index “i” which allows us to only have a new copy of the object inside the database;

Swing GUI:

Parts of the Gui setup are shown below.



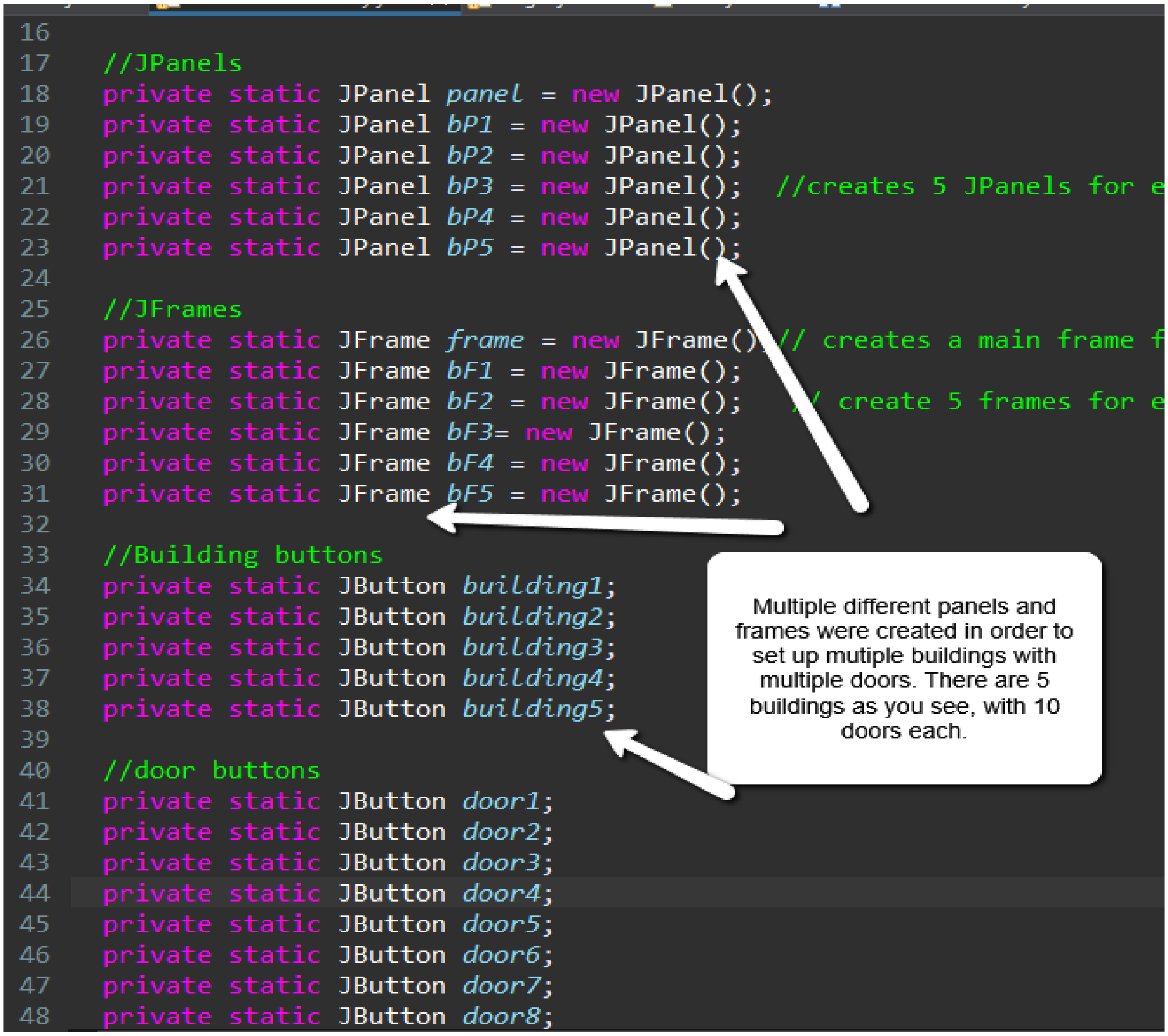
```java
    panel = new JPanel();
    panel.setLayout(null);
    panel.setBackground(Color.black);

    frame = new JFrame();// creates a frame panel
    frame.setTitle("Login"); //sets the title of the gui window
    frame.setResizable(true);
    frame.setSize(420, 200); // sets the size of the gui window
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //closes the application when "x" is clicked
    frame.add(panel); // add panel to frame

    //appearance settings
    ImageIcon parkLogo = new ImageIcon("parkLogo.png"); //imports icon image from project file and stores it in parkLogo
    frame.setIconImage(parkLogo.getImage()); //gets the image from parkLogo and sets it to the window icon
    frame.getContentPane().setBackground(Color.black);  //sets the background color to black

    //creates a username label
    userLabel = new JLabel("Username: "); // creates the username label for the login
    userLabel.setForeground(Color.white);
    userLabel.setBounds(10, 20, 80, 25);
    panel.add(userLabel); // adds the label to the panel so it displays

    //creates a text box to enter the username
    userText = new JTextField(20);
    userText.setBounds(100, 20, 165, 25);
    panel.add(userText);

    //creates a username label
    passwordLabel = new JLabel("Password: "); // creates the password label for the login
    passwordLabel.setForeground(Color.white);
    passwordLabel.setBounds(10, 50, 80, 25);//sets dimensions
    panel.add(passwordLabel);
```

Setting the size of the login frame to 420 (width) by 200(length). Then adding the "login" frame to the panel so it is visible when panel is set to true.

Gets the image from the project folder with the relative path, and sets it up for the gui window. The ICON for the image is personalized according to Mr. Hammond's Park logo.

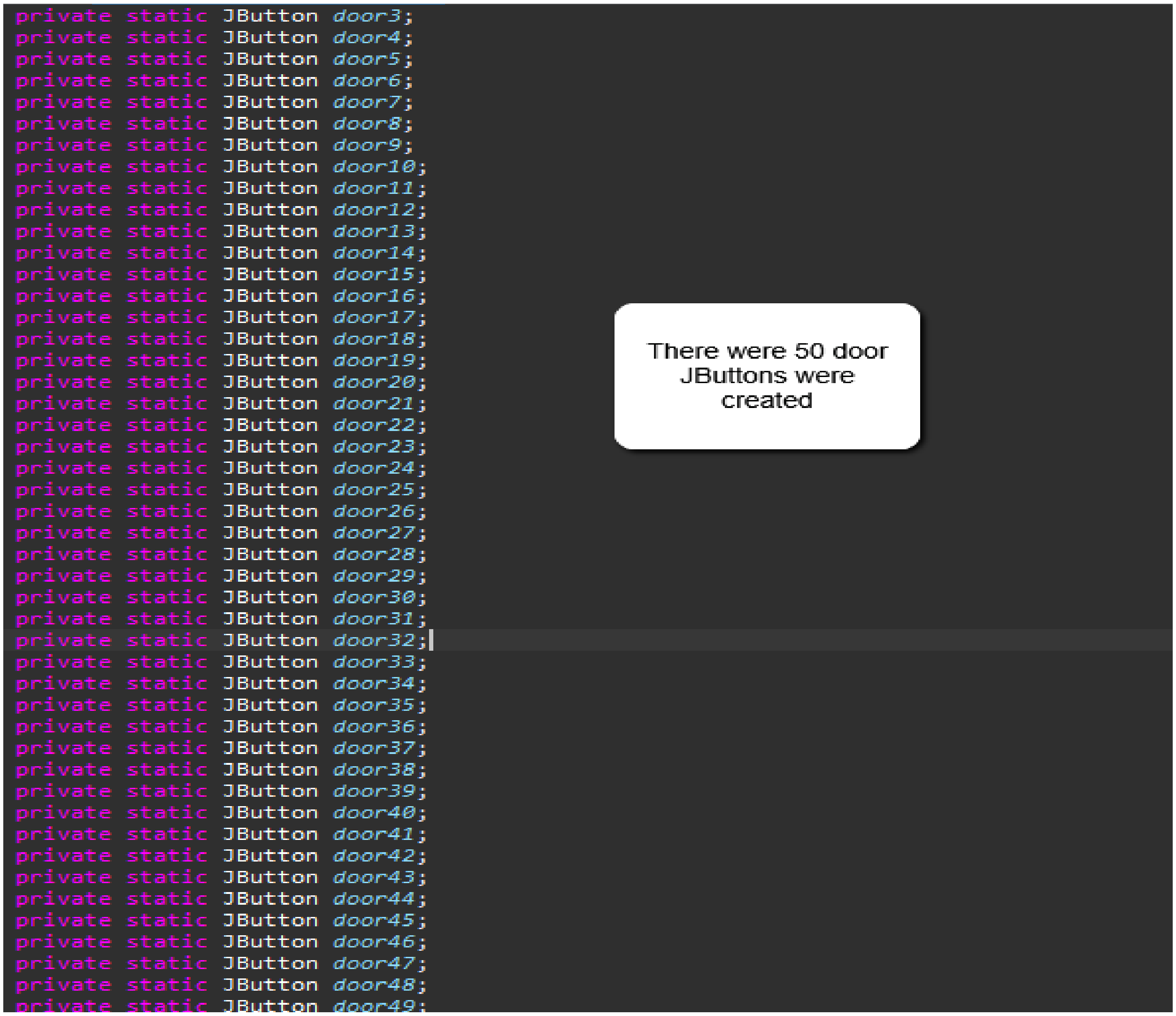
Creates the user label and the username textfield where the user can enter there input which will be colected by the program at a later data.

Setting up the buildings and the doors



```

    //JPanels
    private static JPanel panel = new JPanel();
    private static JPanel bP1 = new JPanel();
    private static JPanel bP2 = new JPanel();
    private static JPanel bP3 = new JPanel();  //creates 5 JPanels for e
    private static JPanel bP4 = new JPanel();
    private static JPanel bP5 = new JPanel();

    //JFrames
    private static JFrame frame = new JFrame();// creates a main frame f
    private static JFrame bF1 = new JFrame();
    private static JFrame bF2 = new JFrame();   // create 5 frames for e
    private static JFrame bF3= new JFrame();
    private static JFrame bF4 = new JFrame();
    private static JFrame bF5 = new JFrame();

    //Building buttons
    private static JButton building1;
    private static JButton building2;
    private static JButton building3;
    private static JButton building4;
    private static JButton building5;

    //door buttons
    private static JButton door1;
    private static JButton door2;
    private static JButton door3;
    private static JButton door4;
    private static JButton door5;
    private static JButton door6;
    private static JButton door7;
    private static JButton door8;
```

Multiple different panels and frames were created in order to set up mutiple buildings with multiple doors. There are 5 buildings as you see, with 10 doors each.

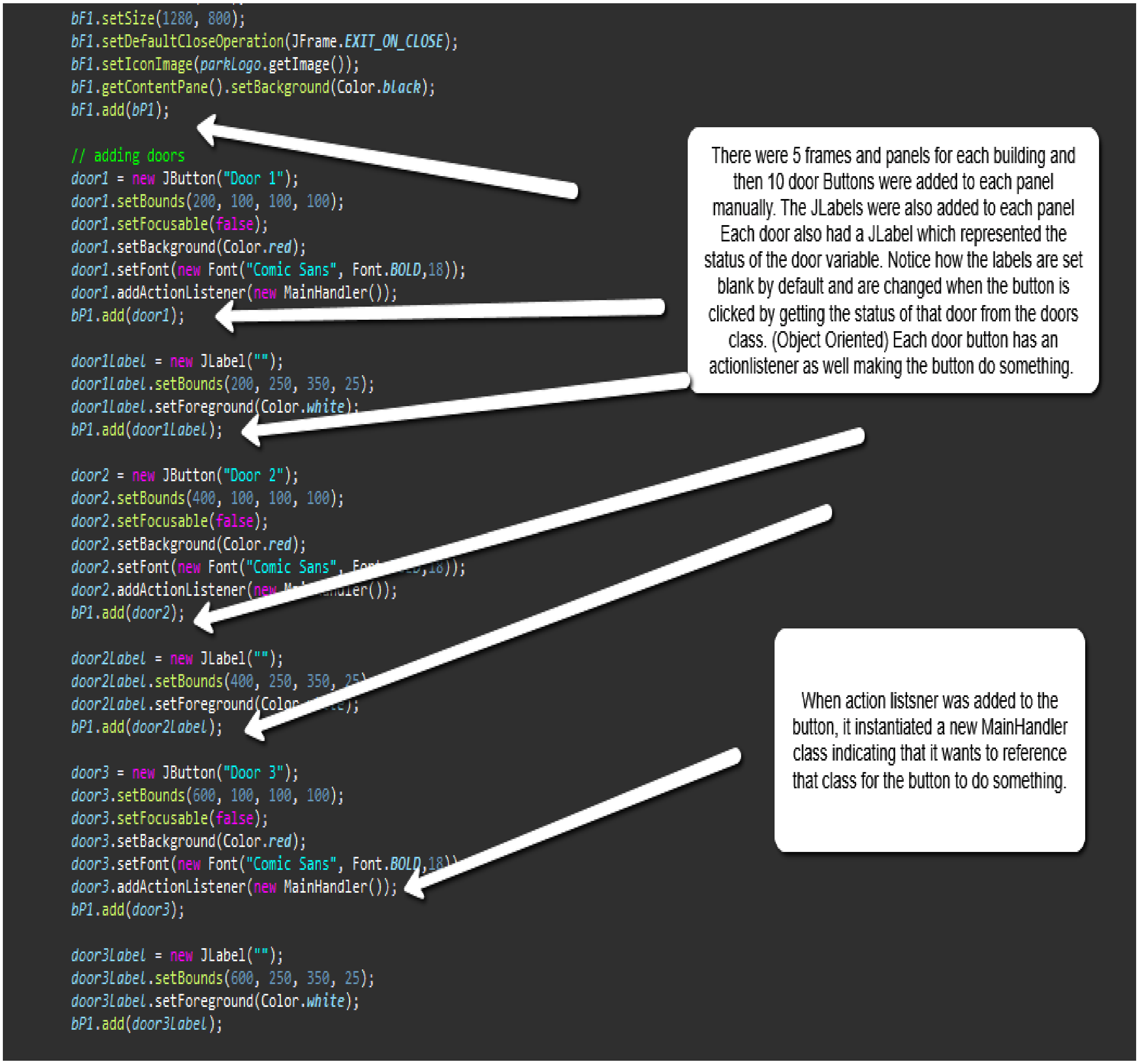


```
private static JButton door3;
private static JButton door4;
private static JButton door5;
private static JButton door6;
private static JButton door7;
private static JButton door8;
private static JButton door9;
private static JButton door10;
private static JButton door11;
private static JButton door12;
private static JButton door13;
private static JButton door14;
private static JButton door15;
private static JButton door16;
private static JButton door17;
private static JButton door18;
private static JButton door19;
private static JButton door20;
private static JButton door21;
private static JButton door22;
private static JButton door23;
private static JButton door24;
private static JButton door25;
private static JButton door26;
private static JButton door27;
private static JButton door28;
private static JButton door29;
private static JButton door30;
private static JButton door31;
private static JButton door32;
private static JButton door33;
private static JButton door34;
private static JButton door35;
private static JButton door36;
private static JButton door37;
private static JButton door38;
private static JButton door39;
private static JButton door40;
private static JButton door41;
private static JButton door42;
private static JButton door43;
private static JButton door44;
private static JButton door45;
private static JButton door46;
private static JButton door47;
private static JButton door48;
private static JButton door49;
```

There were 50 door JButtons were created

The doors were declared static, so the variables are the same throughout the methods of the class.

JFrames and JPanels setup according to each building



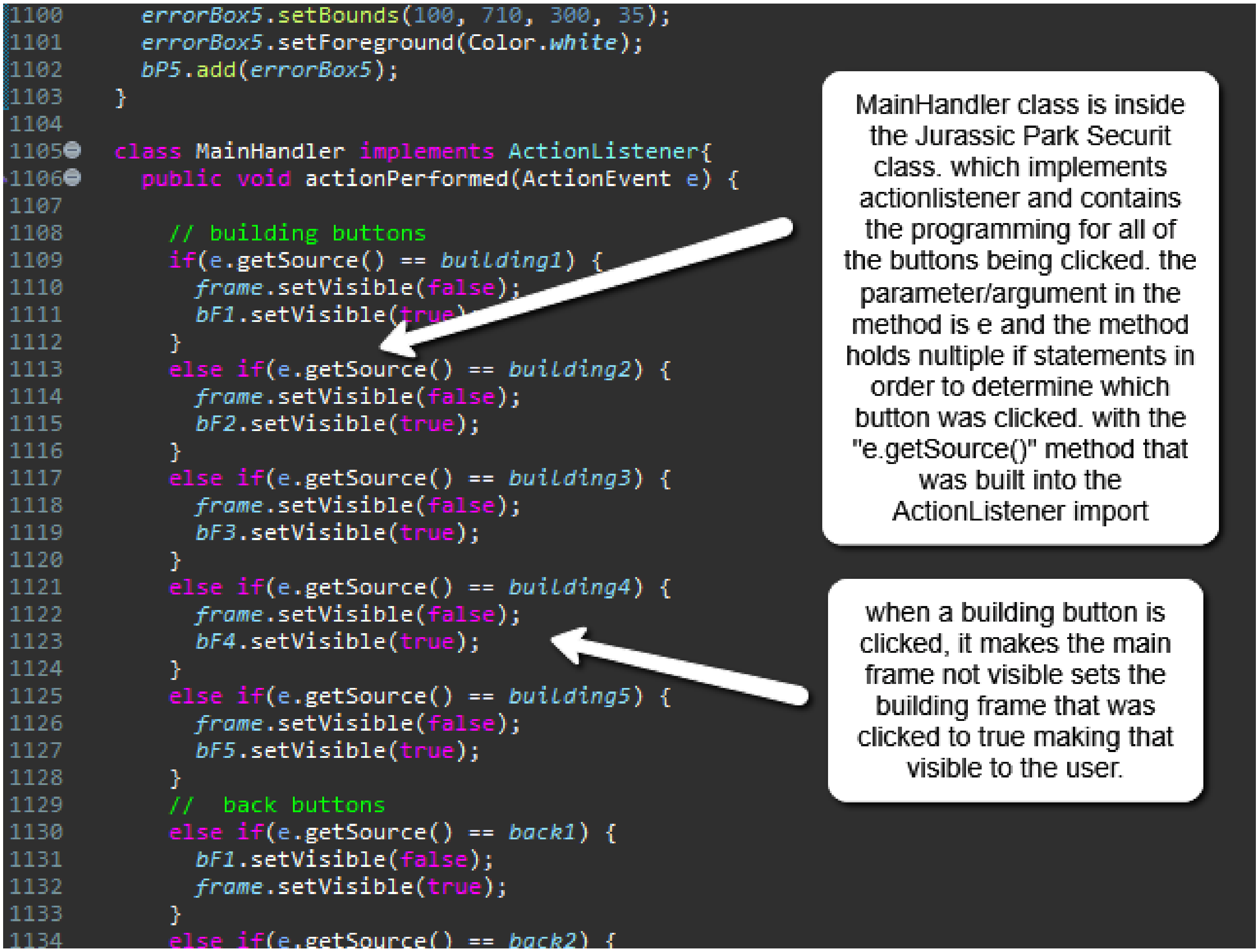
```
        bF1.setSize(1280, 800);
        bF1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        bF1.setIconImage(parkLogo.getImage());
        bF1.getContentPane().setBackground(Color.black);
        bF1.add(bP1);

        // adding doors
        door1 = new JButton("Door 1");
        door1.setBounds(200, 100, 100, 100);
        door1.setFocusable(false);
        door1.setBackground(Color.red);
        door1.setFont(new Font("Comic Sans", Font.BOLD,18));
        door1.addActionListener(new MainHandler());
        bP1.add(door1);

        door1Label = new JLabel("");
        door1Label.setBounds(200, 250, 350, 25);
        door1Label.setForeground(Color.white);
        bP1.add(door1Label);

        door2 = new JButton("Door 2");
        door2.setBounds(400, 100, 100, 100);
        door2.setFocusable(false);
        door2.setBackground(Color.red);
        door2.setFont(new Font("Comic Sans", Font.BOLD,18));
        door2.addActionListener(new MainHandler());
        bP1.add(door2);

        door2Label = new JLabel("");
        door2Label.setBounds(400, 250, 350, 25);
        door2Label.setForeground(Color.white);
        bP1.add(door2Label);

        door3 = new JButton("Door 3");
        door3.setBounds(600, 100, 100, 100);
        door3.setFocusable(false);
        door3.setBackground(Color.red);
        door3.setFont(new Font("Comic Sans", Font.BOLD,18));
        door3.addActionListener(new MainHandler());
        bP1.add(door3);

        door3Label = new JLabel("");
        door3Label.setBounds(600, 250, 350, 25);
        door3Label.setForeground(Color.white);
        bP1.add(door3Label);
```

There were 5 frames and panels for each building and then 10 door Buttons were added to each panel manually. The JLabels were also added to each panel Each door also had a JLabel which represented the status of the door variable. Notice how the labels are set blank by default and are changed when the button is clicked by getting the status of that door from the doors class. (Object Oriented) Each door button has an actionlistener as well making the button do something.

When action listsner was added to the button, it instantiated a new MainHandler class indicating that it wants to reference that class for the button to do something.

Nested Classes:

The benefit of using nested classes or local classes rather, is that it inherits all the variables from the super/parent class making it easier for to user. It also creates good encapsulation, which allows us to group the logic inside one class where it can be used. It does not exist in other classes, making it easier for the programmer to understand because all the action listeners are in one class.



```
  errorBox5.setBounds(100, 710, 300, 35);
  errorBox5.setForeground(Color.white);
  bP5.add(errorBox5);
}

class MainHandler implements ActionListener{
  public void actionPerformed(ActionEvent e) {

    // building buttons
    if(e.getSource() == building1) {
      frame.setVisible(false);
      bF1.setVisible(true);
    }
    else if(e.getSource() == building2) {
      frame.setVisible(false);
      bF2.setVisible(true);
    }
    else if(e.getSource() == building3) {
      frame.setVisible(false);
      bF3.setVisible(true);
    }
    else if(e.getSource() == building4) {
      frame.setVisible(false);
      bF4.setVisible(true);
    }
    else if(e.getSource() == building5) {
      frame.setVisible(false);
      bF5.setVisible(true);
    }
    //  back buttons
    else if(e.getSource() == back1) {
      bF1.setVisible(false);
      frame.setVisible(true);
    }
    else if(e.getSource() == back2) {
```

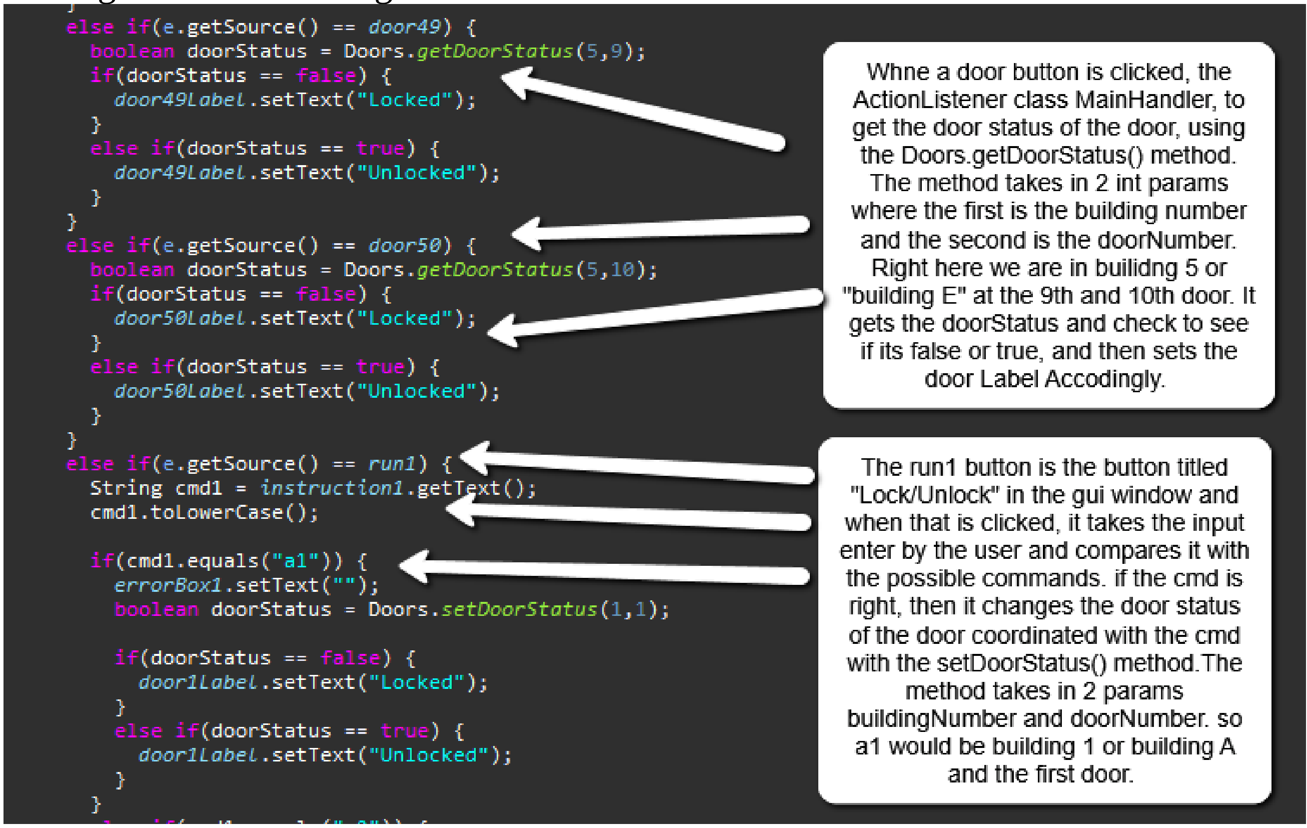
MainHandler class is inside the Jurassic Park Securit class. which implements actionlistener and contains the programming for all of the buttons being clicked. the parameter/argument in the method is e and the method holds nultiple if statements in order to determine which button was clicked. with the "e.getSource()" method that was built into the ActionListener import

when a building button is clicked, it makes the main frame not visible sets the building frame that was clicked to true making that visible to the user.

Object Oriented Programming:

There are a total of 4 main classes used with certain subclasses that implement action Listeners. The User class is used to create user objects that store the username and password. The Doors class creates buildings and doors by default mimicking Mr. Hammond’s Park. The Login Class and the Jurassic Park Security Class are GUI classes made for the user unlike the User and Doors classes. All the classes can be referenced in the Appendix in the souce code.

Explaining what happens when you click a doorButton. This also explains how the locking and the unlocking of doors work



```
    else if(e.getSource() == door49) {
      boolean doorStatus = Doors.getDoorStatus(5,9);
      if(doorStatus == false) {
        door49Label.setText("Locked");
      }
      else if(doorStatus == true) {
        door49Label.setText("Unlocked");
      }
    }
    else if(e.getSource() == door50) {
      boolean doorStatus = Doors.getDoorStatus(5,10);
      if(doorStatus == false) {
        door50Label.setText("Locked");
      }
      else if(doorStatus == true) {
        door50Label.setText("Unlocked");
      }
    }
    else if(e.getSource() == run1) {
      String cmd1 = instruction1.getText();
      cmd1.toLowerCase();

      if(cmd1.equals("a1")) {
        errorBox1.setText("");
        boolean doorStatus = Doors.setDoorStatus(1,1);

        if(doorStatus == false) {
          door1Label.setText("Locked");
        }
        else if(doorStatus == true) {
          door1Label.setText("Unlocked");
        }
      }
```

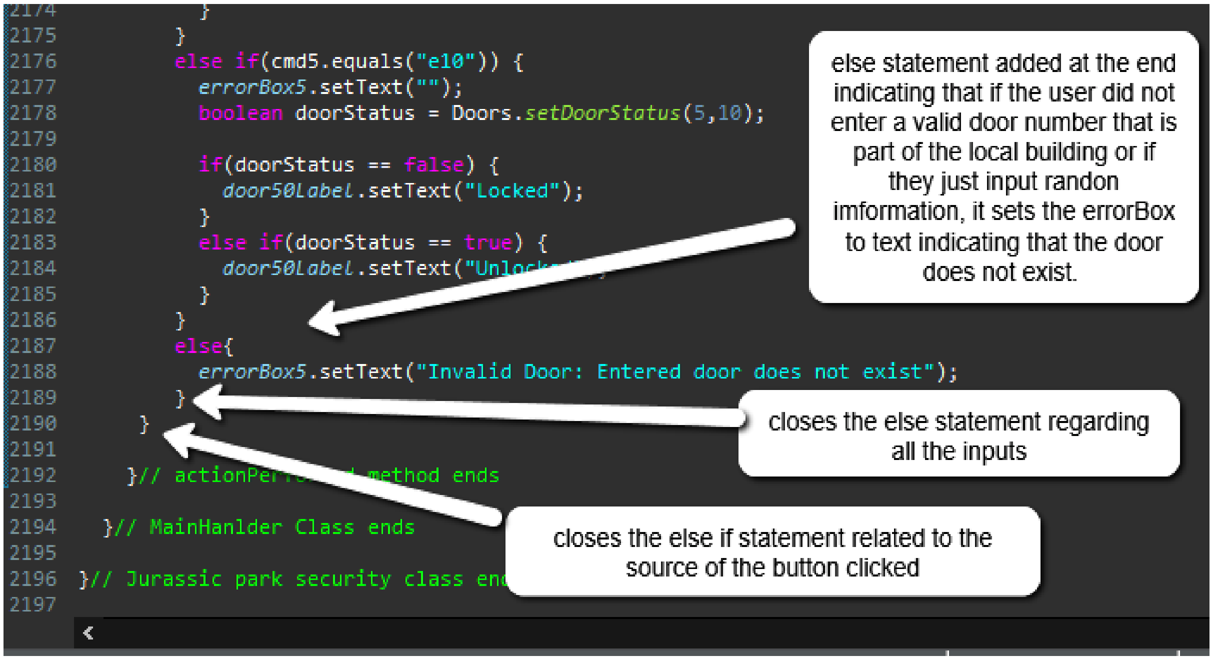
Whne a door button is clicked, the ActionListener class MainHandler, to get the door status of the door, using the Doors.getDoorStatus() method. The method takes in 2 int params where the first is the building number and the second is the doorNumber. Right here we are in builidng 5 or "building E" at the 9th and 10th door. It gets the doorStatus and check to see if its false or true, and then sets the door Label Accodingly.

The run1 button is the button titled "Lock/Unlock" in the gui window and when that is clicked, it takes the input enter by the user and compares it with the possible commands. if the cmd is right, then it changes the door status of the door coordinated with the cmd with the setDoorStatus() method.The method takes in 2 params buildingNumber and doorNumber. so a1 would be building 1 or building A and the first door.

This code also contains nest if statements providing the logic behind the door status.

Nested if statements:

Used for coding complex logic



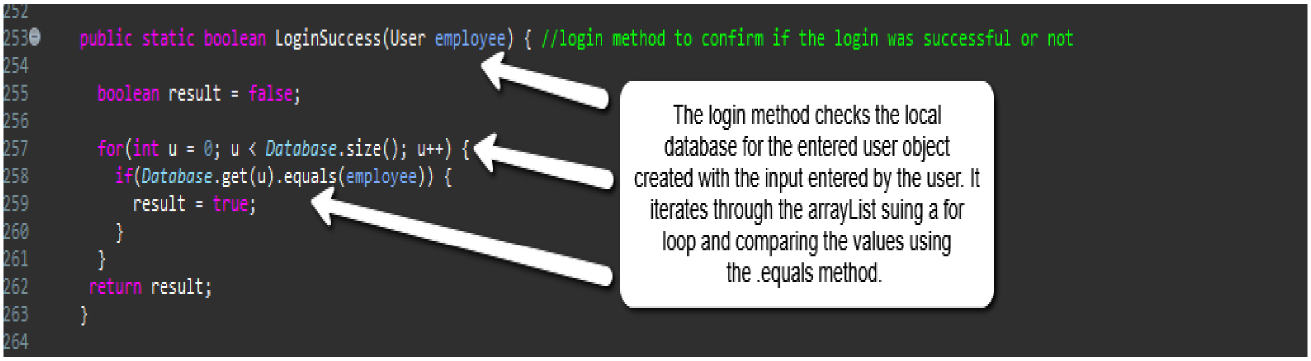
```
     }
    }
    else if(cmd5.equals("e10")) {
     errorBox5.setText("");
     boolean doorStatus = Doors.setDoorStatus(5,10);

     if(doorStatus == false) {
      door50Label.setText("Locked");
     }
     else if(doorStatus == true) {
      door50Label.setText("Unlocked");
     }
    }
    else{
     errorBox5.setText("Invalid Door: Entered door does not exist");
    }
   }

  }// actionPerformed method ends

 }// MainHanlder Class ends

}// Jurassic park security class ends
```

else statement added at the end indicating that if the user did not enter a valid door number that is part of the local building or if they just input randon imformation, it sets the errorBox to text indicating that the door does not exist.

closes the else statement regarding all the inputs

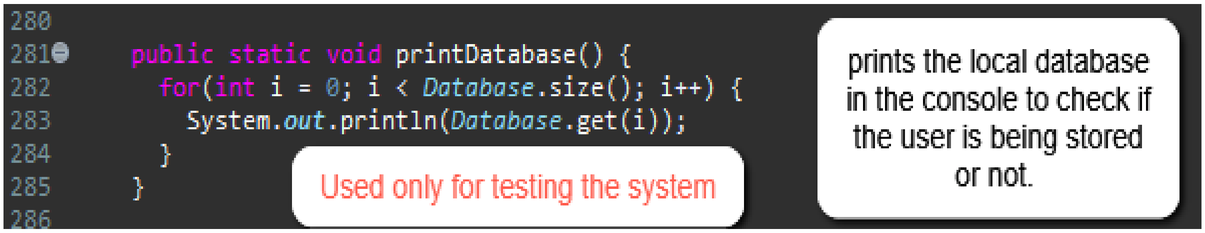
closes the else if statement related to the source of the button clicked

Searching and Testing:



```
 public static boolean LoginSuccess(User employee) { //login method to confirm if the login was successful or not

  boolean result = false;

  for(int u = 0; u < Database.size(); u++) {
   if(Database.get(u).equals(employee)) {
    result = true;
   }
  }
 return result;
 }
```

The login method checks the local database for the entered user object created with the input entered by the user. It iterates through the arrayList suing a for loop and comparing the values using the .equals method.



```
 public static void printDatabase() {
  for(int i = 0; i < Database.size(); i++) {
   System.out.println(Database.get(i));
  }
 }
```

Used only for testing the system

prints the local database in the console to check if the user is being stored or not.

Word Count: 894

Works Cited:

“Lesson: Laying Out Components Within a Container.” Lesson: Laying Out Components Within a Container (The Java™ Tutorials > Creating a GUI With JFC/Swing), docs.oracle.com/javase/tutorial/uiswing/layout/index.html.

“Lesson: Using Swing Components.” Lesson: Using Swing Components (The Java™ Tutorials > Creating a GUI With (JFC/Swing), docs.oracle.com/javase/tutorial/uiswing/components /index.html.

“Lesson: Writing Event Listeners.” Lesson: Writing Event Listeners (The Java™ Tutorials > Creating a GUI With JFC/Swing), docs.oracle.com/javase/tutorial/uiswing/events/ index.html.

“JsonArray (Java(TM) EE 7 Specification APIs).” *Java Oracle*, 1 June 2015, docs.oracle.com/javaee/7/api/javax/json/JsonArray.html.

“HttpURLConnection (Java SE 11 and JDK 11).” *Java Oracle*, 9 Dec. 2021, docs.oracle.com/en/java/javase/11/docs/api/java.base/java/net/HttpURLConnection.html.

Baeldung. “Making a JSON POST Request With HttpURLConnection.” *Baeldung*, 14 Oct. 2021, www.baeldung.com/httpurlconnection-post.

“Unit 7 Lesson 1 - ArrayLists.” *Project Stem*, projectstem.org/users/sign_in?module_item_id=1981903. Accessed 28 Feb. 2022.

Watmore, Jason. “Fetch - HTTP DELETE Request Examples.” *Jasonwatmore.com*, 21 Sept. 2021, https://jasonwatmore.com/post/2021/09/21/fetch-http-delete-request-examples.