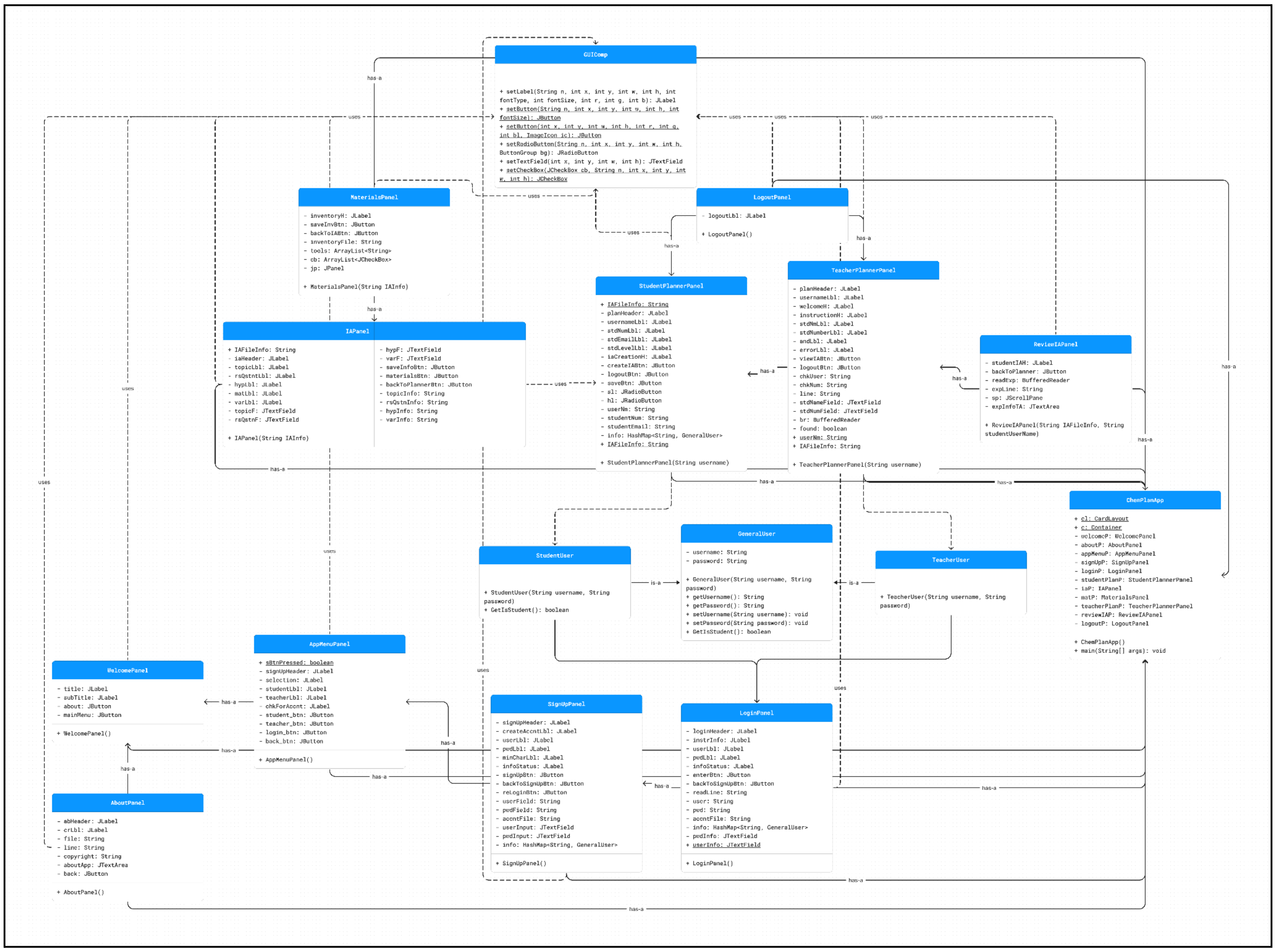
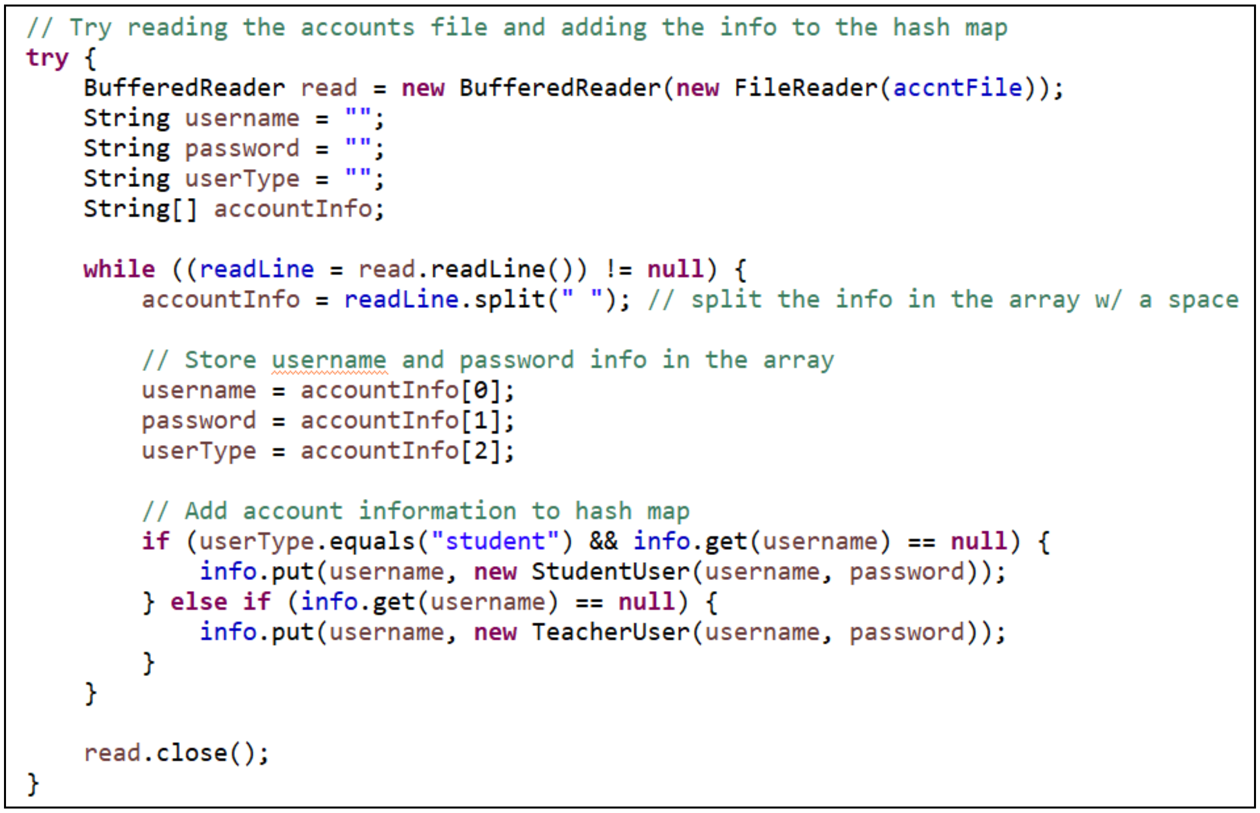
Criterion C: Development

UML Diagram

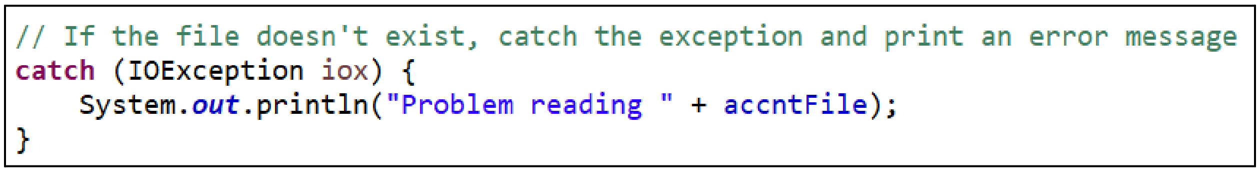


Techniques Used

1. Try/Catch Exception Handling



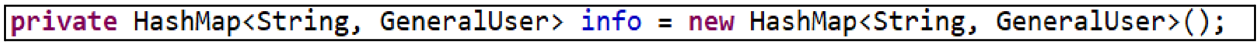
```
// Try reading the accounts file and adding the info to the hash map
try {
    BufferedReader read = new BufferedReader(new FileReader(accntFile));
    String username = "";
    String password = "";
    String userType = "";
    String[] accountInfo;

    while ((readLine = read.readLine()) != null) {
        accountInfo = readLine.split(" "); // split the info in the array w/ a space

        // Store username and password info in the array
        username = accountInfo[0];
        password = accountInfo[1];
        userType = accountInfo[2];

        // Add account information to hash map
        if (userType.equals("student") && info.get(username) == null) {
            info.put(username, new StudentUser(username, password));
        } else if (info.get(username) == null) {
            info.put(username, new TeacherUser(username, password));
        }
    }

    read.close();
}
```



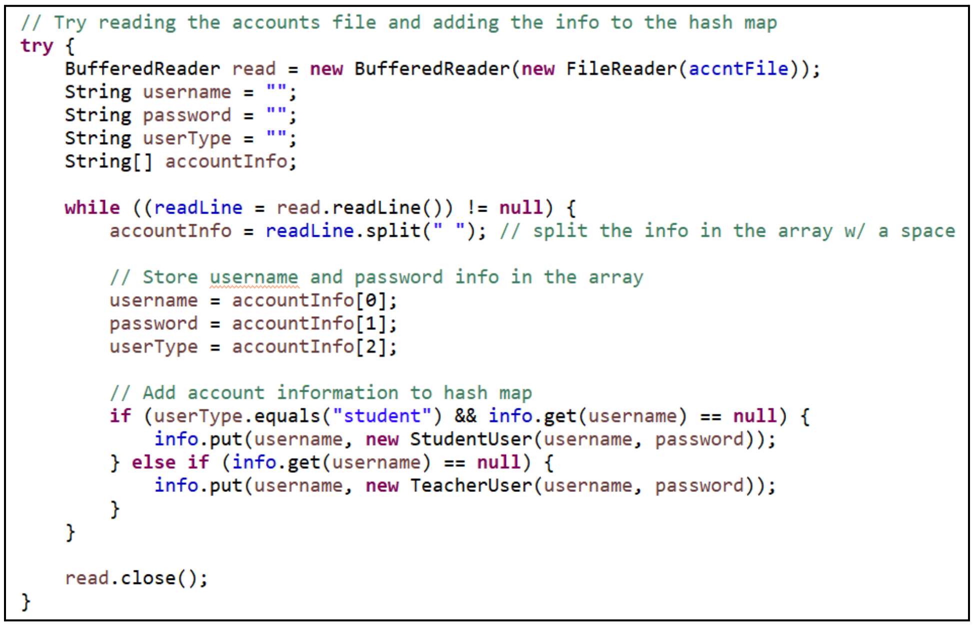
```
// If the file doesn't exist, catch the exception and print an error message
catch (IOException iox) {
    System.out.println("Problem reading " + accntFile);
}
```

The following code is within the LoginPanel class in the LoginPanel() constructor. The use of the try/catch method involves dealing with a file input/output exception. In this particular block of code, the BufferedReader attempts to read the account file that is created in the SignUpPanel() constructor of the SignUpPanel class. This code splits the info in the String array accountInfo[] and then corresponds each index of the array to a specific piece of information: specifically the user's username, password, and user type. An if/else if code block checks for the type of user and the value of the username String and accordingly adds the contents to the *info* HashMap. The main benefit of using the try/catch method is to ensure that if the file is unavailable, the code does not crash and instead, the user is informed that the file does not exist. Therefore, the code executes based on the condition that the file contains the necessary details.

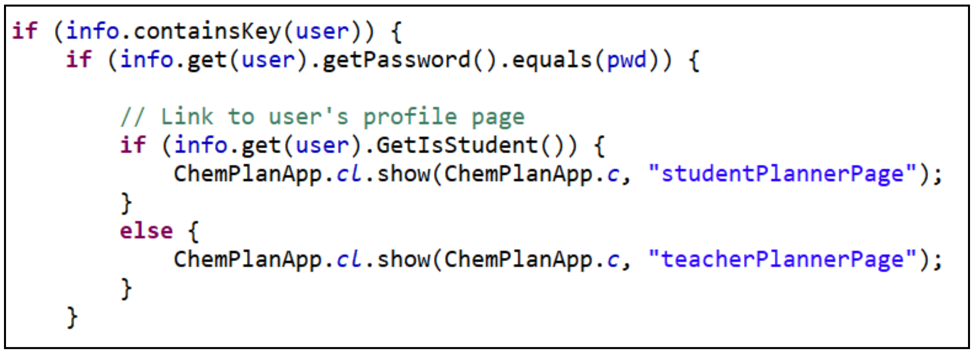
2. Hash Map



```
private HashMap<String, GeneralUser> info = new HashMap<String, GeneralUser>();
```



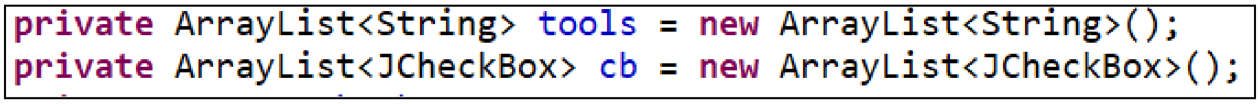
```
// Try reading the accounts file and adding the info to the hash map
try {
    BufferedReader read = new BufferedReader(new FileReader(accntFile));
    String username = "";
    String password = "";
    String userType = "";
    String[] accountInfo;

    while ((readLine = read.readLine()) != null) {
        accountInfo = readLine.split(" "); // split the info in the array w/ a space

        // Store username and password info in the array
        username = accountInfo[0];
        password = accountInfo[1];
        userType = accountInfo[2];

        // Add account information to hash map
        if (userType.equals("student") && info.get(username) == null) {
            info.put(username, new StudentUser(username, password));
        } else if (info.get(username) == null) {
            info.put(username, new TeacherUser(username, password));
        }
    }

    read.close();
}
```



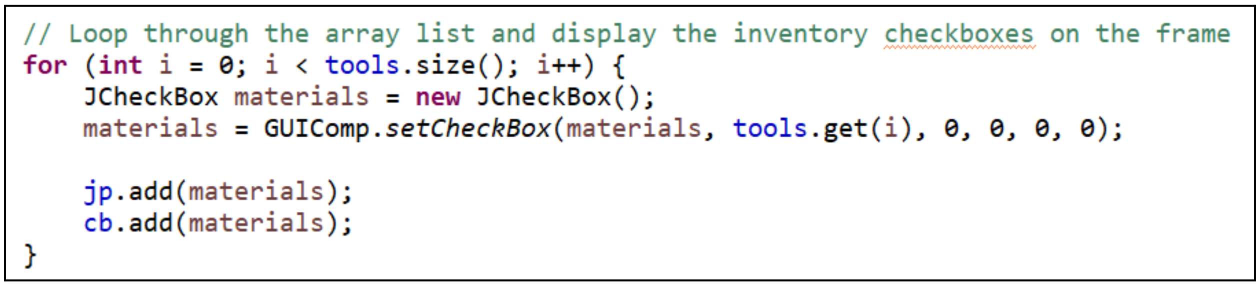
```
if (info.containsKey(user)) {
    if (info.get(user).getPassword().equals(pwd)) {

        // Link to user's profile page
        if (info.get(user).GetIsStudent()) {
            ChemPlanApp.cl.show(ChemPlanApp.c, "studentPlannerPage");
        }
        else {
            ChemPlanApp.cl.show(ChemPlanApp.c, "teacherPlannerPage");
        }
    }
```

A hash map is an ADT that stores data in pairs with parameters for a key and value. The first image shows the initialization of the map as a private data type with the parameters String and GeneralUser which indicates that the values passed into map will include an object of the inherited sub-classes of GeneralUser. In the try/catch block, the put() method for the HashMap is called allowing the parameters to be added to the map. In this instance, the parameters read by the HashMap are the user’s username and the correct inherited object depending on the user’s type (StudentUser or TeacherUser). In the second code block, a double if statement is used to check the HashMap for the appropriate user info. The first if statement uses the containsKey() method to check for the first key in the map, while the second if statement uses the get() method to check for the corresponding password value that matches the key. This code highlights the main benefit of the HashMap as the information entered by the user during sign-up can easily be matched because each key contains a corresponding value. Rather than splitting an array, the map checks for the dual information and accordingly checks the user’s original account information, simplifying the login process of the program.

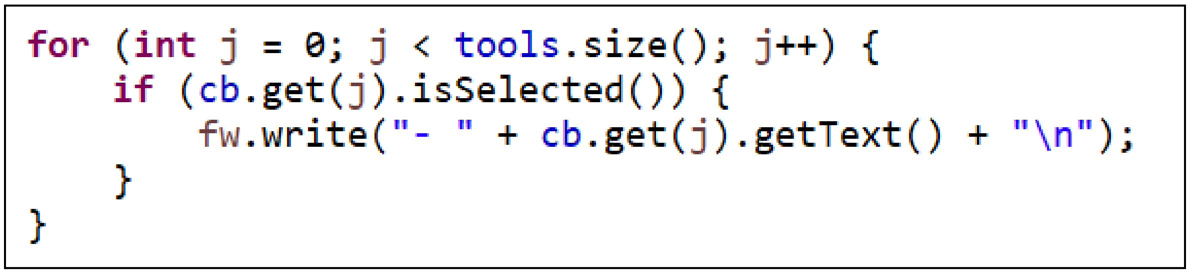
3. ArrayList



```java
private ArrayList<String> tools = new ArrayList<String>();
private ArrayList<JCheckBox> cb = new ArrayList<JCheckBox>();
```



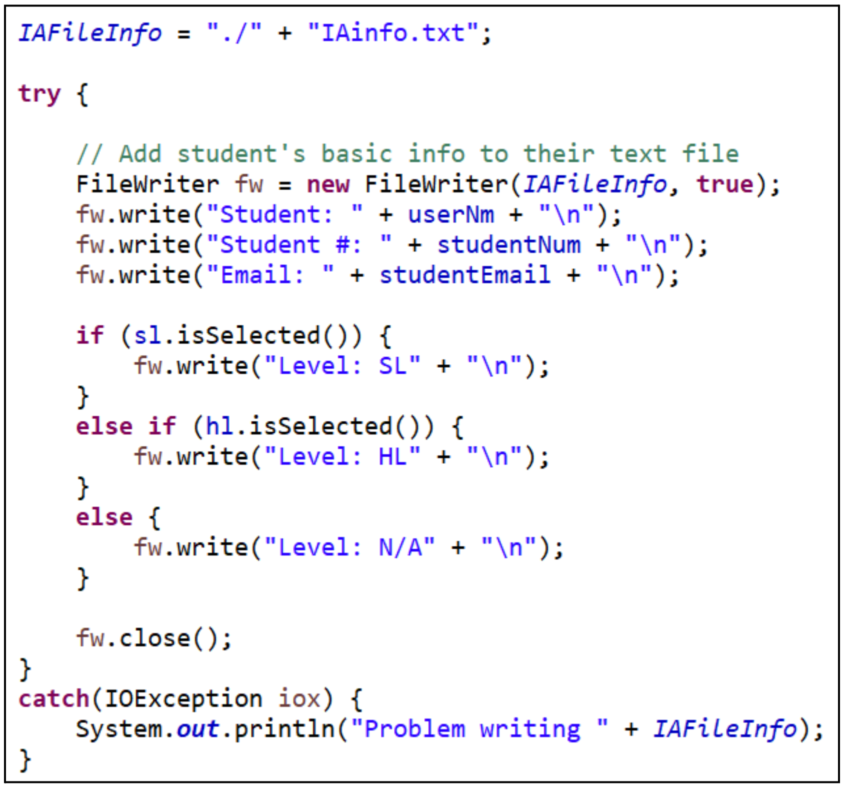
```java
// Loop through the array list and display the inventory checkboxes on the frame
for (int i = 0; i < tools.size(); i++) {
    JCheckBox materials = new JCheckBox();
    materials = GUIComp.setCheckBox(materials, tools.get(i), 0, 0, 0, 0);

    jp.add(materials);
    cb.add(materials);
}
```



```java
for (int j = 0; j < tools.size(); j++) {
    if (cb.get(j).isSelected()) {
        fw.write("- " + cb.get(j).getText() + "\n");
    }
}
```

The following code blocks involve the usage of ArrayLists to store the materials and access them in the program as a comprehensive inventory list. The benefits of this ADT within my program is primarily related to the capability for an ArrayList to hold a large number of elements without a need to define its size. Unlike an array which can only contain a predetermined number of elements, an ArrayList can grow its size during the runtime of the program which is necessary in this situation as the material list that is read in from a text file is not fully determined and can thus change its size if other inventory is added. The third code block shows the get() and isSelected() methods of the ArrayList which checks for the user's material selection and the program then dynamically writes their choices into the custom text file.

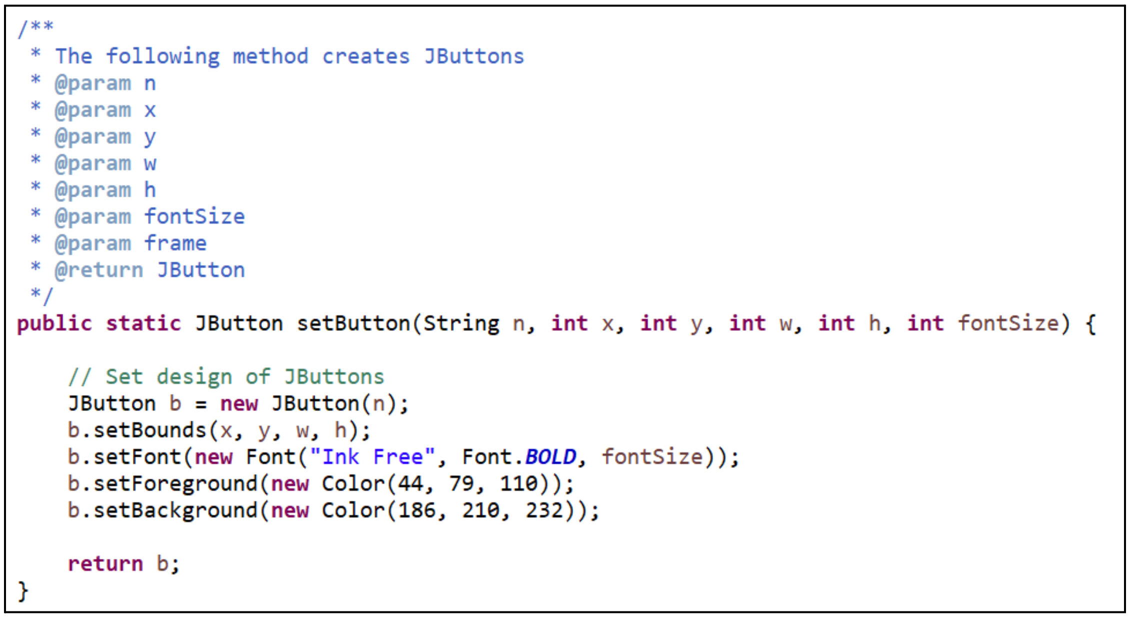
4. File Handling



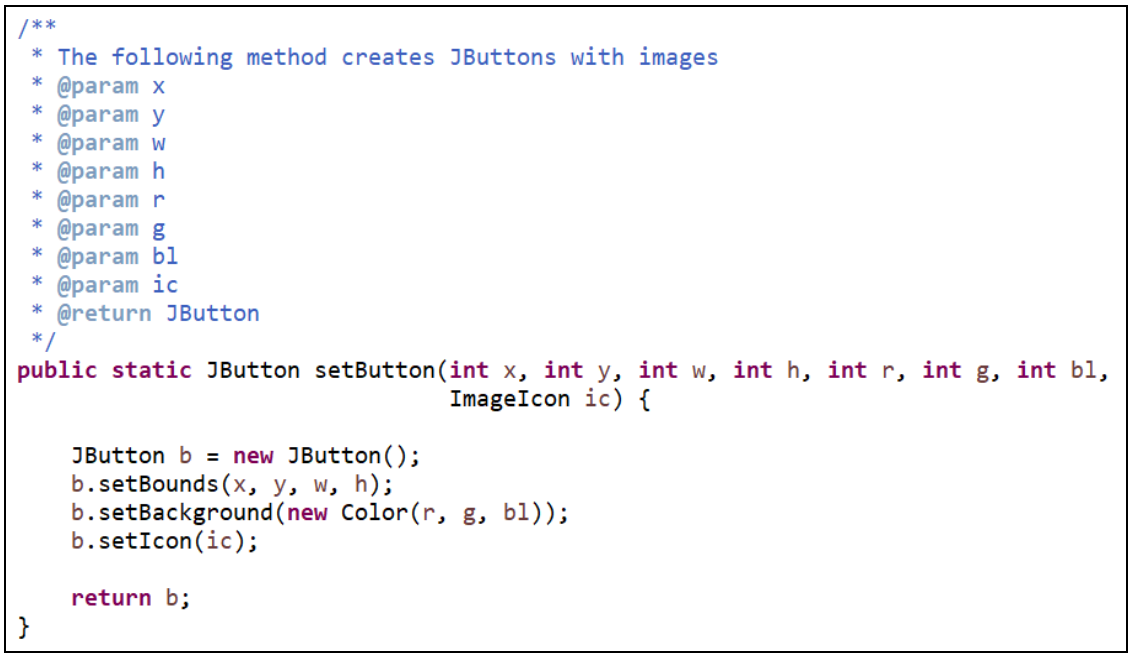
```java
IAFileInfo = "./" + "IAinfo.txt";

try {

    // Add student's basic info to their text file
    FileWriter fw = new FileWriter(IAFileInfo, true);
    fw.write("Student: " + userNm + "\n");
    fw.write("Student #: " + studentNum + "\n");
    fw.write("Email: " + studentEmail + "\n");

    if (sl.isSelected()) {
        fw.write("Level: SL" + "\n");
    }
    else if (hl.isSelected()) {
        fw.write("Level: HL" + "\n");
    }
    else {
        fw.write("Level: N/A" + "\n");
    }

    fw.close();
}
catch(IOException iox) {
    System.out.println("Problem writing " + IAFileInfo);
}
```

The following code employs FileWriter and the creation of the student's IA file. The variable *IAFileInfo* is static so when the student presses the save button on their planner, a customized file with the student's name is added to the file. This process allows the users to have their own distinguishable files which can then have their data be written in separately using FileWriter. Using the write() method allows the FileWriter to add info and have it stored dynamically between multiple users, creating easy accessibility for the teacher users of the program.

5. Method Overloading - Static Polymorphism



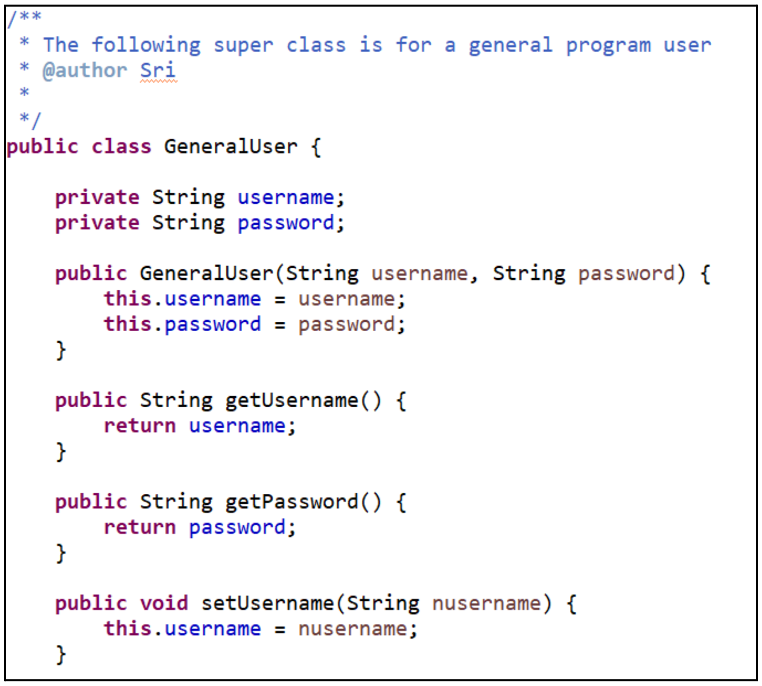
```java
/**
 * The following method creates JButtons
 * @param n
 * @param x
 * @param y
 * @param w
 * @param h
 * @param fontSize
 * @param frame
 * @return JButton
 */
public static JButton setButton(String n, int x, int y, int w, int h, int fontSize) {

    // Set design of JButtons
    JButton b = new JButton(n);
    b.setBounds(x, y, w, h);
    b.setFont(new Font("Ink Free", Font.BOLD, fontSize));
    b.setForeground(new Color(44, 79, 110));
    b.setBackground(new Color(186, 210, 232));

    return b;
}
```



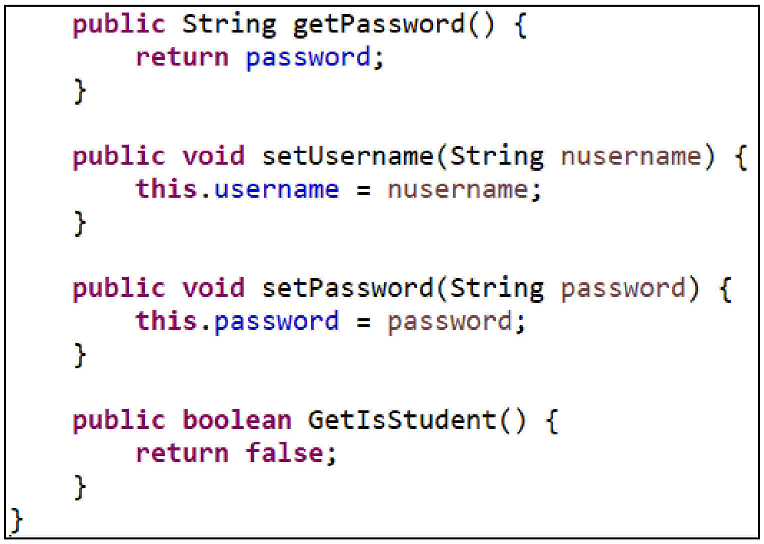
```java
/**
 * The following method creates JButtons with images
 * @param x
 * @param y
 * @param w
 * @param h
 * @param r
 * @param g
 * @param bl
 * @param ic
 * @return JButton
 */
public static JButton setButton(int x, int y, int w, int h, int r, int g, int bl,
                                ImageIcon ic) {

    JButton b = new JButton();
    b.setBounds(x, y, w, h);
    b.setBackground(new Color(r, g, bl));
    b.setIcon(ic);

    return b;
}
```

Polymorphism is a main feature of OOP that can be used for code efficiency and organization. Having two methods called *setButton()* with different parameters is a form of method overloading where Java can distinguish between the two methods and call the appropriate one based on the parameters passed. The first code block features parameters for the button's name, coordinates, and font size, since the buttons are used to primarily direct users to specific panels. However the same method is overloaded with new parameters such as colours and an ImageIcon because the button contains graphics to represent two different users. This employment of static polymorphism through method overloading enables code reuse and efficiency so that the code blocks are organized and have improved readability.

6. Inheritance

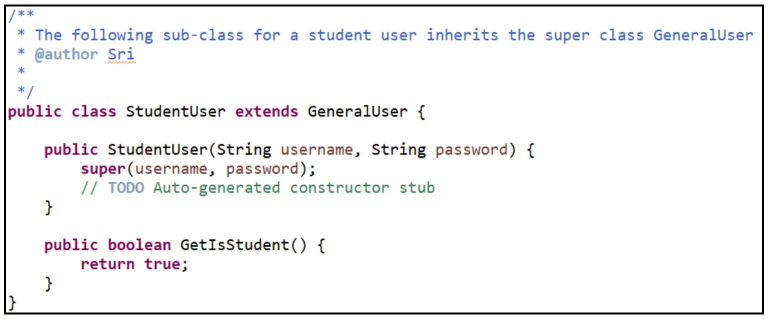


```java
/**
 * The following super class is for a general program user
 * @author Sri
 *
 */
public class GeneralUser {

    private String username;
    private String password;

    public GeneralUser(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public void setUsername(String nusername) {
        this.username = nusername;
    }
```

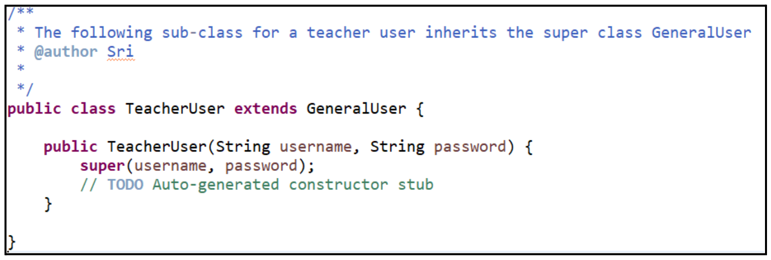


```java
    public String getPassword() {
        return password;
    }

    public void setUsername(String nusername) {
        this.username = nusername;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public boolean GetIsStudent() {
        return false;
    }
}
```

Block 1

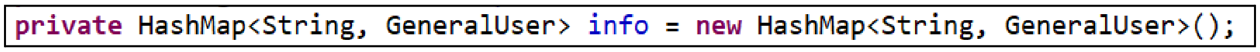


```java
/**
 * The following sub-class for a student user inherits the super class GeneralUser
 * @author Sri
 *
 */
public class StudentUser extends GeneralUser {

    public StudentUser(String username, String password) {
        super(username, password);
        // TODO Auto-generated constructor stub
    }

    public boolean GetIsStudent() {
        return true;
    }
}
```

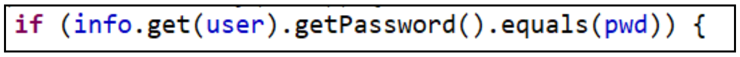


```java
/**
 * The following sub-class for a teacher user inherits the super class GeneralUser
 * @author Sri
 *
 */
public class TeacherUser extends GeneralUser {

    public TeacherUser(String username, String password) {
        super(username, password);
        // TODO Auto-generated constructor stub
    }

}
```

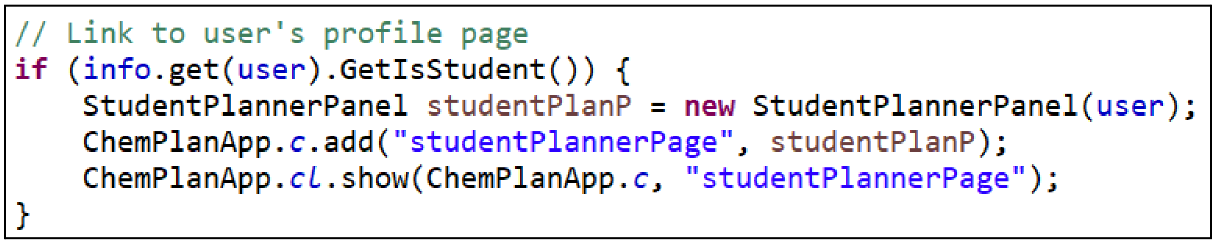
Block 2



```java
private HashMap<String, GeneralUser> info = new HashMap<String, GeneralUser>();
```



```java
if (info.get(user).getPassword().equals(pwd)) {
```

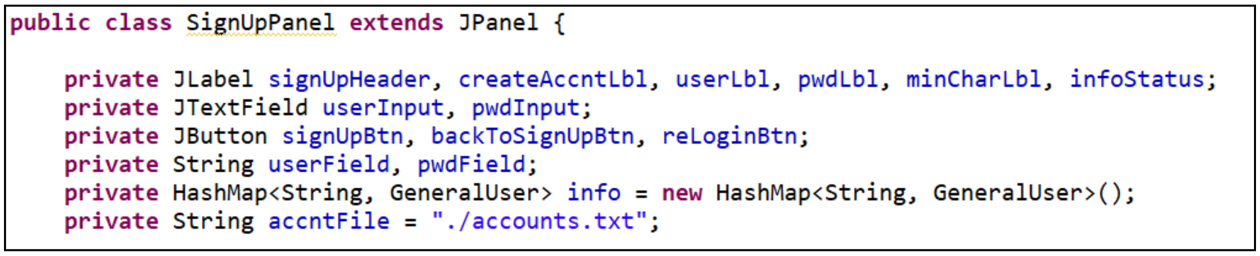


```java
// Link to user's profile page
if (info.get(user).GetIsStudent()) {
    StudentPlannerPanel studentPlanP = new StudentPlannerPanel(user);
    ChemPlanApp.c.add("studentPlannerPage", studentPlanP);
    ChemPlanApp.cl.show(ChemPlanApp.c, "studentPlannerPage");
}
```

Block 3

The following blocks of code highlight the use of inheritance, a main feature of OOP throughout this program. The first block is the parent class GeneralUser which has various getter and setter methods that are used within the program. This provides the structure for the StudentUser and TeacherUser classes in the second block which are sub-classes of the GeneralUser class, as shown by the keyword extends. Both classes inherit the methods in GeneralUser, however modifications pertaining to the type of user can also be added. For example, in the StudentUser class, the private boolean method GetIsStudent() returns true rather than false, which is a change that is made in the sub-class despite the inheriting of this method. Block three features examples of how inheritance is used within the program. The HashMap info takes a GeneralUser object as a parameter. This means that when methods such as getPassword() and GetIsStudent() are called on the map, Java is able to use inheritance to refer to the appropriate methods in the StudentUser class without the need for the declaration of a new StudentUser object because of the inheritance between the classes. Inheritance ultimately helps connect different classes, improving both code efficiency and dynamic functionality within the application.

7. Encapsulation



```java
public class SignUpPanel extends JPanel {

    private JLabel signUpHeader, createAccntLbl, userLbl, pwdLbl, minCharLbl, infoStatus;
    private JTextField userInput, pwdInput;
    private JButton signUpBtn, backToSignUpBtn, reLoginBtn;
    private String userField, pwdField;
    private HashMap<String, GeneralUser> info = new HashMap<String, GeneralUser>();
    private String accntFile = "./accounts.txt";
```

Encapsulation, another main feature of OOP involves creating private variables that are only accessible within a specific class. The following GUI components are all created as private variables as they are only displayed on the panel of the SignUpPanel() class. Rather than having these variables accessible across multiple classes, leading to code inconsistencies, using encapsulation ensures that the correct attributes are only accessed in their specific classes. This also helps hide data from the user. The user is not forced to understand how the variables and info is manipulated or accessed within the program and are just expected to enter their information with the understanding that this data is being passed into specific attributes.

Words: 1057