Code Review of The Software Project:

Blood Management System

Course Name: Software Development Project

Course No: CSE 3106

Submitted to:

Dr. Amit Kumar Mandal

Associate Professor

Computer Science & Engineering Discipline

Group Members:

1. Sourav Shome(210217)
2. Pushpita Chakma (210240)
3. Mst. Eshrat Jahan Esha(210233

Lack of Encapsulation: The SignUp class contains both GUI setup logic (initcomponents()) and business logic (handling user sign-up and writing data to a file). This violates the Single Responsibility Principle (SRP), as the class is responsible for more than one thing

```
package bloodmanagementsystem;

import java.io.FileWriter;
import javax.swing.JOptionPane;


public class SignUp extends javax.swing.JFrame {


    public SignUp() {
        initComponents();
    }
```

Method Length: The Ok buttonactionperformed method is relatively long and contains multiple responsibilities, such as retrieving input values, writing data to a file, and displaying a success message. Breaking down this method into smaller, more focused methods can improve readability and maintainability.

Unused Imports: The code includes unused imports (import javax.swing.RowFilter), which should be removed to keep the code clean.

```
import javax.swing.RowFilter;
```

Non-Descriptive Variable Names: Variable names like jPanel1, jLabel1, jTextField1, etc., are not very descriptive. Using more meaningful names can improve code readability.

Inconsistent Formatting:

The code contains inconsistent formatting practices, such as varying indentation levels and spacing. Consistent formatting improves code readability and maintainability.

```
if (file.exists()) {
  try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
    String line;
    while ((line = reader.readLine()) != null) {
      String[] data = line.split(",");
      if (data.length == 5) {
        model.addRow(data);
      }
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}
}
```

Large Functions & Class :

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton1ActionPerformed
    // TODO add your handling code here:

    String name = tfName.getText();
    String email = tfEmail.getText();
    String phone = tfPhone.getText();
    String blood = tfBlood.getText();
    String address = tfAddress.getText();
    String data = name + "," + email +" ,"+ phone+" ," +blood+","+address; //Storing all variable in one variable

    if(name.isEmpty() || email.isEmpty() ||blood.isEmpty()|| phone.isEmpty() ||address.isEmpty())
    {
        JOptionPane.showMessageDialog(this,"Please enter all fields","Try again",JOptionPane.ERROR_MESSAGE);

    }
    else
    {
        DefaultTableModel model = (DefaultTableModel) jtable.getModel();
        model.addRow(new Object[]{name,email,phone,blood,address});
        tfName.setText("");
        tfEmail.setText("");
        tfPhone.setText("");
        tfBlood.setText("");
        tfAddress.setText("");

    }
    saveToFile(data);
}//GEN-LAST:event_jButton1ActionPerformed
```

Large no of if else statements:

```
if(bg.equals("A+"))
{
    a_pos++;
    td++;

}
else if(bg.equals("B+"))
{
     b_pos++;
      td++;
}
else if(bg.equals("AB+"))
{
     ab_pos++;
      td++;
}
else if(bg.equals("O+"))
{
     o_pos++;
      td++;
}
else if(bg.equals("A-"))
{
     a_neg++;
      td++;
}
     else if(bg.equals("AB+"))
     {
          ab_pos++;
           td++;
     }
     else if(bg.equals("O+"))
     {
          o_pos++;
           td++;
     }
     else if(bg.equals("A-"))
     {
          a_neg++;
           td++;
     }
     else if(bg.equals("B-"))
     {
          b_neg++;
           td++;
     }
     else if(bg.equals("O-"))
     {
          o_neg++;
           td++;
     }
     else if(bg.equals("AB-"))
     {
         ab_neg++;
          td++;
     }
```

Lack of Comments:

The code lacks comments to explain the purpose of certain sections of code or complex logic. Adding comments can improve code understandability, especially for other developers or for future reference.

```
private void Back_to_Home_ButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_Back_to_Home_ButtonActionPerformed
        Home h = new Home();
        h.setVisible(true);
        this.dispose();
}//GEN-LAST:event_Back_to_Home_ButtonActionPerformed

private void saveToFile(String data) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter("data.txt", true))) {
        writer.write(data + "\n");
    } catch (IOException e) {

    }
}
 private void loadDataFromFile() {
    File file = new File("data.txt");
     DefaultTableModel model = (DefaultTableModel) jtable.getModel();

       // model.addRow(data);

    if (file.exists()) {
        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] data = line.split(",");
                if (data.length == 5) {
                    model.addRow(data);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- Refactoring: There is some code duplication that could be reduced by extracting common functionality into methods.

- Error Handling: More comprehensive error handling could be implemented to handle cases such as invalid input data.
- User Interface: Enhancing the UI design and adding labels for input fields could improve usability.
- Comments: Adding comments to explain complex logic or to provide context would improve code readability.
- Naming: Using more descriptive variable names can make the code easier to understand.