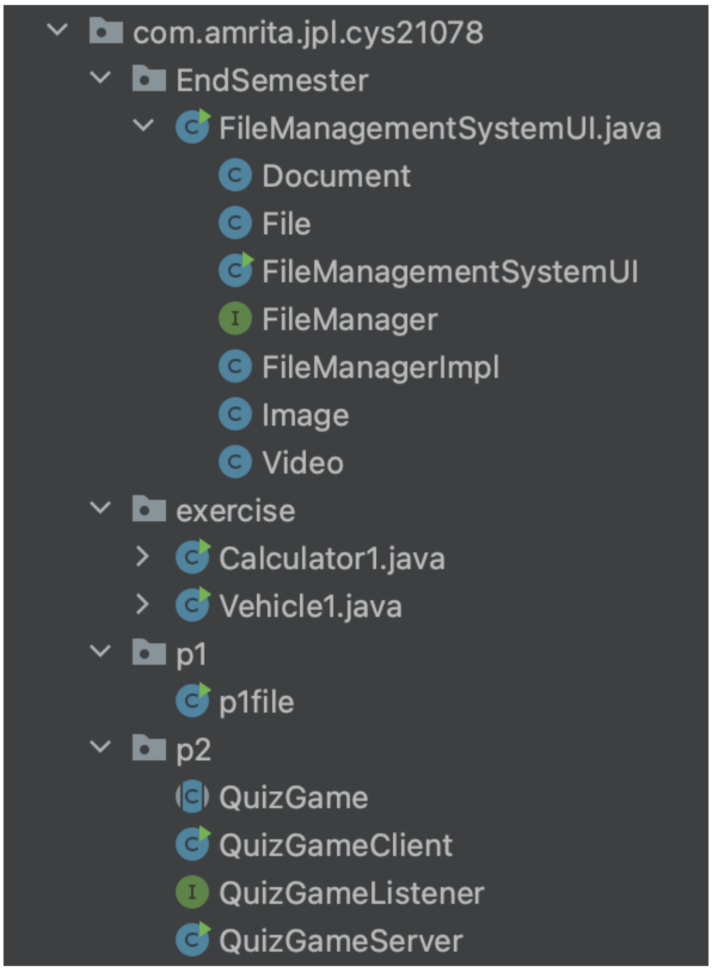
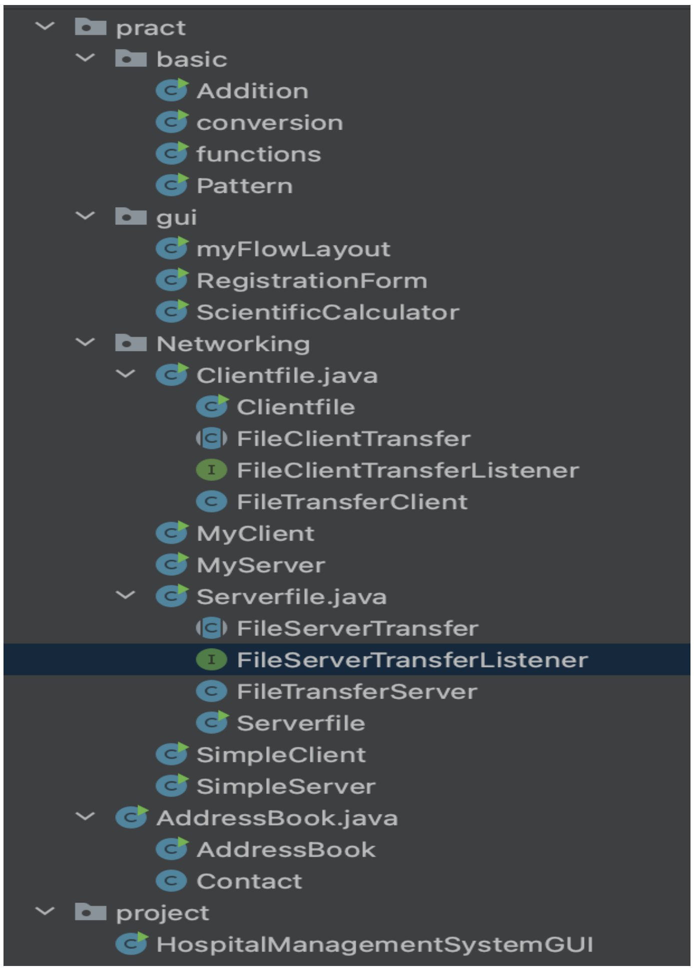
Package Structure



```
com.amrita.jpl.cys21078
  EndSemester
    FileManagementSystemUI.java
      Document
      File
      FileManagementSystemUI
      FileManager
      FileManagerImpl
      Image
      Video
  exercise
    Calculator1.java
    Vehicle1.java
  p1
    p1file
  p2
    QuizGame
    QuizGameClient
    QuizGameListener
    QuizGameServer
```



```
pract
  basic
    Addition
    conversion
    functions
    Pattern
  gui
    myFlowLayout
    RegistrationForm
    ScientificCalculator
  Networking
    Clientfile.java
      Clientfile
      FileClientTransfer
      FileClientTransferListener
      FileTransferClient
    MyClient
    MyServer
    Serverfile.java
      FileServerTransfer
      FileServerTransferListener
      FileTransferServer
      Serverfile
    SimpleClient
    SimpleServer
  AddressBook.java
    AddressBook
    Contact
project
  HospitalManagementSystemGUI
```

The below mentioned packages are inside the main package “com.amrita.jpl.cys21078”.

• In the package “EndSemester” I have included the code for the endsem question. In this document, it was explained under the exercise type:Exam.

• In the package “exercise” I have included the questions that were practiced in Lab hours. In this document, it was explained under exercise type: Exercise.

• The package “p1” contains the code for the p1 question. In this document, it was explained under the exercise type: Exam

• The package “p2” contains the code for the p2 question. In this document, it was explained under the exercise type: Exam.

• The package "practice" contains all the codes which were practised. In this document, it was explained under the exercise type: Practice.

• The package "project" contains the code for project. In this document, it was explained under the exercise type: Project.

Addition of Two Numbers

Exercise Type: Practice

Program No: 1

Github Commit Date: June 25th ,2023

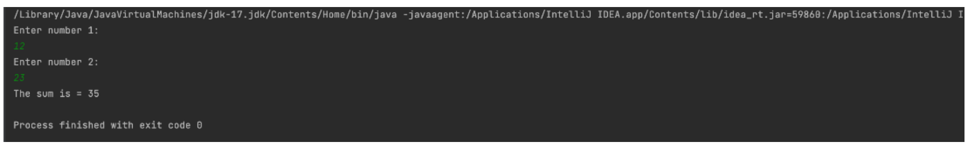
Problem Definition

The problem is to write a program that takes two numbers as input from the user and calculates their sum. The program should prompt the user to enter the numbers, read the input, parse it as integers, perform the addition operation, and finally display the result to the user.

Algorithm

1. Import the necessary package: **com.amrita.jpl.cys21078.pract.basic** and **java.util.Scanner**.
2. Declare a class named **Addition**.
3. Define the **main** method as the entry point of the program.
4. Create a new instance of the **Scanner** class to read input from the user.
5. Print the message "Enter number 1: ".
6. Read the input from the user as a string using the **nextLine()** method of the **Scanner** class.
7. Parse the input string to an integer using the **Integer.parseInt()** method and assign it to the variable **a**.
8. Print the message "Enter number 2: ".
9. Read the input from the user as a string using the **nextLine()** method.
10. Parse the input string to an integer and assign it to the variable **b**.
11. Calculate the sum of **a** and **b** and assign it to the variable **sum**.
12. Print the message "The sum is = " followed by the value of **sum**.

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=59860:/Applications/IntelliJ I
Enter number 1:
12
Enter number 2:
23
The sum is = 35

Process finished with exit code 0
```

Decimal to Binary and Hexadecimal Converter Program

Exercise Type: Practice

Program No: 2

Github Commit Date: June 25th ,2023

Problem Definition

The problem is to write a program that converts a decimal number entered by the user into its binary and hexadecimal representations. The program should prompt the user to enter a decimal number, read the input, perform the conversions using the provided methods, and display the binary and hexadecimal representations of the number to the user.

Algorithm

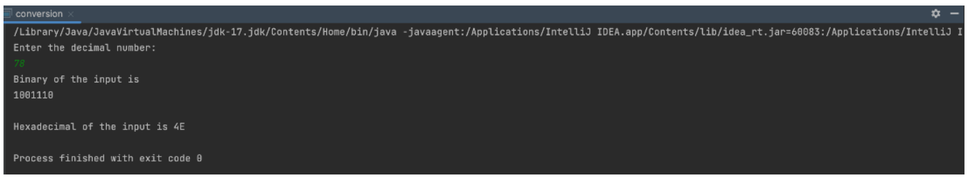
```
    1.Import the necessary package: com.amrita.jpl.cys21078.pract.basic and
java.util.Scanner.

    2.Declare a class named conversion.
    3.Add a documentation comment to provide an overview of the class and its purpose.
    4.Define the toBinary method that takes an integer decimal as input and converts it to
a binary representation.
    5.Inside the toBinary method:
        5.1. Declare an integer array binary with a size of 50 to store the binary digits.
        5.2. Declare an integer variable index and initialize it to 0.
        5.3. While decimal is greater than 0:
            ▪ Calculate the remainder of decimal divided by 2 and store it in
              binary[index].
            ▪ Divide decimal by 2 and update the value of decimal.
            ▪ Increment index by 1.
        5.4. Print the message "Binary of the input is ".
        5.5. Iterate from index - 1 to 0 in reverse order:
            ▪ Print the binary digits binary[i].
    6. Define the toHex method that takes an integer decimal as input and converts it to
a  hexadecimal representation.
    7.Inside the toHex method:
        7.1. Declare a StringBuilder named hex to build the hexadecimal
representation.
```

7.2. Declare a character array **hexchars** containing hexadecimal digits.
7.3. While **decimal** is greater than 0:
- Calculate the remainder of **decimal** divided by 16 and store it in **i**.
- Insert the corresponding hexadecimal digit **hexchars[i]** at the beginning of the **hex** string.
- Divide **decimal** by 16 and update the value of **decimal**.

7.4. Print a new line.
7.5. Print the message "Hexadecimal of the input is " followed by the value of **hex**.

8.Define the **main** method as the entry point of the program.
9.Inside the **main** method:
9.1. Declare an integer variable **decimal**.
9.2. Create a new instance of the **Scanner** class to read input from the user.
9.3. Print the message "Enter the decimal number: ".
9.4. Read an integer from the user and assign it to **decimal**.
9.5. Call the **toBinary** method with the **decimal** as the argument to convert it to binary.
9.6. Call the **toHex** method with the **decimal** as the argument to convert it to hexadecimal.

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60083:/Applications/IntelliJ I
Enter the decimal number:
78
Binary of the input is
1001110

Hexadecimal of the input is 4E

Process finished with exit code 0
```

Grade Calculation

Exercise Type: Practice

Program No: 3

Github Commit Date: June 25th ,2023

Problem Definition

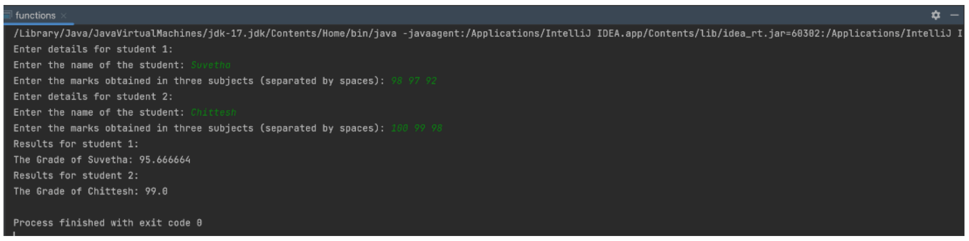
The problem is to write a program that takes details of two students, including their names and marks in three subjects, and calculates their grades based on the average marks. The program should prompt the user to enter the details for each student, calculate their grades, and display the results, including the student's name and grade.

Algorithm

```
1.Import the necessary package: com.amrita.jpl.cys21078.pract.basic and java.util.Scanner.

2.Define a class called "functions" with the necessary instance variables: grade, Name, Mark1, Mark2, and Mark3.
3.Implement a method called "publishresults()" to display the grade of the student along with their name.
    3.1.Print "The Grade of " concatenated with the Name variable, followed by the grade.
4.Implement a method called "Getdetails()" to get the details of a student from the user.
    4.1. Create a Scanner object to read user input.
    4.2. Prompt the user to enter the name of the student and store it in the Name variable.
    4.3. Prompt the user to enter the marks obtained in three subjects (separated by spaces) and store them in Mark1, Mark2, and Mark3 variables.
    4.4. Call the "calculated()" method to calculate the grade based on the entered marks.
5.Implement a method called "calculated()" to assign the grade of the student based on the calculated average marks.
    5.1. Calculate the average of Mark1, Mark2, and Mark3 by summing them up and dividing by 3.
    5.2. Assign the calculated average to the grade variable.
6.In the main method:
```

6.1. Create two instances of the "functions" class named student1 and student2.
6.2. Print "Enter details for student 1:" to prompt the user.
6.3. Call the "Getdetails()" method for student1 to get the details from the user.
6.4. Print "Enter details for student 2:" to prompt the user.
6.5. Call the "Getdetails()" method for student2 to get the details from the user.
6.6. Print "Results for student 1:" to display the heading for student1's results.
6.7. Call the "publishresults()" method for student1 to publish the results.
6.8. Print "Results for student 2:" to display the heading for student2's results.
6.9. Call the "publishresults()" method for student2 to publish the results.

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60302:/Applications/IntelliJ I
Enter details for student 1:
Enter the name of the student: Suvetha
Enter the marks obtained in three subjects (separated by spaces): 98 97 92
Enter details for student 2:
Enter the name of the student: Chittesh
Enter the marks obtained in three subjects (separated by spaces): 100 99 98
Results for student 1:
The Grade of Suvetha: 95.666664
Results for student 2:
The Grade of Chittesh: 99.0

Process finished with exit code 0
```

Pattern Printing

Exercise Type: Practice

Program No: 4

Github Commit Date: June 25th ,2023

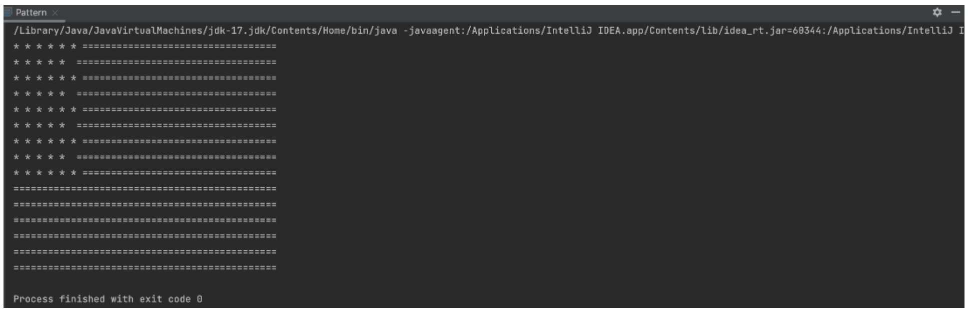
Problem Definition

The given code represents a class named pattern that generates and prints a specific pattern using asterisks and equal signs. The pattern consists of alternating lines of asterisks and equal signs, with a total of 9 asterisk lines and 6 equal sign lines.

Algorithm

1.Define a package called "com.amrita.jpl.cys21078.pract.basic".
2.Within the package, define a class called "Pattern".
3.In the main method:
3.1. Use a for loop to iterate from 0 to 8 (inclusive) for the outer loop for pattern generation.
3.2. Inside the loop, check if the current iteration is even (i%2 == 0).
3.3. If the current iteration is even, print "* * * * * * ====================================" to represent an asterisk line.
3.4. If the current iteration is odd, print "* * * * * =====================================" to represent an equal sign line.
3.5. Repeat steps 4.1 to 4.4 for a total of 9 times to generate the alternating pattern of asterisks and equal signs.
3.6. Use another for loop to iterate from 0 to 5 (inclusive) for the outer loop for the bottom line of equal signs.
3.7. Inside the loop, print "==" to represent the bottom line of equal signs.
3.8. Repeat steps 4.6 to 4.7 for a total of 6 times to generate the bottom line of equal signs.

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60344:/Applications/IntelliJ I
* * * * * * ==================================
* * * * *  ==================================
* * * * * * ==================================
* * * * *  ==================================
* * * * * * ==================================
* * * * *  ==================================
* * * * * * ==================================
* * * * *  ==================================
* * * * * * ==================================
==============================================
==============================================
==============================================
==============================================
==============================================
==============================================

Process finished with exit code 0
```

Inheritance

Exercise Type: Exercise

Program No: 5

Github Commit Date: June 25th ,2023

Problem Definition

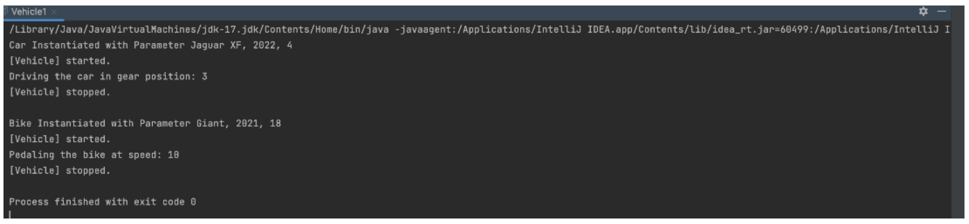
The objective is to create a Java program that defines a package structure, along with several classes, to model vehicles and their behaviors.

Algorithm

```
1.Define a package called "com.amrita.jpl.cys21078.exercise".
2.Create a class called "Vehicle".
    2.1. Declare a protected boolean variable called "run_status" to represent the running status of the vehicle.
    2.2. Implement a method called "start()" that sets the run_status to true and prints "[Vehicle] started".
    2.3. Implement a method called "stop()" that sets the run_status to false and prints "[Vehicle] stopped".
3.Create a class called "Car" that extends the "Vehicle" class.
    3.1. Declare private instance variables: model (String), year (int), and numOfWheels (int) to represent the car's model, year, and number of wheels.
    3.2. Implement a constructor to initialize the model, year, and numOfWheels variables.
    3.3. Implement a method called "drive(int gearPosition)" that checks if the car is running (run_status is true).
    3.4. If the car is running, print "Driving the car in gear position: " followed by the gearPosition.
    3.5. If the car is not running, print "Error: Cannot drive the car as it is not running.".
4.Create a class called "Bike" that extends the "Vehicle" class.
    4.1. Declare private instance variables: brand (String), year (int), and numOfGears (int) to represent the bike's brand, year, and number of gears.
    4.2. Implement a constructor to initialize the brand, year, and numOfGears variables.
    4.3. Implement a method called "pedal(int pedalSpeed)" that checks if the bike is running (run_status is true).
    4.4. If the bike is running, print "Pedaling the bike at speed: " followed by the pedalSpeed.
```

```
            4.5. If the bike is not running, print "Error: Cannot pedal the bike as it is not running.".
    5.Create a class called "Vehicle1".
            5.1. In the main method:
                5.1.1. Print "Car Instantiated with Parameter Jaguar XF, 2022, 4".
                5.1.2. Create an instance of the "Car" class named "car" with the parameter values "Jaguar XF", 2022, 4.
                5.1.3. Call the "start()" method on the "car" instance.
                5.1.4. Call the "drive(3)" method on the "car" instance.
                5.1.5. Call the "stop()" method on the "car" instance.
                5.1.6. Print an empty line.
                5.1.7. Print "Bike Instantiated with Parameter Giant, 2021, 18".
                5.1.8. Create an instance of the "Bike" class named "bike" with the parameter values "Giant", 2021, 18.
                5.1.9. Call the "start()" method on the "bike" instance.
                5.1.10. Call the "pedal(10)" method on the "bike" instance.
                5.1.11. Call the "stop()" method on the "bike" instance.
    6.End the program execution.
```

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60499:/Applications/IntelliJ I
Car Instantiated with Parameter Jaguar XF, 2022, 4
[Vehicle] started.
Driving the car in gear position: 3
[Vehicle] stopped.

Bike Instantiated with Parameter Giant, 2021, 18
[Vehicle] started.
Pedaling the bike at speed: 10
[Vehicle] stopped.

Process finished with exit code 0
```

Interface

Exercise Type: Exercise

Program No: 6

Github Commit Date: June 25th ,2023

Problem Definition

It is to create a calculator program that allows the user to perform various arithmetic operations on two numbers.

Algorithm

1.Import the required packages: **java.util.Scanner**.

2.Declare an interface **Calculator** with two methods:

2.1. calculate(double num1, double num2) - calculates the result of an operation on **num1** and **num2**.

2.2. getOperationSymbol() - returns a string representing the operation symbol.

3.Implement four classes (**Addition**, **Subtraction**, **Multiplication**, and **Division**) that implement the **Calculator** interface:

3.1. Each class provides its own implementation for the **calculate** and **getOperationSymbol** methods.

3.2. The **Division** class handles the special case of division by zero by displaying an error message and returning -1.

4.Define the **Calculator1** class with the **main** method:

4.1. Create a **Scanner** object to read input from the user.

4.2. Read two double values (**num1** and **num2**) from the user using the **Scanner**.

4.3. Close the **Scanner**.

4.4. Create an array **calculators** of type **Calculator** containing instances of all four calculator classes.

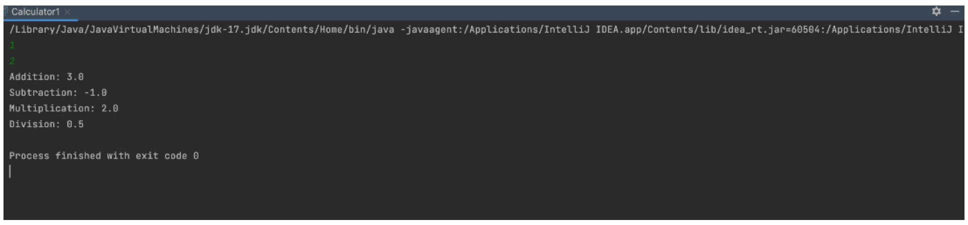
4.5.Iterate through each **Calculator** object in the **calculators** array:

- Try to calculate the result by calling the **calculate** method with **num1** and **num2**.
- If an **ArithmeticException** is thrown (division by zero), catch it and print the error message.
- Print the operation symbol and the result.

5.End the **main** method.

6.End the **Calculator1** class

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60504:/Applications/IntelliJ I
1
2
Addition: 3.0
Subtraction: -1.0
Multiplication: 2.0
Division: 0.5

Process finished with exit code 0
```

GUI

Exercise Type: Practice

Program No: 7

Github Commit Date: June 25th ,2023

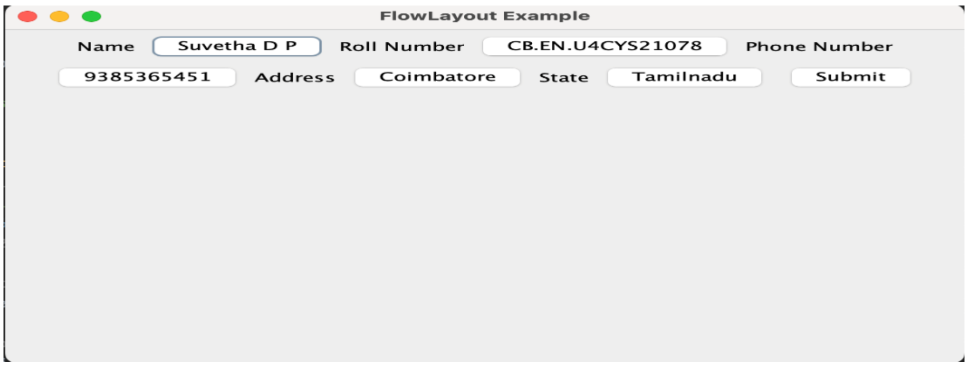
Problem Definition

Creating a program that allows users to manage records by performing various operations such as adding, deleting, and updating records. The program should provide an intuitive user interface and ensure data integrity and security.

Algorithm

1.Import the required package **javax.swing** for GUI components.
2.Define a class **myFlowLayout** that extends **JFrame** to create a form using **FlowLayout**.
3.In the class constructor **myFlowLayout()**:
    3.1. Set the title of the JFrame to "FlowLayout Example".
    3.2. Set the size of the JFrame to width 600 and height 80.
    3.3. Set the default close operation to **EXIT_ON_CLOSE**.
    3.4. Set the layout manager of the JFrame to **FlowLayout**.
    3.5. Create five **JLabel** objects (**label1**, **label2**, **label3**, **label4**, **label5**) with appropriate names.
    3.6. Create six **JButton** objects (**button1**, **button2**, **button3**, **button4**, **button5**, **button6**) with appropriate labels.
    3.7. Add the labels and buttons to the frame using the **add()** method in the order they should appear.
    3.8. Set the visibility of the JFrame to **true**.
4.Define the **main** method:
    4.1. Create an instance of **myFlowLayout** using the constructor.
5.End the **main** method.
6.End the **myFlowLayout** class.

Output



Name Suvetha D P Roll Number CB.EN.U4CYS21078 Phone Number
9385365451 Address Coimbatore State Tamilnadu Submit

Registration Form

Exercise Type: Practice

Program No: 8

Github Commit Date: June 25th ,2023

Problem Definition

Create a registration form GUI with name, email, and password fields. Implement registration logic upon button click and display a success message. Ensure proper imports and utilize the main method to launch the GUI on the Event Dispatch Thread.

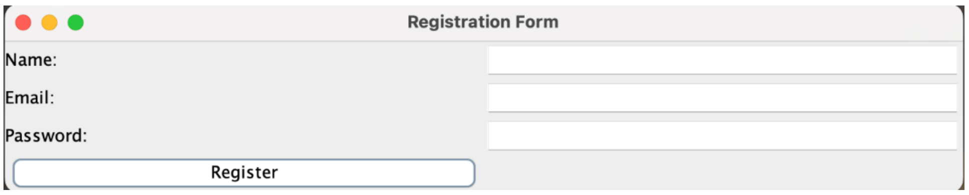
Algorithm

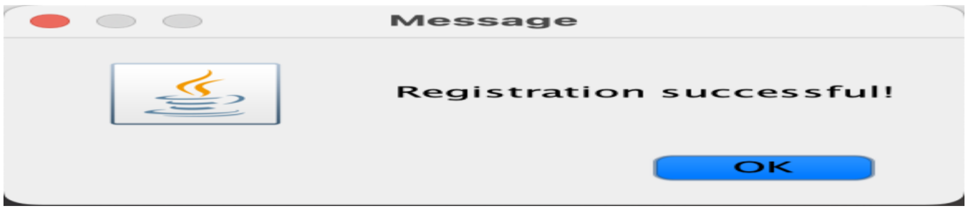
```
1.Import the necessary classes and packages for creating the GUI.
2.Define a class named RegistrationForm and declare the required instance variables: frame, panel, nameLabel, emailLabel, passwordLabel, nameField, emailField, passwordField, and registerButton.
3.Create a constructor for the RegistrationForm class.
    3.1. Initialize the frame object with the title "Registration Form".
    3.2. Create a new JPanel object with a GridLayout of 4 rows and 2 columns and assign it to the panel variable.
    3.3. Create JLabel objects for the name, email, and password labels.
    3.4. Create JTextField objects for the name and email fields.
    3.5. Create a JPasswordField object for the password field.
    3.6. Create a JButton object for the register button.
    3.7. Add the labels, fields, and button to the panel in the desired order.
    3.8. Add the panel to the frame.
    3.9. Set the default close operation for the frame to exit the application when closed.
    3.10. Pack the frame to adjust its size based on the components.
    3.11. Set the frame visibility to true. l. Register an ActionListener for the register button.
        3.11.1.When the button is clicked, the actionPerformed method will be invoked.
        3.11.2.Get the text from the name and email fields.
        3.11.3.Get the password as a character array from the password field.
        3.11.4.Perform the registration logic here (not specified in the given code).
        3.11.5.Display a message dialog with the "Registration successful!" message.
4.Define the main method.
```

4.1. Invoke the **SwingUtilities.invokeLater** method to ensure the GUI is created and updated on the Event Dispatch Thread.

4.2. Inside the **run** method of the **Runnable** interface, create a new instance of the **RegistrationForm** class.

Output





Registration successful!

Scientific Calculator

Exercise Type: Practice

Program No: 9

Github Commit Date: June 25th ,2023

Problem Definition

Create a scientific calculator application with GUI components using Java Swing. Implement functionality for arithmetic operations, trigonometric functions, and logarithms. Allow user input through buttons and keyboard events, and display results on a non-editable text field.

Algorithm

```
1.Create a class named "ScientificCalculator" that extends JFrame to represent the scientific calculator application.
2.Declare private instance variables for the display JTextField, operator, operand1, and a boolean flag to track if an operator button is clicked.
3.Define a constructor for the ScientificCalculator class:
    3.1. Set the title of the JFrame to "Scientific Calculator".
    3.2. Set the default close operation to exit the application when the frame is closed.
    3.3. Create the display JTextField with a width of 10 and set it to be non-editable.
    3.4. Create number buttons (0-9) and operator buttons (+, -, *, /, sin, cos, tan, log).
    3.5. Create the clear button. f. Set the layout of the JFrame to BorderLayout.
    3.6. Create panels for the number buttons and operator buttons.
    3.7. Create a main panel to hold the display and buttons, and add the panels to it.
    3.8. Add the main panel to the content pane of the JFrame.
    3.9. Attach action listeners to the number buttons, operator buttons, clear button, and display field's key events.
    3.10. Pack and center the JFrame on the screen.
4.Define an inner class NumberButtonListener that implements ActionListener to handle number button clicks:
    4.1. Get the clicked button's text and append it to the display text.
5.Define an inner class OperatorButtonListener that implements ActionListener to handle operator button clicks:
    5.1. Get the clicked button's text and current display text.
    5.2. If an operator button was not previously clicked, store the first operand, operator, and clear the display.
```

5.3. If an operator button was previously clicked, store the second operand, perform the calculation using the first operand, second operand, and operator, display the result, update the first operand, and operator.

6.Define an inner class ClearButtonListener that implements ActionListener to handle the clear button click:

6.1. Clear the display text, reset the first operand, operator, and set the operator clicked flag to false.

7.Define an inner class DisplayKeyListener that implements KeyListener to handle key events in the display field:

7.1. Implement the keyTyped method (not used).

7.2. Implement the keyPressed method:

7.2.1. Get the key code.

7.2.2. If the key is a numeric key, append the corresponding digit to the display text.

7.2.3.If the key is an operator key, call the operatorButtonClicked method with the operator as an argument. c. Implement the keyReleased method (not used).

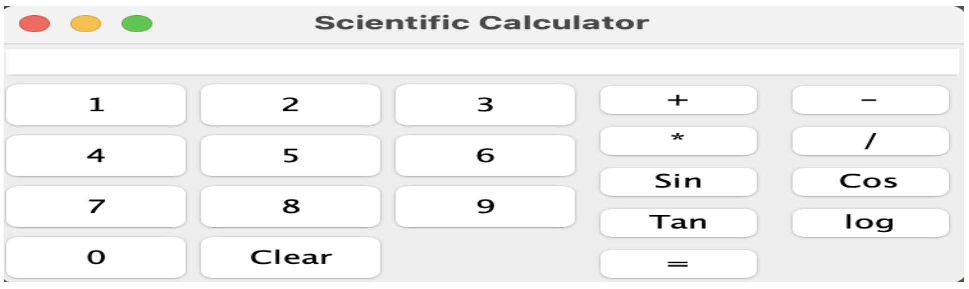
8.Define a performCalculation method that takes operand1, operand2, and operator as arguments and performs the calculation based on the operator, returning the result.

9.Define a performCalculation method that takes operand1 and Operator as arguments and performs the calculation based on the operator (sin, cos, tan, log), returning the result.

10.In the main method:

10.1. Use SwingUtilities.invokeLater to create and show an instance of the ScientificCalculator class.

Output



Socket Programming

Exercise Type: Practice

Program No: 10

Github Commit Date: June 25th ,2023

Problem Definition

The problem involves implementing a client-server communication system using sockets in Java. The server listens for incoming client connections, receives messages from the client, and responds accordingly. The client sends messages to the server and displays the server's responses.

Server

Algorithm

1.Import the required classes from the java.net and java.io packages.
2.Define a class named "MyServer".
3.In the main method, declare a ServerSocket object "ss" and initialize it with a new ServerSocket instance, passing the port number 2445 as an argument.
4.Call the accept() method on the ServerSocket object "ss" to listen for incoming client connections. This method will block until a client connection is established.
5.Once a client connection is accepted, create a Socket object "s" to represent the client socket.
6.Create a DataInputStream object "din" and initialize it with the input stream from the client socket "s" using the getInputStream() method.
7.Create a DataOutputStream object "dout" and initialize it with the output stream from the client socket "s" using the getOutputStream() method.
8.Create a BufferedReader object "br" and initialize it with a new InputStreamReader instance, which reads input from the standard input (System.in).
9.Declare two String variables "str" and "str2" to store the messages received from the client and entered by the server, respectively.
10.Enter a while loop with the condition "!str.equals("stop")" to keep the server running until the client sends the "stop" message.
11.Inside the loop, read a UTF-8 encoded string from the client using the readUTF() method of the DataInputStream "din" and store it in the "str" variable.
12.Print the received message from the client on the server console using System.out.println().
13.Read a line of text entered by the server from the BufferedReader "br" and store it in the "str2" variable.

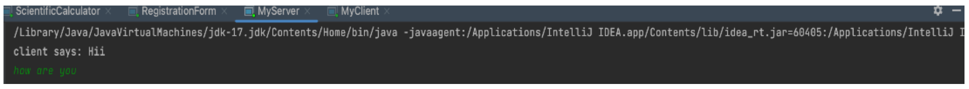
14.Write the value of "str2" as a UTF-8 encoded string to the client using the writeUTF() method of the DataOutputStream "dout".
15.Flush the data in the output stream using the flush() method of the DataOutputStream "dout" to ensure it is sent immediately.
16.Repeat steps 11 to 15 until the client sends the "stop" message.
17.After the loop exits, close the DataInputStream "din", the client Socket "s", and the ServerSocket "ss" to release the associated resources.

Client

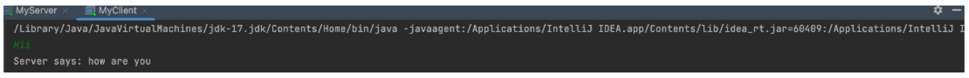
Algorithm

1.Import the required classes from the java.net and java.io packages.
2.Define a class named "MyClient".
3.In the main method, declare a Socket object "s" and initialize it with a new Socket instance, passing the host name "localhost" (referring to the local machine) and the port number 2445 as arguments.
4.Create a DataInputStream object "din" and initialize it with the input stream from the client socket "s" using the getInputStream() method.
5.Create a DataOutputStream object "dout" and initialize it with the output stream from the client socket "s" using the getOutputStream() method.
6.Create a BufferedReader object "br" and initialize it with a new InputStreamReader instance, which reads input from the standard input (System.in).
7.Declare two String variables "str" and "str2" to store the messages entered by the client and received from the server, respectively.
8.Enter a while loop with the condition "!str.equals("stop")" to keep the client running until the client sends the "stop" message.
9.Inside the loop, read a line of text entered by the client from the BufferedReader "br" and store it in the "str" variable.
10.Write the value of "str" as a UTF-8 encoded string to the server using the writeUTF() method of the DataOutputStream "dout".
11.Flush the data in the output stream using the flush() method of the DataOutputStream "dout" to ensure it is sent immediately.
12.Read a UTF-8 encoded string from the server using the readUTF() method of the DataInputStream "din" and store it in the "str2" variable.
13.Print the received message from the server on the client console using System.out.println().
14.Repeat steps 9 to 13 until the client sends the "stop" message.
15.After the loop exits, close the DataOutputStream "dout" and the client Socket "s" to release the associated resources.

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60405:/Applications/IntelliJ I
client says: Hii
how are you
```



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60409:/Applications/IntelliJ I
Hii
Server says: how are you
```

GUI Using Address Book

Exercise Type: Practice

Program No: 11

Github Commit Date: June 25th ,2023

Problem Definition

The problem involves creating an address book application using Java Swing. The application should allow users to add, edit, and delete contacts, displaying them in a table with columns for name, phone number, and email. Input validation for phone numbers and emails should be implemented, and the application should provide a graphical user interface for interaction.

Algorithm

1.Import the required classes from the javax.swing package, javax.swing.table.DefaultTableModel, java.awt, and java.util packages.

2.Define a class named "AddressBook".

3.Declare instance variables: JFrame "frame" to represent the application window, JTable "table" to display the contact information, DefaultTableModel "tableModel" to manage the table data, and List<Contact> "contacts" to store the contacts.

4.Create a constructor for the AddressBook class. Initialize the contacts list as an empty ArrayList and call the initialize() method.

5.Define the initialize() method:

a. Create a new JFrame and set its title, size, and default close operation.

b. Set the layout of the frame's content pane to BorderLayout.

c. Create the table with columns by initializing a DefaultTableModel and adding three columns: "Name", "Phone Number", and "Email".

d. Create a JTable using the table model and wrap it in a JScrollPane. Add the scroll pane to the center of the frame's content pane.

e. Create buttons using a JPanel and add it to the bottom of the frame's content pane.

f. Add an ActionListener to the "Add" button to display the add contact dialog when clicked.

g. Add an ActionListener to the "Edit" button to retrieve the selected row from the table, retrieve the corresponding Contact object from the contacts list, and display the edit contact dialog.

h. Add an ActionListener to the "Delete" button to retrieve the selected row from the table, remove the corresponding Contact object from the contacts list, and remove the row from the table model.

i. Set the frame visible.

6.Define the showAddContactDialog() method:

a. Create JTextFields for name, phone number, and email.

b. Create a JPanel with a GridLayout to display the input fields and labels.

c. Show a JOptionPane dialog with the panel and OK/Cancel buttons.

d. If the user clicks OK, retrieve the text entered in the fields and validate the input.

e. If the input is valid, create a new Contact object and add it to the contacts list.

f. Add a new row to the table model with the contact details.

g. If the input is invalid, show an error message using JOptionPane.

7.Define the showEditContactDialog(Contact contact) method:

a. Create JTextFields for name, phone number, and email and initialize them with the values from the given Contact object.

b. Create a JPanel with a GridLayout to display the input fields and labels.

c. Show a JOptionPane dialog with the panel and OK/Cancel buttons.

d. If the user clicks OK, retrieve the text entered in the fields and validate the input.

e. If the input is valid, update the Contact object with the new values.

f. Update the corresponding row in the table model with the updated contact details.

g. If the input is invalid, show an error message using JOptionPane.

7.Define the isValidPhoneNumber(String phoneNumber) method:

a. Implement the phone number validation logic.

b. Return true if the phone number is valid; otherwise, return false.

8.Define the isValidEmail(String email) method:

a. Implement the email validation logic.

b. Return true if the email is valid; otherwise, return false.

9.Define the main(String[] args) method:

a. Use the EventQueue.invokeLater() method to create an instance of the AddressBook class on the event dispatch thread.

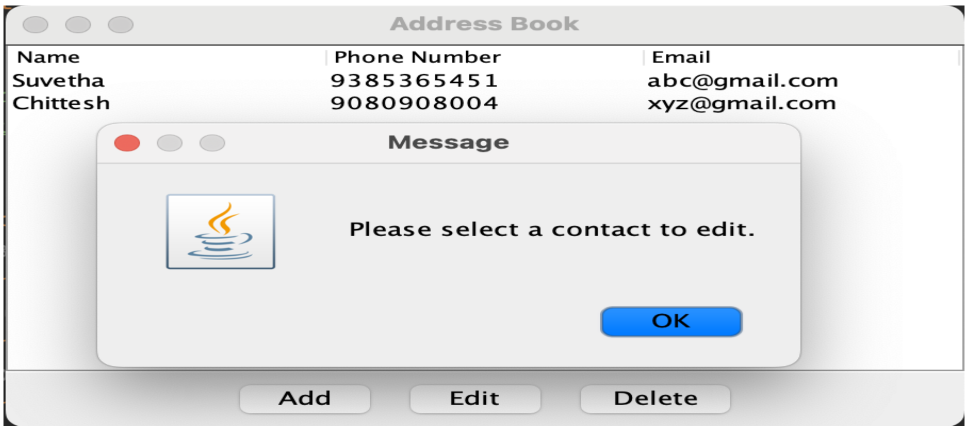
10.Define a Contact class to represent a contact with name, phone number, and email fields.

a. Define instance variables for the contact's name, phone number, and email.

b. Create a constructor to initialize the contact's fields.

c. Define getter and setter methods for the name, phone number, and email fields.

Output



```
Name            Phone Number    Email
Suvetha         9385365451      abc@gmail.com
Chittesh        9080908004      xyz@gmail.com
```

Menu Driven Program

Exercise Type: Periodical 1

Program No: 12

Github Commit Date: June 25th ,2023

Problem Definition

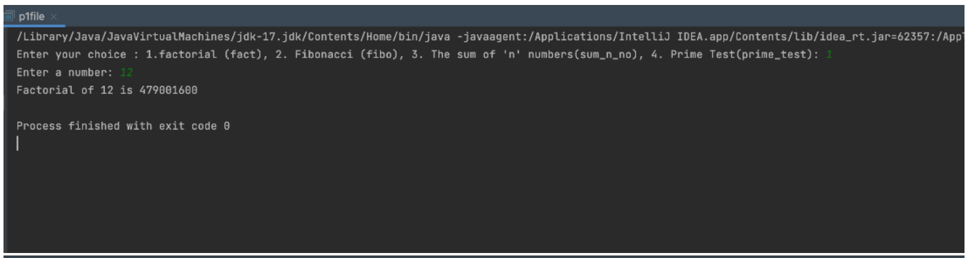
The problem is to create a Java program called "p1file" that presents a menu to the user with options for factorial calculation, Fibonacci sequence generation, sum of 'n' numbers calculation, and prime number testing. The program should read user input, perform the selected operation based on the input, and display the result accordingly. The program should also include methods for each operation, such as fact() for factorial calculation, fibo() for Fibonacci sequence generation, sum_n_no() for sum calculation, and prime_test() for prime number testing.

Algorithm

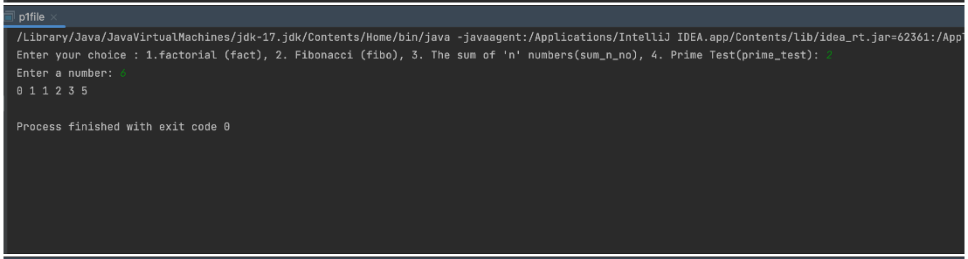
```
1.Import the necessary packages and classes.
2.Define a class named "p1file".
3.Inside the class, define the main method.
4.Create a Scanner object to read user input.
5.Display a menu of choices for the user to select from: factorial, Fibonacci, sum of 'n' numbers, or prime test.
6.Read the user's choice using the nextInt() method of the Scanner object.
7.Use a switch-case statement based on the user's choice to execute the corresponding operation.
8.For choice 1 (factorial):
    a. Prompt the user to enter a number.
    b. Read the number from the user.
    c. If the number is less than 0, display an error message and break.
    d. Call the fact() method with the entered number to calculate the factorial.
    e. Display the factorial value.
9.For choice 2 (Fibonacci):
    a. Prompt the user to enter a number.
    b. Read the number from the user.
    c. If the number is less than 0, display an error message and break.
    d. Call the fibo() method with the entered number to print the Fibonacci sequence.
10.For choice 3 (sum of 'n' numbers):
    a. Prompt the user to enter a number.
    b. Read the number from the user.
```

```
        c. If the number is less than or equal to 0, display an error message and break.
        d. Call the sum_n_no() method with the entered number to calculate the sum.
        e. Display the sum of the first 'n' numbers.
    11.For choice 4 (prime test):
        a. Prompt the user to enter a number.
        b. Read the number from the user.
        c. If the number is less than 0, display an error message and break.
        d. Call the prime_test() method with the entered number to check if it is prime.
        e. Display whether the number is prime or not.
    12.If none of the valid choices are selected, display an error message.
    13.End the switch-case statement.
    14.Define the fact() method:
        a. Check if the input number is 0 and return 1.
        b. Otherwise, recursively call the fact() method with n-1 and multiply it with n.
    15.Define the fibo() method:
        a. Initialize variables a0 and a1 as 0 and 1 respectively.
        b. Print the initial values a0 and a1.
        c. Use a loop to calculate the Fibonacci sequence by adding a0 and a1, updating the variables, and printing the result.
    16.Define the sum_n_no() method:
        a. Initialize a variable sum as 0.
        b. Use a loop to iterate from 1 to n and add each number to the sum.
        c. Return the sum.
    17.Define the prime_test() method:
        a. Check if the input number is less than or equal to 1 and return false.
        b. Use a loop to iterate from 2 to the square root of the number.
        c. If the number is divisible by any value in the loop, return false.
        d. If the loop completes without finding a divisor, return true.
    18.End the class.
```

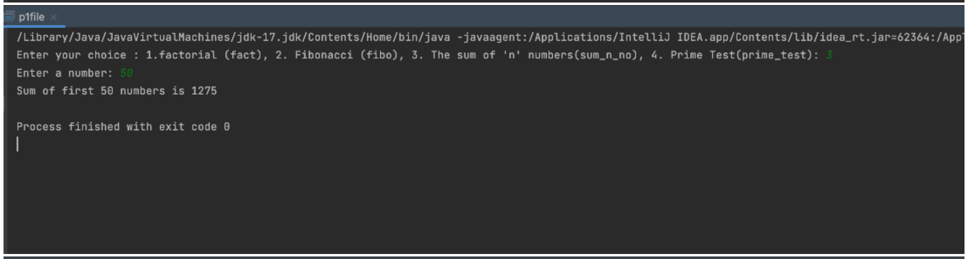
Output



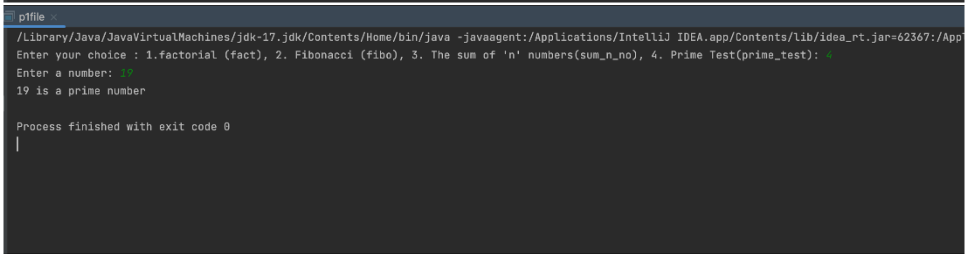
```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62357:/App
Enter your choice : 1.factorial (fact), 2. Fibonacci (fibo), 3. The sum of 'n' numbers(sum_n_no), 4. Prime Test(prime_test): 1
Enter a number: 12
Factorial of 12 is 479001600

Process finished with exit code 0
```



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62361:/App
Enter your choice : 1.factorial (fact), 2. Fibonacci (fibo), 3. The sum of 'n' numbers(sum_n_no), 4. Prime Test(prime_test): 2
Enter a number: 6
0 1 1 2 3 5

Process finished with exit code 0
```



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62364:/App
Enter your choice : 1.factorial (fact), 2. Fibonacci (fibo), 3. The sum of 'n' numbers(sum_n_no), 4. Prime Test(prime_test): 3
Enter a number: 50
Sum of first 50 numbers is 1275

Process finished with exit code 0
```



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62367:/App
Enter your choice : 1.factorial (fact), 2. Fibonacci (fibo), 3. The sum of 'n' numbers(sum_n_no), 4. Prime Test(prime_test): 4
Enter a number: 19
19 is a prime number

Process finished with exit code 0
```

Socket Programming

Exercise Type: periodical 2

Program No: 13

Github Commit Date: June 25th ,2023

Problem Definition

Implement a Quiz Game server and client system that allows multiple clients to connect and participate in a quiz game. The server should be responsible for hosting the game, sending questions to clients, receiving answers, and evaluating them. The client should connect to the server, receive questions, provide answers, and receive evaluation results.

Algorithm

Server

1.The **QuizGameServer** class extends the **QuizGame** class, indicating that it inherits some functionality from it.

2.The **QuizGameServer** class initializes two lists: **questions** and **answers**, containing the quiz questions and their corresponding answers, respectively.

3.The **startGame()** method is overridden from the **QuizGame** class and serves as the entry point for the game. It performs the following steps:

3.1. Creates a **ServerSocket** on port 1234 to listen for incoming client connections.

3.2. Accepts a client connection and establishes communication via a socket.

3.3. Creates **ObjectOutputStream** and **ObjectInputStream** to send and receive objects over the network.

3.4. Iterates through each question in the **questions** list:

- Retrieves the current question from the **questions** list.
- Sends the question to the client using the output stream.
- Waits for the client's answer by reading an object from the input stream.
- Evaluates the client's answer against the correct answer from the **answers** list.
- Sends the evaluation result (true for correct, false for incorrect) to the client using the output stream.

3.5. Sends "Game Over" message to the client to indicate the end of the game.

3.6. Closes the input/output streams and the client socket.

3.7. Prints a message indicating the client has disconnected.

3.8. Prints a message indicating the game is over.

4.The **askQuestion()** and **evaluateAnswer()** methods are overridden from the **QuizGame** class but are empty in the **QuizGameServer** class. These methods are not used in this specific implementation.

5.The **askQuestion()** method takes a **question** and an **ObjectOutputStream** as parameters, sends the question to the client via the output stream, and prints the question to the console.

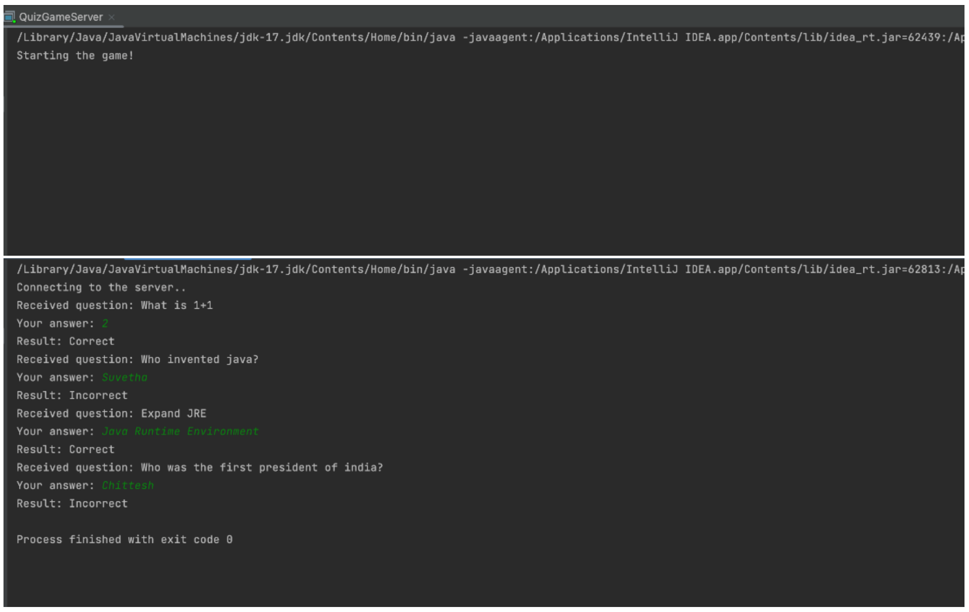
6.The **evaluateAnswer()** method takes an **answer** and an **ObjectOutputStream** as parameters. It compares the provided answer with the correct answer from the **answers** list, sends the evaluation result to the client via the output stream, and prints the answer, correct answer, and the result (correct/incorrect) to the console.

7.The **main()** method creates an instance of **QuizGameServer**, calls the **startGame()** method to initiate the quiz game, and runs the game on the server.

Client

1.Import the required packages (java.io., *java.net.*, java.util.*).
2.Define a class named "QuizGameClient" that extends the "QuizGame" class.
3.Override the "startGame()" method inherited from the "QuizGame" class.
4.Inside the "startGame()" method:
4.1. Print "Connecting to the server.." message.
4.2. Create a new Socket object, connecting to the server using the localhost address and port 1234.
4.3. Create ObjectOutputStream and ObjectInputStream objects using the socket's output and input streams.
4.4. Start a loop that continues until "Game Over" message is received.
4.5. Read a question (as a String) from the server using the ObjectInputStream.
4.6. If the received question is "Game Over," break the loop.
4.7. Print the received question.
4.8. Prompt the user to enter their answer and read it using a Scanner object.
4.9. Write the answer to the server using the ObjectOutputStream and flush it.
4.10. Read the correctness of the answer (as a boolean) from the server using the ObjectInputStream.
4.11. Print the result (either "Correct" or "Incorrect") based on the received correctness.
4.12. Close the ObjectOutputStream and ObjectInputStream.
5.Override the "askQuestion()" method (empty implementation).
6.Override the "evaluateAnswer(String answer)" method (empty implementation).
7.Define the "main(String[] args)" method:
7.1. Create an instance of the QuizGameClient class.
7.2. Call the "startGame()" method on the created instance.

Output



```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62439:/Ap
Starting the game!
```

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62813:/Ap
Connecting to the server..
Received question: What is 1+1
Your answer: 2
Result: Correct
Received question: Who invented java?
Your answer: Suvetha
Result: Incorrect
Received question: Expand JRE
Your answer: Java Runtime Environment
Result: Correct
Received question: Who was the first president of india?
Your answer: Chittesh
Result: Incorrect

Process finished with exit code 0
```

GUI

Exercise Type: End Semester

Program No: 14

Github Commit Date: June 25th ,2023

Problem Definition

Create a file management system with a graphical user interface (GUI) that allows users to add, delete, and view files. The system should support different types of files such as documents, images, and videos, each with specific attributes. The GUI should provide input fields for file details, buttons for performing operations, and a table to display the files. The system should use a FileManagerImpl class that implements the FileManager interface, which provides methods for adding, deleting, and retrieving files. The GUI should handle button clicks and update the file list displayed in the table accordingly. The goal is to develop a user-friendly file management system that simplifies file handling tasks for users

Algorithm

1.Import the required packages and classes for GUI components.

2.Define the **File** class with **fileName** and **fileSize** attributes along with getter, setter, and display methods.

3.Define the **Document**, **Image**, and **Video** classes that extend the **File** class, each having additional attributes specific to their type.

4.Define the **FileManager** interface with methods **addFile**, **deleteFile**, and **getFiles**.

5.Implement the **FileManagerImpl** class that implements the **FileManager** interface. It uses an **ArrayList<File>** to store the files and provides the implementation for the methods.

6.Define the **FileManagementSystemUI** class that creates the graphical user interface for the file management system.

7.Initialize the **FileManagerImpl** and create the UI components.

8.Implement methods in the **FileManagementSystemUI** class to handle button clicks and perform the necessary operations.

9.In the **addFileButtonClicked** method, retrieve the file details from the input fields and create a **Document**, **Image**, or **Video** object based on the selected file type. Add the file to the **FileManagerImpl** and update the table model.

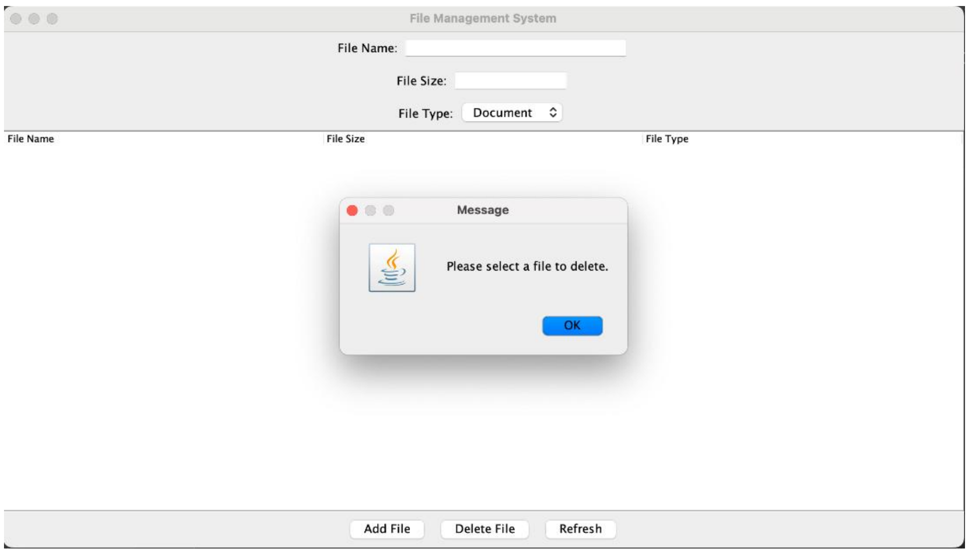
10.In the **deleteFileButtonClicked** method, get the selected row from the table, retrieve the file name, and delete the file from the **FileManagerImpl** and table model.

11.In the **refreshButtonClicked** method, clear the table and retrieve the files from the **FileManagerImpl**. Iterate over the files and add rows to the table based on the file type.

12.Implement utility methods **clearFields** and **clearTable** to clear the input fields and table, respectively.

13.In the **main** method, invoke the **FileManagementSystemUI** constructor using **SwingUtilities.invokeLater** to create the GUI on the event dispatch thread.

Output



Please select a file to delete.

Hospital Management System

Exercise Type: Project

Program No: 15

Github Commit Date: June 25th ,2023

Problem Definition

Develop a user-friendly Hospital Management System GUI to streamline administrative tasks and enhance patient record management. The system should provide secure login interfaces for doctors and patients, allowing doctors to view patient details and the appointments and allowing patients to see their medical history, scheduled appointments and view test reports. The system must prioritize data security and privacy to prevent unauthorized access to sensitive patient information.

Algorithm

1.The HospitalManagementSystemGUI class initializes the GUI components in the initialize() method and sets up the main frame.

2.When the "Doctor" or "Patient" button is clicked, it calls the respective methods: showDoctorLoginPage() or showPatientLoginPage(). These methods create separate login frames for doctors and patients.

3.The login frames contain input fields for username and password. When the login button is clicked, the corresponding authenticateDoctor() or authenticatePatient() method is called to authenticate the user's credentials.

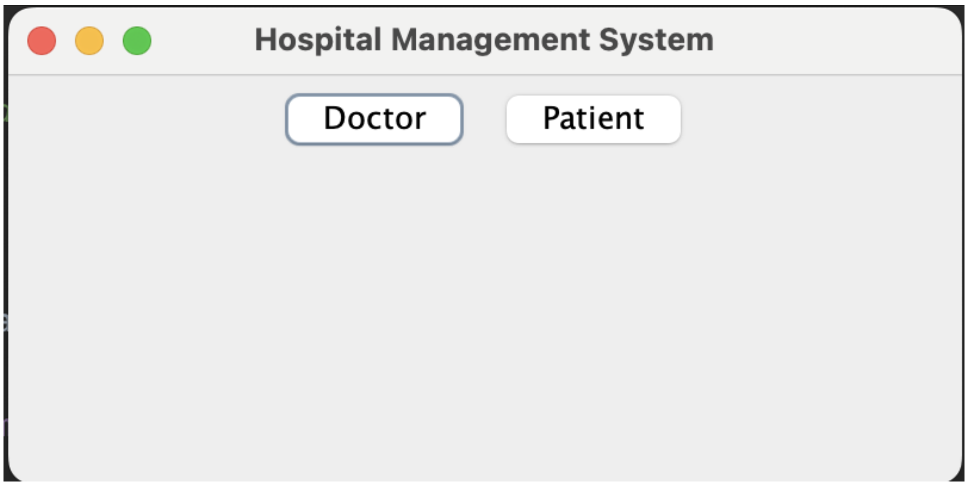
4.If the authentication is successful, the login frame is closed, and the respective showDoctorPage() or showPatientPage() method is called to display the doctor's or patient's main page.

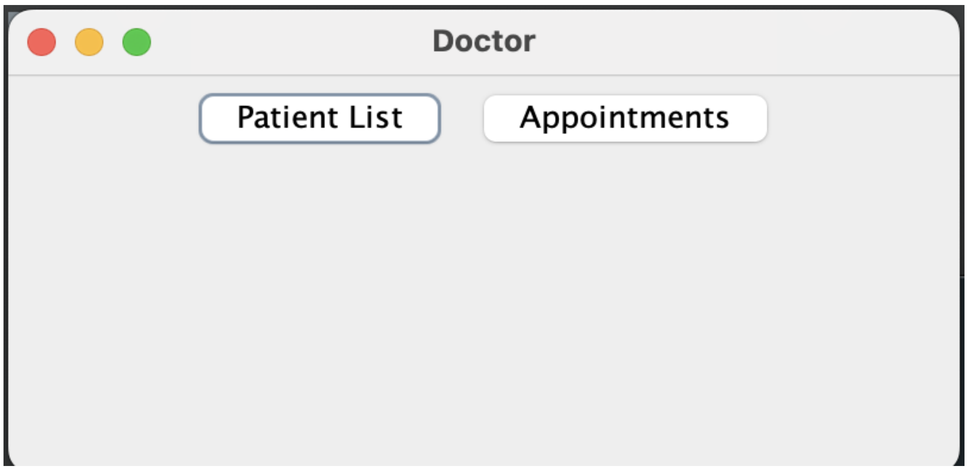
5.The main pages contain buttons for different actions such as viewing patient lists, appointments, medical history, and test reports.

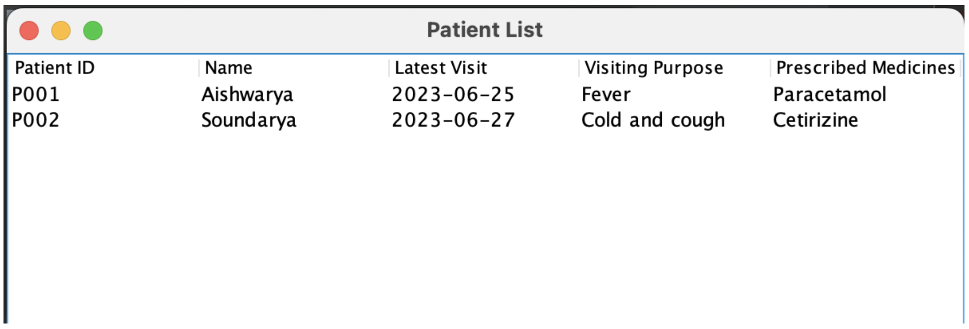
6.When a button is clicked, it calls the corresponding methods such as showPatientListTable(), showAppointmentsPage(), showMedicalHistoryPage(), or showTestReportsPage(). These methods create separate frames to display the requested information.

7.The frames for patient lists, appointments, medical history, and test reports contain tables with relevant data.

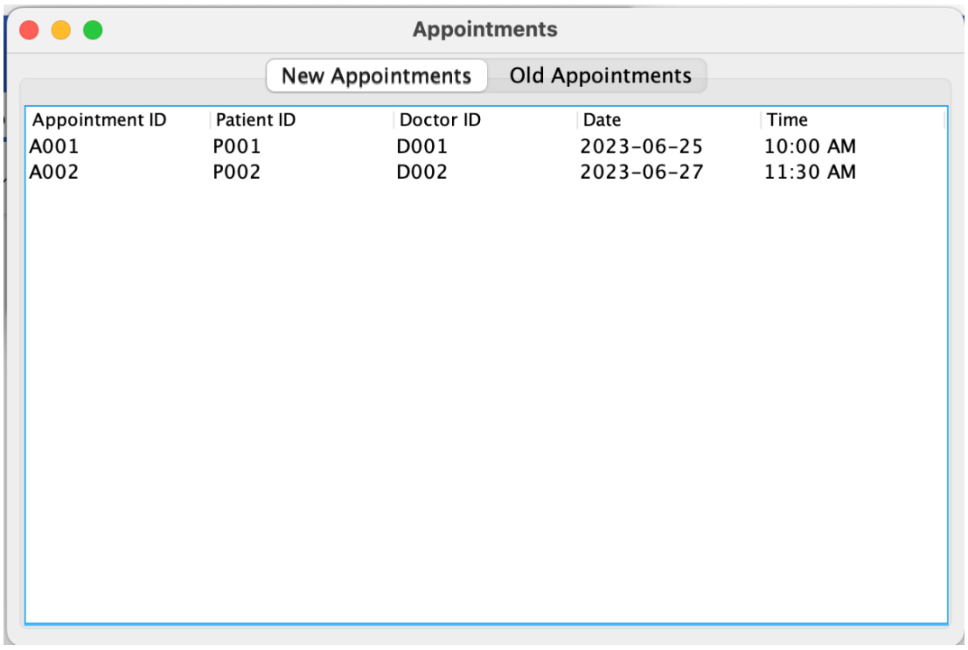
8.The user can interact with the displayed information and close the frames when done.



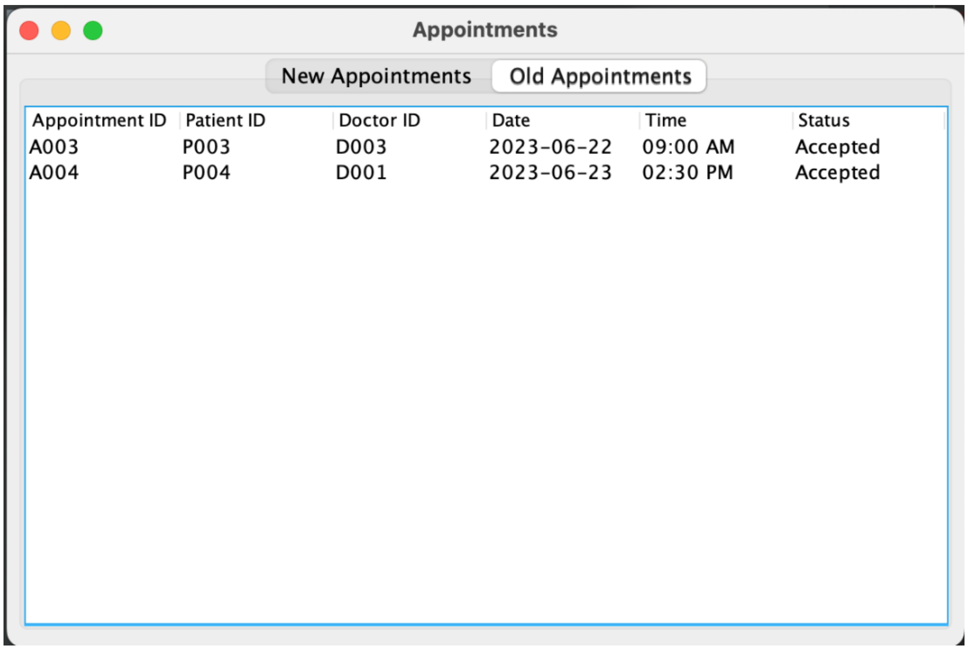



Patient List

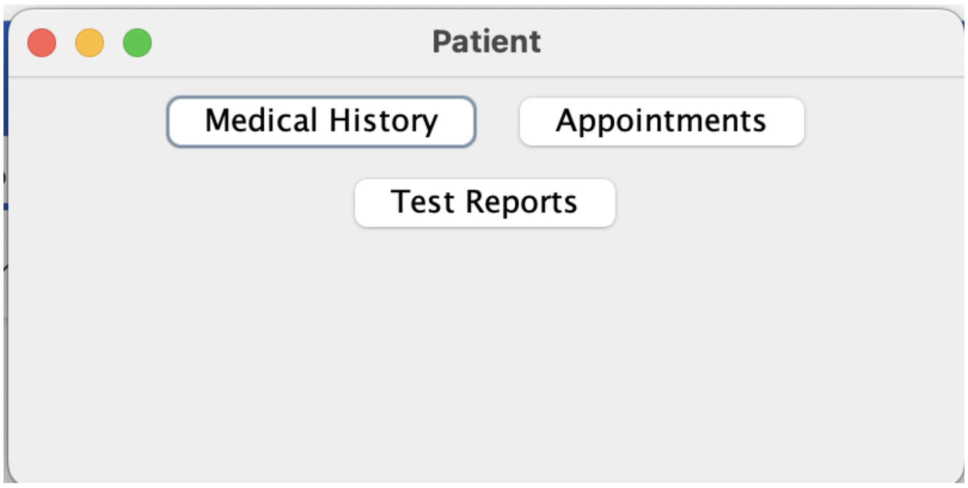
Patient ID	Name	Latest Visit	Visiting Purpose	Prescribed Medicines
P001	Aishwarya	2023-06-25	Fever	Paracetamol
P002	Soundarya	2023-06-27	Cold and cough	Cetirizine

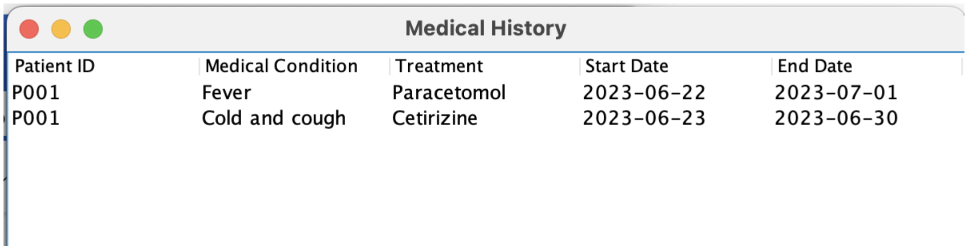


```
Appointment ID  Patient ID  Doctor ID  Date        Time
A001            P001        D001       2023-06-25  10:00 AM
A002            P002        D002       2023-06-27  11:30 AM
```

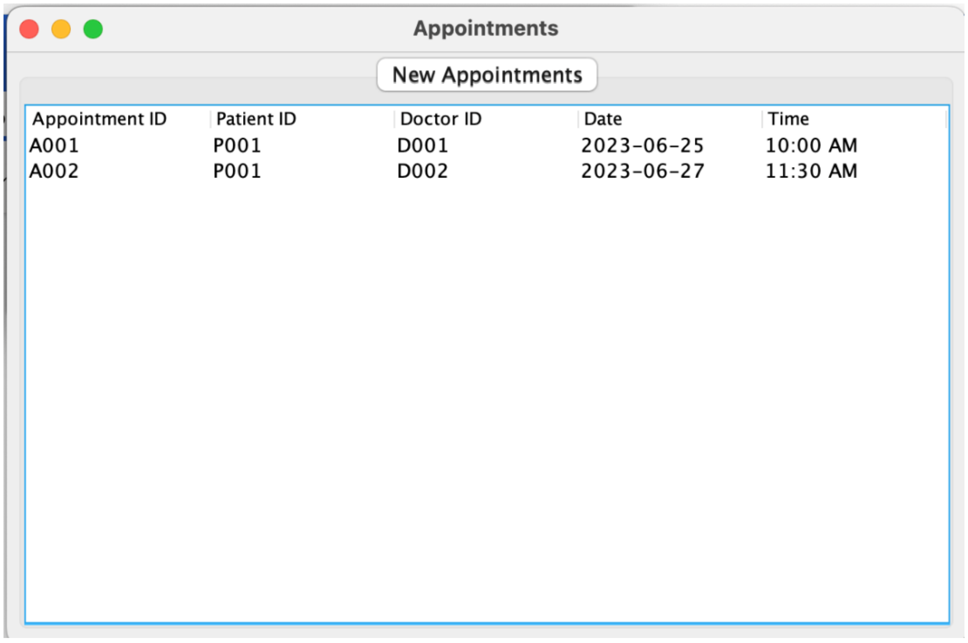


```
Appointment ID  Patient ID  Doctor ID  Date        Time      Status
A003            P003        D003       2023-06-22  09:00 AM  Accepted
A004            P004        D001       2023-06-23  02:30 PM  Accepted
```

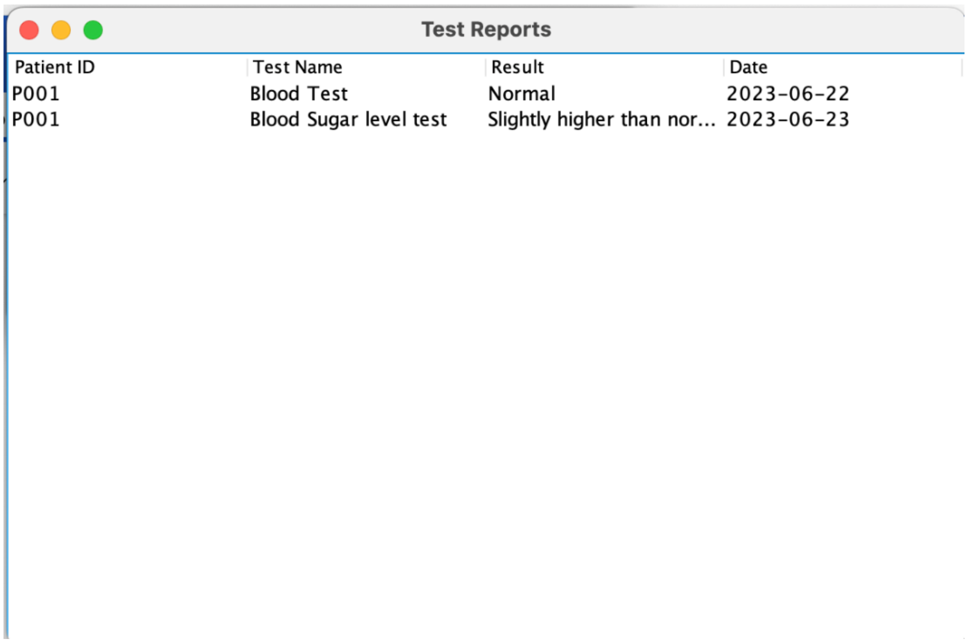



Medical History

Patient ID	Medical Condition	Treatment	Start Date	End Date
P001	Fever	Paracetomol	2023-06-22	2023-07-01
P001	Cold and cough	Cetirizine	2023-06-23	2023-06-30


Appointments

Appointment ID	Patient ID	Doctor ID	Date	Time
A001	P001	D001	2023-06-25	10:00 AM
A002	P001	D002	2023-06-27	11:30 AM



Test Reports

Patient ID	Test Name	Result	Date
P001	Blood Test	Normal	2023-06-22
P001	Blood Sugar level test	Slightly higher than nor...	2023-06-23