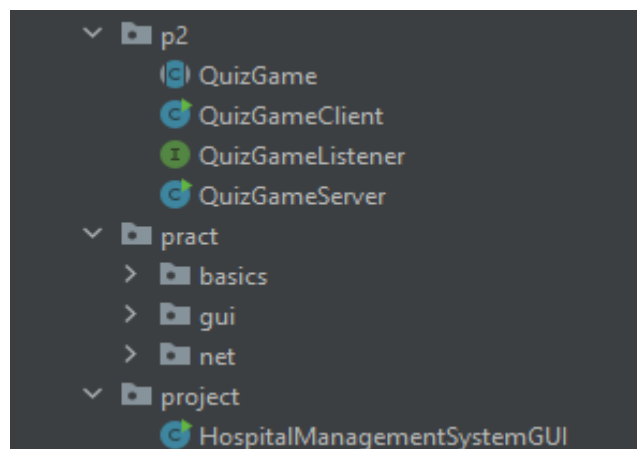
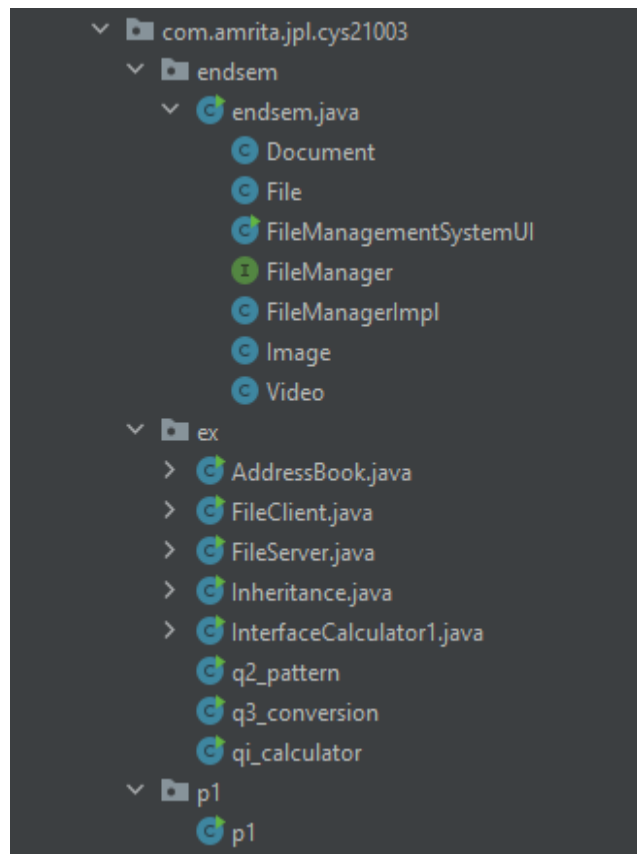
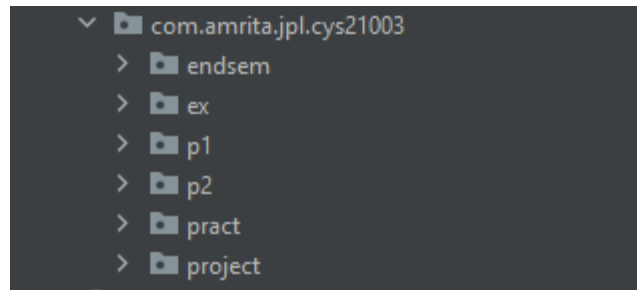
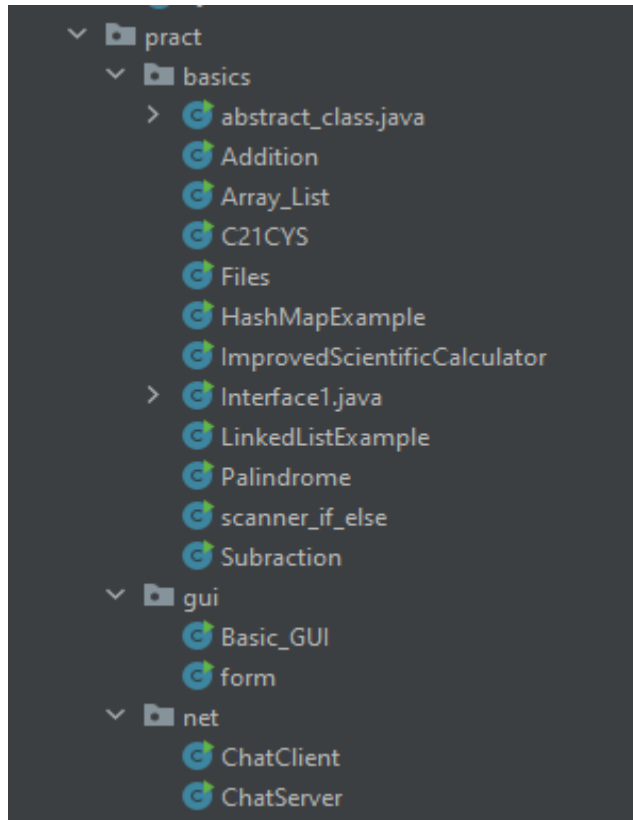


## Package Structure:





### **Com.amrita.jpl.cys21003 has**

- >endsem – Contain End Semester exam code
- >ex- Contain all the assignments question
- >p1- Contain Periodical 1 code
- >p2- Contain Periodical 2 code
- >pract- Contain Basics,GUI,net
- >project- Contain project code

### **Com.amrita.jpl.cys21003.endsem**

- >endsem.java

### **Com.amrita.jpl.cys21003.ex**

- >AddressBook.java
- >FileServer.java

->FileClient.java

->InterfaceCalculator1.java

->Inhertiance.java

->q1\_calculator.java

->q2\_pattern.java

->q3\_conversion.java

**Com.amrita.jpl.cys21003.p1**

->p1.java

**Com.amrita.jpl.cys21003.p2**

->QuizClient.java

->QuizServer.java

**Com.amrita.jpl.cys21003.pract**

->basics

->gui

->net

**Com.amrita.jpl.cys21003.pract.basics**

->abstract\_class.java

->Addition.java

->Subraction.java

->Palindrome.java

->ArrayList.java

->C21CYS.java

->files.java

->HashMapExample.java

->LinkedListExample.java

->ImprovedScientificCalculator.java

->Interface1.java

->scanner\_if\_else.java

**Com.amrita.jpl.cys21003.pract.gui**

->Basic\_GUI.java

->form.java

**Com.amrita.jpl.cys21003.pract.net**

->ChatServer.java

->ChatClient.java

**Com.amrita.jpl.cys21003.project**

->HospitalManagementSystemGUI.java

# Simple Calculator

**Exercise Type:** Assignment

**GitHub Commit Date:** 20<sup>th</sup> March 2023

**Program Number:** 01

## Problem Definition:

This code Create a simple calculator which takes two numbers as input and third input selecting the operation (addition, subtraction, multiplication and division). Handle all the negative cases and print appropriate error messages.

## Algorithm:

1. Start main function
2. Create a class qi\_calaculator
3. Create a new scanner object sc to read input from user
4. Ask user to enter first number.
5. Read integer input from user using sc.nextInt() and store it in a
6. Ask user to enter second number.
7. Read integer input from user using sc.nextInt() and store it in b
8. Ask user to enter the operation to be performed(+,-,\*,/)
9. Read String input from user using sc.next() and store it in c
- 10.If "c" equals "+":
- 11.Calculate the sum of "a" and "b" and store it in variable "t".
- 12.Print the message "Addition = " followed by the value of "t".
13. If "c" equals "-":
- 14.Calculate the difference between "a" and "b" and store it in variable "t".
- 15.Print the message "Subtraction = " followed by the value of "t".
16. If "c" equals "\*":
- 17.Calculate the product of "a" and "b" and store it in variable "t".
- 18.Print the message "Product = " followed by the value of "t".
19. If "c" equals "/":

20. Calculate the division of "a" by "b" and store it in variable "t".
21. Print the message "Division = " followed by the value of "t".
22. If none of the above conditions match:
23. Print the message "Enter a valid operator:".
24. End the inner if-else statements.
25. End the outer if-else statements.
26. End the main function.

**Output:**

```
Enter 1st number:
5
Enter 2nd number:
10
Enter the operation:
+
Addition = 15

Process finished with exit code 0
```

# Pattern Printing

**Exercise Type:** Assignment

**GitHub Commit Date:** 20<sup>th</sup> March 2023

**Program Number:** 02

## Problem Definition:

This program Create a program to print the below pattern

```
* * * * * * =====
* * * * * * =====
* * * * * * =====
* * * * * * =====
* * * * * * =====
* * * * * * =====
* * * * * * =====
* * * * * * =====
=====
=====
=====
=====
=====
=====
```

## Algorithm:

1. Start the main function
2. Declare integer variables i and j.
3. Start a loop for i from 1 to 15 (inclusive).
4. Within the outer loop, start a loop for j from 1 to 40 (exclusive).
5. Check the following conditions:
  - a. If i is less than 10 and j is less than 7:
 

Check if i is even and j is equal to 5.

If the condition is true, use the "continue" statement to skip the rest of the code inside the inner loop and move to the next iteration.

If the condition is false, print an asterisk "\*" using the `System.out.print()` method.

b. If the above condition is false (i.e.,  $i$  is greater than or equal to 10 or  $j$  is greater than or equal to 7):

Print an equal sign "=" using the `System.out.print()` method.

6. After the inner loop ends, print a new line using the `System.out.println()` method to move to the next line.

7.Repeat steps 4-6 until the outer loop completes.

8.End the main function.

**Output:**

[illegible]



## Decimal to binary and hexadecimal Convertor

**Exercise Type:** Assignment

**GitHub Commit Date:** 20<sup>th</sup> March 2023

**Program Number:** 03

### Problem Definition:

Create a simple convertor for decimal to binary and hexadecimal. For a given input in decimal, the output should be binary and hexadecimal. Note: Only Integer input accepted. Handle all negative use cases and print appropriate error messages.

### Algorithm:

- 1.Start the main function.
- 2.Create a Scanner object "sc" to read user input.
- 3.Print the message "Enter the decimal number to convert: ".
- 4.Read an integer input from the user and store it in variable "dec".
- 5.Check if "dec" is less than 0:
  - a. If true, print the message "Decimal number cannot be negative" and end the program.
  - b. If false, proceed to the next step.
- 6.Convert the decimal number "dec" to its binary representation by using the Integer.toString() method. Store the result in a string variable "bin".
- 7.Convert the decimal number "dec" to its hexadecimal representation by using the Integer.toHexString() method. Store the result in a string variable "hex".
- 8.Print the message "Binary equivalent: " followed by the value of "bin".
- 9.Print the message "Hexadecimal equivalent: " followed by the value of "hex".
- 10.End the main function.

**Output:**

```
Enter the decimal number to convert: 3  
Binary equivalent: 11  
Hexadecimal equivalent: 3  
  
Process finished with exit code 0
```

# Interface

**Exercise Type:** Assignment

**GitHub Commit Date:** 25<sup>th</sup> June 2023

**Program Number:** 04

## Problem Definition:

Create calculator using interface class that get input and show the result of addition, subtraction, multiplication, Division

## Algorithm:

1. Define an interface called "Calculator" with the following methods:
  - add(double num1, double num2) : double
  - subtract(double num1, double num2) : double
  - multiply(double num1, double num2) : double
  - divide(double num1, double num2) : double
2. Implement the interface by creating a class called "BasicCalculator" that implements the Calculator interface.
3. In the BasicCalculator class, implement the methods defined in the Calculator interface as per their respective functionalities.
4. In the main function of the InterfaceCalculator1 class:
  - a. Create a Scanner object named "scanner" to read user input.
  - b. Create an instance of the BasicCalculator class named "BasicCalculatorExample".
  - c. Read the first double input from the user and store it in the variable "num1".
  - d. Read the second double input from the user and store it in the variable "num2".
  - e. Declare a double variable named "result" and initialize it to 0.0.
  - f. Calculate the addition of "num1" and "num2" using the BasicCalculatorExample.add() method and store the result in "result".
  - g. Print the message "Addition: "

followed by the value of "result". h. Calculate the subtraction of "num1" and "num2" using the BasicCalculatorExample.subtract() method and store the result in "result". i. Print the message "Subtraction: " followed by the value of "result". j. Calculate the multiplication of "num1" and "num2" using the BasicCalculatorExample.multiply() method and store the result in "result". k. Print the message "Multiplication: " followed by the value of "result". l. Calculate the division of "num1" and "num2" using the BasicCalculatorExample.divide() method and store the result in "result". m. Print the message "Division: " followed by the value of "result".

5. End the main function.
6. End the InterfaceCalculator1 class.
7. End the BasicCalculator class.
8. End the Calculator interface.

**Output:**

```
5
10
Addition: 15.0
Subtraction: -5.0
Multiplication: 50.0
Division: 0.5

Process finished with exit code 0
```

# Inheritance

**Exercise Type:** Assignment

**GitHub Commit Date:** 25<sup>th</sup> June 2023

**Program Number:** 05

## Problem Definition:

Create a program that demonstrate Inheritance

## Algorithm:

1. Define a class called "Vehicle" with a protected boolean variable "run\_status" and the following methods:
  - start(): Set "run\_status" to true and print "[Vehicle] started."
  - stop(): Set "run\_status" to false and print "[Vehicle] stopped."
2. Define a class called "Car" that extends "Vehicle" and has private variables "modelname", "year", and "numofWheels". Include the following methods:
  - Car(String modelname, int year, int numofWheels): Initialize the instance variables and print a message with the parameter values.
  - drive(int gearPosition): If "run\_status" is true, print "Driving the car in gear position: " followed by the gear position. Otherwise, print "Error."
3. Define a class called "Bike" that extends "Vehicle" and has private variables "brandname", "year", and "numofGears". Include the following methods:
  - Bike(String brandname, int year, int numofGears): Initialize the instance variables and print a message with the parameter values.
  - pedal(int pedalSpeed): If "run\_status" is true, print "Pedaling the bike at speed: " followed by the pedal speed. Otherwise, print "Error."

4. In the "Inheritance" class:

- Create a "Car" object named "car" with the model name "Jaguar XF", year 2022, and 4 wheels.
- Call the "start" method on the "car" object.
- Call the "drive" method on the "car" object with gear position 3.
- Call the "stop" method on the "car" object.
- Create a "Bike" object named "bike" with the brand name "Giant", year 2021, and 18 gears.
- Call the "start" method on the "bike" object.
- Call the "pedal" method on the "bike" object with pedal speed 10.
- Call the "stop" method on the "bike" object.

5. End the main function.

6. End the "Inheritance" class.

7. End the "Bike" class.

8. End the "Car" class.

9. End the "Vehicle" class.

**Output:**

```
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.
Disconnected from the target VM, address: '127.0.0.1:52030', transport: 'socket'

Process finished with exit code 0
|
```

# File Server Program

**Exercise Type:** Assignment

**GitHub Commit Date:** 12<sup>th</sup> June 2023

**Program Number:** 06

## Problem Definition:

Implement a file server application that allows clients to transfer files to the server.

## Algorithm:

1. Define an abstract class called "File\_Server\_Transfer" with the following abstract methods:
  - saveFile(byte[] fileData, String filename): This method is responsible for saving the file data.
  - sendFile(String filename): This method is responsible for sending the file.
2. Define an interface called "FileTransferListener" with the following methods:
  - onFileSent(String filename): This method is called when a file is sent.
  - onFileSaved(String filename): This method is called when a file is saved.
3. Define a class called "FileTransferServer" that extends "File\_Server\_Transfer":
  - Declare a private variable "listener" of type "FileTransferListener".
4. Implement the abstract methods in the "FileTransferServer" class:



- Override the "saveFile" method: Implement the logic to save the file data.
  - Override the "sendFile" method: Not implemented for the server.
5. Define a method called "start" in the "FileTransferServer" class:
- This method starts the server and listens for incoming connections.
  - Create a new ServerSocket on port 8083.
  - Inside an infinite loop, accept incoming client connections using "serverSocket.accept()".
  - For each client connection, create a new thread with a "ClientHandler" object and start the thread.
6. Create a method called "setListener" in the "FileTransferServer" class:
- This method sets the listener for the server.
  - Assign the provided listener to the "listener" variable.
7. Define a private inner class called "ClientHandler" that implements the "Runnable" interface:
- This class handles each client connection.
  - Declare a private variable "socket" of type "Socket" to store the client socket.
8. Implement the "run" method in the "ClientHandler" class:
- Inside the "run" method, implement the logic to handle the client connection.
  - Get the input stream from the socket.
  - Read the filename sent by the client using a DataInputStream.
  - Create a new FileOutputStream with the filename.

- Read the file data from the input stream and save it to the `FileOutputStream`.
- Close the streams.
- Notify the listener that the file is saved.
- Close the socket.

9. In the "FileServer" class:

- Create an instance of "FileTransferServer" named "server".
- Implement the "FileTransferListener" interface using an anonymous inner class:
  - Override the "onFileSent" method: Not implemented for the server.
  - Override the "onFileSaved" method: Print "File saved: " followed by the filename.
- Set the listener for the "server" object using the "setListener" method.
- Start the server using the "start" method.

10. End the main program.

### Output:

```
Server started. Listening on port 8080...
```

```
Server started. Listening on port 8080...  
File saved: received.txt  
|
```

# File Client

**Exercise Type:** Assignment

**GitHub Commit Date:** 12<sup>th</sup> June 2023

**Program Number:** 07

## Problem Definition:

A simple file transfer client that sends a file to a server. It establishes a connection to the server and streams the file data over the network.

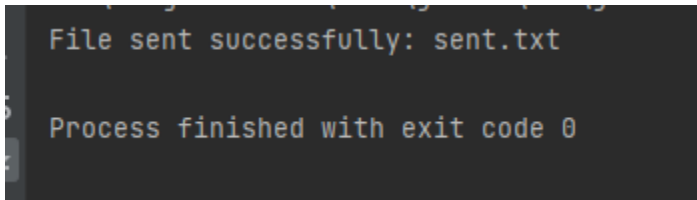
## Algorithm:

1. Define an abstract class called "File\_Client\_Transfer" with the following abstract methods:
  - `saveFile(byte[] fileData, String filename)`: This method is responsible for saving the file data.
  - `sendFile(String filename)`: This method is responsible for sending the file.
2. Define an interface called "File\_Client\_Transfer\_Listener" with the following methods:
  - `onFileSent(String filename)`: This method is called when a file is sent.
  - `onFileSaved(String filename)`: This method is called when a file is saved.
3. Define a class called "FileTransferClient" that extends "File\_Client\_Transfer":
  - Declare a private variable "listener" of type "FileTransferListener".
4. Implement the abstract methods in the "FileTransferClient" class:
  - Override the "saveFile" method: Not implemented for the client.

- Override the "sendFile" method: Implement the logic to send the file using socket programming.
  - Create a new Socket with the server IP address and port number.
  - Get the output stream from the socket.
  - Create a new File object with the provided filename.
  - Create a new FileInputStream with the file object.
  - Create a byte array buffer.
  - Read the file data into the buffer and write it to the output stream.
  - Close the file input stream, output stream, and socket.
  - Notify the listener that the file is sent.
- 5. Create a method called "setListener" in the "FileTransferClient" class:
  - This method sets the listener for the client.
  - Assign the provided listener to the "listener" variable.
- 6. In the "FileClient" class:
  - Define a constant string "SERVER\_IP" with the server's IP address.
  - Define a constant integer "SERVER\_PORT" with the server's port number.
- 7. Implement the "main" method in the "FileClient" class:
  - Declare two strings: "filename" for the local filename and "serverFilename" for the desired filename on the server.
  - Create a new Socket with the server IP address and port number.

- Get the output stream from the socket.
- Create a new `DataOutputStream` with the output stream.
- Write the `serverFilename` to the data output stream.
- Create a new `FileInputStream` with the filename.
- Create an output stream with the socket.
- Create a byte array buffer.
- Read the file data into the buffer and write it to the output stream.
- Close the output stream, file input stream, and socket.
- Print "File sent successfully: " followed by the filename.

8. End the main program.

**Output:**A screenshot of a terminal window with a dark background. It shows two lines of text: "File sent successfully: sent.txt" on the first line and "Process finished with exit code 0" on the second line. The text is in a light-colored, monospaced font.

```
File sent successfully: sent.txt  
Process finished with exit code 0
```

# Address Book

**Exercise Type:** Assignment

**GitHub Commit Date:** 15<sup>th</sup> June 2023

**Program Number:** 08

## Problem Definition:

Create an Address Book GUI using Java Swing Module.

## Algorithm:

1. Create a class called "Contact" with private variables for name, phoneNumber, and email. Include getters and setters for each variable.
2. Create a class called "AddressBook" that extends JFrame:
  - Declare private variables for tableModel, table, addButton, editButton, deleteButton, and contacts (a List of Contact objects).
3. Implement the constructor of "AddressBook":
  - Initialize the contacts list.
  - Set the title of the JFrame to "Address Book".
  - Set the default close operation to exit on close.
  - Set the size of the JFrame to 500x300 pixels.
  - Set the location of the JFrame to the center of the screen.
4. Create a method called "showContactDialog" in the "AddressBook" class:
  - Create a new JDialog with the title "Contact Details" and set it as modal.
  - Set the size of the dialog to 300x200 pixels.

- Set the location of the dialog relative to the main JFrame.
  - Set the layout of the dialog to a 4x2 GridLayout.
  - Create JLabels and JTextFields for name, phoneNumber, and email.
  - If a contact object is provided, set the text of the fields with the contact's information.
  - Create a "Save" JButton and add an ActionListener to handle the save operation.
  - Implement validation for the name, phoneNumber, and email fields.
  - If the fields are valid, update the contact object or create a new contact and add it to the contacts list.
  - Refresh the table in the main JFrame.
  - Dispose of the dialog.
5. Create a private method called "refreshTable" in the "AddressBook" class:
- Clear the rows of the tableModel.
  - Iterate through the contacts list and add each contact's information as a new row to the tableModel.
6. Implement the main method in the "AddressBook" class:
- Use SwingUtilities.invokeLater to create an instance of the "AddressBook" class and make it visible.
7. Implement ActionListener for addButton:
- Call showContactDialog with a null argument to create a new contact.
8. Implement ActionListener for editButton:
- Get the selected row from the table.

- If a row is selected, retrieve the contact from the contacts list based on the selected row.
- Call showContactDialog with the selected contact to edit it.
- If no row is selected, display a JOptionPane message to select a contact to edit.

9. Implement ActionListener for deleteButton:

- Get the selected row from the table.
- If a row is selected, remove the contact from the contacts list and the corresponding row from the tableModel.
- If no row is selected, display a JOptionPane message to select a contact to delete.

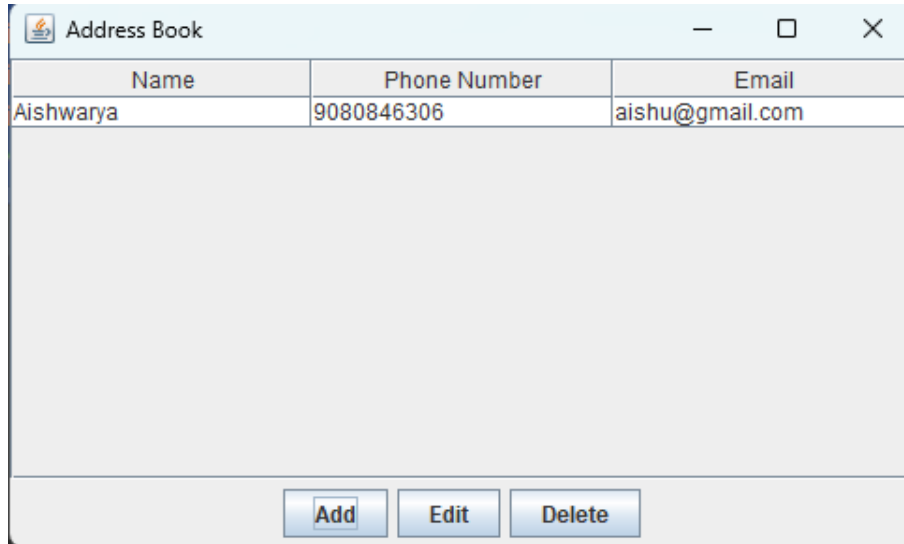
10. Implement isValidPhoneNumber method:

- Implement your phone number validation logic here.
- Return true if the phone number is valid; otherwise, return false.

11. Implement isValidEmail method:

- Implement your email validation logic here.
- Return true if the email is valid; otherwise, return false.



**Output:**

Name	Phone Number	Email
Aishwarya	9080846306	aishu@gmail.com

Add Edit Delete

# Addition

**Exercise Type:** Practice

**GitHub Commit Date:** 16<sup>th</sup> March 2023

**Program Number:** 09

## Problem Definition:

Create a program that add two numbers

## Algorithm:

1. Create a public class called "Addition".
2. Create a new instance of the Scanner class, passing System.in as the argument.
3. Display a message asking the user to enter the first number.
4. Read an integer from the user and assign it to the variable 'a'.
5. Display a message asking the user to enter the second number.
6. Read an integer from the user and assign it to the variable 'b'.
7. Close the scanner.
8. Add the values of 'a' and 'b' and assign the result to the variable 'add'.
9. Display the addition result using the System.out.println() method.

**Output:**

```
enter first number  
5  
enter second number  
5  
Addition is:10  
  
Process finished with exit code 0
```

# Subraction

**Exercise Type:** Practice

**GitHub Commit Date:** 16<sup>th</sup> March 2023

**Program Number:** 10

## Problem Definition:

Create a program that subtract two numbers

## Algorithm:

1. Create a new instance of the Scanner class to read input from the console.
2. Display the message "Enter the first number" to prompt the user for input.
3. Read the first number entered by the user and store it in the variable 'a'.
4. Display the message "Enter the second number" to prompt the user for input.
5. Read the second number entered by the user and store it in the variable 'b'.
6. Close the Scanner to release system resources.
7. Subtract the value of 'b' from 'a' and store the result in the variable 'sub'.
8. Display the message "Subtraction is: " followed by the value of 'sub' to show the subtraction result.

**Output:**

```
enter first number
10
enter second number
5
sub is:5

Process finished with exit code 0
|
```

# Palindrome

**Exercise Type:** Practice

**GitHub Commit Date:** 25.06.2023

**Program Number:** 11

## Problem Definition:

Create a program that check whether the number is palindrome number or not.

## Algorithm:

1. Start the program.
2. Create a new instance of the Scanner class to read input from the console.
3. Display the message "Enter an integer: " to prompt the user for input.
4. Read an integer from the user and store it in the variable 'n'.
5. Create a variable 'temp' and assign the value of 'n' to it for later comparison.
6. Initialize variables 'r', 'sum' to 0.
7. Perform the following steps while 'n' is greater than 0: a. Calculate the remainder 'r' by taking the modulus of 'n' divided by 10. b. Multiply the 'sum' by 10 and add 'r' to it. c. Divide 'n' by 10 to remove the rightmost digit.
8. After the while loop, check if 'temp' is equal to 'sum'.
9. If they are equal, display the message "Palindrome number" indicating that the number is a palindrome.
10. If they are not equal, display the message "Not palindrome" indicating that the number is not a palindrome.

11.End the program.

**Output:**

```
Enter an integer: 10  
not palindrome  
  
Process finished with exit code 0
```

# Student Grade printing

**Exercise Type:** Practice

**GitHub Commit Date:** 17<sup>th</sup> April 2023

**Program Number:** 12

## Problem Definition:

Create a program that print grade of a student

## Algorithm:

1. Start the program.
2. Declare instance variables 'assignGrade', 'name', and 'roll' of appropriate data types.
3. Define a static method 'publishResult()' that prints the message "Results are published".
4. Define a non-static method 'grades(double grade)' that takes a 'grade' parameter and prints the message "Grade is: " followed by the value of 'grade'.
5. Define a non-static method 'nameOfStud(String name)' that takes a 'name' parameter and prints the message "Name: " followed by the value of 'name'.
6. Define a non-static method 'RollNo(String roll)' that takes a 'roll' parameter and prints the message "Roll Number: " followed by the value of 'roll'.
7. In the 'main' method:
  - a. Create an instance of the 'C21CYS' class named 'myclass'.
  - b. Call the 'publishResult()' method to display the message "Results are published".
  - c. Assign a value of 9.69 to the 'assignGrade' variable of 'myclass'.



- d. Assign a value of "Aishwarya G" to the 'name' variable of 'myclass'.
  - e. Assign a value of "CB.EN.U4CYS21003" to the 'roll' variable of 'myclass'.
  - f. Call the 'nameOfStud()' method of 'myclass' and pass the 'name' variable as an argument.
  - g. Call the 'RollNo()' method of 'myclass' and pass the 'roll' variable as an argument.
  - h. Call the 'grades()' method of 'myclass' and pass the 'assignGrade' variable as an argument.
8. End the program.

**Output:**

```
Results are published  
Name:Aishwarya G  
Roll Number:CB.EN.U4CYS21003  
Grade is:9.69  
  
Process finished with exit code 0
```

# Array List

**Exercise Type:** Practice

**GitHub Commit Date:** 31<sup>st</sup> May 2023

**Program Number:** 13

## Problem Definition:

Creates an ArrayList of student roll numbers, adds student roll numbers to it, and uses an Iterator to traverse and print the student roll numbers.

## Algorithm:

1. Start the program.
2. Import the required package: **java.util.ArrayList** and **java.util.Iterator**.
3. Define the **Array\_List** class.
4. Define the **main** method that takes an array of strings as input parameters.
5. Inside the **main** method:
  - Create an **ArrayList** object named **u21cys** to store integer values.
  - Add the values 21001,21002,21003,21004 and 21005 to the **u21cys** list using the **add()** method.
  - Create an **Iterator** object named **it** by calling the **iterator()** method on the **u21cys** list.
  - Print the first element of the list using the **next()** method of the iterator (**it.next()**) and display it on the console.
  - Print the second element of the list using the **next()** method of the iterator and display it on the console.

- Print the third element of the list using the **next()** method of the iterator and display it on the console.
- Print the fourth element of the list using the **next()** method of the iterator and display it on the console.

6. End the program.

**Output:**

```
21001
21002
21003
21004

Process finished with exit code 0
```

## Improved Scientific Calculator

**Exercise Type:** Practice

**GitHub Commit Date:** 12<sup>th</sup> June 2023

**Program Number:** 14

### Problem Definition:

Create a scientific calculator using GUI-Swing

### Algorithm:

1. Start the program.
2. Import the required packages: **javax.swing.JFrame**, **javax.swing.JTextField**, **javax.swing.JButton**, **javax.swing.JPanel**, **java.awt.BorderLayout**, **java.awt.GridLayout**, **java.awt.Container**, **java.awt.event.ActionEvent**, **java.awt.event.ActionListener**, **java.awt.event.KeyEvent**, **java.awt.event.KeyListener**, and **javax.swing.SwingUtilities**.
3. Define a class called **ImprovedScientificCalculator** that extends **JFrame**.
4. Declare private instance variables:
  - **display** of type **JTextField** to represent the calculator display.
  - **operator** of type **String** to store the current operator.
  - **operand1** of type **double** to store the first operand.
  - **isOperatorClicked** of type **boolean** to track if an operator button was clicked.
5. Implement a constructor for **ImprovedScientificCalculator**:
  - Set the title of the frame to "Scientific Calculator".
  - Set the default close operation to **JFrame.EXIT\_ON\_CLOSE**.

- Create the **display** field with a width of 10 and make it non-editable.
- Create the number buttons (**button1** to **button9**) with corresponding labels.
- Create the operator buttons (**buttonPlus**, **buttonMinus**, **buttonMultiply**, **buttonDivide**, **buttonSin**, **buttonCos**, **buttonTan**, **buttonEquals**) with corresponding labels.
- Create the clear button (**buttonClear**) with the label "Clear".
- Set the layout of the frame to **BorderLayout**.
- Create a panel (**numberPanel**) with a grid layout of 4 rows and 3 columns.
- Add the number buttons and the clear button to the **numberPanel**.
- Create a panel (**operatorPanel**) with a grid layout of 5 rows and 1 column.
- Add the operator buttons to the **operatorPanel**.
- Create a panel (**mainPanel**) with a border layout.
- Add the **display** field to the north region, the **numberPanel** to the center region, and the **operatorPanel** to the east region of the **mainPanel**.
- Get the content pane of the frame and add the **mainPanel** to it.
- Create action listeners for the number buttons, operator buttons, and the clear button.
- Add the corresponding action listeners to the buttons.
- Create a key listener for the display field.
- Add the key listener to the display field.

- Pack the frame to adjust its size based on the components.
  - Set the location of the frame to be centered on the screen.
6. Implement the **NumberButtonListener** class as a private inner class:
- Implement the **actionPerformed** method:
    - Get the button that triggered the event.
    - Get the text of the button.
    - Append the button's text to the text in the display field.
7. Implement the **OperatorButtonListener** class as a private inner class:
- Implement the **actionPerformed** method:
    - Get the button that triggered the event.
    - Get the current text in the display field.
    - If an operator button was not previously clicked:
      - Parse the current text to a double and assign it to **operand1**.
      - Get the text of the button and assign it to **operator**.
      - Clear the display field.
      - Set **isOperatorClicked** to **true**.
      - If the operator is one of "Sin", "Cos", or "Tan":
        - Call the **performCalculation** method with **operand1** and **operator**.
        - Convert the result to a string and set it as the text in the display field.
    - If an operator button was previously clicked:

- Parse the current text to a double and assign it to **operand2**.
- Call the **performCalculation** method with **operand1**, **operand2**, and **operator**.
- Convert the result to a string and set it as the text in the display field.
- Assign the result to **operand1**.
- Get the text of the button and assign it to **operator**.

8. Implement the **ClearButtonListener** class as a private inner class:

- Implement the **actionPerformed** method:
  - Clear the text in the display field.
  - Reset **operand1** to 0.
  - Reset **operator** to **null**.
  - Set **isOperatorClicked** to **false**.

9. Implement the **DisplayKeyListener** class as a private inner class:

- Implement the **keyPressed** method:
  - Get the key code from the event.
  - If the key code represents a numeric key (0-9), append the corresponding digit to the display field.
  - If the key code represents an operator key (+, -, \*, /), call the **operatorButtonClicked** method with the operator.
  - If the key code represents the equals key (=) or the enter key, call the **operatorButtonClicked** method with "=".
- Implement the **keyTyped** and **keyReleased** methods (not used).

- Implement the **operatorButtonClicked** method:
  - Get the current text in the display field.
  - If an operator button was not previously clicked:
    - Parse the current text to a double and assign it to **operand1**.
    - Assign the operator passed as an argument to **operator**.
    - Clear the display field.
    - Set **isOperatorClicked** to **true**.
  - If an operator button was previously clicked:
    - Parse the current text to a double and assign it to **operand2**.
    - Call the **performCalculation** method with **operand1**, **operand2**, and **operator**.
    - Convert the result to a string and set it as the text in the display field.
    - Assign the result to **operand1**.
    - Assign the operator passed as an argument to **operator**.

10. Implement the **performCalculation** method:

- Accept **operand1**, **operand2**, and **operator** as parameters.
- Declare a variable **result** of type **double** and initialize it to 0.0.
- Use a switch statement to perform the calculation based on the operator:



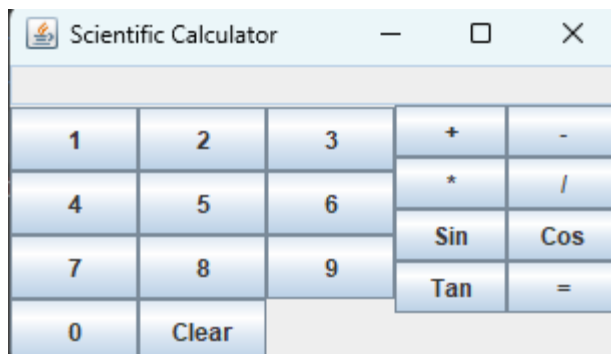
- For the operators "+", "-", "\*", and "/", perform the corresponding arithmetic operation on **operand1** and **operand2**.
- For the operators "Sin", "Cos", and "Tan", use the **Math.sin**, **Math.cos**, and **Math.tan** functions on **operand1**.
- Assign the calculated result to **result**.
- Return **result**.

11. Implement the **main** method:

- Use **SwingUtilities.invokeLater** to create and display an instance of **ImprovedScientificCalculator**.

12. End the program.

**Output:**



# Abstract class calculator

**Exercise Type:** Practice

**GitHub Commit Date:** 22th May 2023

**Program Number:** 15

## Problem Definition:

Create a calculator using abstract class

## Algorithm:

1. Start the program.
2. Import the required package: **java.util.Scanner**.
3. Define an abstract class called **calculator** with the following abstract methods:
  - **add(double num1, double num2)** - returns the sum of **num1** and **num2**.
  - **subtract(double num1, double num2)** - returns the difference between **num1** and **num2**.
  - **multiply(double num1, double num2)** - returns the product of **num1** and **num2**.
  - **divide(double num1, double num2)** - returns the division result of **num1** divided by **num2**.
4. Define a concrete class called **calc** that extends the **calculator** class:
  - Implement the **add**, **subtract**, **multiply**, and **divide** methods based on the corresponding operations.
  - Handle the case where the denominator (**num2**) is zero in the **divide** method by printing an error message and returning 0.0.

5. Define a public class called **abstract\_class**.

6. Define the **main** method within the **abstract\_class** class:

- Create a new instance of **Scanner** named **scanner** to read input from the user.
- Create an instance of the **calc** class named **calculator**.
- Enter a while loop that runs indefinitely until the user chooses to exit.
- Within the loop, display a menu of available operations (addition, subtraction, multiplication, division, or exit).
- Prompt the user to enter their choice (an integer from 1 to 5).
- If the user chooses 5, print "Exit" and break out of the loop.
- Prompt the user to enter the first number and store it in the variable **num1**.
- Prompt the user to enter the second number and store it in the variable **num2**.
- Declare a variable **result** and initialize it to 0.0.
- Use a switch statement to perform the selected operation based on the user's choice:
  - Call the corresponding method on the **calculator** instance and pass **num1** and **num2** as arguments.
  - Store the returned result in the **result** variable.
- Print the calculated result.

7. End the program.

**Output:**

```
choose 1 to perform Addition
choose 2 to perform Subtraction
choose 3 to perform Multiplication
choose 4 to perform Division
choose 5 to Exit
Enter your choice (1-5): 1
Enter the first number: 10
Enter the second number: 20
Result: 30.0
```

# Interface Calculator

**Exercise Type:** Practice

**GitHub Commit Date:** 22<sup>nd</sup> May 2023

**Program Number:** 16

## Problem Definition:

Create a calculator using Interface class

## Algorithm:

1. Start the program.
2. Import the required packages: **java.util.Scanner**.
3. Define an interface called **Calculator** with the following methods:
  - **double add(double num1, double num2);**
  - **double subtract(double num1, double num2);**
  - **double multiply(double num1, double num2);**
  - **double divide(double num1, double num2);**
4. Create a class called **calc1** that implements the **Calculator** interface.
5. Implement the **add** method:
  - Add **num1** and **num2** and return the result.
6. Implement the **subtract** method:
  - Subtract **num2** from **num1** and return the result.
7. Implement the **multiply** method:
  - Multiply **num1** and **num2** and return the result.

8. Implement the **divide** method:

- Check if **num2** is not equal to 0.
  - If true, divide **num1** by **num2** and return the result.
  - If false, print "denominator can't be zero" and return 0.0.

9. Create a class called **Interface1** with the **main** method.

10. Create a **Scanner** object to read user input.

11. Create an instance of the **calc1** class and assign it to the **calculator** variable of type **Calculator**.

12. Start a while loop with the condition **true** for continuously accepting user input.

13. Print the menu options for the calculator:

- "choose 1 to perform Addition"
- "choose 2 to perform Subtraction"
- "choose 3 to perform Multiplication"
- "choose 4 to perform Division"
- "choose 5 to Exit"

14. Prompt the user to enter their choice (1-5).

15. Use a switch statement based on the user's choice:

- If the choice is 5, print "Exit" and break out of the loop.
- If the choice is 1, prompt the user to enter the first number and the second number.
  - Call the **add** method on the **calculator** object with the entered numbers and assign the result to the **result** variable.

- If the choice is 2, prompt the user to enter the first number and the second number.
  - Call the **subtract** method on the **calculator** object with the entered numbers and assign the result to the **result** variable.
- If the choice is 3, prompt the user to enter the first number and the second number.
  - Call the **multiply** method on the **calculator** object with the entered numbers and assign the result to the **result** variable.
- If the choice is 4, prompt the user to enter the first number and the second number.
  - Call the **divide** method on the **calculator** object with the entered numbers and assign the result to the **result** variable.
- If the choice is not 1-5, print "Invalid choice. Please try again."

16. Print the calculated result.

17. End the while loop and continue to accept user input.

18. End the program.

### Output:

```
choose 1 to perform Addition
choose 2 to perform Subtraction
choose 3 to perform Multiplication
choose 4 to perform Division
choose 5 to Exit
Enter your choice (1-5): 1
Enter the first number: 10
Enter the second number: 20
Result: 30.0
```

# Basic GUI

**Exercise Type:** Practice

**GitHub Commit Date:** 25<sup>th</sup> June 2023

**Program Number:** 17

## Problem Definition:

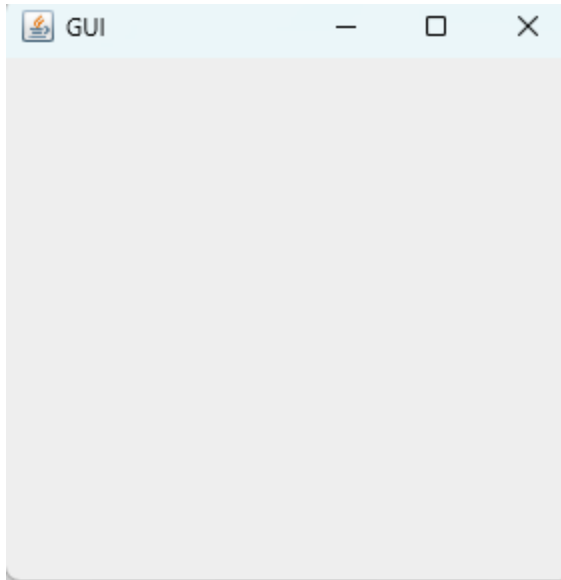
Create a basic graphical user interface (GUI) using Java Swing

## Algorithm:

1. Start the program.
2. Create a class called **Basic\_GUI** that extends **JFrame**.
3. Inside the **Basic\_GUI** class, define a constructor.
4. Set the title of the GUI window to "GUI" using the **setTitle** method.
5. Set the default close operation to **EXIT\_ON\_CLOSE** using the **setDefaultCloseOperation** method.
6. Set the size of the GUI window to 300x300 pixels using the **setSize** method.
7. Center the GUI window on the screen using the **setLocationRelativeTo** method with the argument **null**.
8. Set the visibility of the GUI window to **true** using the **setVisible** method.
9. Define the **main** method.
10. Create an instance of the **Basic\_GUI** class using the **new** keyword.
11. End the program.



**Output:**



# Personal details Form

**Exercise Type:** Practice

**GitHub Commit Date:** 05<sup>th</sup> June 2023

**Program Number:** 18

## Problem Definition:

Create a GUI which will show the personal details of a student

## Algorithm:

1. Start the program.
2. Declare a new class "form" that extends JFrame.
3. Define the constructor of the "form" class.
4. Set the title of the JFrame to "FlowLayout Example".
5. Set the size of the JFrame to (40, 220).
6. Set the default close operation of the JFrame to EXIT\_ON\_CLOSE.
7. Set the layout manager of the JFrame to a new FlowLayout object.
8. Create six JLabel objects with the following text: " Full Name:", "Roll Number:", "Course:", "Father name:", "Mother name:", "City:".
9. Create seven JButton objects with the following text: "Aishwarya G", "CB.EN.U4CYS21003", "B-TECH CYS", "Govindasamy N", "Balamani G", "Namakkal", "Submit".
10. Add all the JLabel and JButton objects to the content pane of the JFrame using the add() method.
11. Make the JFrame visible using the setVisible() method.
12. Define the main method of the program.

13.Create a new object of the "form" class using the constructor.

14.End the main method.

15.End the program.

### Output:



The screenshot shows a Java Swing window titled "FlowLayout Example". Inside the window, there is a horizontal flow layout of labels and text input fields. The labels and their corresponding values are: "Full Name:" followed by "Aishwarya G", "Roll Number:" followed by "CB.EN.U4CYS21003", "Course:" followed by "B-TECH CYS", "Father name:" followed by "Govindasamy N", "Mother name:" followed by "Balamani G", and "City:" followed by "Namakkal". To the right of the "City" field is a "Submit" button.

# Chat Server

**Exercise Type:** Practice

**GitHub Commit Date:** 29<sup>th</sup> May 2023

**Program Number:** 19

## Problem Definition:

Create a simple server application that listens on a specific port and receives messages from a client. The server should print the received messages along with the timestamp.

## Algorithm:

1. Define a public class named **ChatServer**.
2. Define the **main** method as the entry point of the program, accepting **String[] args** as command line arguments.
3. Wrap the code within a try-catch block to handle any potential exceptions.
4. Inside the try block: a. Create a **ServerSocket** object **ss** and bind it to port 2444.
5. Accept a client connection using the **accept()** method on the **ServerSocket** object **ss**. This will block until a client establishes a connection and return a **Socket** object **s** representing the client-server connection.
6. Create a **Scanner** object **msg** to read user input from the console.
7. Create a **DataInputStream** object **dis** using **s.getInputStream()** to receive messages from the client.
8. Create a **DataOutputStream** object **dout** using **s.getOutputStream()** to send messages to the client.

9. Enter a do-while loop: a. Read the message from the client using **dis.readUTF()** and assign it to the **str** variable. b. Print the received message to the console. c. Check if the received message is not equal to "EXIT" (to terminate the loop and the program). d. If it's not "EXIT", prompt the user to enter a reply via the console. e. Read the reply entered by the user using **msg.nextLine()** and assign it to the **reply** variable. f. Send the reply to the client using **dout.writeUTF(reply)** and flush the stream using **dout.flush()**.
10. After exiting the loop, print "TERMINATED" to the console.
11. Close the server socket using **ss.close()**.
12. Catch any IOException that might occur and print an error message to the console.

**Output:**

```
Message Received: hi
Type your reply:
hello
|
```

# Chat Client

**Exercise Type:** Practice

**GitHub Commit Date:** 29<sup>th</sup> May 2023

**Program Number:** 20

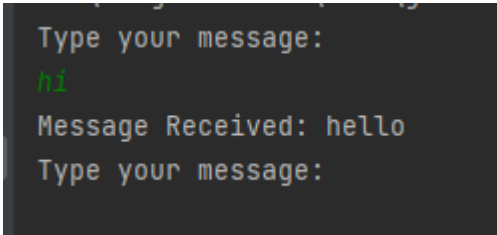
## Problem Definition:

Create a simple client application that connects to a server and sends messages entered by the user. The client should establish a connection with the server using a specific IP address and port number. It should prompt the user to enter a message, send it to the server, and repeat the process until the user enters the "exit" command.

## Algorithm:

1. Define a public class named **ChatClient**.
2. Define the **main** method as the entry point of the program, accepting **String[] args** as command line arguments.
3. Wrap the code within a try-catch block to handle any potential exceptions.
4. Inside the try block: a. Create a **Socket** object **s** and connect it to the server at "localhost" on port 2444.
5. Create a **DataInputStream** object **dis** using **s.getInputStream()** to receive messages from the server.
6. Create a **DataOutputStream** object **dout** using **s.getOutputStream()** to send messages to the server.
7. Create a **Scanner** object **input** to read user input from the console.
8. Declare string variables **message** and **str** to hold user input and received messages, respectively.

9. Enter a do-while loop: a. Prompt the user to enter a message via the console using **System.out.println("Type your message: ")**. b. Read the user's input using **input.nextLine()** and assign it to the **message** variable. c. Send the message to the server using **dout.writeUTF(message)** and flush the stream using **dout.flush()**. d. Check if the sent message is not equal to "EXIT" (to terminate the loop and the program). e. If it's not "EXIT", read the reply from the server using **dis.readUTF()** and assign it to the **str** variable. f. Print the received message to the console using **System.out.println("Message Received: " + str)**.
10. After exiting the loop, close the output stream using **dout.close()** and close the socket using **s.close()**.
11. Catch any **IOException** that might occur and print an error message to the console.

**Output:**

```
Type your message:
hi
Message Received: hello
Type your message:
```

# Linked List

**Exercise Type:** Practice

**GitHub Commit Date:** 25<sup>th</sup> June 2023

**Program Number:** 21

## Problem Definition:

Create a program to illustrate Linked list using java

## Algorithm:

1. Start the program.
2. Declare a class named "LinkedListExample".
3. Write a method named "main" that takes an array of strings as input.
4. Inside the "main" method:
  - a. Create a new object of the LinkedList class called "u21cys" with type parameter string.
  - b. Use the "add" method of the LinkedList to add elements to the list.
    - Add the element "CB.EN.U4CYS22031".
    - Add the element "CB.EN.U4CYS22032".
    - Add the element "CB.EN.U4CYS22033".
    - Add the element "CB.EN.U4CYS22034".
    - Add the element "CB.EN.U4CYS22035".
  - c. Print the contents of the LinkedList using the "println" method.
  - d. Use the "get" method of the LinkedList to retrieve and store the element at index 2 in a variable called "element".
  - e. Print the value of "element".
  - f. Use the "add" method of the LinkedList to add a new element at index 1.



- Add the element "CB.EN.U4CYS22036". f. Use the "remove" method of the LinkedList to remove the element "CB.EN.U4CYS22033". g. Use the "contains" method of the LinkedList to check if the element "CB.EN.U4CYS22035" exists in the list and store the result in a boolean variable called "containsElement".
- Print the value of "containsElement". h. Use the "size" method of the LinkedList to get the size of the list and store it in an integer variable called "size".
- Print the value of "size". i. Use a for-each loop to iterate over the elements in the LinkedList and print each element.
- For each element "x" in "u21cys", do the following:
  - Print the value of "x".

5. End the program.

### Output:

```
[CB.EN.U4CYS22031, CB.EN.U4CYS22032, CB.EN.U4CYS22033, CB.EN.U4CYS22034, CB.EN.U4CYS22035]
Element at index 2: CB.EN.U4CYS22033
Contains 'CB.EN.U4CYS22035'? true
Size of the LinkedList: 5
Element: CB.EN.U4CYS22031
Element: CB.EN.U4CYS22036
Element: CB.EN.U4CYS22032
Element: CB.EN.U4CYS22034
Element: CB.EN.U4CYS22035

Process finished with exit code 0
```

# Hash Map

**Exercise Type:** Practice

**GitHub Commit Date:** 25<sup>th</sup> June 2023

**Program Number:** 22

## Problem Definition:

Create a program to illustrate Hash map using java

## Algorithm:

1. Start the program.
2. Declare a class named "HashMapExample".
3. Write a method named "main" that takes an array of strings as input.
4. Inside the "main" method:
  - a. Create a new object of the HashMap class called "rollName" with key and value types as strings.
  - b. Use the "put" method of the HashMap to add key-value pairs representing roll numbers and names.
    - Add the key "CB.EN.U4CYS21001" and the value "Abinesh G".
    - Add the key "CB.EN.U4CYS21011" and the value "A S Deepan".
    - Add the key "CB.EN.U4CYS21021" and the value "Gundala Kushal Bhavani Reddy".
    - Add the key "CB.EN.U4CYS21031" and the value "Kishanth K".
    - Add the key "CB.EN.U4CYS21041" and the value "Middivari Charan Kumar Reddy".
    - Add the key "CB.EN.U4CYS21051" and the value "Nithin S".
    - Add the key "CB.EN.U4CYS21061" and the value "Roshni V".

- Add the key "CB.EN.U4CYS21071" and the value "Sourabh Sasikanthan".
- Add the key "CB.EN.U4CYS21081" and the value "Koti Venkatadinesh Reddy". c. Use the "get" method of the HashMap to retrieve and print the name associated with the roll number "CB.EN.U4CYS21011". d. Use the "put" method of the HashMap to add a new key-value pair representing a roll number and name.
- Add the key "CB.EN.U4CYS21091" and the value "Scoob". e. Use the "remove" method of the HashMap to remove the key-value pair associated with the roll number "CB.EN.U4CYS21061". f. Use the "containsKey" method of the HashMap to check if the roll number "CB.EN.U4CYS21041" exists in the HashMap and store the result in a boolean variable "containsRollNumber".
- Print the value of "containsRollNumber". g. Use the "containsValue" method of the HashMap to check if the name "Nithin S" exists in the HashMap and store the result in a boolean variable "containsName".
- Print the value of "containsName". h. Use an iterator to iterate over the entries in the HashMap and print each roll number and name.
- Get an iterator by calling the "entrySet" method of the HashMap and then the "iterator" method.
- While there are more entries, do the following:
  - Get the next entry from the iterator and store it in a variable "entry".
  - Get the key and value from the "entry".
  - Print the roll number and name.

5. End the program.

**Output:**

```
A S Deepan
Contains roll number: true
Contains name: true
Roll Number: CB.EN.U4CYS21001, Name: Abinesh G
Roll Number: CB.EN.U4CYS21021, Name: Gundala Kushal Bhavani Reddy
Roll Number: CB.EN.U4CYS21011, Name: A S Deepan
Roll Number: CB.EN.U4CYS21041, Name: Middivari Charan Kumar Reddy
Roll Number: CB.EN.U4CYS21031, Name: Shaggy
Roll Number: CB.EN.U4CYS21051, Name: Nithin S
Roll Number: CB.EN.U4CYS21081, Name: Koti Venkatadinesh Reddy
Roll Number: CB.EN.U4CYS21071, Name: Sourabh Sasikanthan
Roll Number: CB.EN.U4CYS21091, Name: Scoob

Process finished with exit code 0
|
```

# File Handling

**Exercise Type:** Practice

**GitHub Commit Date:** 25<sup>th</sup> June 2023

**Program Number:** 23

## Problem Definition:

Create a program that illustrates basic file handling

## Algorithm:

1. Create a new class called "Files" and import the necessary packages.
2. Declare the main method within the "Files" class.
3. Wrap the entire code within a try-catch block to handle potential IOExceptions.
4. Inside the try block, create a new File object named "myFile" with the filename "20cys383.txt".
5. Use the `createNewFile()` method of the File object to check if the file exists or needs to be created. If it's a new file, print "File created: <filename>". If the file already exists, print "[INFO] File Exists."
6. Print the static information about the file, such as the absolute path, writeability, readability, file size in bytes, and the hash code of the file.
7. Catch any IOException that occurs and print the error message and stack trace.
8. Create a new FileWriter object named "myWriter" and pass the filename "20cys383.txt" as an argument.
9. Use the `write()` method of the FileWriter object to write the string "Hi Aishu" to the file.

10. Close the FileWriter using the close() method.
11. Catch any IOException that occurs and print the error message and stack trace.
12. Create a new File object named "myFile" with the filename "20cys383.txt".
13. Create a new Scanner object named "myScanner" and pass the "myFile" object as an argument. This will allow reading from the file.
14. Use a while loop to iterate through each line in the file while the scanner has more lines.
15. Inside the loop, retrieve the next line from the scanner using the nextLine() method and store it in a string variable called "data".
16. Print the content of each line using the "data" variable.
17. Catch any IOException that occurs and print the error message and stack trace.
18. End the main method.
19. End the class.

**Output:**

```
File created: 20cys383.txt
Static contents are added to the file.
Absolute path: C:\Users\govindasamy\IdeaProjects\Aishwarya_jpl\20cys383.txt
Writeable: true
Readable true
File size in bytes 0
Hash Code-1898025024
File content: Hi Aishu

Process finished with exit code 0
```

# Menu driven program for factorial,Fibonacci,sum of n numbers,prime or not

**Exercise Type:** Periodical-1

**GitHub Commit Date:** 24<sup>th</sup> April 2023

**Program Number:** 24

## Problem Definition:

Write a Menu Driven Program to do the following :

- 1) To Calculate the Factorial of a number.
- 2) To Calculate the Fibonacci series.
- 3) To calculate the sum of n natural numbers.
- 4) To check whether the number is prime or not.

## Algorithm:

- 1.Start the program
- 2.Create class P1
- 3.Define function fact()
4. Define function fibo()
5. Define function sum\_of\_n\_no()
6. Define function prime\_test()
7. Start the main function.
8. Create an instance of the class `p1` called `mycClass`.
9. Create a new instance of the `Scanner` class called `input`.

10. Print the message "Enter choice: ".
11. Read an integer input from the user using the `nextInt()` method of the `Scanner` class and store it in the variable `choice`.
12. Use a switch statement to perform different actions based on the value of `choice`:
  - > Case 1:
    - If `choice` is equal to 1, Call the `fact()` method.
  - > Case 2:
    - If `choice` is equal to 2, Call the `fibo()` method.
  - > Case 3:
    - If `choice` is equal to 3, Call the `sum_n_no()`.
  - > Case 4:
    - If `choice` is equal to 4, Call the `prime_test()` method.
  - Default case:
    - Print the message "Invalid choice".
13. End the switch statement.
14. End the main function.

**Output:**

```
enter choice:
1
Enter an integer: 5
Factorial of 5 is: 120

Process finished with exit code 0
|
```



# Quiz Game

**Exercise Type:** Periodical-2

**GitHub Commit Date:** 13<sup>th</sup> June 2023

**Program Number:** 25

## Problem Definition:

The problem discusses about a Quiz Game Client that connects to a Quiz Game Server. The client receives questions from the server and sends answers back. The server evaluates the answers and provides feedback to the client. The objective is to create a functioning client that communicates with the server to participate in the quiz game.

## Algorithm:

Algorithm for QuizGame Abstract Class:

1. Start
2. Define the abstract class QuizGame.
3. Declare the abstract method startGame() in the QuizGame class.
4. Declare the abstract method askQuestion() in the QuizGame class.
5. Declare the abstract method evaluateAnswer(answer: String) in the QuizGame class.
6. Define the QuizGame class as abstract.
7. Define the concrete subclass ConcreteQuizGame that extends the QuizGame class.
8. Implement the startGame() method in the ConcreteQuizGame subclass.
9. Implement the askQuestion() method in the ConcreteQuizGame subclass.

10. Implement the evaluateAnswer(answer: String) method in the ConcreteQuizGame subclass.
11. Define the main method.
12. Create an instance of ConcreteQuizGame.
13. Call the startGame() method on the instance.
14. Call the askQuestion() method on the instance.
15. Read the user input into the variable answer.
16. Call the evaluateAnswer(answer) method on the instance.
17. End.

#### Algorithm for QuizGameListener

1. Start
2. Define the interface QuizGameListener.
3. Declare the method onQuestionAsked(question: String) in the QuizGameListener interface.
4. Declare the method onAnswerEvaluated(isCorrect: boolean) in the QuizGameListener interface.
5. Define the class that implements the QuizGameListener interface.
6. Implement the onQuestionAsked(question: String) method in the class.
7. Implement the onAnswerEvaluated(isCorrect: boolean) method in the class.
8. Define the main method.
9. Create an instance of the class that implements QuizGameListener.
10. Call the onQuestionAsked(question) method on the instance, passing the question as an argument.

11. Call the `onAnswerEvaluated(isCorrect)` method on the instance, passing the evaluation result as an argument.

12. End.

Algorithm for QuizGameClient:

1. Start

2. Define the class `QuizGameClient` that extends the `QuizGame` abstract class.

3. Override the `startGame()` method from the `QuizGame` class.

4. Inside the `startGame()` method, establish a connection to the server and handle the question-answer flow:

5. Print "Connecting to the server.." message.

6. Try connecting to the server using a `Socket` and initializing `ObjectOutputStream` and `ObjectInputStream`.

7. Enter an infinite loop:

7.1. Read a question from the input stream (`q`).

7.2. If `q` is "Game Over", break out of the loop.

7.3. Print the received question (`q`).

7.4. Read the user's answer from the console and store it in the answer variable.

7.5. Write answer to the output stream and flush it.

7.6. Read the evaluation result (`isCorrect`) from the input stream.

7.7. Print the result ("Correct" or "Incorrect") based on `isCorrect`.

8. Close the output stream and the input stream.

9. Handle exceptions and display an error message if necessary.

10. Override the askQuestion() method from the QuizGame class and leave it empty.
11. Override the evaluateAnswer(String answer) method from the QuizGame class and leave it empty.
12. Define the main method.
13. Inside the main method:
14. Create an instance of the QuizGameClient class and assign it to a variable (client).
15. Call the startGame() method on the client instance.
16. End.

Algorithm for QuizGameServer:

1. Start
2. Import the necessary libraries (java.io., java.net., java.util.\*).
3. Define the class QuizGameServer that extends the QuizGame abstract class.
4. Declare the instance variables: questions, answers, and currIndex.
5. Define the constructor for QuizGameServer:
  - 5.1. Initialize the questions and answers lists with the predefined questions and answers.
6. Override the startGame() method from the QuizGame class.
  - 6.1. Print "Starting the game!" message.
  - 6.2. Try the following:
    - 6.2.1. Create a ServerSocket instance and listen for client connections on the specified port.

6.2.2. Accept a client connection and print "Client connected."

6.2.3. Create an ObjectOutputStream and an ObjectInputStream for the client's socket.

6.2.4. Iterate through the questions:

6.2.4.1. Get the current question from the questions list.

6.2.4.2. Call the askQuestion(question, outputStream) method to send the question to the client.

6.2.4.3. Read the client's answer from the input stream and call the evaluateAnswer(answer, outputStream) method to evaluate it.

6.2.5. Write "Game Over" to the output stream and flush it.

6.2.6. Close the output stream, input stream, and the client socket.

6.2.7. Print "Client disconnected."

6.3. Catch IOException and print "Error accepting client connection." if an exception occurs.

6.4. Print "All questions answered. Game over!" message.

7. Override the askQuestion() method from the QuizGame class and leave it empty.

8. Override the evaluateAnswer(String answer) method from the QuizGame class and leave it empty.

9. Define the askQuestion(String question, ObjectOutputStream outputStream) method:

9.1. Write the question to the output stream and flush it.

9.2. Print "Question: " followed by the question.

10. Define the evaluateAnswer(String answer, ObjectOutputStream outputStream) method:

10.1. Get the correct answer for the current question from the answers list.

10.2. Compare the client's answer with the correct answer case-insensitively to determine if it's correct.

10.3. Write the evaluation result (isCorrect) to the output stream and flush it.

10.4. Print the client's answer, the correct answer, and the result ("Correct" or "Incorrect").

11. Define the main method.

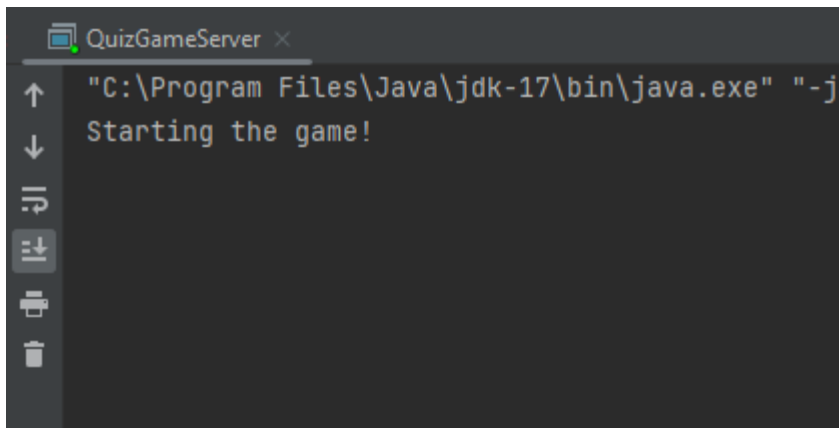
12. Inside the main method:

12.1. Create an instance of the QuizGameServer class and assign it to a variable (server).

12.2. Call the startGame() method on the server instance.

13. End.

### Output:



```
QuizGameServer x
"C:\Program Files\Java\jdk-17\bin\java.exe" "-j
Starting the game!
```

```

C:\Program Files\Java\jdk-17\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.
Connecting to the server..
Received question: What is 1+1
Your answer: 2
Result: Correct
Received question: Who captained the Argentina National team for 2022 World Cup?
Your answer: Lionel Messi
Result: Correct
Received question: Where is the tallest building in the world located?
Your answer: Dubai
Result: Correct
Received question: What is the capital of America?
Your answer: Washington DC
Result: Correct

Process finished with exit code 0
|

```

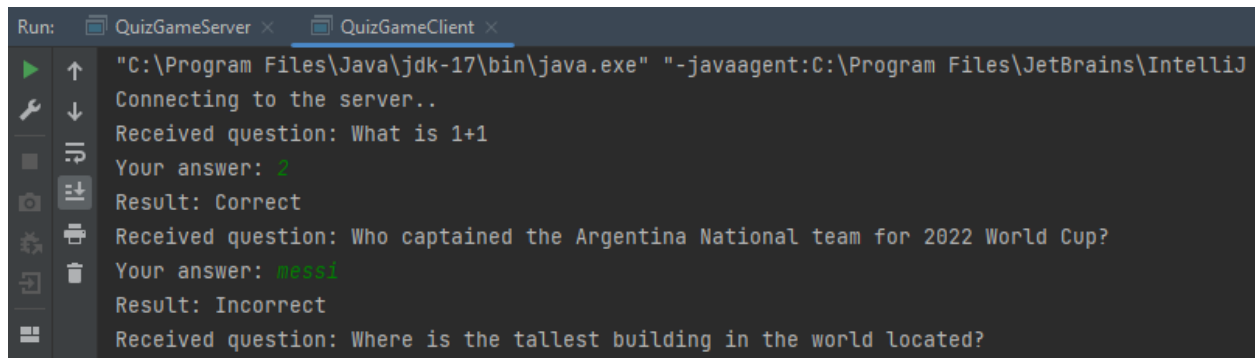
```

QuizGameServer x QuizGameClient x
↑ "C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.
↓ Starting the game!
: Client connected.
: Question: What is 1+1
: Answer: 2
: Correct Answer: 2
: Result: Correct
: Question: Who captained the Argentina National team for 2022 World Cup?
: Answer: Lionel Messi
: Correct Answer: Lionel Messi
: Result: Correct
: Question: Where is the tallest building in the world located?
: Answer: Dubai
: Correct Answer: Dubai
: Result: Correct
: Question: What is the capital of America?
: Answer: Washington DC
: Correct Answer: Washington DC
: Result: Correct
: Client disconnected.
: All questions answered. Game over!

Process finished with exit code 0
|

```

45 //To make user finish a set of questions, we use f



```
Run: QuizGameServer x QuizGameClient x
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
Connecting to the server..
Received question: What is 1+1
Your answer: 2
Result: Correct
Received question: Who captained the Argentina National team for 2022 World Cup?
Your answer: messi
Result: Incorrect
Received question: Where is the tallest building in the world located?
```



# GUI based File Management System

**Exercise Type:** End semester

**GitHub Commit Date:** 18<sup>th</sup> June 2023

**Program Number:** 26

## Problem Definition:

Write a Java program implements a File Management System application that allows users to manage different types of files. It provides functionality to add files of various types (Document, Image, Video), delete files, display all files, and save/load file details to/from a file.

## Algorithm:

1. Start the program.
2. Declare a class named "File" with private instance variables: fileName (String) and fileSize (int).
  - Include a constructor to initialize the fileName and fileSize.
  - Include getters and setters for fileName and fileSize.
  - Include a method named "displayFileDetails" to print the file name and file size.
3. Declare a subclass named "Document" that extends the "File" class.
  - Add a private instance variable "documentType" (String).
  - Include a constructor to initialize the fileName, fileSize, and documentType.
  - Include getters and setters for the documentType.
  - Override the "displayFileDetails" method to display the document type in addition to the file name and file size.

4. Declare a subclass named "Image" that extends the "File" class.
  - Add a private instance variable "resolution" (String).
  - Include a constructor to initialize the fileName, fileSize, and resolution.
  - Include getters and setters for the resolution.
  - Override the "displayFileDetails" method to display the resolution in addition to the file name and file size.
5. Declare a subclass named "Video" that extends the "File" class.
  - Add a private instance variable "duration" (String).
  - Include a constructor to initialize the fileName, fileSize, and duration.
  - Include getters and setters for the duration.
  - Override the "displayFileDetails" method to display the duration in addition to the file name and file size.
6. Declare an interface named "FileManager" with the following methods:
  - addFile(File file): Add a file to the file manager.
  - deleteFile(String fileName): Delete a file from the file manager using the file name.
  - getFiles(): Get a list of files stored in the file manager.
7. Implement the "FileManager" interface with a class named "FileManagerImpl".
  - Include a private instance variable "files" (ArrayList<File>) to store the files.
  - Implement the methods of the "FileManager" interface.
    - addFile(File file): Add the file to the "files" ArrayList.

- `deleteFile(String fileName)`: Iterate over the "files" ArrayList and remove the file with the matching file name.
- `getFiles()`: Return the "files" ArrayList.

8. Declare a class named "FileManagementSystemUI".

- Include private instance variables: `fileManager (FileManager)`, `frame (JFrame)`, `table (JTable)`, `tableModel (DefaultTableModel)`, `fileNameTextField (JTextField)`, `fileSizeTextField (JTextField)`, `fileTypeComboBox (JComboBox<String>)`.
- Create a constructor for "FileManagementSystemUI" that initializes the `fileManager` and calls the "createUI" method.
- Implement the "createUI" method to create the user interface using Swing components:
  - Create a `JFrame` with the title "File Management System".
  - Create a `JPanel` for the file details.
  - Add a `JLabel` and a `JTextField` for the file name.
  - Add a `JLabel` and a `JTextField` for the file size.
  - Add a `JLabel` and a `JComboBox` for the file type.
  - Create a `JPanel` for the buttons.
  - Add `JButtons` for "Add File", "Delete File", and "Refresh".
  - Create a `JTable` with a `DefaultTableModel` to display the file details.
  - Create a main `JPanel` and add the file details panel, `JScrollPane` for the table, and the button panel.
  - Set the content pane of the `JFrame` and make it visible.

9. Implement action listeners for the buttons in the "FileManagementSystemUI" class:

- Implement the "addFileButtonClicked" method to handle the "Add File" button click:
  - Get the values from the text fields and combo box for file name, file size, and file type.
  - Based on the selected file type, prompt the user for additional information (document type, resolution, or duration).
  - Create the respective file object (Document, Image, or Video) and add it to the fileManager.
  - Add a row to the table model with the file details.
  - Clear the input fields.
- Implement the "deleteFileButtonClicked" method to handle the "Delete File" button click:
  - Get the selected row from the table.
  - Retrieve the file name from the selected row.
  - Remove the file from the fileManager and remove the corresponding row from the table model.
- Implement the "refreshButtonClicked" method to handle the "Refresh" button click:
  - Clear the table.
  - Get the list of files from the fileManager.
  - Iterate over the files and add rows to the table model based on the file type (Document, Image, or Video).
- Implement the "clearFields" method to clear the input fields.

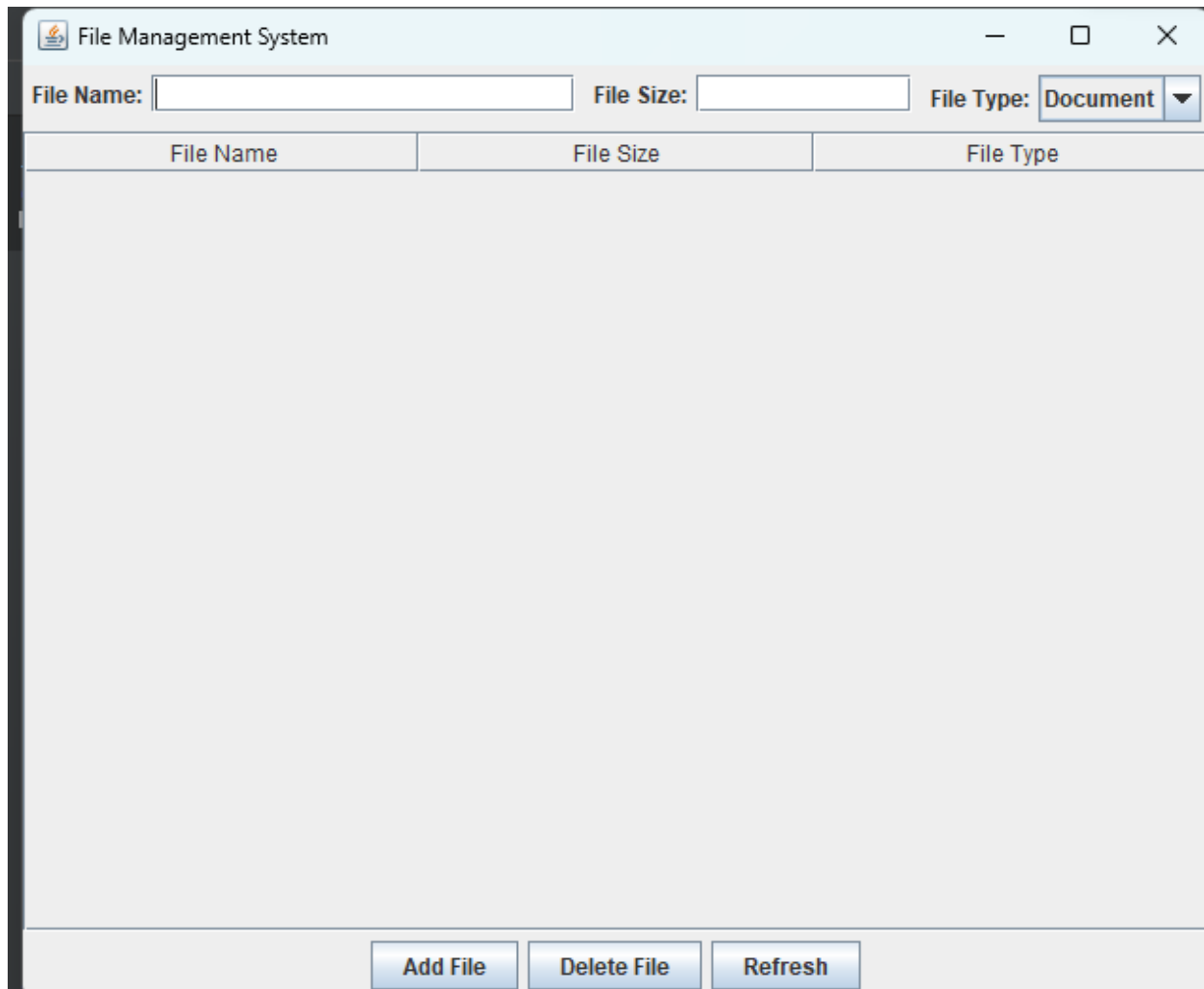
- Implement the "clearTable" method to clear the table model.

10. Implement the main method:

- Create an instance of the "FileManagementSystemUI" class within the "SwingUtilities.invokeLater" to ensure the UI is created on the event dispatch thread.

11. End the program.

### Output:



# Hospital Management System GUI

**Exercise Type:** Project

**GitHub Commit Date:** 25<sup>th</sup> June 2023

**Program Number:** 27

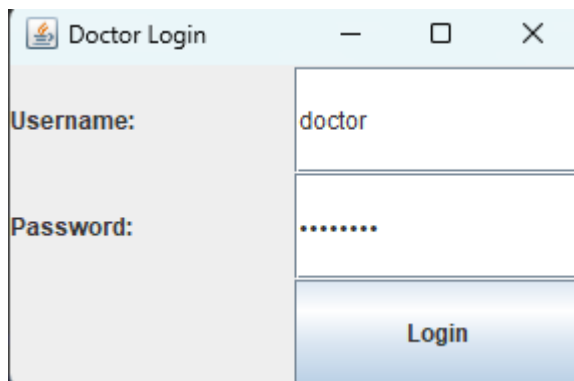
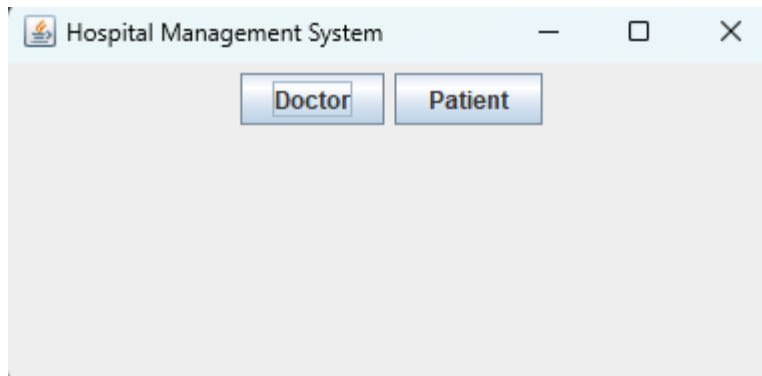
## Problem Definition:

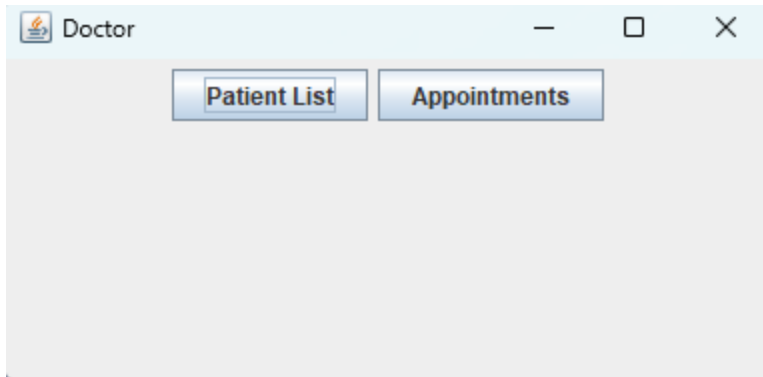
Develop a user-friendly Hospital Management System GUI to streamline administrative tasks and enhance patient record management. The system should provide secure login interfaces for doctors and patients, allowing doctors to view patient details and the appointments and allowing patients to see their 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.

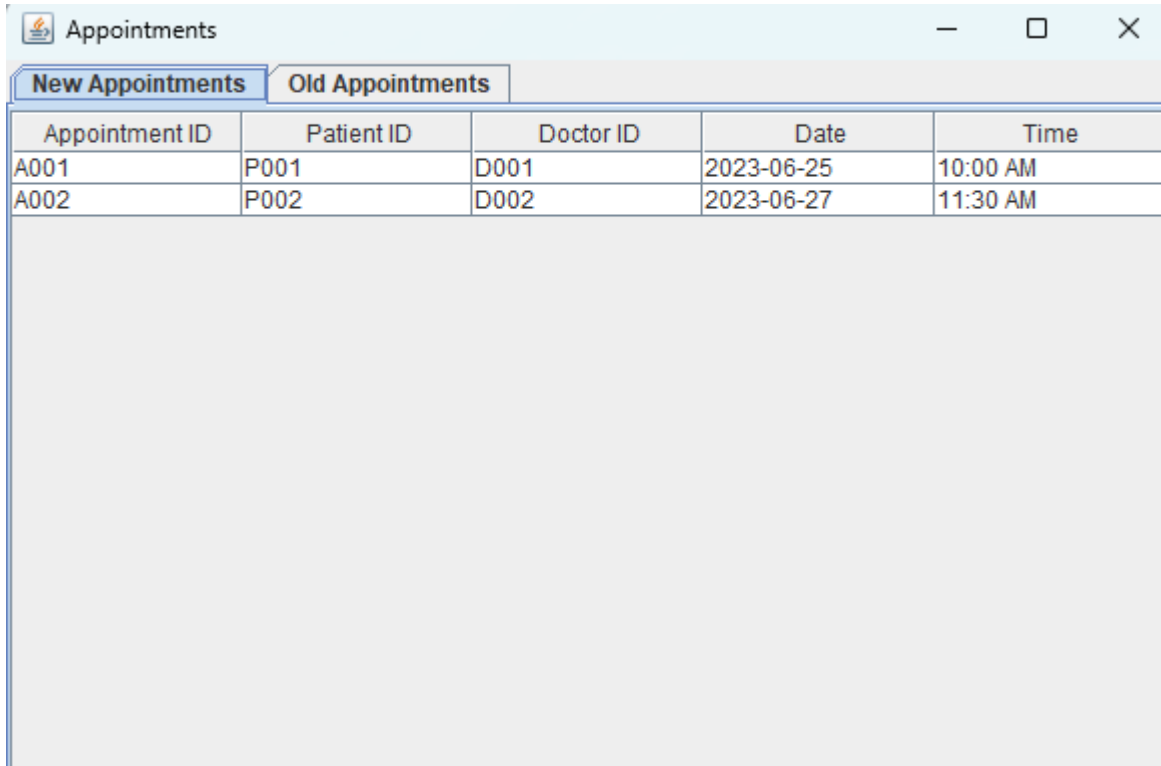
**Output:**



A screenshot of a Java Swing window titled "Patient List". The window has a light blue title bar with standard minimize, maximize, and close buttons. Below the title bar, there is a table with five columns: "Patient ID", "Name", "Latest Visit", "Visiting Purpose", and "Prescribed Medicin...". The table contains two rows of data. Below the table, there is a large, empty light gray rectangular area.

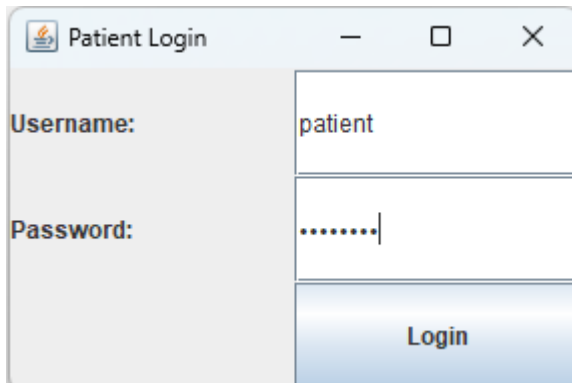
Patient ID	Name	Latest Visit	Visiting Purpose	Prescribed Medicin...
P001	Aishwarya	2023-06-25	Fever	Paracetamol
P002	Soundarya	2023-06-27	Cold and cough	Cetirizine



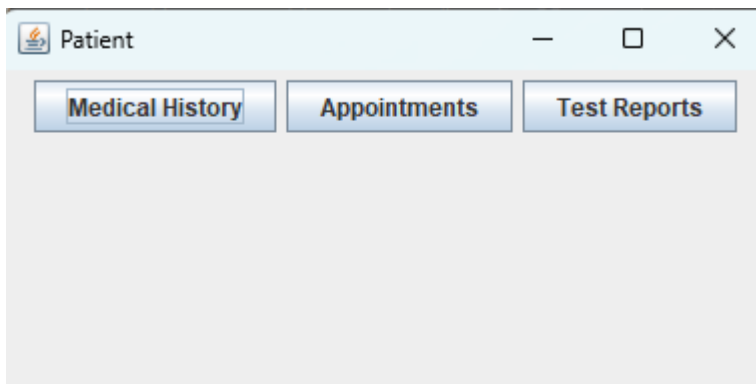


The 'Appointments' window has a title bar with a standard icon, the text 'Appointments', and window control buttons. It features two tabs: 'New Appointments' (selected) and 'Old Appointments'. Below the tabs is a table with five columns: Appointment ID, Patient ID, Doctor ID, Date, and Time. The table contains two rows of data.

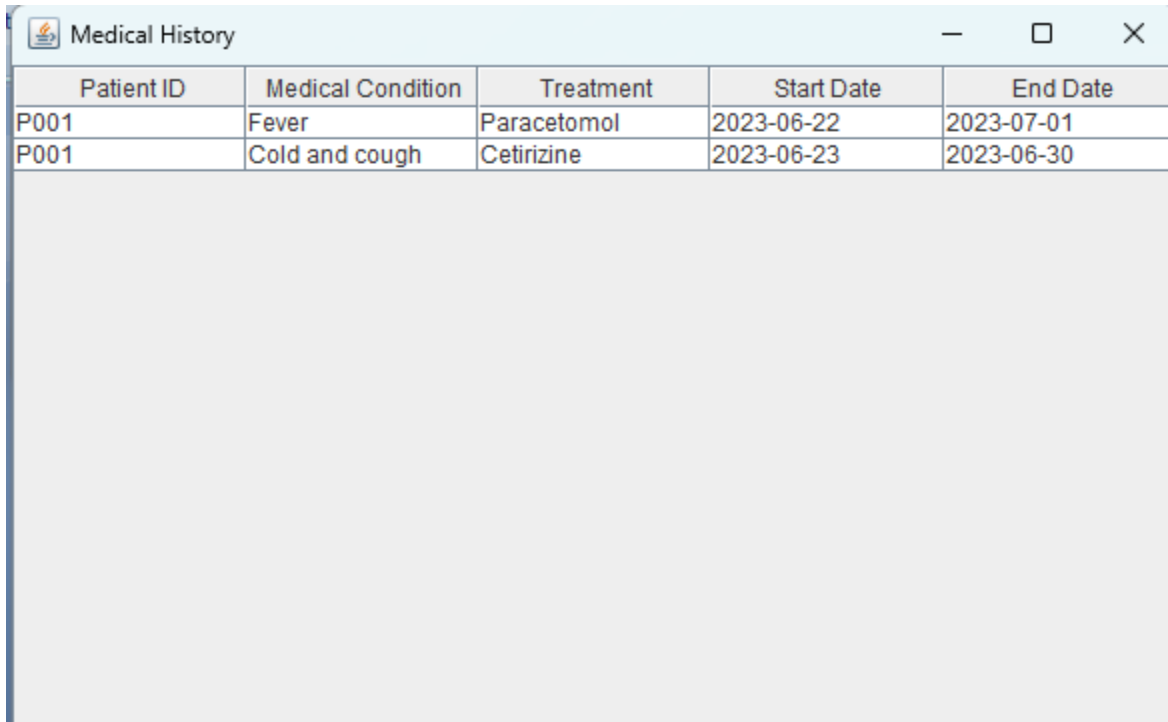
Appointment ID	Patient ID	Doctor ID	Date	Time
A001	P001	D001	2023-06-25	10:00 AM
A002	P002	D002	2023-06-27	11:30 AM



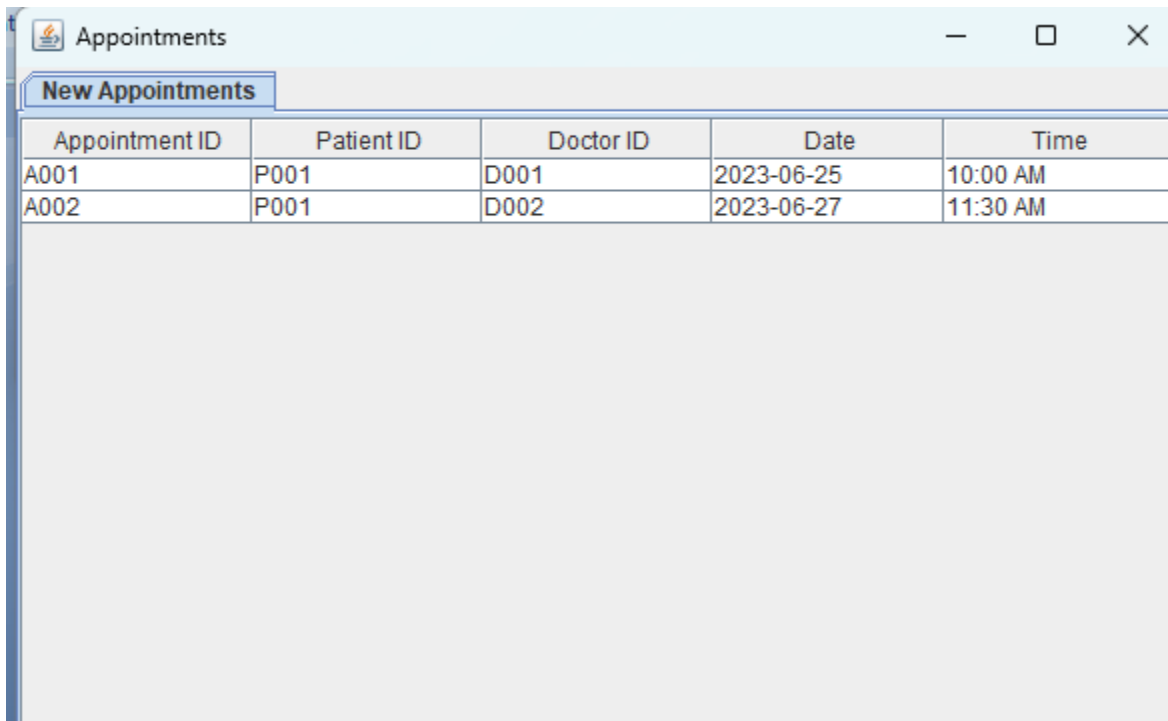
The 'Patient Login' window has a title bar with a standard icon, the text 'Patient Login', and window control buttons. It contains two labels, 'Username:' and 'Password:', each followed by a text input field. The 'Username' field contains the text 'patient'. The 'Password' field contains seven dots. Below the input fields is a 'Login' button.



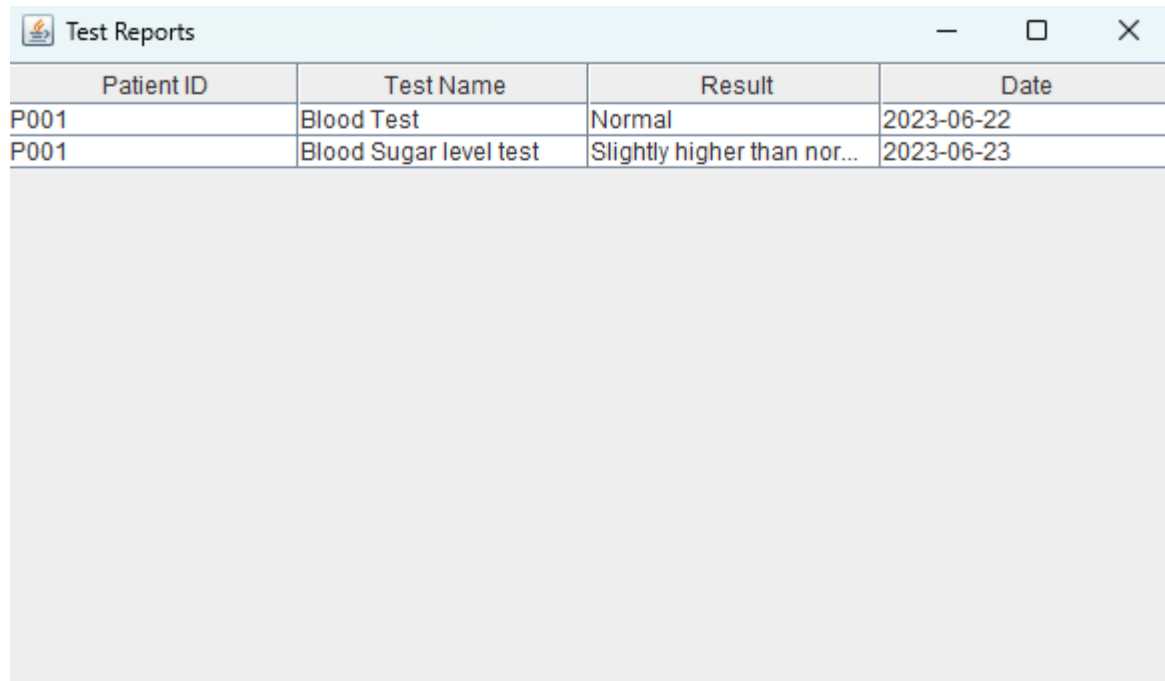
The 'Patient' window has a title bar with a standard icon, the text 'Patient', and window control buttons. It contains three buttons: 'Medical History', 'Appointments', and 'Test Reports'.

A screenshot of a Java Swing window titled "Medical History". It features a table with five columns: Patient ID, Medical Condition, Treatment, Start Date, and End Date. The table contains two rows of data. Below the table is a large, empty rectangular area.

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

A screenshot of a Java Swing window titled "Appointments". It has a tabbed interface with a single tab labeled "New Appointments". Below the tab is a table with five columns: Appointment ID, Patient ID, Doctor ID, Date, and Time. The table contains two rows of data. Below the table is a large, empty rectangular area.

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

A screenshot of a Java Swing window titled "Test Reports". The window has a light blue title bar with standard minimize, maximize, and close buttons. Inside the window is a table with four columns: "Patient ID", "Test Name", "Result", and "Date". The table contains two rows of data. The first row shows Patient ID "P001", Test Name "Blood Test", Result "Normal", and Date "2023-06-22". The second row shows Patient ID "P001", Test Name "Blood Sugar level test", Result "Slightly higher than nor...", and Date "2023-06-23". The table is set against a light gray background.

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