Experiment No : 01

Experiment Name : Write a JAVA Program to Display Image using JFrame

Theory : Java provides a comprehensive set of libraries for building Graphical User Interfaces (GUIs), and one of the key components in GUI development is the JFrame class. This lab explores the creation of a simple Java program to display an image using the Swing library and JFrame. The primary objectives include understanding the basic principles of GUI programming and gaining familiarity with key Swing components.

The JFrame class is a fundamental component of Swing, representing the main window of a GUI application. It provides functionalities such as title setting, size configuration, and default close operations. Through JFrame, developers can create a window to host various GUI components.

Program Structure:

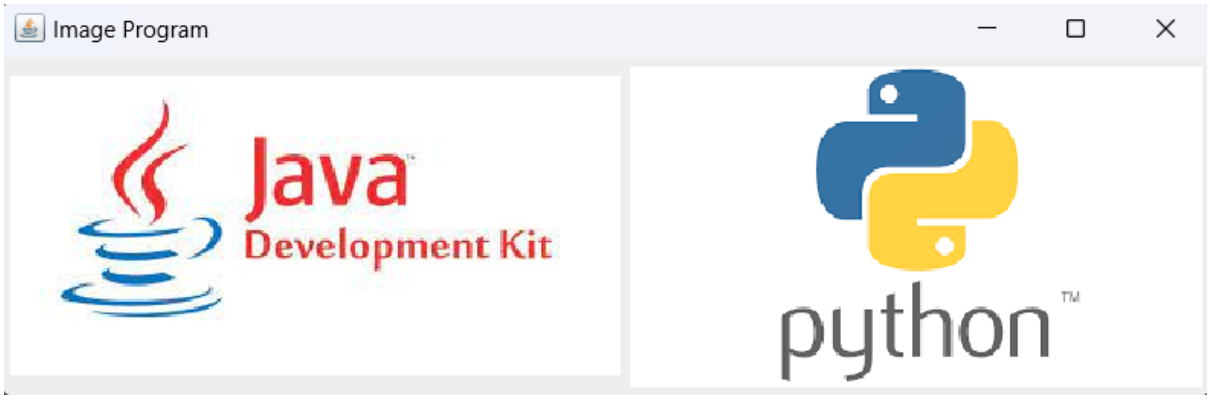
- **ImageDisplay Class:** The ImageDisplay class is defined, extending the JFrame class. In the constructor, essential properties of the JFrame, such as title, size, and default close operation, are configured. Additionally, an instance of ImageIcon is created to represent the image, and a JLabel is used to display the image within the JFrame.
- **Main Method:** The main method serves as the entry point for the program. It ensures that the GUI-related tasks are executed on the Event Dispatch Thread (EDT) using SwingUtilities.invokeLater. An instance of the ImageDisplay class is created, and the JFrame is set to be visible.

Source Code:

```
import java.awt.FlowLayout;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class Jframe extends JFrame {
private ImageIcon image1;
private JLabel label1;
private ImageIcon image2;
private JLabel label2;
Jframe(){
setLayout(new FlowLayout());
image1 = new ImageIcon(getClass().getResource("jdk.jpeg"));
label1 = new JLabel(image1);
```

```
add(label1);
image2 = new ImageIcon(getClass().getResource("python.png"));
label2 = new JLabel(image2);
add(label2);
}
public static void main(String args[]) {
Jframe gui = new Jframe();
gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
gui.setVisible(true);
gui.pack();
gui.setTitle("Image Program");}
```

Output:



Experiment No : 02

Experiment Name : Write a JAVA Program for generating Restaurant Bill

Theory : The objective of this lab is to develop a Java program that generates a restaurant bill based on items ordered by the customer. The program utilizes fundamental concepts of Java programming, including data structures, input/output handling, and basic arithmetic operations. By completing this lab, students gain practical experience in software development and learn how to implement a simple billing system.

Program Structure:

- **Menu Initialization:** The program initializes a menu containing various food items and their corresponding prices. This information is stored in a Java Map data structure, with the item names as keys and the prices as values.
- **Order Placement:** Using the menu, the program prompts the user to place an order by selecting items from the menu and specifying the

quantities. The user input is processed to record the ordered items and their respective quantities, which are stored in another Map data structure.

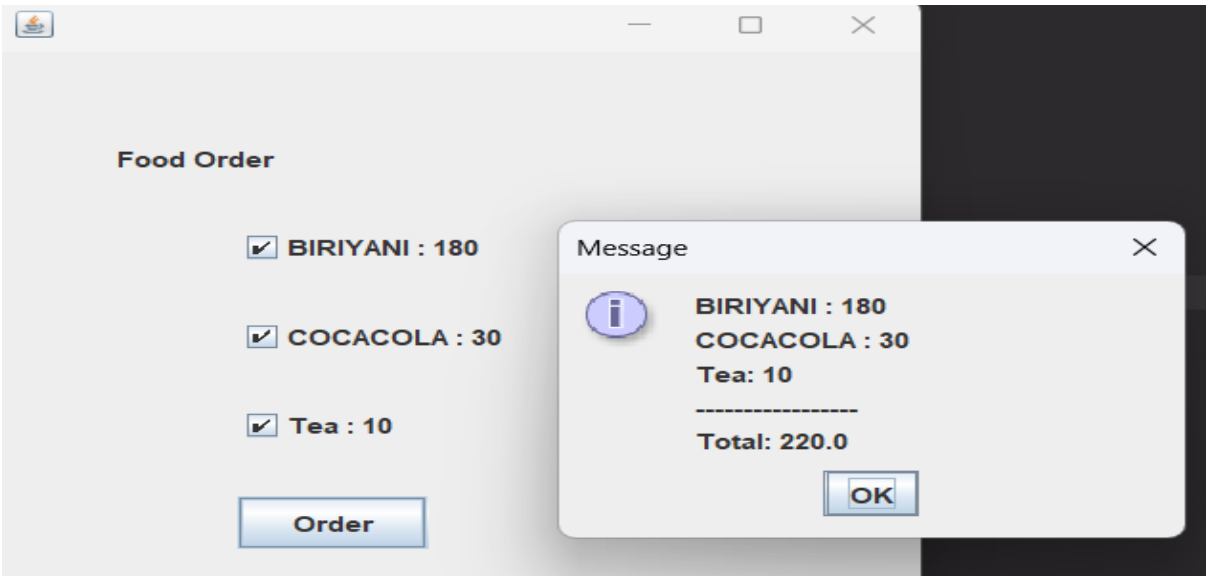
- **Bill Calculation:** Based on the recorded order, the program calculates the total bill by multiplying the price of each item by its quantity and summing up the results.
- **Displaying the Bill:** Finally, the program displays the generated bill, listing the ordered items, their quantities, individual prices, and the total bill amount.

Source Code :

```
import javax.swing.*;
import java.awt.event.*;
public class billrest extends JFrame implements ActionListener{
JLabel l;
JCheckBox cb1,cb2,cb3;
JButton b;
billrest(){
l=new JLabel("Food Order");
l.setBounds(50,50,300,20);
cb1=new JCheckBox("BIRIYANI : 180");
cb1.setBounds(100,100,150,20);
cb2=new JCheckBox("COCACOLA : 30");
cb2.setBounds(100,150,150,20);
cb3=new JCheckBox("Tea : 10");
cb3.setBounds(100,200,150,20);
b=new JButton("Order");
b.setBounds(100,250,80,30);
b.addActionListener(this);
add(l);add(cb1);add(cb2);add(cb3);add(b);
setSize(400,400);
setLayout(null);
setVisible(true);
setDefaultCloseOperation(EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent e){
float amount=0;
String msg="";
if(cb1.isSelected()){
amount+=180;
```

```
msg="BIRIYANI : 180\n";
}
if(cb2.isSelected()){
amount+=30;
msg+="COCACOLA : 30\n";
}
if(cb3.isSelected()){
amount+=10;
msg+="Tea: 10\n";
}
msg+="----------------\n";
JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
}
public static void main(String[] args) {
new billrest();
}
}
```

Output :



Experiment No : 03

Experiment Name : Write a JAVA Program to Create a Student form in GUI

Theory :

The goal of this lab is to design and implement a graphical user interface (GUI) for a student information form using Java's Swing library. The lab aims to familiarize students with GUI programming concepts, event handling, and the creation of interactive forms in Java.

Graphical User Interfaces (GUIs) are essential components in modern software applications, providing users with an interactive and user-friendly experience. Java's Swing library offers a rich set of tools for creating GUIs. This lab focuses on the creation of a student form, which typically includes input fields for personal information.

Program Structure :

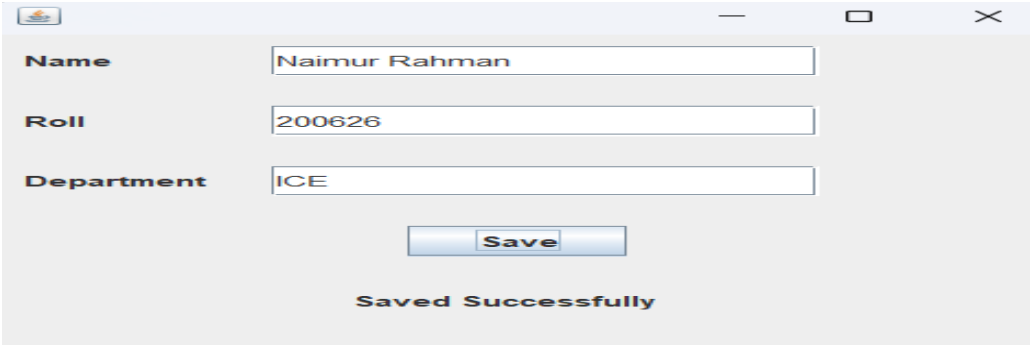
- **JFrame and Layout:** The program utilizes the JFrame class as the main window for the GUI. A GridLayout is used to organize and arrange the components in a grid structure.
- **Form Components:** The form components include JTextField for inputting text, JRadioButton for selecting gender, JComboBox for selecting the grade, and a JButton for submitting the form.
- **Action Handling:** An ActionListener is implemented to handle the action when the submit button is clicked. The entered information is then processed, and a dialog is displayed to present the submitted data.
- **Implementation:** The Java code provided in the previous response serves as the implementation of the student form. It utilizes Swing components to create an interactive and user-friendly form.

Source Code :

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class studentinfo implements ActionListener {
private static JLabel success;
private static JFrame frame;
private static JLabel label1,label2,label3;
private static JTextField userText1, userText2, userText3;
public static void main(String[] args) {
frame = new JFrame();
```

```
panel = new JPanel();
frame.setSize(400, 300);
label1= new JLabel("Name");
label1.setBounds(10,10,80,25);
userText1 = new JTextField("Enter Your Name");
userText1.setBounds(100,10,200,25);
panel.add(userText1);
JTextField userText2 = new JTextField("Enter Your Name");
userText2.setBounds(100,60,200,25);
panel.add(userText2);
JTextField userText3 = new JTextField("Enter Your Name");
button.setBounds(150, 160, 80, 25);
button.addActionListener(new studentinfo());
panel.add(button);
success = new JLabel("");
success.setBounds(130,210,300,25);
panel.add(success);
frame.setVisible(true);
}
@Override
public void actionPerformed(ActionEvent e) {
success.setText("Saved Successfully");
}
}
```

Output :



Experiment No : 04

Experiment Name : Write a JAVA Program to simple calculator in GUI.

Theory : The goal of this lab is to design and implement a graphical user interface (GUI) for a student information form using Java's Swing library. The lab aims to familiarize students with GUI programming concepts, event handling, and the creation of interactive forms in Java. Graphical User Interfaces (GUIs) are essential components in modern software applications, providing users with an interactive and user-friendly experience. Java's Swing library offers a rich set of tools for creating GUIs. This lab focuses on the creation of a student form, which typically includes input fields for personal information.

Program Structure

- **JFrame and Layout:** The program utilizes the JFrame class as the main window for the GUI. A GridLayout is used to organize and arrange the components in a grid structure.
- **Form Components:** The form components include JTextField for inputting text, JRadioButton for selecting gender, JComboBox for selecting the grade, and a JButton for submitting the form.
- **Action Handling:** An ActionListener is implemented to handle the action when the submit button is clicked. The entered information is then processed, and a dialog is displayed to present the submitted data.
- **Implementation:** The Java code provided in the previous response serves as the implementation of the student form. It utilizes Swing components to create an interactive and user-friendly form.

Source Code :

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class test extends JFrame implements ActionListener {
   protected Container c;
   protected Font f2 = new Font("Times New Roman", Font.ROMAN_BASELINE, 38);
   protected Font f1 = new Font("Times New Roman", Font.BOLD, 48);
   protected JButton btn[]=new JButton[18];
   protected JPanel pnl1, pnl2;
   protected JTextArea res;
   protected static int i,r,n1,n2;
   protected static String text="";
   char op;
```

```
boolean opf=false;
test() {
   this.setBounds(20, 29, 526, 635);
   this.setTitle("Calculator");
   Rectangle rctngl = new Rectangle(280, 50, 850, 635);
   this.setMaximizedBounds(rctngl);
   this.setLocationRelativeTo(null);
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   pnl1 = new JPanel();
   pnl1.setLayout(new GridLayout());
   res = new JTextArea("");
   res.setFont(f1);
   pnl1.add(res);
   pnl2 = new JPanel();
   pnl2.setBounds(5, 110, 500, 480);
   pnl2.setLayout(new GridLayout(4, 4));
   for(int i=0;i<=9;i++)
   {
            btn[i]=new JButton(i+"");
   }
   for(int i=7;i<=9;i++) {
       pnl2.add(btn[i]);
   }
   btn[13]= new JButton("/");
   pnl2.add(btn[13]);  // division
   for(int i=4;i<=6;i++) {
       pnl2.add(btn[i]);
   }
   btn[12] = new JButton("x");
   pnl2.add(btn[12
   for(int i=1;i<=3;i++) {
       pnl2.add(btn[i]);
   }
     btn[11] = new JButton("-");
   pnl2.add(btn[11]);
   btn[15] = new JButton("AC");
   btn[15].setForeground(Color.RED);
   pnl2.add(btn[15]);                    // AC
   pnl2.add(btn[0]);            // zero
   btn[10] = new JButton("+");
   pnl2.add( btn[10]);          // Addition
   btn[14] = new JButton("=");
   pnl2.add(btn[14]);     // Result
   for(int i=0;i<=15;i++) {
       btn[i].setFont(f2);
            btn[i].setBackground(Color.MAGENTA);    }
public void actionPerformed(ActionEvent e) {
JButton pb=(JButton)e.getSource();
```

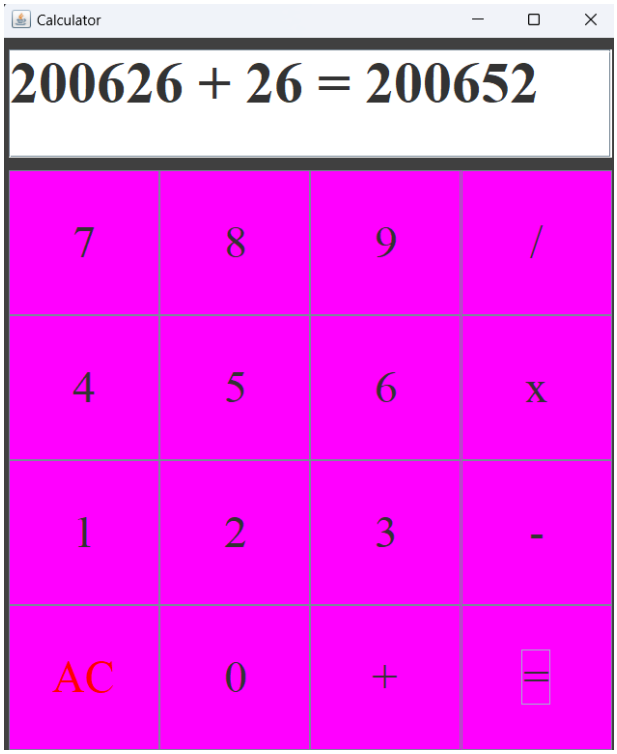
```
        if(pb==btn[15]){
                r=n1=n2=0;
                text="";
                opf=false;
                res.setText("");
        }
        else if(pb==btn[14]){
                n2 = Integer.parseInt(text);
                System.out.println(n2);
//              n2=3;
                eval();
                res.append(" = "+r);
                text="";
        }
        else{for(int j=10;j<=13;j++){
                        if(pb==btn[j]){
                                n1 = Integer.parseInt(res.getText());
                                if(pb==btn[10]){ op='+'; res.append(" + "); opf=true;}
                                if(pb==btn[11]){ op='-'; res.append(" - "); opf=true;}
                                if(pb==btn[12]){ op='*'; res.append(" x "); opf=true;}
                                if(pb==btn[13]){ op='/'; res.append(" / "); opf=true;}
                        }}

                if(opf==false){
                        for(i=0;i<=9;i++){
                                if(pb==btn[i]){
                                        String t=res.getText();
                                        t+=i;
                                        res.setText(t);
                                  }
                           }
                }
                else{
                        for(i=0;i<=9;i++){
                                if(pb==btn[i]){
                                        text +=i;
                                        res.append(Integer.toString(i));
                                  }
                           }
                }
        }
   }
        int eval(){
                switch(op)
                {
                        case '+': r=n1+n2;  break;
                        case '-': r=n1-n2;  break;
                        case '*': r=n1*n2;  break;
```

```
                        case '/': r=n1/n2;  break;
                    }
public static void main(String[] args) {
      test frm = new test();
      frm.setVisible(true);
   }

}
```

Output :



Experiment No : 05

Experiment Title: Write a JAVA Program to Create a Thread using Thread Class

Theory: In Java, multithreading is a technique that allows concurrent execution of two or more parts of a program, thereby improving performance and responsiveness. Threads can be created by extending the Thread class or implementing the Runnable interface. In this experiment, we focus on creating a thread by extending the Thread class.

Program Structure:

- **MyThread Class:**
  - Extends the Thread class.
  - Overrides the run() method to define the code to be executed by the thread.
- **Main Class:**
  - Contains the main method.
  - Creates an instance of the MyThread class.
  - Starts the thread using the start() method.

Source Code:

```
class A extends Thread{
   public void run(){
      for(int i=1;i<=5;i++)
      {
         System.out.println("\t From Thread A : i= "+i);
      }
      System.out.println("Exit from A");
   }
}
 class B extends Thread {

   public void run( ){
      for(int j=1;j<=5;j++){
         System.out.println("\t From Thread B : j= "+j);
      }
      System.out.println("Exit from B");
   }
```

```
}
class C extends Thread {

   public void run( ){
      for(int k=1;k<=5;k++){
         System.out.println("\t From Thread C : K= "+k);
      }
      System.out.println("Exit from C");
   }
}
public class TreadTest {
   public static void main(String[] args) {
      System.out.println("The Thread is running");
      new A( ).start( );
      new B( ).start( );
      new C( ).start( );
   }
}
```

Output:

```
The Thread is running
            From Thread A : i= 1
            From Thread A : i= 2
            From Thread B : j= 1
            From Thread B : j= 2
            From Thread C : K= 1
            From Thread C : K= 2
            From Thread A : i= 3
            From Thread B : j= 3
            From Thread C : K= 3
            From Thread A : i= 4
            From Thread B : j= 4
            From Thread C : K= 4
            From Thread A : i= 5
Exit from A
            From Thread B : j= 5
            From Thread C : K= 5
Exit from C
Exit from B
```

Experiment No: 06
Experiment Title: Write a JAVA Program to Call Threads using run() Method.

Theory: In Java, multithreading allows concurrent execution of multiple parts of a program, enhancing performance and responsiveness. Threads can be created by extending the **Thread** class or implementing the **Runnable** interface. When a thread is created by extending the **Thread** class, we can start it by calling the **start()** method, which internally calls the **run()** method. Alternatively, we can directly call the **run()** method to execute the thread, but it won't create a new thread of execution; instead, it runs in the context of the current thread. In this experiment, we explore calling threads using the **run()** method.

Program Structure:

- **MyThread Class:**
    - ❖ Extends the **Thread** class.
    - ❖ Overrides the **run()** method to define the code to be executed by the thread.
- **Main Class:**
    - ❖ Contains the main method.
    - ❖ Creates an instance of the **MyThread** class.
    - ❖ Calls the **run()** method directly.

Source Code:

```
class X implements Runnable {
   public void run() {
      for (int i = 1; i <= 10; i++) {
         System.out.println("\tThreadX :  " + i);
      }
      System.out.println("End of ThreadX");
   }
}

public class thrdclassRunIn {
   public static void main(String args[]) {
      X runnable = new X();
      Thread threadX = new Thread(runnable);
      threadX.start();
      System.out.println("End of main Thread ");
```

```
   }
public static void main(String args[]) {
      X runnable = new X();
      Thread threadX = new Thread(runnable);
      threadX.start();

}
```

Output:

```
        ThreadX :  2
        ThreadX :  3
        ThreadX :  4
        ThreadX :  5
        ThreadX :  6
        ThreadX :  7
        ThreadX :  8
        ThreadX :  9
        ThreadX :  10
End of ThreadX
```

Experiment No: 07
Experiment Title: Write a JAVA Program to Illustrate yield() and sleep() Methods using Threads

Theory: In Java, threads can be used to achieve concurrent execution of tasks. The **yield()** and **sleep()** methods are used to control the execution of threads.

- **yield() Method:** This method is used to pause the currently executing thread temporarily, allowing other threads of the same priority to execute. If there are no threads of the same priority, or if they are already running, the yielded thread may continue its execution.
- **sleep() Method:** This method is used to pause the execution of the currently executing thread for a specified amount of time. It allows other threads to execute while the current thread is sleeping.

Program Structure:

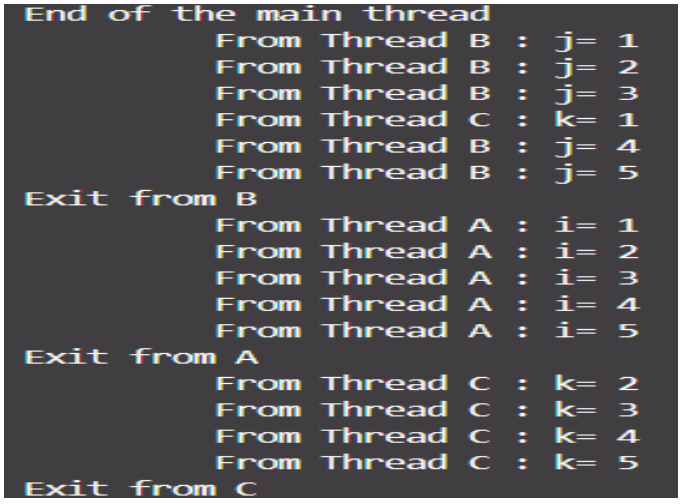
- **YieldThread Class:**
  - ❖ Extends the **Thread** class.
  - ❖ Overrides the **run()** method to demonstrate the usage of the **yield()** method.
- **SleepThread Class:**
  - ❖ Extends the **Thread** class.
  - ❖ Overrides the **run()** method to demonstrate the usage of the **sleep()** method.

Source Code:

```
class A extends Thread {
   public void run() {
      for (int i = 1; i <= 5; i++) {
         if (i == 1)
         Thread.yield();
         System.out.println("\t From Thread A : i= " + i);
class B extends Thread {
   public void run() {
      for (int j = 1; j <= 5; j++) {
         System.out.println("\t From Thread B : j= " + j);
         if (j == 3) {
            Thread.yield();
         }
```

```
      System.out.println("Exit from B");
   }
class C extends Thread {
   public void run() {
      for (int k = 1; k <= 5; k++) {
         System.out.println("\t From Thread C : k= " + k);
         if (k == 1) {
            try {
               sleep(1000); // Pause for 1 second
            }
      System.out.println("Exit from C");
   }
public class ThreadMethod {
   public static void main(String[] args) {
      A threadA = new A();
      B threadB = new B();
      C threadC = new C();
      threadA.start();
      threadB.start();
      System.out.println("End of the main thread");
   }
}
```

Output:



Experiment No: 08
Experiment Title: Write a JAVA Program to Use Priority of Thread.

Theory: In Java, threads are scheduled for execution by the JVM based on their priority. Thread priority is an integer value that determines the importance or urgency of a thread's execution. The thread scheduler attempts to allocate CPU time to higher priority threads before lower priority threads.

Thread priorities are represented by integer values ranging from 1 to 10, where 1 is the lowest priority and 10 is the highest. Threads with higher priority values are more likely to be scheduled for execution over threads with lower priority values. By default, all threads inherit the priority of the parent thread, which is usually the main thread. However, you can explicitly set the priority of a thread using the **setPriority()** method.

Program Structure:

- **PriorityThread Class:**
    - ❖ Extends the **Thread** class.
    - ❖ Overrides the **run()** method to display the thread's name and priority.
- **Main Class:**
    - ❖ Contains the main method.
    - ❖ Creates instances of **PriorityThread** class with different priorities.
    - ❖ Starts the threads to demonstrate the effect of thread priority on scheduling.

Source Code:

```
class PriorityThread extends Thread {
   public void run() {
      // Display the name and priority of the thread
      System.out.println("Thread Name: " +
Thread.currentThread().getName() + ", Priority: " +
Thread.currentThread().getPriority());
   }
}

public class Main {
   public static void main(String[] args) {
```

```
        // Create instances of PriorityThread with different priorities
        PriorityThread thread1 = new PriorityThread();
        PriorityThread thread2 = new PriorityThread();
        PriorityThread thread3 = new PriorityThread();

        // Set priorities for the threads
        thread1.setPriority(Thread.MIN_PRIORITY); // Minimum priority
        thread2.setPriority(Thread.NORM_PRIORITY); // Normal priority
        thread3.setPriority(Thread.MAX_PRIORITY); // Maximum priority

        // Start the threads
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

Output:

```
Thread C started
        From Thread C : i= 1
        From Thread C : i= 2
        From Thread C : i= 3
        From Thread C : i= 4
        From Thread C : i= 5
Exit from C
        From Thread B : i= 1
        From Thread A : i= 1
        From Thread B : i= 2
        From Thread A : i= 2
        From Thread B : i= 3
        From Thread B : i= 4
        From Thread B : i= 5
Exit from B
        From Thread A : i= 3
        From Thread A : i= 4
        From Thread A : i= 5
Exit from A
```

Experiment No : 09

Experiment Title: Write a client and server program in Java to establish a connection between them.

Theory:
In Java, client-server communication is facilitated through the use of sockets. Sockets provide the communication mechanism between two processes, typically a client and a server, running on different machines or within the same machine.

- **Server:** The server program waits for client requests and responds to them accordingly.
- **Client:** The client program initiates communication with the server by sending requests and receiving responses.

Program Structure:

- **Server Program:**
    - ❖ Listens for incoming connections from clients.
    - ❖ Accepts client connections and handles client requests.
- **Client Program:**
    - ❖ Establishes a connection to the server.
    - ❖ Sends requests to the server and receives responses.

Source Code:
Server sided Code:

```
import java.io.*;
import java.net.*;

public class Server {
   public static void main(String[] args) {
      try {
         // Create a server socket bound to port 9999
         ServerSocket serverSocket = new ServerSocket(9999);
         System.out.println("Server started. Waiting for client connection...");

         // Accept client connection
         Socket clientSocket = serverSocket.accept();
         System.out.println("Client connected.");
```

```
        // Get input and output streams
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);

        // Read client message
        String message = in.readLine();
        System.out.println("Received from client: " + message);

        // Send response to client
        out.println("Message received by server.");

        // Close streams and sockets
        in.close();
        out.close();
        clientSocket.close();
        serverSocket.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
   }
}
```

Client Sided Code:

```
import java.io.*;
import java.net.*;

public class Client {
   public static void main(String[] args) {
      try {
        // Create a socket to connect to the server running on localhost at
port 9999
        Socket socket = new Socket("localhost", 9999);

        // Get input and output streams
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        // Send message to server
```

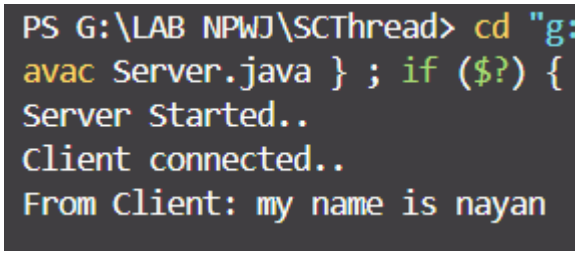
```
            out.println("Hello from client.");

            // Receive response from server
            String response = in.readLine();
            System.out.println("Response from server: " + response);

            // Close streams and socket
            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

Server Waiting For Client……..!!



Client Connected from Server…….!!