CONTENT

Ex.No:1	Implementation of and Exception handling concepts with different type of Exception
DATE:	

AIM:

To create a Implementation of and Exception handling concepts with different type of Exception.

ALGORITHM:

Step 1 : **Try Block**: Code that might throw an exception is placed inside a try block.

Step 2 : **Catch Block**: When an exception occurs, control is transferred to the catch block, where the exception can be handled.

Step 3 : **Finally Block**: A block that executes after try and catch, regardless of whether an exception was thrown or not. It's often used for cleanup tasks.

Step 4 : **Throw Statement**: Allows a programmer to manually trigger an exception.

Step 5 : **Checked Exceptions**: Must be either caught or declared in the method signature.

Step 6 : **Unchecked Exceptions**: Subclasses of Runtime Exception and do not need to be explicitly handled.

Step 7 : Run the Program.

PROGRAM:

```
import java.util.Scanner;
public class ExceptionHandling{
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        int numerator,denominator;
        double result;
        try{
            System.out.println("Enter the numerator:");
            numerator = scanner.nextInt();
            System.out.println("Enter the denominator:");
            denominator = scanner.nextInt();
            result = divide(numerator,denominator);
            System.out.println("Result and  division:"+result);
        }
        catch(ArithmeticException e){
            System.out.println("ArithmeticException:"+e.getMessage());
            System.out.println("cannot divide by zero, pls enter a non zero denominator");
        }
        catch(NumberFormatException e){
            System.out.println("NumberFormatException"+e.getMessage());
            System.out.println("Invalid input please enter valid interger");
        }
        catch(Exception e){
            System.out.println("Exception  occured:"+e.getMessage());
        }
        finally{
            scanner.close();
            System.out.println("program execution completed");
        }
    }

    public static double divide(int numerator, int denominator){
        return (double) numerator/denominator;
    }
}
```

OUTPUT:

```
Command Prompt
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\STUDENT LOGIN>d:

D:\>cd java

D:\java>set path=C:\Program Files (x86)\Java\jdk1.8.0_66\bin

D:\java>javac ExceptionHandling.java

D:\java>java ExceptionHandling
Enter the numerator:50
Enter the denominator:10
Result of division:5
Scanner closed. Finally block always executes

D:\java>
```

RESULT:

Thus the program executed successfully.

Ex.No:2	Build a Swing application to implement metric conversion
DATE:	

AIM:

To Create a Build a Swing application to implement metric conversion.

ALGORITHM:

Step 1 : Set Up Your Development Environment

Make sure you have Java and an IDE like IntelliJ IDEA or Eclipse installed.

Step 2 : **Imports**: The necessary Swing and AWT classes are imported.

Step 3 : **Class Declaration**: The MetricConverter class extends JFrame and implements ActionListener.

Step 4 : Sets up the frame, size, and layout.

Step 5 : Creates input fields, a combo box for conversion types, and a button to trigger conversion.

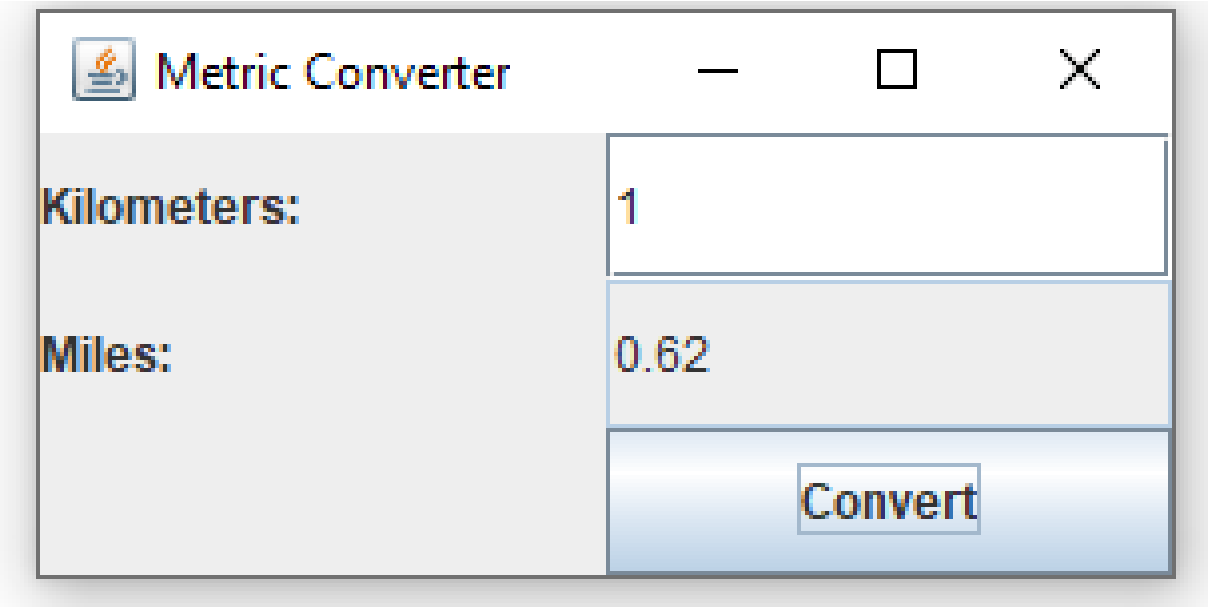
Step 6 : Adds these components to the frame.

PROGRAM:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MetricConverterApp extends JFrame {

    private JLabel inputLabel;
    private JTextField inputField;
    private JLabel outputLabel;
    private JTextField outputField;
    private JButton convertButton;

    public MetricConverterApp() {
        setTitle("Metric Converter");
        setSize(300, 150);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLayout(new GridLayout(3, 2));

        inputLabel = new JLabel("Kilometers:");
        inputField = new JTextField();
        outputLabel = new JLabel("Miles:");
        outputField = new JTextField();
        outputField.setEditable(false);
        convertButton = new JButton("Convert");

        add(inputLabel);
        add(inputField);
        add(outputLabel);
        add(outputField);
        add(new JLabel());
        add(convertButton);

        convertButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                convert();
            }
        });
    }

    private void convert() {
        try {
            double kilometers = Double.parseDouble(inputField.getText());
            double miles = kilometers * 0.621371;
            outputField.setText(String.format("%.2f", miles));
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, "Invalid input. Please enter a number.", "Error", JOption-
Pane.ERROR_MESSAGE);
        }
    }
```

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

      SwingUtilities.invokeLater(new Runnable() {
          @Override
          public void run() {
              MetricConverterApp app = new MetricConverterApp();
              app.setVisible(true);
          }
      });
    }
}
```

OUTPUT:



RESULT:

Thus the program executed successfully.

Ex.No: 3	Use Grid Layout to design a calculator and simulate the functions of a simple calculator
DATE:	

AIM:

To Use Grid Layout to design a calculator and simulate the functions of a simple calculator.

ALGORITHM:

Step 1 : **Imports**: The necessary Swing and AWT classes are imported for GUI components and event handling.

Step 2 : **Class Declaration**: The Simple Calculator class extends JFrame and implements ActionListener.

Step 3 : **Constructor**

- Creates a display field at the top to show input and results.
- Sets up the main frame and its layout.
- Sets up a button panel using `GridLayout`, organizing buttons into a 4x4 grid.

Step 4 : **Button Creation** - An array of button labels is defined, and buttons are creat- ed and added to the panel with an action listener.

Step 5 : **ActionListener Implementation** - The actionPerformed method processes button clicks.

Step 6 : **Calculation Logic** - The calculate method performs the arithmetic operations based on the selected operator.

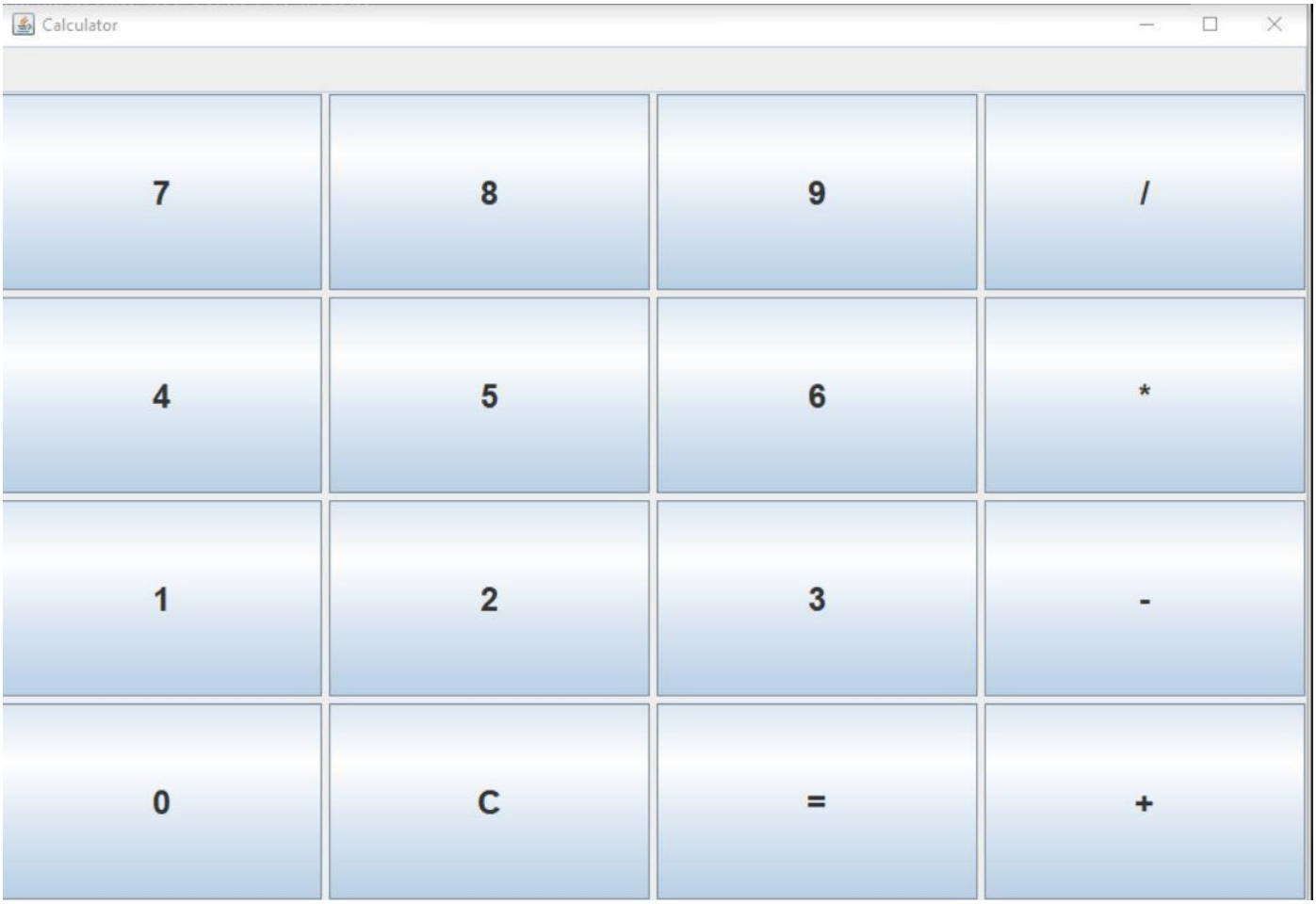
Step 7 : **Main Method**: Initializes the Swing application using Swing Utilities.invokeLater.

PROGRAM:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Calculator extends JFrame implements ActionListener {
    private JTextField display;
    private JButton[] buttons;
    private String[] labels = {
        "7", "8", "9", "/", "4", "5", "6", "*", "1", "2", "3", "-", "0", "c", "=", "+"
    };
    private String operand1 = "";
    private String operand2 = "";
    private String operator = "";

    public Calculator() {
        display = new JTextField();
        display.setEditable(false);
        display.setHorizontalAlignment(JTextField.RIGHT);
        display.setFont(new Font("Arial", Font.BOLD, 24));

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(4, 4, 5, 5));

        buttons = new JButton[16];
        for (int i = 0; i < 16; i++) {
            buttons[i] = new JButton(labels[i]);
            buttons[i].setFont(new Font("Arial", Font.BOLD, 24));
            buttons[i].addActionListener(this);
            panel.add(buttons[i]);
        }

        setLayout(new BorderLayout());
        add(display, BorderLayout.NORTH);
        add(panel, BorderLayout.CENTER);
        setTitle("Calculator");
        setSize(400, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if (command.charAt(0) >= '0' && command.charAt(0) <= '9') {
            if (operator.equals("")) {
                operand1 += command;
                display.setText(operand1);
            } else {
                operand2 += command;
                display.setText(operand2);
            }
```

```
        } else if (command.equals("c")) {
            operand1 = operand2 = operator = "";
            display.setText("");
        } else if (command.equals("=")) {
            if (!operand1.isEmpty() && !operand2.isEmpty() && !operator.isEmpty()) {
                int result = 0;
                switch (operator) {
                    case "+":
                        result = Integer.parseInt(operand1) + Integer.parseInt(operand2);
                        break;
                    case "-":
                        result = Integer.parseInt(operand1) - Integer.parseInt(operand2);
                        break;
                    case "*":
                        result = Integer.parseInt(operand1) * Integer.parseInt(operand2);
                        break;
                    case "/":
                        result = Integer.parseInt(operand1) / Integer.parseInt(operand2);
                        break;
                }
                display.setText("" + result);
                operand1 = "" + result;
                operand2 = "";
                operator = "";
            }
        }else{
            if (operator.isEmpty() || operand2.isEmpty()) {
                operator = command;
            }else{
                int result = 0;
                switch (operator) {
                    case "+":
                        result = Integer.parseInt(operand1) + Integer.parseInt(operand2);
                        break;
                    case "-":
                        result = Integer.parseInt(operand1) - Integer.parseInt(operand2);
                        break;
                    case "*":
                        result = Integer.parseInt(operand1) * Integer.parseInt(operand2);
                        break;
                    case "/":
                        result = Integer.parseInt(operand1) / Integer.parseInt(operand2);
                        break;
                }
                operand1 = "" + result;
                operator = command;
                operand2 = "";
                display.setText(operand1);
            }
        }
    }
    public static void main(String[] args) {
        new Calculator();
    }
}
```

OUTPUT:



RESULT:

Thus the program executed successfully.

Ex.No: 4	Create a Color palette with a matrix of buttons using Applet
DATE:	

AIM:

To create a Create a Color palette with a matrix of buttons using Applet.

ALGORITHM:

Step 1 : **Imports**: The necessary classes are imported from the java.applet, java.awt, and java.awt.event packages.

Step 2 : **Class Declaration**: The ColorPaletteApplet class extends Applet and implements ActionListener.

Step 3 : **Color Matrix**: A 2D array of `Color` objects defines the colors for the buttons.

Step 4 : **init Method**: This method sets up the applet:

- A GridLayout is created based on the dimensions of the color array.
- For each color, a button is created, its background set to the corresponding color, and an action listener added.

Step 5 : **actionPerformed Method**: When a button is clicked, this method retrieves the background color of the button that was clicked and sets it as the background color of the applet.

Step 6 : **paint Method**: This method can be used to draw text or other graphics. In this case, it displays a message prompting the user to click a color.

PROGRAM:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ColorPalette extends JFrame {
    private JButton[][] buttons;
    private Color[][] colors;

    public ColorPalette() {
        setLayout(new GridLayout(10, 10));
        buttons = new JButton[10][10];
        colors = new Color[10][10];

        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                // Create colors
                int red = Math.min(i * 25, 255);
                int green = Math.min(j * 25, 255);
                int blue = Math.min((i + j) * 25, 255);
                colors[i][j] = new Color(red, green, blue);

                // Create buttons
                buttons[i][j] = new JButton();
                buttons[i][j].setBackground(colors[i][j]);
                buttons[i][j].addActionListener(new ButtonListener());

                // Add button to the frame
                add(buttons[i][j]);
            }
        }

        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    private class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            JButton button = (JButton) e.getSource();
            Color color = button.getBackground();
            System.out.println("Selected color: " + color);
        }
    }

    public static void main(String[] args) {
        new ColorPalette();
    }
}
```

OUTPUT:

ZZ

```
D:\susmitha>javac ColorPaletteApplet.java

D:\susmitha>appletviewer ColorPaletteApplet.html
Warning: Can't read AppletViewer properties file: C:\Users\STUDENT LOGIN\.hotjava\properties Using defaults.
Color Selected: java.awt.Color[r=0,g=0,b=255]
Color Selected: java.awt.Color[r=255,g=0,b=0]
Color Selected: java.awt.Color[r=255,g=255,b=0]
Color Selected: java.awt.Color[r=0,g=255,b=0]
Color Selected: java.awt.Color[r=0,g=255,b=255]
Color Selected: java.awt.Color[r=255,g=0,b=255]
```

RESULT:

Thus the program executed successfully.

Ex.No: 5	To invoke a servlet from HTML forms
DATE:	

AIM:

To create a invoke a servlet from HTML forms.

ALGORITHM:

Step 1 : **Create HTML Form:**

- Define an HTML file that contains the form.
- Set the form's `action` attribute to the servlet's URL.
- Use the `POST` method for form submission.

Step 2 : **Create Servlet:**

- Extend HttpServlet.
- Override the doPost method to handle form data.
- Retrieve parameters using request.getParameter().

Step 3 : **Deploy the Application:**

- Compile the servlet.
- Place the HTML file in the appropriate directory.
- Start the servlet container

Step 4 : **Access the Form:**

- Open a web browser and navigate to the HTML form.
- Fill out the form and submit.

Step 5 : **Display the Result:**

- ✓ The servlet processes the data and returns a response.

Step 6 : Run the program.

PROGRAM:

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Servlet Form</title>
</head>
<body>
    <form action="myservlet" method="post">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

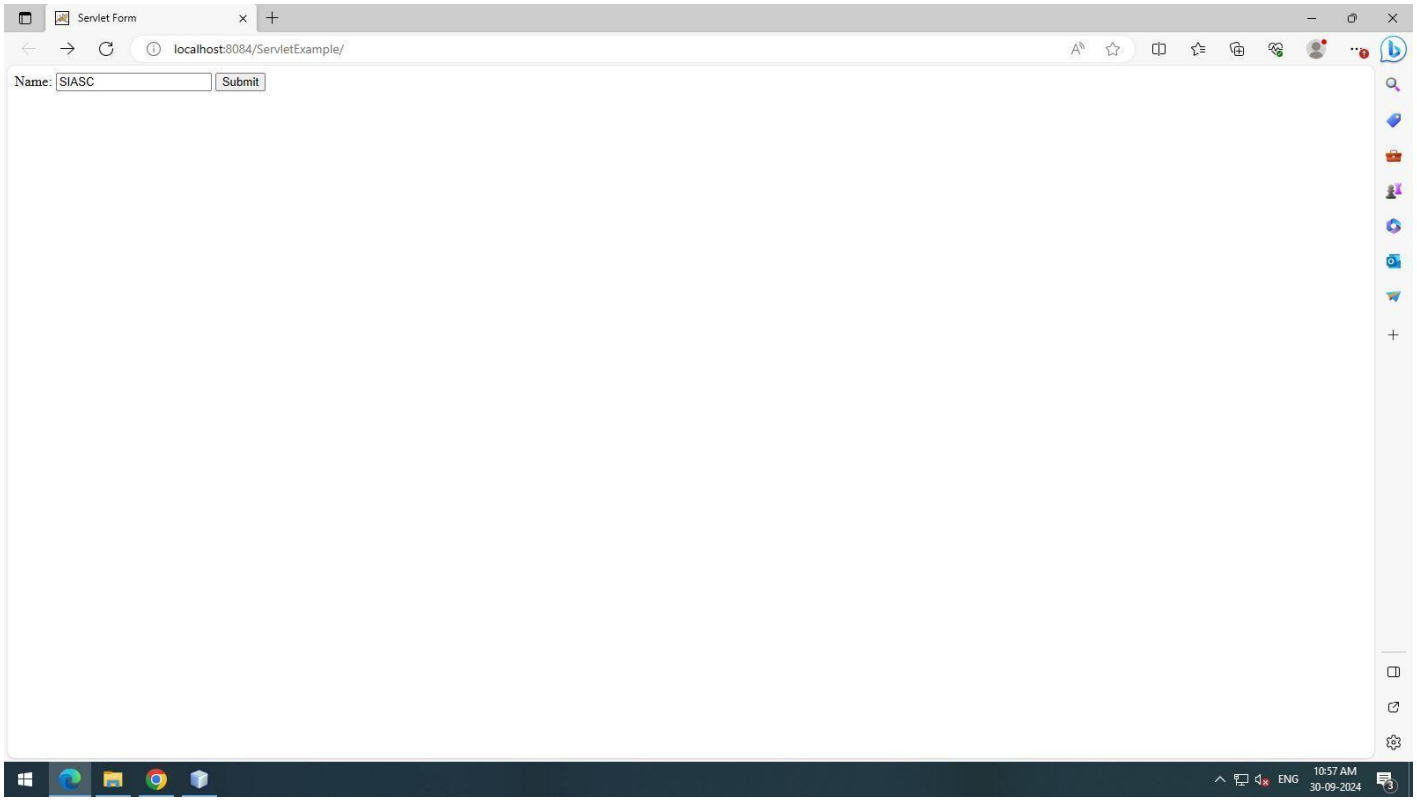
MyServlet.java

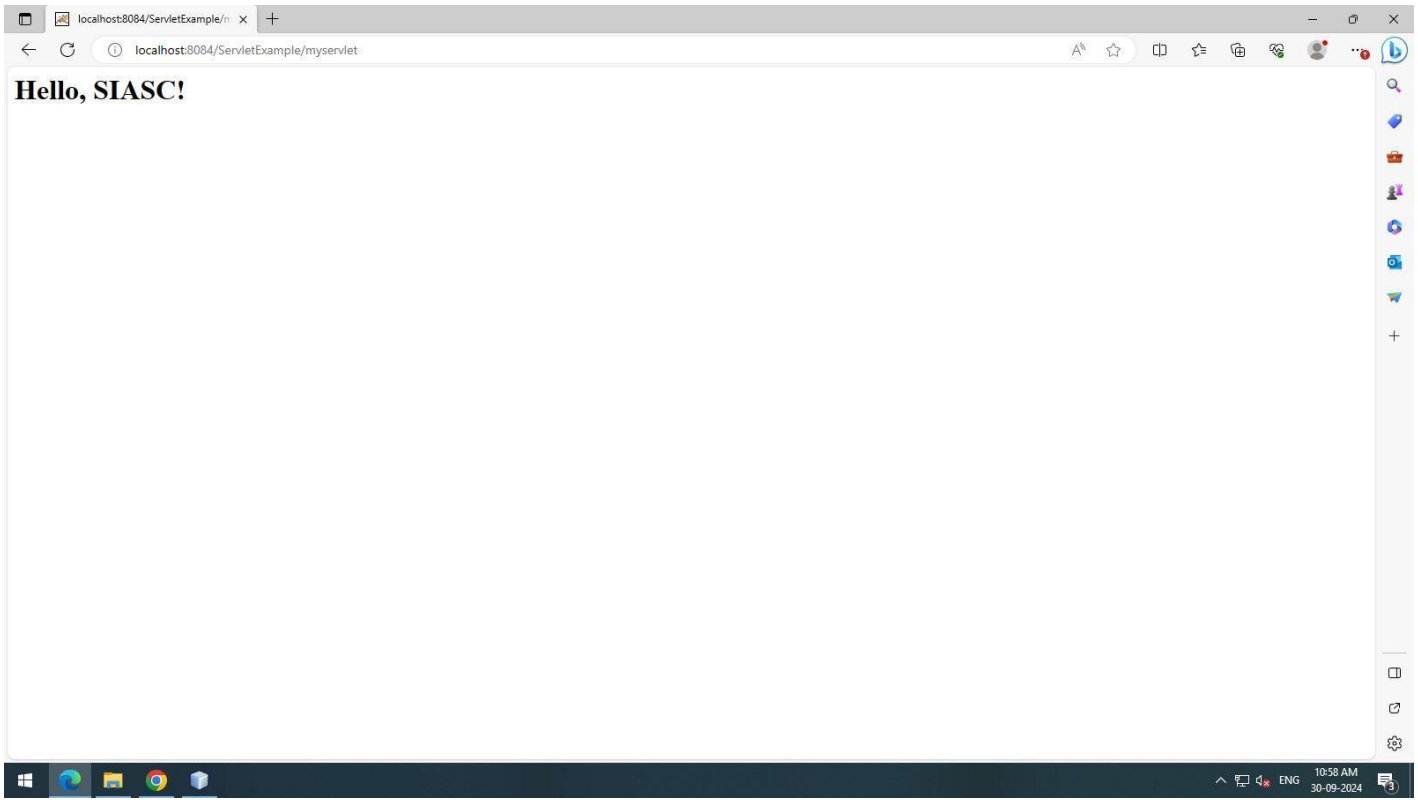
```
package com.example.servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/myservlet")
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = request.getParameter("name");
        response.setContentType("text/html");
        response.getWriter().println("<html><body>");
        response.getWriter().println("<h1>Hello, " + name + "!</h1>");
        response.getWriter().println("</body></html>");
    }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd"
version="3.1">
  <servlet>
      <servlet-name>MyServlet</servlet-name>
      <servlet-class>com.example.servlets.MyServlet</servlet-class>
   </servlet>

   <servlet-mapping>
      <servlet-name>MyServlet</servlet-name>
      <url-pattern>/Myservlet</url-pattern>
   </servlet-mapping>
</web-app>
```

OUTPUT:





RESULT:

Thus the program executed successfully.

Ex.No:6	To invoke servlet from Applets
DATE:	

AIM:

To create a invoke servlet from Applets.

ALGORITHM:

Step 1 : **Set Up the Environment:**

- ✓ Ensure you have a servlet container.
- ✓ Prepare the servlet that will handle requests.

Step 2 : **Create the Applet**:

- ➢ Develop an applet that sends an HTTP request to the servlet.
- ➢ Use HttpURLConnection for the request.

Step 3 : **Compile and Deploy**:

- Compile both the servlet and applet.
- Deploy the servlet on a web server (like Apache Tomcat).

Step 4 : **Run the Applet**:

- Ensure that the applet is executed in an environment that allows network access, possibly requiring signing the applet.

Step 5 : **Handle Security**:

- Be aware of applet security restrictions and consider signing your applet if necessary.

PROGRAM:

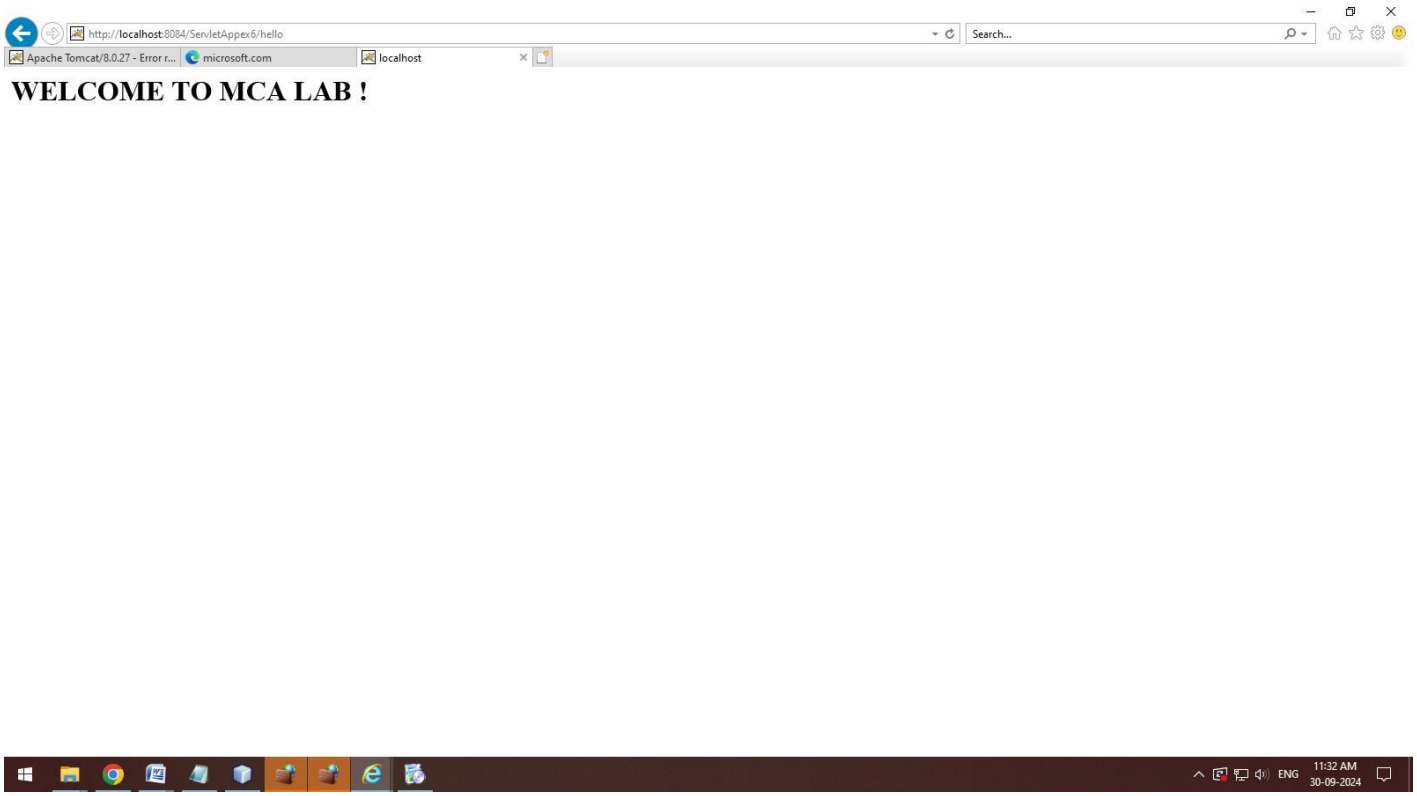
Applet.html

```
<html>
<body>
    <applet code="HelloApplet.class" width="400" height="300"></applet>
</body>
</html>
```

HelloServlet.java

```
package com.example;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        response.getWriter().println("<h1>WELCOME TO MCA LAB !</h1>");
    }
}
```

OUTPUT:



RESULT:

Thus the program executed successfully.

Ex.No: 7	To invoke servlet from JSP
DATE:	

AIM:

To create a invoke servlet from JSP.

ALGORITHM:

Step 1 : Define the Servlet

- Create a Java Servlet class that handles HTTP requests.
  Here's a basic example:

Step 2 : **Client Request**: User submits form data from the JSP page.

Step 3 : **Form Submission**: The form's action points to the servlet's URL pattern.

Step 4 : **Response**: The servlet sends a response back to the client (in this case, the browser).

Step 5 : **Open NetBeans.**

- Create a New Project:
- Go to File > New Project.
- Select Java Web > Web Application.
- Click Next, give your project a name, and specify a location.
- Choose a server (e.g., Apache Tomcat) and click Finish.

Step 6 : **Add a Servlet**:

- Right-click on the Source Packages folder in your project.
- Choose New > Servlet.
- Name your servlet (e.g., MyServlet), and specify the package name.
- Click Next and then Finish.

Step 7 : **Add a JSP File :**

- Right-click on the Web Pages folder.
- Choose New > JSP.
- Name it (e.g., index.jsp).

PROGRAM:

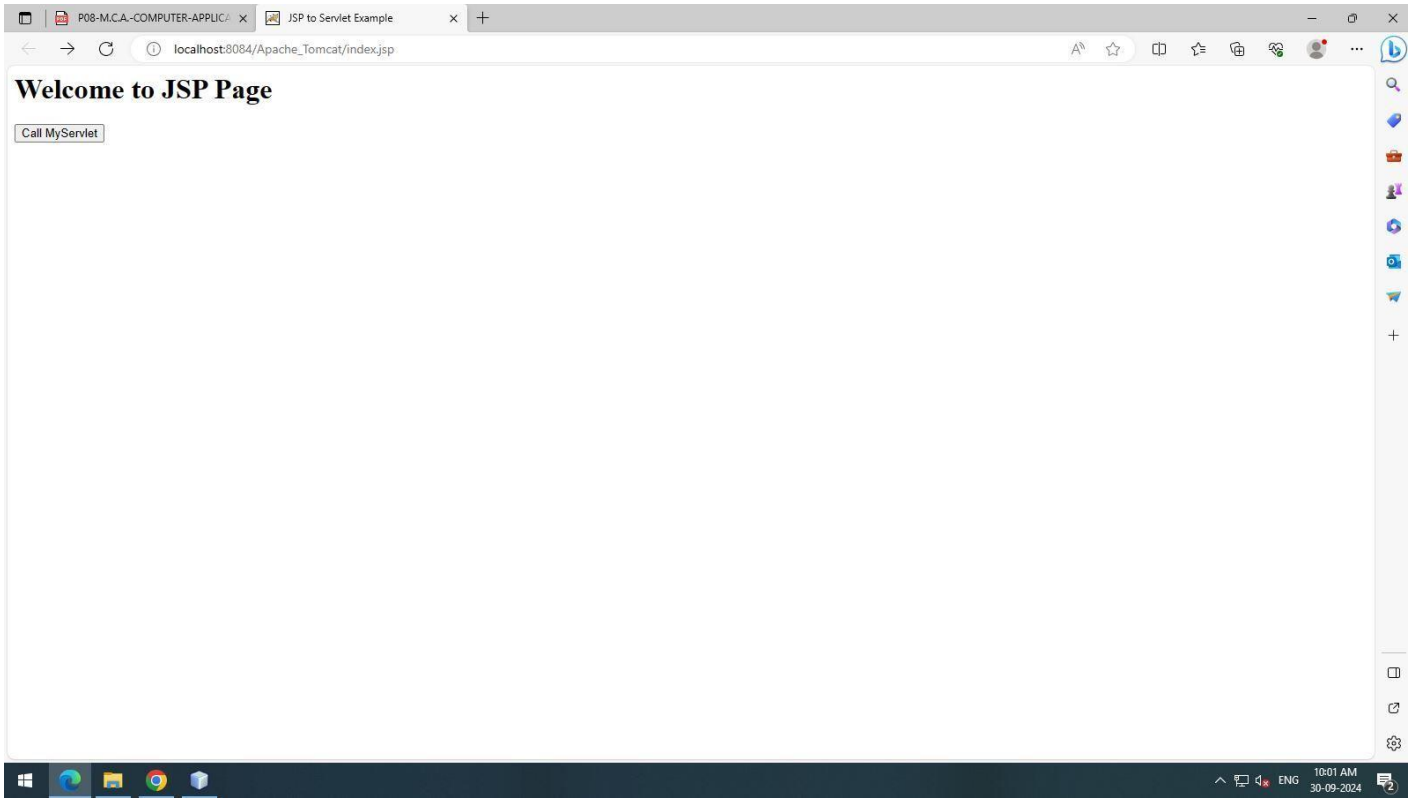
MyServlet.java

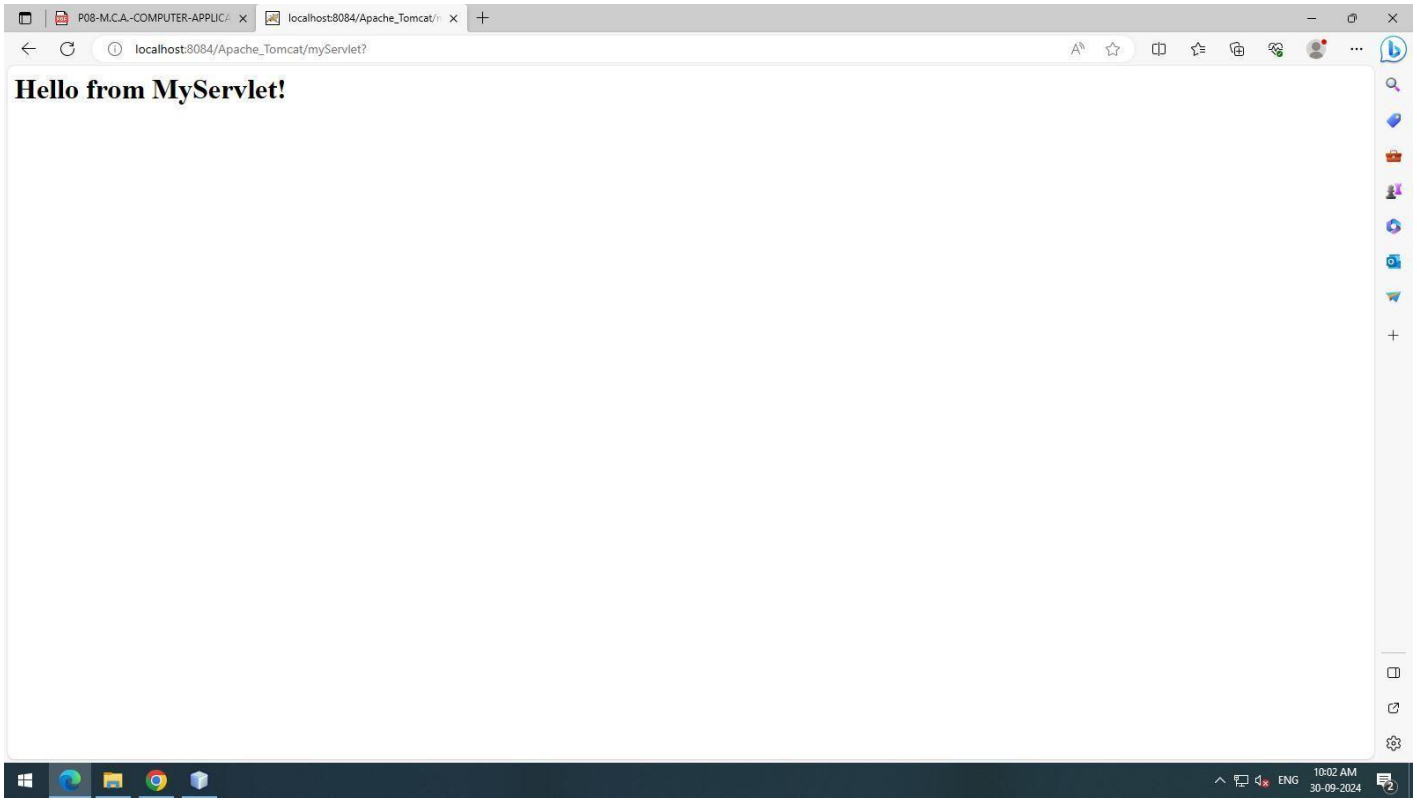
```
package com.example;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/myServlet")
public class MyServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        response.getWriter().println("<h1>Hello from MyServlet!</h1>");
    }
}
```

index.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>JSP to Servlet Example</title>
</head>
<body>
    <h1>Welcome to JSP Page</h1>
    <form action="myServlet" method="get">
        <input type="submit" value="Call MyServlet">
    </form>
</body>
</html>
```

OUTPUT:





RESULT:

Thus the program executed successfully.

Ex.No: 8	Implement message communication using Network Programming
DATE:	

AIM:

To Implement message communication using Network Programming

ALGORITHM:

Step 1 : Server-Side Implementation (Java)

- The server listens on a specific port for client connections and communicates with the clients using sockets.

Step 2 : Client-Side Implementation (Java)

- The client connects to the server, sends a message, and waits for the server's response.

Step 3 : **Process Algorithm** (Message Communication in Network Programming): Server:

- **Start Server Socket**: The server starts a ServerSocket on a specific port.
- **Listen for Client Connections**: The server waits for client connections using the accept() method.

Step 4 : **Respond to Client**: The server sends a response back to the client using the output stream.

Step 5 : **Close the Connection**: After communication is complete, the server closes the socket.

Step 6 **: Client**

- **Create a Client Socket**: The client establishes a connection to the server using Socket(hostname, port).
- **Send a Message**: The client sends a message to the server by writing to the socket's output stream.

Step 7 : . Message Flow:

- ✓ Connection: Client initiates a connection to the server.
- ✓ Message Sent: Client sends a message to the server.

PROGRAM:

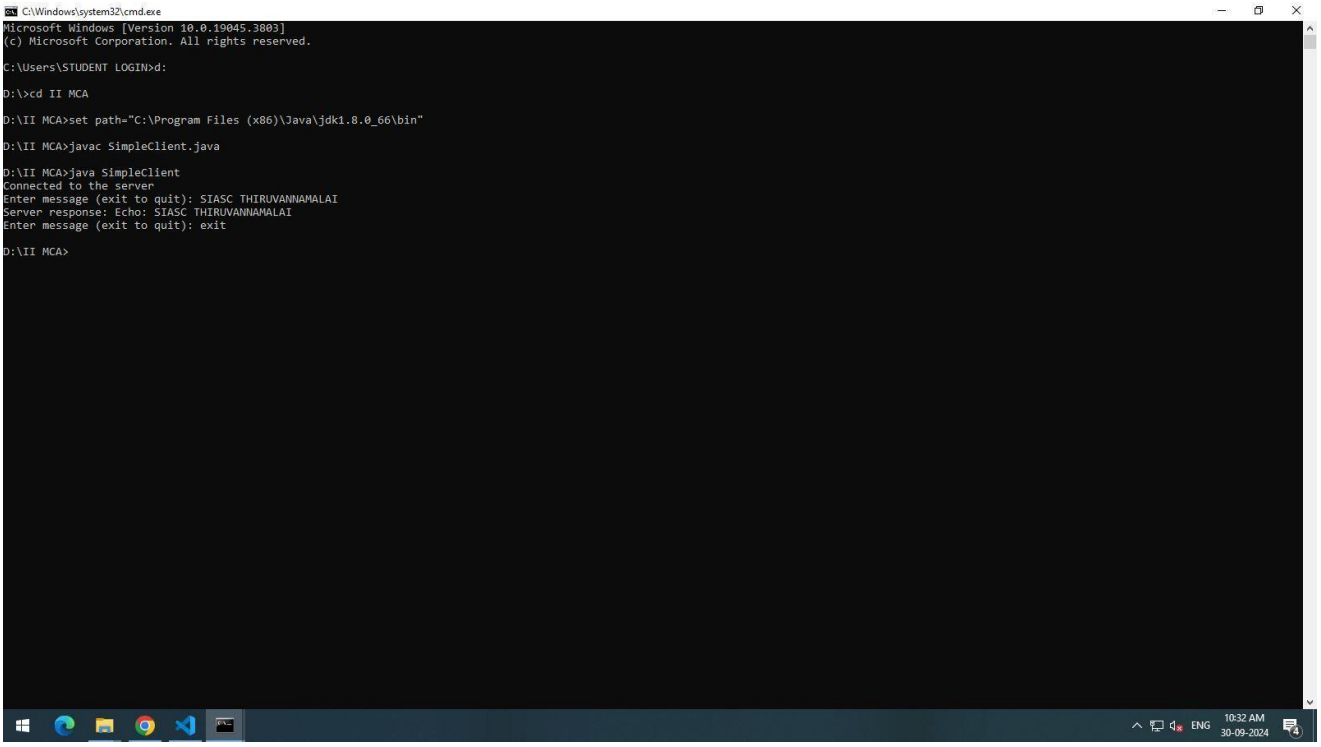
Client side:

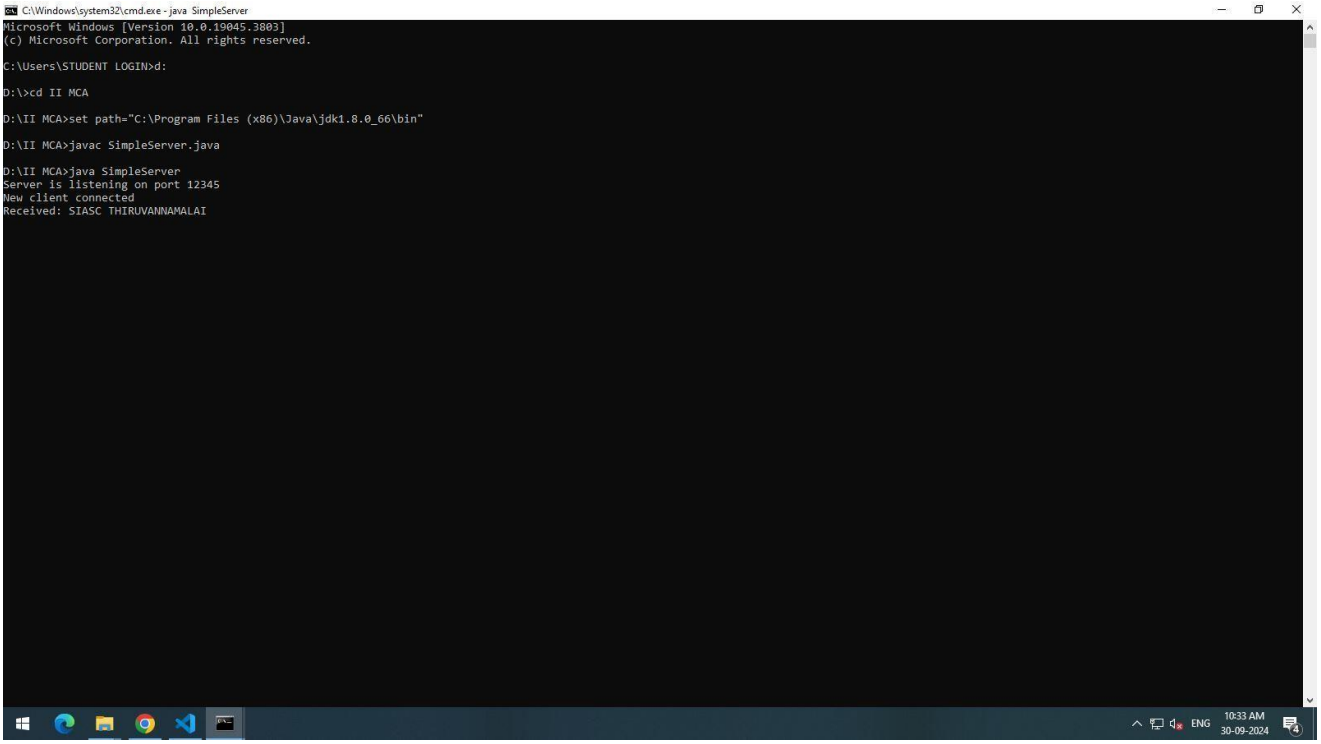
```
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class SimpleClient {
    public static void main(String[] args) {
        String hostname = "localhost"; // Server hostname
        int port = 12345; // Server port

        try (Socket socket = new Socket(hostname, port);
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             Scanner scanner = new Scanner(System.in)) {

            System.out.println("Connected to the server");

            String userInput;
            while (true) {
                System.out.print("Enter message (exit to quit): ");
                userInput = scanner.nextLine();

                if ("exit".equalsIgnoreCase(userInput)) {
                    break; // Exit the loop and close connection
                }

                out.println(userInput); // Send message to server
                String response = in.readLine(); // Read response from server
                System.out.println("Server response: " + response);
            }
        } catch (IOException e) {
            System.out.println("Client error: " + e.getMessage());
        }
    }
}
```

Server side:

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

public class SimpleServer {
    public static void main(String[] args) {
        int port = 12345; // Port number to listen on

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);

            while (true) {
                Socket socket = serverSocket.accept(); // Accept client connection
                System.out.println("New client connected");

                // Handle client in a new thread
                new Thread(new ClientHandler(socket)).start();
            }
        } catch (IOException e) {
            System.out.println("Server error: " + e.getMessage());
        }
    }
}

class ClientHandler implements Runnable {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {

            String message;
            while ((message = in.readLine()) != null) {
                System.out.println("Received: " + message);
                out.println("Echo: " + message); // Echo message back to client
            }
        } catch (IOException e) {
            System.out.println("Client disconnected: " + e.getMessage());
        } finally {
            try {
                socket.close();
            } catch (IOException e) {
                System.out.println("Error closing socket: " + e.getMessage());
            }
        }
    }
}
```

OUTPUT:





RESULT:

Thus the program executed successfully.

Ex.No: 9	Write a program to connect databases using JDBC
DATE:	

AIM:

To create Write a program to connect databases using JDBC.

ALGORITHM:

Step 1 : Connecting to a database using **JDBC (Java Database Connectivity)** involves the following steps:

1. **Load the JDBC Driver**: This is typically done by using Class.forName(), which loads the JDBC driver class.
2. **Establish a Connection**: Create a connection to the database using the DriverManager.getConnection() method.
3. **Create a Statement**: Use the connection object to create a Statement or PreparedStatement to execute SQL queries.

Step 2 : Process Algorithm for JDBC Database Connection:

1. Load JDBC Driver:
   - Use Class.forName("com.mysql.cj.jdbc.Driver") to load the JDBC driver for MySQL (other databases like PostgreSQL or Oracle have different driver classes).
2. Establish Connection:
   - Use DriverManager.get Connection(jdbcUrl, username,password) to establish a connection to the database.
3. Create SQL Statement:
   - Once connected, use the createStatement() method from the Connection object to create a Statement.

Step 3 **:** Driver and Database Setup

- **Driver**: You need to have the correct JDBC driver in your pro- ject. For example, for **MySQL**, you can download the MySQL JDBC driver (e.g., mysql-connector-java.jar) and add it to your project's classpath.

Step 4 : Common JDBC Drivers:

- MySQL: com.mysql.cj.jdbc.Driver
- PostgreSQL: org.postgresql.Driver
- Oracle: oracle.jdbc.driver.OracleDriver
- SQL Server: com.microsoft.sqlserver.jdbc.SQLServerDriver

Step 5 : Database URL Format**:**

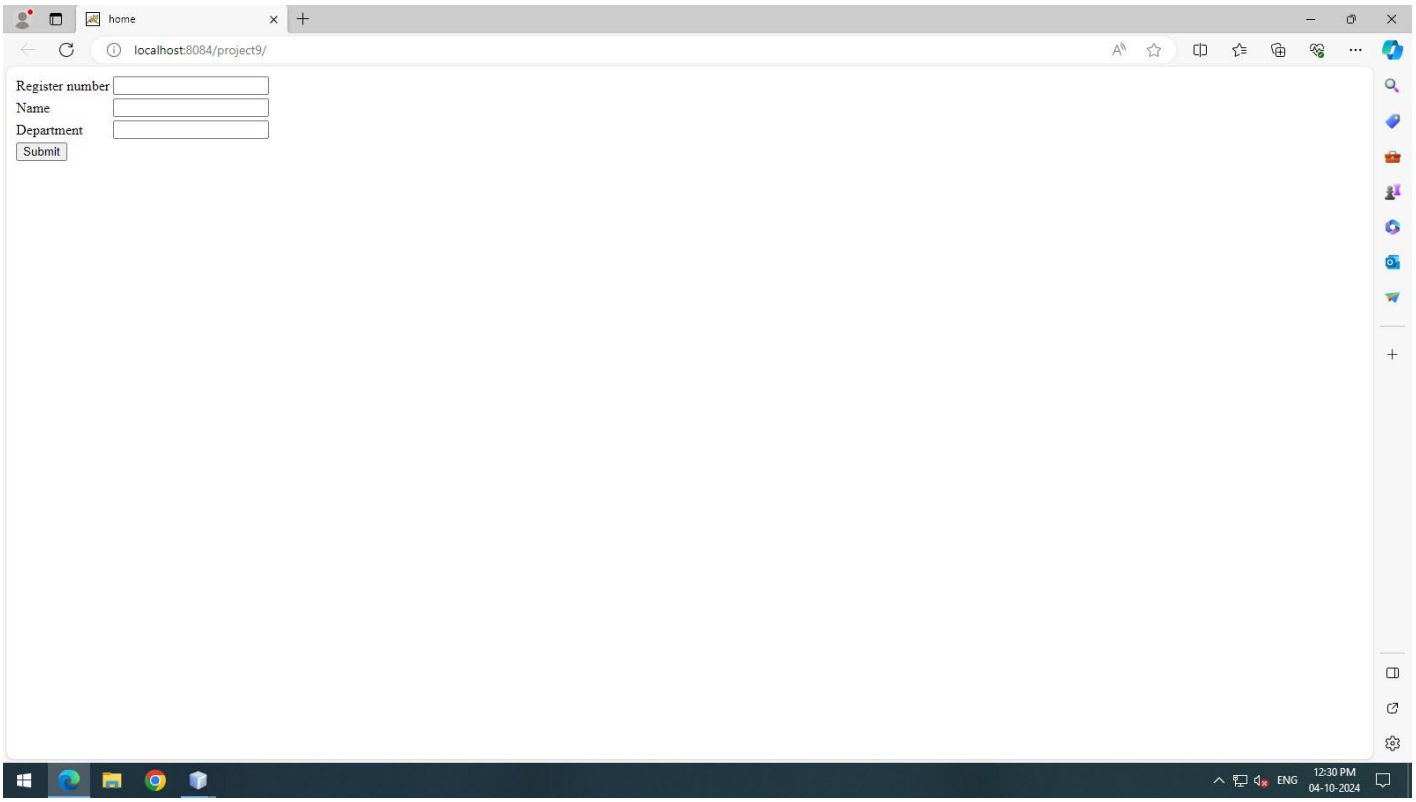
- **MySQL**: jdbc:mysql://hostname:port/dbname
- **PostgreSQL**: jdbc:postgresql://hostname:port/dbname
- **Oracle**: jdbc:oracle:thin:@hostname:port:dbname.

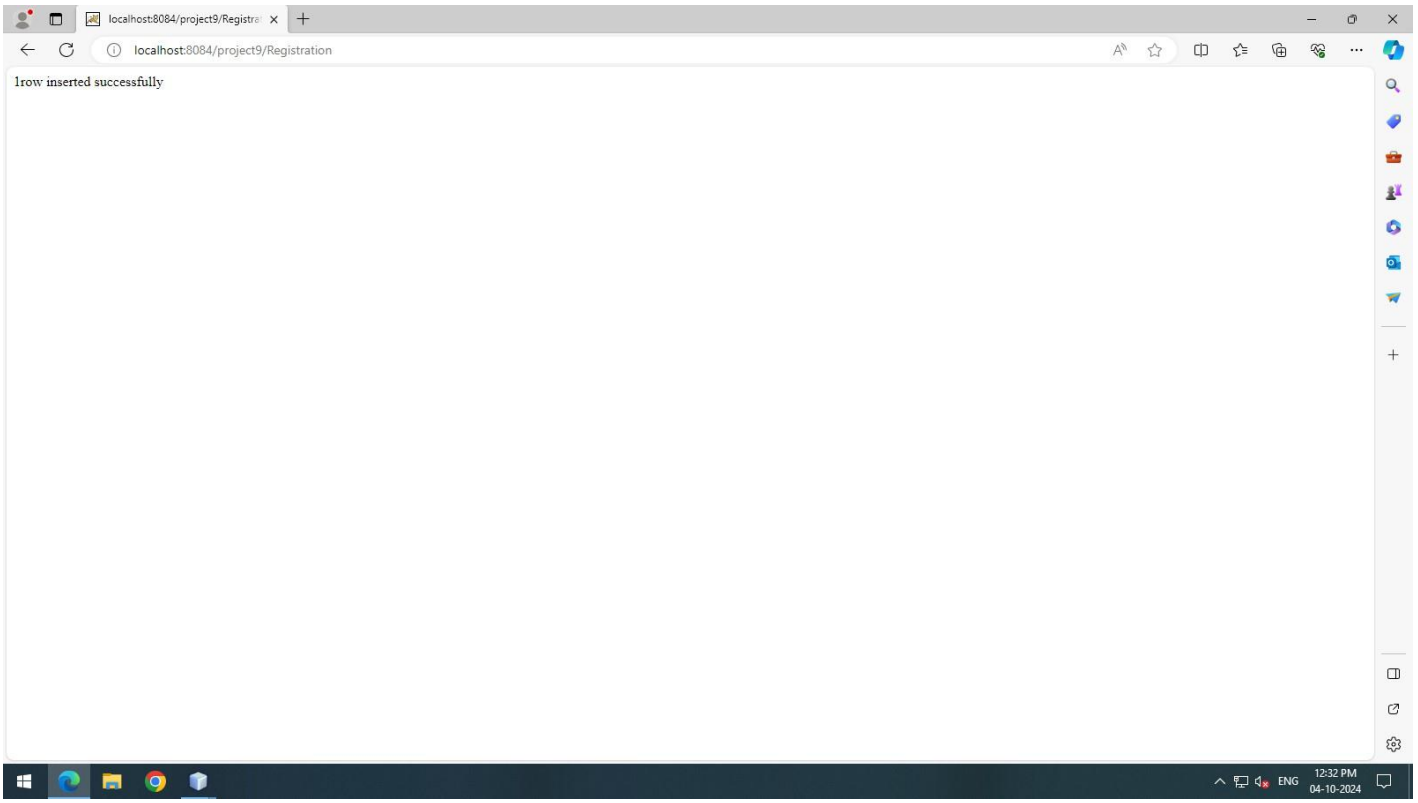
PROGRAM:

REGISTRATION.JAVA

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.io.*;
public class Registration extends HttpServlet {
    public void doPost(HttpServletRequest request,HttpServletResponse response)throws IOException
    {
        response.setContentType("text/html");
        PrintWriter pw=response.getWriter();
        String regno=request.getParameter("rn");
        String name=request.getParameter("nn");
        String dept=request.getParameter("dept");
        try
        {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            Connection
con=DriverManager.getConnection("jdbc:derby://localhost:1527/table","ide","UER2fH5R");
            PreparedStatement ps=con.prepareStatement("insert into detail values(?,?,?)");
            ps.setString(1, regno);
            ps.setString(2,name);
            ps.setString(3,dept);
            int i=ps.executeUpdate();
            pw.println(i+"row inserted successfully");
            con.close();
        }
        catch(Exception e)
        {
            pw.println(e);

        }
    }
}
```

OUTPUT:


home
localhost:8084/project9/
Register number
Name
Department
Submit
ENG
12:30 PM
04-10-2024


localhost:8084/project9/Registra
localhost:8084/project9/Registration
1row inserted successfully
ENG
12:32 PM
04-10-2024

SELECT * FROM IDE.DETAIL ...

Max. rows: 100 Fetched Rows: 3 Matching Rows:

#	REGNO	NAME	DEPT
1	1	Nivetha	MCA
2	2	Sandhiya	MCA
3	3	Susmitha	MCA

2:1 INS

RESULT:

Thus the program executed successfully.

Ex.No: 10	Implementation of Java Beans
DATE:	

AIM:

To Implementation of Java Beans.

ALGORITHM:

Step 1 : Key Characteristics of a JavaBean:

- Public no-argument constructor.
- Private fields (properties).
- Public getter and setter methods.
- Serializable (optional but recommended).

Step 2 : Create a Class with private properties (fields). Provide a No-Argument Constructor**.**

Step 3 : Define the Class**:** Create a class that will act as the JavaBean.

Step 4 : Declare Properties (Fields)**:** Declare the private instance variables that will store the bean's state.

Step 5 : Generate Getters and Setters**:** For each property, define getter and setter methods following the getProperty() and setProperty() naming convention.

Provide a No-Argument Constructor**:** Define a public constructor with no parameters.

Step 6 : Make the Bean Serializable: Implement the java.io.Serializable interface so the bean can be persisted or transferred between systems (optional).

PROGRAM:

Main.java

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

        Person person = new Person();

        person.setName("Siasc");
        person.setAge(27);

        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());

        System.out.println(person);
    }
}
```

Person.java

```
import java.io.Serializable;

public class Person implements Serializable {
    private String name;
    private int age;

    public Person() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}
```

OUTPUT:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.3448]
(c) Microsoft Corporation. All rights reserved.

C:\Users\STUDENT LOGIN>d:

D:\>cd manoj

D:\manoj>javac Person.java Main.java

D:\manoj>java Main
Name: SIASC
Age: 27
Person{name='SIASC', age=27}

D:\manoj>
```

RESULT:

Thus program as executed successfully.