**Write a Java Program to implement multithreading (three threads using single run method).**

class MultiThreadDemo implements Runnable {

private String threadName;

MultiThreadDemo(String name) {

threadName = name;

}

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(threadName + " - Count: " + i);

try {

// Sleep for a while to simulate some work and allow context switching

Thread.sleep(500);

} catch (InterruptedException e) {

System.out.println(threadName + " interrupted.");

}

}

System.out.println(threadName + " exiting.");

}

public static void main(String[] args) {

Thread thread1 = new Thread(new MultiThreadDemo("Thread 1"));

Thread thread2 = new Thread(new MultiThreadDemo("Thread 2"));

Thread thread3 = new Thread(new MultiThreadDemo("Thread 3"));

thread1.start();

thread2.start();

thread3.start();

}}

**Write a Java Program to implement the calculator.**

import java.util.Scanner;

public class Calculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter first number:");

double num1 = scanner.nextDouble();

System.out.println("Enter second number:");

double num2 = scanner.nextDouble();

System.out.println("Enter an operator (+, -, \*, /):");

char operator = scanner.next().charAt(0);

double result;

switch (operator) {

case '+':

result = num1 + num2;

System.out.println("The result is: " + result);

break;

case '-':

result = num1 - num2;

System.out.println("The result is: " + result);

break;

case '\*':

result = num1 \* num2;

System.out.println("The result is: " + result);

break;

case '/':

if (num2 != 0) {

result = num1 / num2;

System.out.println("The result is: " + result);

} else {

System.out.println("Error! Division by zero.");

}

break;

default:

System.out.println("Invalid operator.");

break;

}

scanner.close();

}

}

**Write a Java Program to implement the URL.**

import java.net.MalformedURLException;

import java.net.URL;

public class URLExample {

public static void main(String[] args) {

try {

// Example URL

URL url = new URL("https://www.example.com:8080/path/to/resource?query=example#reference");

// Display the components of the URL

System.out.println("URL: " + url.toString());

System.out.println("Protocol: " + url.getProtocol());

System.out.println("Host: " + url.getHost());

System.out.println("Port: " + (url.getPort() != -1 ? url.getPort() : "default port"));

System.out.println("Path: " + url.getPath());

System.out.println("Query: " + url.getQuery());

System.out.println("Reference: " + url.getRef());

} catch (MalformedURLException e) {

System.out.println("The URL is malformed: " + e.getMessage());

}

}

}

Write a Java Program to implement the InetAddress.

import java.net.InetAddress;

import java.net.UnknownHostException;

public class InetAddressExample {

public static void main(String[] args) {

try {

// Get the local host address

InetAddress localHost = InetAddress.getLocalHost();

System.out.println("Local Host:");

System.out.println("Hostname: " + localHost.getHostName());

System.out.println("IP Address: " + localHost.getHostAddress());

System.out.println();

// Get the IP address from a hostname

String hostname = "www.example.com";

InetAddress hostAddress = InetAddress.getByName(hostname);

System.out.println("Host Information for: " + hostname);

System.out.println("Hostname: " + hostAddress.getHostName());

System.out.println("IP Address: " + hostAddress.getHostAddress());

System.out.println();

// Get the hostname from an IP address

String ipAddress = "93.184.216.34"; // IP address for example.com

InetAddress addressByIP = InetAddress.getByName(ipAddress);

System.out.println("Host Information for IP: " + ipAddress);

System.out.println("Hostname: " + addressByIP.getHostName());

System.out.println("IP Address: " + addressByIP.getHostAddress());

} catch (UnknownHostException e) {

System.out.println("Unknown host: " + e.getMessage());

}

}

}

**Write a Java Program for Sending E-mail in Java.**

import java.util.Properties;

import javax.mail.\*;

import javax.mail.internet.\*;

public class EmailSender {

public static void main(String[] args) {

// Recipient's email ID needs to be mentioned.

String to = "recipient@example.com";

// Sender's email ID needs to be mentioned

String from = "sender@example.com";

final String username = "your-email@example.com"; // change accordingly

final String password = "your-email-password"; // change accordingly

// Assuming you are sending email through relay.jangosmtp.net

String host = "smtp.example.com"; // for example, smtp.gmail.com for Gmail

// Get system properties

Properties properties = new Properties();

properties.put("mail.smtp.auth", "true");

properties.put("mail.smtp.starttls.enable", "true");

properties.put("mail.smtp.host", host);

properties.put("mail.smtp.port", "587"); // or 465 for SSL

// Get the Session object.

Session session = Session.getInstance(properties,

new javax.mail.Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication(username, password);

}

});

try {

// Create a default MimeMessage object.

Message message = new MimeMessage(session);

// Set From: header field of the header.

message.setFrom(new InternetAddress(from));

// Set To: header field of the header.

message.setRecipients(Message.RecipientType.TO,

InternetAddress.parse(to));

// Set Subject: header field

message.setSubject("This is the Subject Line!");

// Now set the actual message

message.setText("This is the actual message");

// Send message

Transport.send(message);

System.out.println("Sent message successfully....");

} catch (MessagingException e) {

throw new RuntimeException(e);

}

}

}

**Write a Java Program to implement Single Client-Server Communication**

Server:

import java.io.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(1234)) {

System.out.println("Server is listening on port 1234");

try (Socket socket = serverSocket.accept()) {

System.out.println("New client connected");

InputStream input = socket.getInputStream();

BufferedReader reader = new BufferedReader(new InputStreamReader(input));

OutputStream output = socket.getOutputStream();

PrintWriter writer = new PrintWriter(output, true);

String text;

do {

text = reader.readLine();

System.out.println("Received: " + text);

writer.println("Server: " + text);

} while (!text.equals("bye"));

System.out.println("Client disconnected");

}

} catch (IOException ex) {

System.out.println("Server exception: " + ex.getMessage());

ex.printStackTrace();

}

}

}

Client:

import java.io.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) {

String hostname = "localhost";

int port = 1234;

try (Socket socket = new Socket(hostname, port)) {

OutputStream output = socket.getOutputStream();

PrintWriter writer = new PrintWriter(output, true);

InputStream input = socket.getInputStream();

BufferedReader reader = new BufferedReader(new InputStreamReader(input));

BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));

String text;

do {

System.out.print("Enter message: ");

text = consoleReader.readLine();

writer.println(text);

String response = reader.readLine();

System.out.println(response);

} while (!text.equals("bye"));

} catch (UnknownHostException ex) {

System.out.println("Server not found: " + ex.getMessage());

} catch (IOException ex) {

System.out.println("I/O error: " + ex.getMessage());

}

}

}

Running the Programs:

Start the Server:

Run the Server program first. The server will start and listen for incoming connections.

Start the Client:

Run the Client program. The client will connect to the server, send messages, and print responses.

Server Output:

Server is listening on port 1234

New client connected

Received: Hello Server

Received: How are you?

Received: bye

Client disconnected

Client Output:

Enter message: Hello Server

Server: Hello Server

Enter message: How are you?

Server: How are you?

Enter message: bye

Server: bye

**Write a Java Program to implement the Login\_Id Form using JDBC**

To implement a login form using JDBC (Java Database Connectivity), you need to follow these steps:

Create a Database and a Table: For this example, let's use a MySQL database with a table named users.

Connect to the Database using JDBC: Establish a connection to the database.

Create a Login Form: Use Swing for the GUI to create the login form.

Validate User Credentials: Check the entered username and password against the database records.

Prerequisites:

MySQL database setup:

Database name: testdb

Table name: users

Columns: username (VARCHAR), password (VARCHAR)

sql

Copy code

CREATE DATABASE testdb;

USE testdb;

CREATE TABLE users (

username VARCHAR(50) NOT NULL,

password VARCHAR(50) NOT NULL

);

INSERT INTO users (username, password) VALUES ('testuser', 'testpass');

JDBC Driver for MySQL: Ensure you have the MySQL JDBC driver (mysql-connector-java.jar) in your project's classpath.

Java Code:

1. Database Connection Utility

java

Copy code

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

public static Connection getConnection() {

Connection connection = null;

try {

Class.forName("com.mysql.cj.jdbc.Driver");

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/testdb", "root", "password");

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

}

return connection;

}

}

2. Login Form with JDBC

java

Copy code

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class LoginForm extends JFrame {

private JTextField usernameField;

private JPasswordField passwordField;

private JButton loginButton;

private JPanel loginPanel;

public LoginForm() {

setTitle("Login Form");

setContentPane(loginPanel);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 200);

setLocationRelativeTo(null);

loginButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String username = usernameField.getText();

String password = new String(passwordField.getPassword());

if (authenticate(username, password)) {

JOptionPane.showMessageDialog(LoginForm.this, "Login successful!");

} else {

JOptionPane.showMessageDialog(LoginForm.this, "Invalid username or password.", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

}

private boolean authenticate(String username, String password) {

boolean isValid = false;

Connection connection = DatabaseConnection.getConnection();

if (connection != null) {

String sql = "SELECT \* FROM users WHERE username = ? AND password = ?";

try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

preparedStatement.setString(1, username);

preparedStatement.setString(2, password);

try (ResultSet resultSet = preparedStatement.executeQuery()) {

if (resultSet.next()) {

isValid = true;

}

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

return isValid;

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

new LoginForm().setVisible(true);

}

});

}

}

Explanation:

DatabaseConnection Class:

Provides a method to establish a connection to the MySQL database.

Replace "root" and "password" with your MySQL username and password.

LoginForm Class:

Creates a simple Swing-based login form with username and password fields.

On clicking the login button, it calls the authenticate method to validate the credentials.

The authenticate method connects to the database, checks the username and password, and returns true if the credentials are valid, otherwise false.

Main Method:

Launches the login form.

Running the Program:

Ensure your MySQL server is running and the testdb database and users table are set up with the appropriate data.

Ensure the MySQL JDBC driver (mysql-connector-java.jar) is in your project's classpath.

Compile and run the LoginForm class. The login form will appear.

Enter the username and password to test the login functionality.

**JTree**nstrate

//

import javax.swing.\*;

import javax.swing.tree.DefaultMutableTreeNode;

public class DynamicTreeExample {

public static void main(String[] args)

{

// Creating the frame

JFrame frame = new JFrame(

"GeeksforGeeks - Java JTree Example");

// Creating the root node

DefaultMutableTreeNode root

= new DefaultMutableTreeNode("Root");

// Creating child nodes

DefaultMutableTreeNode parent1

= new DefaultMutableTreeNode("Parent 1");

DefaultMutableTreeNode child1\_1

= new DefaultMutableTreeNode("Child 1.1");

DefaultMutableTreeNode child1\_2

= new DefaultMutableTreeNode("Child 1.2");

// Adding child nodes to the parent1

parent1.add(child1\_1);

parent1.add(child1\_2);

// Creating another set of child nodes

DefaultMutableTreeNode parent2

= new DefaultMutableTreeNode("Parent 2");

DefaultMutableTreeNode child2\_1

= new DefaultMutableTreeNode("Child 2.1");

DefaultMutableTreeNode child2\_2

= new DefaultMutableTreeNode("Child 2.2");

// Adding child nodes to the parent2

parent2.add(child2\_1);

parent2.add(child2\_2);

// Adding parent nodes to the root

root.add(parent1);

root.add(parent2);

// Creating the JTree

JTree tree = new JTree(root);

// Adding the JTree to the frame within a scroll

// pane

frame.add(new JScrollPane(tree));

// Setting frame properties

frame.setSize(400, 400);

frame.setDefaultCloseOperation(

JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

}

**JTable**

// Packages to import

import javax.swing.JFrame;

import javax.swing.JScrollPane;

import javax.swing.JTable;

public class JTableExamples {

// frame

JFrame f;

// Table

JTable j;

// Constructor

JTableExamples()

{

// Frame initialization

f = new JFrame();

// Frame Title

f.setTitle("JTable Example");

// Data to be displayed in the JTable

String[][] data = {

{ "Kundan Kumar Jha", "4031", "CSE" },

{ "Anand Jha", "6014", "IT" }

};

// Column Names

String[] columnNames = { "Name", "Roll Number", "Department" };

// Initializing the JTable

j = new JTable(data, columnNames);

j.setBounds(30, 40, 200, 300);

// adding it to JScrollPane

JScrollPane sp = new JScrollPane(j);

f.add(sp);

// Frame Size

f.setSize(500, 200);

// Frame Visible = true

f.setVisible(true);

}

// Driver method

public static void main(String[] args)

{

new JTableExamples();

}

}

**Write a Java Program to implement the SQL commands using JDBC.**

iimport java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class MySQLJDBCExample {

// MySQL Database credentials

static final String JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver";

static final String DB\_URL = "jdbc:mysql://localhost:3306/your\_database\_name"; // Replace 'your\_database\_name'

// MySQL credentials

static final String USER = "your\_username"; // Replace 'your\_username'

static final String PASS = "your\_password"; // Replace 'your\_password'

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try {

// Register MySQL JDBC Driver

Class.forName(JDBC\_DRIVER);

// Open a connection

System.out.println("Connecting to database...");

conn = DriverManager.getConnection(DB\_URL, USER, PASS);

// Create a statement object to send SQL commands

stmt = conn.createStatement();

// Create a table

String createTableSQL = "CREATE TABLE Employees "

+ "(id INTEGER not NULL, "

+ " name VARCHAR(255), "

+ " age INTEGER, "

+ " PRIMARY KEY ( id ))";

stmt.executeUpdate(createTableSQL);

System.out.println("Table created successfully...");

// Insert data into table

String insertSQL = "INSERT INTO Employees (id, name, age) VALUES (1, 'John Doe', 30)";

stmt.executeUpdate(insertSQL);

insertSQL = "INSERT INTO Employees (id, name, age) VALUES (2, 'Jane Smith', 25)";

stmt.executeUpdate(insertSQL);

System.out.println("Records inserted successfully...");

// Select and display data from the table

String selectSQL = "SELECT id, name, age FROM Employees";

ResultSet rs = stmt.executeQuery(selectSQL);

System.out.println("Data from Employees table:");

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

int age = rs.getInt("age");

// Display the retrieved data

System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age);

}

// Update data in the table

String updateSQL = "UPDATE Employees SET age = 35 WHERE id = 1";

stmt.executeUpdate(updateSQL);

System.out.println("Record updated successfully...");

// Delete data from the table

String deleteSQL = "DELETE FROM Employees WHERE id = 2";

stmt.executeUpdate(deleteSQL);

System.out.println("Record deleted successfully...");

// Clean-up environment

rs.close();

stmt.close();

conn.close();

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (stmt != null) stmt.close();

if (conn != null) conn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

**Write a Java Program to create the table using JDBC**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

public class CreateTableExample {

// MySQL database credentials

static final String JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver";

static final String DB\_URL = "jdbc:mysql://localhost:3306/your\_database\_name"; // Replace with your database name

static final String USER = "your\_username"; // Replace with your MySQL username

static final String PASS = "your\_password"; // Replace with your MySQL password

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try {

// Step 1: Register JDBC driver

Class.forName(JDBC\_DRIVER);

// Step 2: Open a connection

System.out.println("Connecting to database...");

conn = DriverManager.getConnection(DB\_URL, USER, PASS);

// Step 3: Execute a query to create the table

System.out.println("Creating table in the database...");

stmt = conn.createStatement();

String sql = "CREATE TABLE Employees " +

"(id INT NOT NULL, " +

" name VARCHAR(255), " +

" age INT, " +

" PRIMARY KEY ( id ))";

stmt.executeUpdate(sql);

System.out.println("Table 'Employees' created successfully...");

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

// Step 4: Clean-up environment

if (stmt != null) stmt.close();

if (conn != null) conn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

**Write a Java Program to implement Remote Method Invocation.**

**Steps to implement RMI:**

1. Create a remote interface that defines the methods that can be called remotely.
2. Implement the remote interface on the server side.
3. Create a client that will invoke the remote methods.
4. Set up the RMI registry to register the server.

**Step-by-Step RMI**

1. **Create the Remote Interface**: The interface should extend java.rmi.Remote, and each method should throw java.rmi.RemoteException.

import java.rmi.Remote;

import java.rmi.RemoteException;

// Remote interface

public interface Hello extends Remote {

String sayHello() throws RemoteException;

}

**2. Implement the Remote Interface (Server Implementation)**: The server class implements the remote interface and extends UnicastRemoteObject.

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

// Remote object implementation class

public class HelloImpl extends UnicastRemoteObject implements Hello {

// Constructor that throws RemoteException

public HelloImpl() throws RemoteException {

super();

}

// Implementation of the remote method

@Override

public String sayHello() throws RemoteException {

return "Hello, RMI World!";

}

}

3. **Create the Server Program**: The server program registers the remote object in the RMI registry.

import java.rmi.Naming;

import java.rmi.registry.LocateRegistry;

public class RMIServer {

public static void main(String[] args) {

try {

// Create and export a remote object

HelloImpl obj = new HelloImpl();

// Start the RMI registry on port 1099

LocateRegistry.createRegistry(1099);

// Bind the remote object in the registry with a name "Hello"

Naming.rebind("rmi://localhost:1099/Hello", obj);

System.out.println("RMI Server is ready...");

} catch (Exception e) {

e.printStackTrace();

}

}

}

4. **Create the Client Program**: The client looks up the remote object and invokes the remote method.

import java.rmi.Naming;

public class RMIClient {

public static void main(String[] args) {

try {

// Lookup the remote object in the RMI registry

Hello obj = (Hello) Naming.lookup("rmi://localhost:1099/Hello");

// Call the remote method and print the result

String message = obj.sayHello();

System.out.println("Message from server: " + message);

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Steps to Run the RMI Program:**

1. **Compile all the Java classes**:

javac Hello.java HelloImpl.java RMIServer.java RMIClient.java

2. **Start the RMI registry**: Open a terminal and run the following command to start the RMI registry.

rmiregistry

3. **Run the server**: In a new terminal, run the server program:

java RMIServer

4. **Run the client**: In another terminal, run the client program:

java RMIClient

**Output:**

When the client program is run, it will print:

Message from server: Hello, RMI World!