

## Practical No: 1

**AIM:** Create a Chat application using TCP/P and UDP Protocol

### Description/Theory:

#### TCP Chat Server:

- **ServerSocket:** This class implements a socket that listens for incoming connections from clients.
- **accept()**
- **Socket:** Represents the endpoint of a two-way communication link between two programs running on the network.
- **getInputStream():** Returns an InputStream for reading data from this socket.
- **getOutputStream():** Returns an OutputStream for writing data to this socket.
- **BufferedReader:** Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.
- **PrintWriter:** Prints formatted representations of objects to a text-output stream.
- **IOException:** This is the general class of exceptions produced by failed or interrupted I/O operations.

#### UDP Chat Server:

- **DatagramSocket:** This class represents a socket for sending and receiving datagram packets.
- **receive():** This method waits to receive a datagram packet from this socket.
- **send():** This method sends a datagram packet from this socket.
- **DatagramPacket:** Represents a datagram packet containing the data that is to be sent or received.

#### UDP Chat Client:

- **DatagramSocket:** Same as in the server code.
- **InetAddress:** This class represents an Internet Protocol (IP) address.
- **getByName():** This method returns an InetAddress object given the raw IP address.
- **DatagramPacket:** Same as in the server code.
- **IOException:** Same as in the TCP server code.

### Program:

#### TCP/P

##### Server.java

```
import java.net.*;
import java.io.*;
class Server{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(8000);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
```

```
str2=br.readLine();  
dout.writeUTF(str2);  
dout.flush();  
}  
din.close();  
s.close();  
ss.close();  
}}
```

### **Client.java**

```
import java.net.*;  
  
import java.io.*;  
  
class Client{  
  
public static void main(String args[])throws Exception{  
  
Socket s=new Socket("localhost",8000);  
  
DataInputStream din=new DataInputStream(s.getInputStream());  
DataOutputStream dout=new  
DataOutputStream(s.getOutputStream());  
  
BufferedReader br=new BufferedReader(new  
InputStreamReader(System.in));  
  
String str="",str2="";  
  
while(!str.equals("stop")){  
  
str=br.readLine();  
  
dout.writeUTF(str);  
  
dout.flush();  
  
str2=din.readUTF();  
  
System.out.println("Server says: "+str2);  
  
}  
  
dout.close();  
  
s.close();  
  
}}
```

### **UDP/P**

**UDPChatClient.java**

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

public class UDPChatClient {

    public static void main(String[] args) {

        try {

            DatagramSocket clientSocket = new
DatagramSocket();

            InetAddress IPAddress =
InetAddress.getByName("localhost");

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

            byte[] sendData;

            byte[] receiveData = new byte[1024];

            while (true) {

                String messageToSend =
consoleInput.readLine();

                sendData = messageToSend.getBytes();

                DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length, IPAddress, 12345);

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);

                clientSocket.receive(receivePacket);

                String receivedMessage = new
String(receivePacket.getData(), 0, receivePacket.getLength());

                System.out.println("Server: " +
receivedMessage);

            }

        } catch (IOException e) {
```

```
        e.printStackTrace();
    }
}
}
```

### **UDPChatServer.java**

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

public class UDPChatServer {
    public static void main(String[] args) {
        try {
            DatagramSocket serverSocket = new
DatagramSocket(12345);

            System.out.println("UDP Server running...");

            byte[] receiveData = new byte[1024];

            byte[] sendData = new byte[1024];

            DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);

            while (true) {
                serverSocket.receive(receivePacket);

                String receivedMessage = new
String(receivePacket.getData(), 0, receivePacket.getLength());

                System.out.println("Client: " +
receivedMessage);

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

                String messageToSend =
consoleInput.readLine();

                sendData = messageToSend.getBytes();
```

```
        DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length,
receivePacket.getAddress(), receivePacket.getPort());

        serverSocket.send(sendPacket);

    }

    } catch (IOException e) {

        e.printStackTrace();

    }

}

}
```

### Output: With TCP/P

```
G:\College material\Nupur_Aj\Prac1>javac Client.java

G:\College material\Nupur_Aj\Prac1>java Client
Hello From Nupur Patel
|
```

```
G:\College material\Nupur_Aj\Prac1>javac Server.java

G:\College material\Nupur_Aj\Prac1>java Server
client says: Hello From Nupur Patel
|
```

### With UDP/P

```
G:\College material\Nupur_Aj\Prac1>javac UDPChatServer.java

G:\College material\Nupur_Aj\Prac1>java UDPChatServer
UDP Server running...
Client: Hello From Nupur Patel
BYE
|
```

```
G:\College material\Nupur_Aj\Prac1>javac UDPChatClient.java

G:\College material\Nupur_Aj\Prac1>java UDPChatClient
Hello From Nupur Patel
Server: BYE
```

## Practical No: 2

**AIM:** Write a client server program using UDP where client sends a string expression and server evaluate that expression and send the result. For an example: Client: 10 + 20 <Enter>

Server: Sending Result... Client: Result=30

### Description/Theory:

UDP Server:

- DatagramSocket: Represents a socket for sending and receiving datagram packets.
- receive(): This method waits to receive a datagram packet from this socket.
- send(): This method sends a datagram packet from this socket.
- DatagramPacket: Represents a datagram packet containing the data that is to be sent or received.
- IOException: This is the general class of exceptions produced by failed or interrupted I/O operations.

UDP Client:

- DatagramSocket: Same as in the server code.
- InetAddress: This class represents an Internet Protocol (IP) address.
- getByName(): This method returns an InetAddress object given the raw IP address.
- DatagramPacket: Same as in the server code.
- IOException: Same as in the server code.

### Program:

**UDPServer.java**

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

public class UDPServer {
    public static void main(String[] args) {
        try {
            DatagramSocket serverSocket = new
DatagramSocket(12345);
            System.out.println("UDP Server running...");
            byte[] receiveData = new byte[1024];
            byte[] sendData = new byte[1024];
            while (true) {
                DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
                serverSocket.receive(receivePacket);
                String expression = new
String(receivePacket.getData(), 0, receivePacket.getLength());
                // Evaluate the expression
                int result = evaluateExpression(expression);
                // Prepare the result to send back to the
client

                String resultString = "Result=" + result;
                sendData = resultString.getBytes();

                // Send the result back to the client
```

```

        DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length,
receivePacket.getAddress(), receivePacket.getPort());
        serverSocket.send(sendPacket);
    }
} catch (IOException e) {
    e.printStackTrace(); }
}

private static int evaluateExpression(String expression) {
    // Split the expression into operands and operator
    String[] parts = expression.split("\\s+");
    int operand1 = Integer.parseInt(parts[0]);
    int operand2 = Integer.parseInt(parts[2]);
    String operator = parts[1];
    // Evaluate the expression based on the operator
    switch (operator) {
        case "+":
            return operand1 + operand2;
        case "-":
            return operand1 - operand2;
        case "*":
            return operand1 * operand2;
        case "/":
            if (operand2 != 0) {
                return operand1 / operand2;
            } else {
                throw new ArithmeticException("Division by
zero");
            }
        default:
            throw new IllegalArgumentException("Invalid
operator");
    } }
}

```

#### **UDPCClient.java**

```

import java.io.*;

import java.net.*;

public class UDPCClient {

    public static void main(String[] args) {

        try {

            DatagramSocket clientSocket = new
DatagramSocket();

            InetAddress serverIPAddress =
InetAddress.getByName("localhost");

```

```
        BufferedReader consoleInput = new
BufferedReader(new InputStreamReader(System.in));

        byte[] sendData;

        byte[] receiveData = new byte[1024];

        System.out.print("Enter expression (e.g., 10 +
20): ");

        String expression = consoleInput.readLine();

        sendData = expression.getBytes();

        // Send the expression to the server

        DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length, serverIPAddress,
12345);

        clientSocket.send(sendPacket);

        // Receive the result from the server

        DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);

        clientSocket.receive(receivePacket);

        String result = new
String(receivePacket.getData(), 0, receivePacket.getLength());

        System.out.println("Server: " + result);

        clientSocket.close();

    } catch (IOException e) {

        e.printStackTrace();

    } } }
```



## Output:

```
G:\College material\Nupur_Aj\Prac2>Java UdpClient
Enter a mathematical expression : |
```

```
G:\College material\Nupur_Aj\Prac2>Java UdpClient
Enter a mathematical expression : 10+20
Sent expression: 10+20
Received from server: Result=30
```

```
Sent result: 30
|
```

## Practical No: 3

**AIM:** Write a TCP Client-Server program to get the Date & Time details from Server on the Client request.

### Description/Theory:

Server.java:

- **ServerSocket:** This class implements a socket that listens for incoming connections from clients.
- **accept():** This method waits for a client to connect to the server and returns a Socket object representing the client-server connection.
- **Socket:** Represents the endpoint of a two-way communication link between two programs running on the network.
- **OutputStream:** This is an abstract class representing an output stream of bytes.
- **PrintWriter:** Prints formatted representations of objects to a text-output stream.
- **Date:** Represents a specific instant in time, with millisecond precision.

Client.java:

- **Socket:** Same as in the server code.
- **InputStream:** This is an abstract class representing an input stream of bytes.
- **BufferedReader:** Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

### Program:

#### Server.java

```
import java.io.*;

import java.net.*;

import java.util.Date;

public class Server {

    public static void main(String[] args) {

        try {

            ServerSocket serverSocket = new
ServerSocket(9999);

            while (true) {

                Socket clientSocket =serverSocket.accept();

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

                OutputStream outputStream =
clientSocket.getOutputStream();

                PrintWriter out = new
PrintWriter(outputStream, true);

                Date currentDate = new Date();
```

```
        out.println("Date & Time: " + currentDate);  
  
        clientSocket.close();  
  
        System.out.println("Client disconnected.");  
  
    }  
  
    } catch (IOException e) {  
  
        e.printStackTrace();  
  
    }    }}
```

#### **Client.java**

```
import java.io.*;  
  
import java.net.*;  
  
public class Client {  
  
    public static void main(String[] args) {  
  
        try {  
  
            Socket socket = new Socket("localhost", 9999);  
  
            InputStream inputStream = socket.getInputStream();  
  
            BufferedReader in = new BufferedReader(new  
InputStreamReader(inputStream));  
  
            String dateTime = in.readLine();  
  
            System.out.println("Server response: " +  
dateTime);  
  
            socket.close();  
  
        } catch (IOException e) {  
  
            e.printStackTrace();    }    }}
```

**Output:**

```
G:\College material\Nupur_Aj\Prac3>javac Server.java  
G:\College material\Nupur_Aj\Prac3>java Server
```

```
G:\College material\Nupur_Aj\Prac3>javac Client.java  
G:\College material\Nupur_Aj\Prac3>java Client  
Date:Fri Apr 25 20:04:42 IST 2025
```

## Practical No: 4

**AIM:** Create a multi-threaded server to which more than one client can connect and communicate with Server for sending the string and server returns the reverse of string to each of client.

### Description/Theory:

#### Server

- **ServerSocket:** This class implements a socket that listens for incoming connections from clients.
- **accept():** This method waits for a client to connect to the server and returns a **Socket** object representing the client-server connection.
- **Socket:** Represents the endpoint of a two-way communication link between two programs running on the network.
- **Thread:** Represents a thread of execution.
- **BufferedReader:** Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.
- **PrintWriter:** Prints formatted representations of objects to a text-output stream.
- **IOException:** This is the general class of exceptions produced by failed or interrupted I/O operations.

#### ClientHandler (Inner Class)

- **Thread:** This class represents a thread of execution.
- **start():** This method starts the execution of the thread by invoking its **run()** method.
- **Socket:** Same as in the server code.
- **BufferedReader:** Same as in the server code.
- **PrintWriter:** Same as in the server code.
- **IOException:** Same as in the server code.

### Program:

#### **MultiThreadedServer.java**

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

public class MultiThreadedServer {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(9999)) {
            System.out.println("Server running...");
            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " +
clientSocket);

                // Start a new thread to handle the client
                ClientHandler clientHandler = new
ClientHandler(clientSocket);
                clientHandler.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```
static class ClientHandler extends Thread {
    private final Socket clientSocket;

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

    public void run() {
        try (
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new
PrintWriter(clientSocket.getOutputStream(), true);
        ) {
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                // Reverse the input string
                String reversedString = new
StringBuilder(inputLine).reverse().toString();
                // Send the reversed string back to the client
                out.println(reversedString);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                clientSocket.close();
                System.out.println("Client disconnected: " +
clientSocket);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
Client1.java import java.io.*;
import java.net.*;
```

```
public class Client1 {
    public static void main(String[] args) {
        try (
            // Use try-with-resources to manage resources
            automatically
            Socket socket = new Socket("localhost", 9999);
            BufferedReader consoleInput = new BufferedReader(new
InputStreamReader(System.in));
            PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()))
```



```
) {  
    System.out.println("Connected to server.");  
    String userInput;  
  
    // Continuously read user input from the console  
    while ((userInput = consoleInput.readLine()) != null) {  
        // Send user input to the server  
        out.println(userInput);  
  
        // Receive the reversed string from the server  
        String reversedString = in.readLine();  
  
        // Print the server's response  
        System.out.println("Server response: " +  
reversedString); // Ensure this line ends with a semicolon  
    }  
    System.out.println("Disconnected from server.");  
} catch (IOException e) {  
    e.printStackTrace();  
}  
}
```

#### **Client2.java**

```
import java.io.*;  
import java.net.*;  
  
public class Client2 {  
    public static void main(String[] args) {  
        try {  
            // Establishing a connection to the server  
            Socket socket = new Socket("localhost", 9999);  
            // BufferedReader for user input from the console  
            BufferedReader consoleInput = new BufferedReader(new  
InputStreamReader(System.in));  
            // PrintWriter to send data to the server  
            PrintWriter out = new  
PrintWriter(socket.getOutputStream(), true);  
            // BufferedReader to read data from the server  
            BufferedReader in = new BufferedReader(new  
InputStreamReader(socket.getInputStream()))  
        } {  
            System.out.println("Connected to server.");  
            String userInput;  
  
            // Continuously read user input from the console  
            while ((userInput = consoleInput.readLine()) != null) {  
                // Send user input to the server  
                out.println(userInput);  
  
                // Receive the reversed string from the server  
                String reversedString = in.readLine();  
                System.out.println("Server response: " +  
reversedString);  
            }  
        }  
    }  
}
```

```

        System.out.println("Disconnected from server.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

## Output:

```

G:\College material\Nupur_Aj\Prac4>javac MultiThreadedServer.java

G:\College material\Nupur_Aj\Prac4>java MultiThreadedServer
Server running...
Client connected: Socket[addr=/127.0.0.1,port=63176,localport=9999]
Client connected: Socket[addr=/127.0.0.1,port=63181,localport=9999]
|

```

```

G:\College material\Nupur_Aj\Prac4>javac Client1.java

G:\College material\Nupur_Aj\Prac4>java Client1
Connected to server.
Nupur
Server response: rupuN
|

```

```

G:\College material\Nupur_Aj\Prac4>javac Client2.java

G:\College material\Nupur_Aj\Prac4>java Client2
Connected to server.
Patel
Server response: letaP
|

```



## Practical No: 5

**AIM:** Create a File Server Which transfers a file to Client.

### Description/Theory:

#### Server

- **ServerSocket:** This class implements a socket that listens for incoming connections from clients.
- **accept():** This method waits for a client to connect to the server and returns a Socket object representing the client-server connection.
- **Socket:** Represents the endpoint of a two-way communication link between two programs running on the network.
- **FileInputStream:** A FileInputStream obtains input bytes from a file in a file system.
- **OutputStream:** This is an abstract class representing an output stream of bytes.
- **BufferedOutputStream:** Wraps an existing OutputStream and provides additional functionality.
- **File:** Represents a file or directory pathname.
- **IOException:** This is the general class of exceptions produced by failed or interrupted I/O operations.

#### Client

- **Socket:** Same as in the server code.
- **InputStream:** This is an abstract class representing an input stream of bytes.
- **FileOutputStream:** A FileOutputStream writes data to a file in the file system.
- **IOException:** This is the general class of exceptions produced by failed or interrupted I/O operations.

### Program:

#### Server.java

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

public class Server {
    public static void main(String[] args) {
        final int PORT = 12345;
        ServerSocket serverSocket = null;
        Socket clientSocket = null;
        FileInputStream fileInputStream = null;
        BufferedOutputStream bufferedOutputStream = null;

        try {
            serverSocket = new ServerSocket(PORT);
            System.out.println("Server started. Waiting for
client...");

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

            // Open input stream to read the file
            File file = new File("G:\\College
material\\Nupur_Aj\\Prac5\\Nupur.txt"); // Replace with your file
```



path

```
        fileInputStream = new FileInputStream(file);

        // Open output streams to send the file
        OutputStream outputStream =
clientSocket.getOutputStream();
        bufferedOutputStream = new
BufferedOutputStream(outputStream);

        // Send file to client
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = fileInputStream.read(buffer)) != -1)
{
            bufferedOutputStream.write(buffer, 0, bytesRead);
        }

        bufferedOutputStream.flush();
        System.out.println("File sent successfully.");
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (fileInputStream != null)
fileInputStream.close();
            if (bufferedOutputStream != null)
bufferedOutputStream.close();
            if (clientSocket != null) clientSocket.close();
            if (serverSocket != null) serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

### **Client.java**

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

public class Client {
    public static void main(String[] args) {
        final String SERVER_IP = "127.0.0.1"; // Server IP address
        final int SERVER_PORT = 12345; // Server port number

        Socket socket = null;
        InputStream inputStream = null;
        FileOutputStream fileOutputStream = null;

        try {
            // Connect to server
            socket = new Socket(SERVER_IP, SERVER_PORT);
            System.out.println("Connected to server.");
        }
```



```
// Open input stream to receive file
InputStream = socket.getInputStream();

// Open output stream to write file
FileOutputStream = new
FileOutputStream("received_file.txt"); // Replace with your desired
file name

// Receive file from server
byte[] buffer = new byte[1024];
int bytesRead;
while ((bytesRead = inputStream.read(buffer)) != -1) {
    fileOutputStream.write(buffer, 0, bytesRead);
}

System.out.println("File received successfully.");
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        if (fileOutputStream != null)
fileOutputStream.close();
        if (inputStream != null) inputStream.close();
        if (socket != null) socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

### Output:

```
G:\College material\Nupur_Aj\Prac5>javac Client.java
```

```
G:\College material\Nupur_Aj\Prac5>java Client
```

```
Connected to server.
```

```
File received successfully.
```

```
G:\College material\Nupur_Aj\Prac5>javac Server.java
```

```
G:\College material\Nupur_Aj\Prac5>java Server
```

```
Server started. Waiting for client...
```

```
Client connected: Socket[addr=/127.0.0.1,port=63305,localport=12345]
```

```
File sent successfully.
```

## Practical No: 6

**AIM:** Create a new database named "StudentDB" and also create a new table "Student" under that database. Once database has been created then perform following database operation

**1.** Connect **2.** Create Database **3.** Create Table **4.** Insert Records into respective table **5.** Update records of particular table of database **6.** Delete Records from table. **7.** Delete table and also database. using Statement and ResultSet interface.

### Description:

Class Name

java.sql.Connection

Description: Represents a connection to a database. It can be used to create Statement objects.

### Interface Name

- java.sql.Statement

Description: Used to execute SQL queries and can return ResultSet objects.

- java.sql.ResultSet

Description: Represents the result set of a database query. It provides methods for traversing and manipulating the data returned by a SELECT SQL statement.

### Methods Used

- createStatement():
- close():
- executeQuery(String sql)
- executeUpdate(String sql)
- getString(int columnIndex)
- getInt(int columnIndex) or getInt(String columnLabel)
- updateString(int columnIndex, String x) or updateString(String columnLabel, String x)
- deleteRow()

### Program:

```
import java.sql.*;

public class StudentDBOperations
{
    private static final String JDBC_URL =
"jdbc:mysql://localhost:3306/";

    private static final String DB_NAME = "StudentDB";

    private static final String USER = "root";

    private static final String PASSWORD = "root";

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



```
try
{
    Connection connection =
DriverManager.getConnection(JDBC_URL, USER, PASSWORD);

    //create db

    createDatabase(connection, DB_NAME);

    //use db

    connection.setCatalog(DB_NAME);

    //create table

    createTable(connection);

    //insert records

    insertRecords(connection);

    // //update records

    // updateRecords(connection);

    // //delete records from table

    // deleteRecords(connection);

    // //delete table

    // dropTable(connection);

    // //delete db

    // dropDatabase(connection, DB_NAME);

    //close conn

    connection.close();

} catch (SQLException e) {

    e.printStackTrace();

}

}
```

```
private static void createDatabase(Connection connection,
String dbName) throws SQLException {
```

```
        try (Statement statement =
connection.createStatement()) {

            statement.executeUpdate("CREATE DATABASE " +
dbName);

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

        }

    }

    private static void createTable(Connection connection)
throws SQLException {

        try (Statement statement =
connection.createStatement()) {

            statement.executeUpdate("CREATE TABLE Student (id
INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100), age INT)");

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

        }

    }

    private static void insertRecords(Connection connection)
throws SQLException {

        try (Statement statement =
connection.createStatement()) {

            statement.executeUpdate("INSERT INTO Student
(name, age) VALUES ('Kareena', 21)");

            statement.executeUpdate("INSERT INTO Student
(name, age) VALUES ('xyz', 20)");

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

        }

    }

    // private static void updateRecords(Connection
connection) throws SQLException {
```



```
//      try (Statement statement =
connection.createStatement()) {

    //          statement.executeUpdate("UPDATE Student SET age
= 21 WHERE name = 'Alice'");

    //          System.out.println("Records updated
successfully");

    //      }

    // }

// private static void deleteRecords(Connection
connection) throws SQLException {

    //      try (Statement statement =
connection.createStatement()) {

    //          statement.executeUpdate("DELETE FROM Student
WHERE name = 'Bob'");

    //          System.out.println("Records deleted
successfully");

    //      }

    // }

// private static void dropTable(Connection connection)
throws SQLException {

    //      try (Statement statement =
connection.createStatement()) {

    //          statement.executeUpdate("DROP TABLE Student");

    //          System.out.println("Table dropped
successfully");

    //      }

    // }

// private static void dropDatabase(Connection connection,
String dbName) throws SQLException {

    //      try (Statement statement =
connection.createStatement()) {
```

```
// statement.executeUpdate("DROP DATABASE " +
dbName);

// System.out.println("Database dropped
successfully");

// }

// }

}
```

## Output:

```
C:\Users\anshu\Desktop\AJT\ANSHUMI\java-pracs\prac6>java -cp mysql-connector-j-8.3.0.jar;. StudentDBOperations.java
Database created successfully
Table created successfully
Records inserted successfully
Records updated successfully
```

Result Grid			
Filter Rows:			
	id	name	age
•	NULL	NULL	NULL

```
1 • show databases;
2 • use studentdb;
3 • select * from Student;
```

Database	
▶	information_schema
	mysql
	performance_schema
	sys
	try6

Result Grid			
Filter Rows:			
	id	name	age
▶	1	ANSHUMI	22
	2	SHAH	51
•	NULL	NULL	NULL

Error Code: 1146. Table 'studentdb.student' doesn't exist



## Practical No: 7

**AIM:** Create a new database named "EmployeeDB" and also create new table "Employee" under that database. Once database has been created then perform following database operation

1. Connect 2. Create Database 3. Create Table 4. Insert Records into respective table 5. Update records of particular table of database 6. Delete Records from table. 7. Delete table and also database using PreparedStatement and ResultSet interface.

### Description/Theory/:

#### Class Name

- java.sql.Connection

Description: Represents a connection to a database. It can be used to create PreparedStatement objects.

- Interface Name: java.sql.PreparedStatement

Description: An interface used to execute parameterized SQL queries. It extends the Statement interface.

- Interface Name: java.sql.ResultSet

Description: Represents the result set of a database query. It provides methods for traversing and manipulating the data returned by a SELECT SQL statement.

#### Methods Used

- prepareStatement(String sql)
- executeQuery()
- executeUpdate()
- setXXX(int parameterIndex, XXX value)
- getString(int columnIndex) or getString(String columnLabel)
- getInt(int columnIndex) or getInt(String columnLabel)
- updateString(int columnIndex, String x) or updateString(String columnLabel)
- deleteRow()
- 

#### Program:

```
import java.sql.*;
public class EmployeeDBOperations {
    private static final String DB_URL =
"jdbc:mysql://localhost:3306/";
    private static final String DB_NAME = "EmployeeDB";
    private static final String USER = "root";
    private static final String PASSWORD = "root";
    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
            //connect to db
            conn = DriverManager.getConnection(DB_URL, USER,
PASSWORD);

            // //create db
```

```

        // String createDatabaseQuery = "CREATE DATABASE
IF NOT EXISTS " + DB_NAME;
        // pstmt =
conn.prepareStatement(createDatabaseQuery);
        // pstmt.executeUpdate();
        // //select db
        conn.setCatalog(DB_NAME);
        // //create table
        // String createTableQuery = "CREATE TABLE IF NOT
EXISTS Employee (id INT AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(100), age INT)";
        // pstmt =
conn.prepareStatement(createTableQuery);
        // pstmt.executeUpdate();
        // //insert
        // String insertQuery = "INSERT INTO Employee
(name, age) VALUES (?, ?)";
        // pstmt = conn.prepareStatement(insertQuery);
        // pstmt.setString(1, "karina");
        // pstmt.setInt(2, 21);
        // pstmt.executeUpdate();
        // pstmt.setString(1, "varun");
        // pstmt.setInt(2, 20);
        // pstmt.executeUpdate();
        //update
String updateQuery = "UPDATE Employee SET age = ? WHERE name =
?";

        pstmt = conn.prepareStatement(updateQuery);
        pstmt.setInt(1, 21);
        pstmt.setString(2, "varun");
        pstmt.executeUpdate();
        //delete rows
String deleteQuery = "DELETE FROM Employee WHERE
name = ?";
        pstmt = conn.prepareStatement(deleteQuery);
        pstmt.setString(1, "varun");
        pstmt.executeUpdate();
        // delete table and db
String dropTableQuery = "DROP TABLE IF EXISTS
Employee";
        pstmt = conn.prepareStatement(dropTableQuery);
        pstmt.executeUpdate();
        String dropDatabaseQuery = "DROP DATABASE IF
EXISTS " + DB_NAME;
        pstmt = conn.prepareStatement(dropDatabaseQuery);
        pstmt.executeUpdate();
        System.out.println("Database operations executed
successfully.");

    } catch (SQLException e) {
        e.printStackTrace();
    }

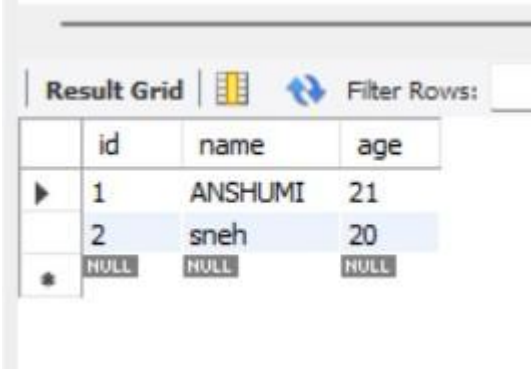
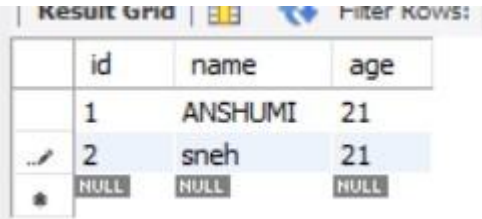


```



```
    } finally {  
        try {  
            if (rs != null) rs.close();  
            if (pstmt != null) pstmt.close();  
            if (conn != null) conn.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}  
}
```

## Output:

```
C:\Users\anshu\Desktop\AJT\ANSHUMI\java-pracs\prac7>java -cp mysql-connector-j-8.3.0.jar;. EmployeeDBOperations.java  
Database operations executed successfully.
```

<pre>show databases; use employeeedb; select * from employee;</pre> 	<pre>select * from employee;</pre> 
 <p>DELETE RECORDS</p>	 <p>DELETE DATABASE</p>
<pre>Error Code: 1146. Table 'employeeedb.employee' doesn't exist</pre> <p>DELETE TABLE</p>	

## Practical No: 8

**AIM:** Create stored Procedure and call that using CallableStatement Interface.

**Description:**

In Java, JDBC (Java Database Connectivity) API provides a way to interact with relational databases. Stored procedures are precompiled SQL statements stored in the database server and executed by invoking a simple name from an application. They enhance performance, maintainability, and security by centralizing SQL code within the database server.

To create a stored procedure in a database, you typically use SQL commands specific to the database management system (e.g., MySQL, Oracle, SQL Server). Here, we'll demonstrate creating and calling a stored procedure in MySQL.

- The CallableStatement interface is used to call stored procedures.
- We establish a connection to the MySQL database using `DriverManager.getConnection()`.
- We prepare a CallableStatement by passing the SQL query with placeholders for parameters ({call getEmployee(?, ?)}).
- Input parameters are set using `setXXX()` methods (e.g., `setInt()`).
- Output parameters are registered using `registerOutParameter()`.
- The stored procedure is executed using `execute()`.
- Output parameters are retrieved using appropriate `getXXX()` methods.
- Finally, resources (statements and connections) are closed to release them.

**Program:**

**Stored Procedure -**

```
DELIMITER //
CREATE PROCEDURE getEmployee(IN employeeId INT, OUT
employeeName VARCHAR(255))
BEGIN
    SELECT name INTO employeeName FROM employees WHERE id =
employeeId;
END //
DELIMITER ;
```

**Main.java -**

```
import java.sql.*;
public class Main1 {
    static final String JDBC_DRIVER =
"com.mysql.cj.jdbc.Driver";
    static final String DB_URL =
"jdbc:mysql://127.0.0.1:3306/prac8";
    static final String USER = "root";
    static final String PASS = "";
    public static void main(String[] args) {
        Connection conn = null;
        CallableStatement cstmt = null;
        try {
```

```
// Register JDBC driver
Class.forName(JDBC_DRIVER);
// Open a connection
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL, USER,
PASS);

// Prepare the callable statement
String sql = "{CALL GetEmployeeCount(?)}";
cstmt = conn.prepareCall(sql);
// Register OUT parameter
cstmt.registerOutParameter(1, Types.INTEGER);
// Execute the stored procedure
cstmt.execute();
// Retrieve the result
int totalEmployees = cstmt.getInt(1);
System.out.println("Total number of employees: " +
totalEmployees);
} catch (SQLException se) {
    se.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (cstmt != null) cstmt.close();
        if (conn != null) conn.close();
    } catch (SQLException se) {
        se.printStackTrace();
    }
}
}
}
```

## Output:



id	name	department
NULL	NULL	NULL

## Practical No: 9

**AIM:** Write a Program to make ResultSet scrollable and updatable and perform insertion and deletion using ResultSet only.

### Description/Theory:

#### Classes

- **ResultSetDemo:** This is the main class that contains the main method. It handles the database connection, statement execution, and manipulation of the ResultSet.

#### Interfaces

- **Connection:** Represents a connection to a database. Used to establish a connection to the MySQL database.
- **Statement:** Represents a static SQL statement that is executed by the database. Used to create a statement object for executing SQL queries.
- **ResultSet:** Represents the result set of a database query. Used to retrieve and manipulate the data returned by the SQL query.

### Program:

```
import java.sql.*;

public class ResultSet1 {

    static final String JDBC_DRIVER =
"com.mysql.cj.jdbc.Driver";

    static final String DB_URL =
"jdbc:mysql://localhost:3306/employee";

    static final String USER = "root";

    static final String PASS = "root";

    public static void main(String[] args) {

        Connection conn = null;

        Statement stmt = null;

        ResultSet rs = null;

        try {

            // Register JDBC driver

            Class.forName (JDBC_DRIVER);

            // Open a connection

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

```
conn = DriverManager.getConnection(DB_URL, USER,  
PASS);
```

```
    // Create a scrollable and updatable statement  
stmt=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
ResultSet.CONCUR_UPDATABLE);
```

```
    // Execute a query
```

```
String sql = "SELECT * FROM employees";
```

```
rs = stmt.executeQuery(sql);
```

```
    // Move to the insert row
```

```
rs.moveToInsertRow();
```

```
    // Insert new record
```

```
    rs.updateString("emp_id", "4");
```

```
rs.updateString("emp_name", "abc");
```

```
    // Update more columns as needed
```

```
System.out.println("added");
```

```
    // Insert the new row
```

```
rs.insertRow();
```

```
    // Move to the last row to perform deletion
```

```
rs.last();
```

```
    // Delete the last row
```

```
rs.deleteRow();
```

```
    // Print the updated result set
```

```
rs.beforeFirst();
```

```
// Move cursor to before the first row
```

```
while (rs.next()) {
```

```
    // Print row data
```

```
}
```



```
    } catch (SQLException se) {  
        se.printStackTrace();  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            if (rs != null) rs.close();  
            if (stmt != null) stmt.close();  
            if (conn != null) conn.close();  
        } catch (SQLException se) {  
            se.printStackTrace();  
        }  
    }  
}  
}
```

### Output:

	emp_id	emp_name
▶	1	ANSHUMI
	2	xyz
	3	kg
•	NULL	NULL

	2	xyz
	3	kg
•	NULL	NULL

	emp_id	emp_name
▶	1	ANSHUMI
	2	xyz
	3	kg
	4	abc
•	NULL	NULL



**2. Write a program to fetch different metadata like TABLE CATALOG, TABLE SCHEMA, TABLE TYPE, Column Count, Column Label from a database.**

### **Description/Theory:**

#### **Interfaces**

- **Connection:** Represents a connection to a database.
- **ResultSet:** Represents the result set of a database query.
- **DatabaseMetaData:** Provides methods to retrieve metadata information about the database.

#### **Methods**

- **Class.forName(String className):** Used to dynamically load the JDBC driver class.
- **DriverManager.getConnection(String url, String user, String password):** Establishes a connection to the database specified by the URL, username, and password.
- **Connection.getMetaData():** Retrieves a DatabaseMetaData object that contains metadata about the database.
- **DatabaseMetaData.getDatabaseProductName():** Returns the name of the database product.
- **DatabaseMetaData.getUserName():** Returns the username used to establish the connection.
- **DatabaseMetaData.getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types):** Retrieves metadata about tables in the database.
- **DatabaseMetaData.getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern):** Retrieves metadata about columns in a table.
- **ResultSet.next():** Moves the cursor to the next row in the ResultSet.
- **ResultSet.getString(String columnLabel):** Retrieves the value of the specified column as a String.

### **Program:**

```
import java.sql.*;

public class MetadataDemo {

    static final String JDBC_DRIVER =
"com.mysql.cj.jdbc.Driver";

    static final String DB_URL =
"jdbc:mysql://localhost:3306/employeedb";

    static final String USER = "root";

    static final String PASS = "root";

    public static void main(String[] args) {

        Connection conn = null;

        ResultSet rs = null;

        try {
```



```
// Register JDBC driver

Class.forName(JDBC_DRIVER);

// Open a connection

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

conn = DriverManager.getConnection(DB_URL, USER,
PASS);

// Get DatabaseMetaData object

DatabaseMetaData metaData = conn.getMetaData();

// Fetch and print metadata

System.out.println("TABLE CATALOG: " +
metaData.getDatabaseProductName());

System.out.println("TABLE SCHEMA: " +
metaData.getUserName());

// Fetch metadata for tables

ResultSet tables = metaData.getTables(null, null,
"employeeedb", null);

if (tables.next()) {

    System.out.println("TABLE TYPE: " +
tables.getString("TABLE_TYPE"));

} else {

    System.out.println("Table not found.");

}

// Fetch column metadata

ResultSet columns = metaData.getColumns(null,
null, "employeeedb", null);

int columnCount = 0;

while (columns.next()) {

    columnCount++;

    System.out.println("Column Label " +
columnCount + ": " + columns.getString("COLUMN_NAME"));
```



```
    }

    System.out.println("Column Count: " +
columnCount);

    } catch (SQLException se) {

        se.printStackTrace();

    } catch (Exception e) {

        e.printStackTrace();

    } finally {

        try {

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

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

        } catch (SQLException se) {

            se.printStackTrace();

        }

    }

}

}
```

### Output:

```
C:\Users\anshu\Desktop\AJT\ANSHUMI\java-pracs\Prac9>java -cp mysql-connector-j-8.3.0.jar;. MetadataDemo.java
Connecting to database...
TABLE CATALOG: MySQL
TABLE SCHEMA: root@localhost
Table not found.
Column Count: 0
```

## Practical No: 10

**AIM:** Create Servlet file and study web descriptor file.

### Description:

#### Methods Used:

- `init(ServletConfig config)`: Initializes the servlet with the given configuration.
- `service(ServletRequest req, ServletResponse res)`: Handles client requests and generates responses.
- `destroy()`: Cleans up resources when the servlet is being unloaded.
- `getServletConfig()`: Returns the servlet's configuration.
- `getServletInfo()`: Returns information about the servlet.

#### Elements Used:

- `<web-app>`: Root element of the descriptor.
- `<servlet>`: Defines a servlet by specifying its name and class.
- `<servlet-name>`: Unique name for the servlet.
- `<servlet-class>`: Fully qualified name of the servlet class.
- `<servlet-mapping>`: Maps a servlet to a URL pattern.
- `<url-pattern>`: URL pattern used to access the servlet.

### Program:

TestThis.class file

```
import jakarta.servlet.*;
import java.io.*;

public class TestThis implements Servlet
{
    ServletConfig conf = null;

    public void init(ServletConfig config)
    {
        this.conf = config;
    }

    public void service(ServletRequest req,
        ServletResponse
        res)throws IOException,ServletException
    {

        res.setContentType("text/html;charset=UTF-8");
        PrintWriter pw = res.getWriter();
        pw.println("<html><body>");
        pw.println("<h1>This is Servlet Page</h1>");
        pw.println("</body></html>");
        pw.close();
    }
}
```

```
public void destroy()
{
    System.out.println("Bye");
}
public ServletConfig getServletConfig()
{
    return conf;
}
public String getServletInfo()
{
    return "Byeeee";
}
}
```

### **web.xml**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
<servlet>
<servlet-name>VisitSer</servlet-name>
<servlet-class>TestThis</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>VisitSer</servlet-name>
<url-pattern>/doVisit</url-pattern>
</servlet-mapping>
</web-app>
```

### **index.html**

```
<html>
<body>
<a href = "doVisit">Visit This</a>
</body>
</html>
```

### **Output:**



## Practical No: 11

**AIM:** Write servlet which displayed following information of client.

- I. Client Browser
- II. Client IP address
- III. Client Port No
- IV. Server Port No
- V. Local Port No
- VI. Method used by client for form submission
- VII. Query String name and values

### Description/Theory/:

Java Servlet API Classes and Interfaces:

- **HttpServlet:** Extends the **GenericServlet** class and provides methods to handle HTTP specific requests.
- **HttpServletRequest:** Represents an HTTP request and provides methods to access request information.
- **HttpServletResponse:** Represents an HTTP response and provides methods to set response content and headers.

Java I/O Classes:

- **PrintWriter:** Writes formatted data to an underlying output stream (in this case, the response output stream).
- **Java Annotation:**
- **@WebServlet:** Annotation used to declare a servlet in a Java class without the need for a web.xml deployment descriptor. In this code, the **@WebServlet** annotation is not used, and instead, the servlet is configured in the web.xml file.

Other Methods:

- **request.getHeader("User-Agent"):** Retrieves the User-Agent header from the HTTP request, which typically contains information about the client's browser.
- **request.getRemoteAddr():** Retrieves the IP address of the client making the request.
- **request.getRemotePort(), request.getServerPort(), request.getLocalPort():** Retrieve the port numbers of the client, server, and local machine, respectively.
- **request.getMethod():** Retrieves the HTTP method used by the client for the request (e.g., GET, POST).

- `request.getQueryString()`: Retrieves the query string portion of the URL from the HTTP request

**Program:**

ClientInfoServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
public class ClientInfoServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Client Information:</h2>");
        // Client Browser
        String userAgent = request.getHeader("User-Agent");
        out.println("<strong>Client Browser:</strong> " +
            userAgent + "<br>");
        // Client IP address
        String clientIP = request.getRemoteAddr();
        out.println("<strong>Client IP Address:</strong> "
            +
            clientIP + "<br>");
        // Client Port No
        int clientPort = request.getRemotePort();
        out.println("<strong>Client Port Number:</strong> "
            +
            clientPort + "<br>");
        // Server Port No
        int serverPort = request.getServerPort();
        out.println("<strong>Server Port Number:</strong> "
            +
            serverPort + "<br>");
        // Local Port No
        int localPort = request.getLocalPort();
```

```

    out.println("<strong>Local Port Number:</strong> "
+
localPort + "<br>");
    // Method used by client for form submission
    String method = request.getMethod();
    out.println("<strong>Method Used for Form
Submission:</strong> " + method + "<br>");
    // Query String name and values
    String queryString = request.getQueryString();
    if (queryString != null) {
        out.println("<strong>Query String:</strong> " +
queryString + "<br>");
    }

    out.println("</body></html>");
    out.close();
}
}

```

#### **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/java
ee

http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
version="4.0">

    <servlet>
        <servlet-name>ClientInfoServlet</servlet-name>
        <servlet-class>ClientInfoServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ClientInfoServlet</servlet-name>
        <url-pattern>/clientInfo</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

```



```
</web-app>
index.html
<!DOCTYPE html>
<html>
<head>
  <title>Client Information</title>
</head>
<body>
  <h2>Client Information</h2>
  <form action="clientInfo" method="GET">
    <input type="submit" value="Get Client Info">
  </form>
</body>
</html>
```

### Output :



## Client Information

Get Client Info



### Client Information:

Client Browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36  
Client IP Address: 0:0:0:0:0:0:1  
Client Port Number: 53617  
Server Port Number: 8080  
Local Port Number: 8080  
Method Used for Form Submission: GET  
Query String:

## Practical No: 12

**AIM:** Design a form to input details of an employee and submit the data to a servlet. Write code for servlet that will save the entered details as a new record in database table Employee with fields (EmpId, EName, Email, Age).

### Description:

Java Servlet API Classes and Interfaces:

- **HttpServlet:** Extends the GenericServlet class and provides methods to handle HTTP- specific requests.
- **HttpServletRequest:** Represents an HTTP request and provides methods to access request information.
- **HttpServletResponse:** Represents an HTTP response and provides methods to set response content and headers.

Java I/O Classes:

- **PrintWriter:** Writes formatted data to an underlying output stream (in this case, the response output stream).
- **Java JDBC Classes:**
- **Connection:** Represents a connection to a database.
- **DriverManager:** Manages JDBC drivers and establishes database connections.
- **PreparedStatement:** Represents a precompiled SQL statement and allows parameterized queries.

### Program:

#### EmployeeServlet.java

```
import java.io.IOException; import java.io.PrintWriter;
import java.sql.Connection; import java.sql.DriverManager;
import java.sql.PreparedStatement; import
jakarta.servlet.ServletException; import
jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest; import
jakarta.servlet.http.HttpServletResponse;
public class EmployeeServlet extends HttpServlet { private
static final String DB_URL =
"jdbc:mysql://localhost:3306/";
private static final String DB_NAME = "EmployeeDB";
private static final String USER = "root";
private static final String PASSWORD = "root"; protected
void doPost(HttpServletRequest request,
HttpServletResponse response)
```

```
throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();

// Retrieve form data
String empId = request.getParameter("empId"); String
empName = request.getParameter("empName"); String email =
request.getParameter("email"); int age =
Integer.parseInt(request.getParameter("age"));
// Database connection and insert query
try {
Class.forName("com.mysql.cj.jdbc.Driver"); Connection con
=
DriverManager.getConnection(DB_URL, USER, PASSWORD);
PreparedStatement pst =
con.prepareStatement("INSERT INTO Employee (EmpId, EName,
Email, Age) VALUES (?, ?, ?, ?)");
pst.setString(1, empId); pst.setString(2, empName);
pst.setString(3, email); pst.setInt(4, age);
int rowsAffected = pst.executeUpdate();
if (rowsAffected > 0) { out.println("<h3>Employee details
saved
successfully!</h3>");
} else {
out.println("<h3>Error saving employee details.</h3>");
}
con.close();
} catch (Exception e) {
out.println("<h3>Error: " + e.getMessage() +
"</h3>");
}
out.close();
}
}
```

#### **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_4_0.xsd"
version="4.0">
<servlet>
<servlet-name>EmployeeServlet</servlet-name>
<servlet-class>EmployeeServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
<servlet-name>EmployeeServlet</servlet-name>
<url-pattern>/EmployeeServlet</url-pattern>
</servlet-mapping>
</web-app>
index.html
<!DOCTYPE html>
<html>
<head>
<title>Employee Details Form</title>
</head>
<body>
<h2>Enter Employee Details:</h2>
<form action="EmployeeServlet" method="POST">
<label for="empId">Employee ID:</label>
<input type="text" id="empId" name="empId"
required><br><br>
<label for="empName">Employee Name:</label>
<input type="text" id="empName" name="empName"
required><br><br>
<label for="email">Email:</label>
<input type="email" id="email" name="email"
required><br><br>
<label for="age">Age:</label>
<input type="number" id="age" name="age" required><br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

## Output:

**Employee added successfully!**

### Employee Details:

Employee ID: 23781

Employee Name: Java

Email: j@gmail.com

Age: 21

### Login

EmployeeID:	<input type="text" value="23781"/>
Employee Name:	<input type="text" value="Java"/>
Email:	<input type="text" value="j@gmail.com"/>
Age:	<input type="text" value="21"/>
<input type="submit" value="Submit"/>	

## Practical No: 13

**AIM:** Create login form and perform state management using Cookies, HttpSession and URL Rewriting

**Description:**

**Methods:**

- `doPost(HttpServletRequest request, HttpServletResponse response)`: This method is overridden from the `HttpServlet` class. It handles HTTP POST requests in the `LoginServlet`.
- `doGet(HttpServletRequest request, HttpServletResponse response)`: This method is overridden from the `HttpServlet` class. It handles HTTP GET requests in the `LogoutServlet`.
- `response.setContentType(String type)`: This method sets the content type of the response.
- `response.getWriter()`: This method returns a `PrintWriter` object that can send character text to the client.
- `request.getParameter(String name)`: This method retrieves the value of the specified request parameter.
- `request.getSession(boolean create)`: This method retrieves the current session associated with the request, or creates a new session if one does not exist.
- `session.setAttribute(String name, Object value)`: This method binds an object to a given attribute name in the session.
- `Cookie(String name, String value)`: This is the constructor for creating a new cookie with the specified name and value.
- `cookie.setMaxAge(int expiry)`: This method sets the maximum age of the cookie in seconds.
- `response.addCookie(Cookie cookie)`: This method adds the specified cookie to the response.

**Interfaces:**

- `HttpServlet`: This is an abstract class that provides methods for handling HTTP requests. Both `LoginServlet` and `LogoutServlet` extend this class.

- **HttpServletRequest:** This is an interface representing an HTTP request. It provides methods for accessing request information, such as parameters and session data.
- **HttpServletResponse:** This is an interface representing an HTTP response. It provides methods for setting response content and headers.
- **HttpSession:** This is an interface representing a user session. It provides methods for managing session attributes and lifecycle.
- **Cookie:** This is a class representing an HTTP cookie. It implements the Serializable interface.

## **Program:**

### **LoginServlet.java**

```
import java.io.*;
import
jakarta.servlet.*;
import
jakarta.servlet.http
.*;

public class LoginServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {

        response.setContentType("text/html
        "); PrintWriter out =
        response.getWriter();
        String username =
        request.getParameter("username"); String
        password =
        request.getParameter("password");
        // For simplicity, let's assume username
        and password are "admin"
        if
        (username.equals("admin") &&
        password.equals("admin")) {
            HttpSession session =
            request.getSession();
            session.setAttribute("username",
            username);
```

```
// Create a Cookie for username

Cookie usernameCookie = new Cookie("username",
username);

usernameCookie.setMaxAge(30 * 60); // Cookie will
last 30 minutes

response.addCookie(usernameCookie);

"!</h2>");
out.println("<html><body>"); out.println("<h2>Welcome, " +
username +
out.println("<a href='LogoutServlet'>Logout</a>");
out.println("</body></html>");
} else {
    out.println("<html><body>");
    out.println("<h2>Invalid username or
password.

Please try again.</h2>");

    out.println("<a
href='login.html'>Back to Login</a>");

    out.println("</body></html>");

}

}

}
```

### **LogoutServlet.java**

```
import java.io.*;
import jakarta.servlet.*; import jakarta.servlet.http.*;
public class LogoutServlet extends HttpServlet { private static
final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html
"); PrintWriter out =
response.getWriter();
```



```
HttpSession session =
request.getSession(false); if
(session != null) {

    session.invalidate(); // Invalidate the session

}

// Delete the username cookie

Cookie[] cookies =
request.getCookies(); if
(cookies != null) {
    for (Cookie cookie : cookies) {

        if (cookie.getName().equals("username"))
            { cookie.setMaxAge(0); // Setting
              maxAge to

                    }

            }

    }

}

response.addCookie(cookie);

    out.println("<html><body>");
    out.println("<h2>You have been
logged out
successfully.</h2>");
    out.println("<a href='login.html'>Back to
Login</a>"); out.println("</body></html>");

}

}
```

#### **index.html**

```
<!DOCTYPE html>
<html>
<head>
<title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <form action="LoginServlet" method="post">
```





```
Username: <input type="text"
name="username"><br> Password: <input
type="password" name="password"><br>
<input type="submit" value="Login">
</form>
</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\_4\_0.xsd"

    version="4.0">
    <!-- Define the servlet for LoginServlet -->

    <servlet>

        <servlet-name>LoginServlet</servlet-name>

        <servlet-class>LoginServlet</servlet-class>

    </servlet>

    <!-- Map the servlet to a URL pattern -->

    <servlet-mapping>

        <servlet-name>LoginServlet</servlet-name>

        <url-pattern>/LoginServlet</url-pattern>

    </servlet-mapping>
    <!-- Define the servlet for LogoutServlet -->

    <servlet>

        <servlet-name>LogoutServlet</servlet-name>

        <servlet-class>LogoutServlet</servlet-class>

    </servlet>
```



```
<!-- Map the servlet to a URL pattern -->

<servlet-mapping>

    <servlet-name>LogoutServlet</servlet-name>

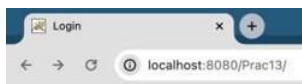
    <url-pattern>/LogoutServlet</url-pattern>

</servlet-mapping>

<!-- Session Configuration -->

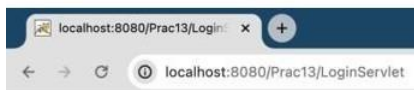
<session-config>
    <session-timeout>30</session-timeout>
<!-- Session timeout in minutes -->
</session-config>
</web-app>
```

## Output:



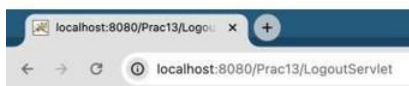
### Login

Username:   
Password:



**Welcome, admin!**

[Logout](#)



**You have been logged out successfully.**

[Back to Login](#)

## Practical No: 14

**AIM:** Implement Authentication filter using filter API

### Description:

### Methods:

- `init(FilterConfig fConfig)`: This method is from the Filter interface and is called by the servlet container to indicate to a filter that it is being placed into service.
- `doFilter(ServletRequest request, ServletResponse response, FilterChain chain)`: This method is from the Filter interface and is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
- `context.log(String message)`: This method is used to log messages from within the filter.
- `getRequestURI()`: This method retrieves the request URI from the `HttpServletRequest` object.
- `getSession(boolean create)`: This method retrieves the current session associated with the request, or creates a new session if one does not exist.
- `session.getAttribute(String name)`: This method retrieves the value of the named attribute as an Object from the session.
- `sendRedirect(String location)`: This method sends a temporary redirect response to the client using the specified redirect location URL.
- `destroy()`: This method is from the Filter interface and is called by the container when it determines the filter is no longer in use.

### Interfaces:

- **Filter**: This interface represents a filter that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource, or both.

### Program:

```
AuthenticationFilt  
er.java import  
java.io.IOExceptio  
n; import  
javax.servlet.*;  
import
```



```
javax.servlet.http
.*;
public class AuthenticationFilter implements Filter { private
ServletContext context

    @Override

    public void init(FilterConfig
fConfig) throws ServletException {

        this.context = fConfig.getServletContext();
        this.context.log("AuthenticationFilter initialized");

    }

    @Override

    public void
doFilter(ServletRequest request,
ServletResponse response, FilterChain
chain)

        throws IOException, ServletException {

        HttpServletRequest req =
        (HttpServletRequest) request;
        HttpServletResponse res =
        (HttpServletResponse)
response;
        String uri = req.getRequestURI();
        this.context.log("Requested
Resource::" + uri);
        HttpSession session =

        req.getSession(false); boolean

        loggedIn = session != null &&

        session.getAttribute("username")

        != null;

        boolean loginRequest =
uri.endsWith("login.html") ||
uri.endsWith("LoginServlet");
```



```
        if (loggedIn || loginRequest)
            { chain.doFilter(request,
                response);
        } else {
            this.context.log("Unauthorized access
            request");
            res.sendRedirect("login.html");
        }
    }

    @Override
    public void destroy() {

        // Close any resources here if needed

    }
}
```

### **index.html**

```
<!DOCTYPE html>

<html>

<head>

    <title>Login</title>

</head>

<body>

    <h2>Login</h2>

    <form action="LoginServlet" method="post">

        Username: <input type="text"
        name="username"><br> Password: <input
        type="password" name="password"><br>
        <input type="submit" value="Login">

    </form>

</body>

</html>
```

**web.xml**

```
<filter>

    <filter-name>AuthenticationFilter</filter-name>

    <filter-class>AuthenticationFilter</filter-class>

</filter>

<filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>

    <url-pattern>/*</url-pattern>
</filter-mapping>
```

## Practical No: 15

**AIM:** Write a JSP program to find out details of user available in the system.

**Description:**

**Methods:**

- `doGet(HttpServletRequest request, HttpServletResponse response)`: This method is overridden from the `HttpServlet` class. It handles HTTP GET requests.
- `System.getProperty(String key)`: This method retrieves the system property indicated by the specified key (in this case, "user.name" and "user.home").
  - `FileSystems.getDefault()`: This method returns the default file system.
- `Path.getPath(String first, String... more)`: This method constructs a `Path` by parsing a given string, or part of it, according to the default `FileSystem`.
- `request.setAttribute(String name, Object o)`: This method sets an attribute in the request scope, which can be accessed later.
- `request.getRequestDispatcher(String path)`: This method returns a `RequestDispatcher` object that acts as a wrapper for the resource located at the given path.
- `forward(ServletRequest request, ServletResponse response)`: This method forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

**Classes and Interfaces:**

- `HttpServletRequest`: This is an interface representing an HTTP request. It provides methods for accessing request information.
- `HttpServletResponse`: This is an interface representing an HTTP response. It provides methods for setting response content and headers.
- `FileSystems`: This is a final class providing methods for interacting with the file system.
- `Path`: This is an interface representing a path in the file system. It provides methods for manipulating paths.
  - `ServletException`: This is an exception indicating a servlet-specific problem.
  - `IOException`: This is an exception indicating an I/O-related problem.

- **@WebServlet:** This is an annotation used to define a servlet component in a web application. It maps the servlet to a URL pattern.
- **HttpServlet:** This is an abstract class that provides methods for handling HTTP requests. It is extended to create servlets.
- **RequestDispatcher:** This is an interface used to dispatch requests from a servlet to another resource.

## Program:

### UserDetailsServlet.java

```
import java.io.IOException; import java.nio.file.FileSystems;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet; import
jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest; import
jakarta.servlet.http.HttpServletResponse;
@WebServlet("/user-details")

public class UserDetailsServlet
    extends HttpServlet { private
        static final long serialVersionUID
            = 1L;
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)

        throws ServletException, IOException {
        // Get the current user's name and home
        directory String userName =
        System.getProperty("user.name"); String
        userHome =

        FileSystems.getDefault().getPath(System.getProperty("user.hom
        e ")).toString();

        details
        // Set attributes in request to be accessed in JSP
        request.setAttribute("userName", userName);
        request.setAttribute("userHome", userHome);

        // Forward the request to the JSP page to display user

        request.getRequestDispatcher("/user-details.jsp").forward(r
        equest, response);}}
```

### 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_4_0.xsd"

    version="4.0">
<servlet>

    <servlet-name>UserDetailsServlet</servlet-name>

    <servlet-class>UserDetailsServlet</servlet-class>

</servlet>

<servlet-mapping>

    <servlet-name>UserDetailsServlet</servlet-name>

    <url-pattern>/user-details</url-pattern>

</servlet-mapping>
</web-app>

user-details.jsp

<!DOCTYPE html>

<html>

<head>

    <title>User Details</title>

</head>

<body>

    <h2>User Details</h2>

    <p><strong>Username:</strong> ${userName}</p>

    <p><strong>User Home Directory:</strong> ${userHome}</p>

</body>
```

</html>

**index.html**

<!DOCTYPE html>

<html>

<head>

    <title>User Details</title>

</head>

<body>

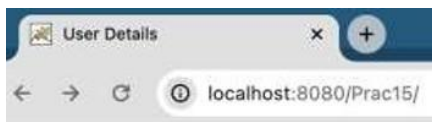
    <h2>User Details</h2>

    <a href="user-details">View User Details</a>

</body>

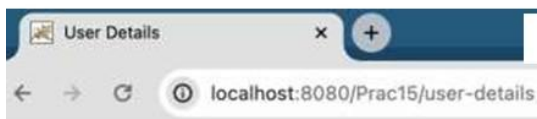
</html>

### Output:



### User Details

[View User Details](#)



### User Details

**Username: anshumishah**

**User Home Directory: /user/anshumishah**