UNIT IV – JAVA NETWORK PROGRAMMING & JDBC

1. SOCKET PROGRAMMING IN JAVA

Definition

Socket programming allows communication **between two programs** (client and server) running on the same or different machines using **TCP/IP protocol**.

What is a Socket?

A **Socket** is an **endpoint** for communication between two machines.

Types of Sockets in Java

Type	Protocol	Class Used	Features
Stream Socket	TCP (Transmission Control Protocol)	Socket / ServerSocket	Reliable, connection-oriented
Datagram Socket	UDP (User Datagram Protocol)	DatagramSocket / DatagramPacket	Fast, connectionless

2. BASIC SOCKET COMMUNICATION SETUP

Steps to Run

1. Create a **Server Program** (run first).
2. Create a **Client Program** (run next).
3. Use **localhost** or IP address (e.g., "127.0.0.1") and a **port number** (e.g., 5000).
4. Communicate using InputStream and OutputStream.

Example: Simple TCP Socket Program

Server Program

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

public class SimpleServer {
   public static void main(String[] args) throws IOException {
      ServerSocket ss = new ServerSocket(5000); // Create server socket
      System.out.println("Server started... Waiting for client");
      Socket s = ss.accept(); // Wait for client
```

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

        DataInputStream dis = new DataInputStream(s.getInputStream());
        String message = dis.readUTF();
        System.out.println("Message from client: " + message);

        dis.close();
        s.close();
        ss.close();
    }
}
```

Client Program

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

public class SimpleClient {
    public static void main(String[] args) throws IOException {
        Socket s = new Socket("localhost", 5000); // Connect to server
        DataOutputStream dos = new DataOutputStream(s.getOutputStream());

        dos.writeUTF("Hello Server, I am Client!");
        dos.flush();

        dos.close();
        s.close();
    }
}
```

Steps to Execute:

1. Compile both files.
2. Run SimpleServer first → Wait for connection.
3. Run SimpleClient → Sends message.
4. Observe output on server side.

3. INETADDRESS CLASS

Definition

InetAddress class represents an **IP address** of a host.

Common Methods

Method	Description
getLocalHost()	Returns IP of current machine
getByName("hostname")	Returns IP of a given host
getHostName()	Returns host name
getHostAddress()	Returns IP address as string

Example

```
import java.net.*;
public class InetExample {
   public static void main(String[] args) throws UnknownHostException {
      InetAddress address = InetAddress.getLocalHost();
      System.out.println("Host Name: " + address.getHostName());
      System.out.println("IP Address: " + address.getHostAddress());
   }
}
```

4. URL CLASS

Definition

The URL (Uniform Resource Locator) class represents a web address.

Important Methods

Method	Description
getProtocol()	Returns protocol (http/https)
getHost()	Returns host name
getPort()	Returns port number
getFile()	Returns file name
openStream()	Opens connection stream

Example

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

public class URLExample {
   public static void main(String[] args) throws Exception {
      URL url = new URL("https://www.example.com");
      System.out.println("Protocol: " + url.getProtocol());
      System.out.println("Host: " + url.getHost());
      System.out.println("Port: " + url.getPort());
      System.out.println("File: " + url.getFile());
   }
}
```

5. TCP & UDP PROTOCOLS IN JAVA

TCP (Transmission Control Protocol)

- **Connection-oriented**
- Reliable delivery (error checking, acknowledgment)
- Used for chat, file transfer, web communication
- Classes: Socket, ServerSocket

UDP (User Datagram Protocol)

- **Connectionless**
- Faster but **unreliable**
- Used in real-time apps (video/audio streaming)
- Classes: DatagramSocket, DatagramPacket

Example: UDP Communication

Server

```
import java.net.*;

public class UDPServer {
   public static void main(String[] args) throws Exception {
      DatagramSocket ds = new DatagramSocket(3000);
      byte[] buf = new byte[1024];
      DatagramPacket dp = new DatagramPacket(buf, 1024);
      ds.receive(dp);
      String msg = new String(dp.getData(), 0, dp.getLength());
      System.out.println("Message: " + msg);
```

```
    ds.close();
  }
}
```

Client

```
import java.net.*;
public class UDPClient {
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Hello UDP Server!";
    InetAddress ip = InetAddress.getByName("localhost");
    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}
```

6. MULTI-THREADED SERVERS

Definition

A multi-threaded server can handle **multiple clients at the same time**, each client on a separate thread.

Example

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

class ClientHandler extends Thread {
  Socket socket;
  ClientHandler(Socket s) { socket = s; }

  public void run() {
    try {
      DataInputStream dis = new DataInputStream(socket.getInputStream());
      String msg = dis.readUTF();
      System.out.println("Received: " + msg);
      dis.close();
      socket.close();
    } catch(Exception e) { System.out.println(e); }
  }
}
```

```
public class MultiThreadServer {
   public static void main(String[] args) throws IOException {
      ServerSocket ss = new ServerSocket(6000);
      System.out.println("Server ready...");
      while (true) {
         Socket s = ss.accept();
         new ClientHandler(s).start();
      }
   }
}
```

7. REMOTE METHOD INVOCATION (RMI)

Definition

RMI allows a Java program to **invoke methods of remote objects** running on a different JVM or machine.

Steps to Create RMI Application

1. Create a **Remote Interface** (extends Remote)
2. Implement the interface
3. Create **Server** to bind object
4. Create **Client** to call methods
5. Start **RMI Registry**
6. Run Server → then Client

Example: RMI Program

Step 1: Remote Interface

```
import java.rmi.*;

public interface AddInterface extends Remote {
   int add(int a, int b) throws RemoteException;
}
```

Step 2: Implementation

```
import java.rmi.*;
import java.rmi.server.*;

public class AddImpl extends UnicastRemoteObject implements AddInterface {
   AddImpl() throws RemoteException { super(); }
   public int add(int a, int b) { return a + b; }
```

```
}
```

Step 3: Server

```
import java.rmi.*;
import java.rmi.registry.*;

public class AddServer {
   public static void main(String args[]) {
      try {
         AddImpl obj = new AddImpl();
         LocateRegistry.createRegistry(1099);
         Naming.rebind("AddService", obj);
         System.out.println("RMI Server ready...");
      } catch(Exception e) { System.out.println(e); }
   }
}
```

Step 4: Client

```
import java.rmi.*;

public class AddClient {
   public static void main(String args[]) {
      try {
         AddInterface stub = (AddInterface) Naming.lookup("rmi://localhost/AddService");
         System.out.println("Sum: " + stub.add(10, 20));
      } catch(Exception e) { System.out.println(e); }
   }
}
```

Running RMI Program

1. Compile all files (javac *.java)
2. Start RMI registry → rmiregistry
3. Run Server → java AddServer
4. Run Client → java AddClient

8. OBJECT SERIALIZATION IN RMI

Definition

Serialization means converting an object into a byte stream to send over a network or save to a file.
Java objects must **implement Serializable** interface.

Example

```
import java.io.*;
class Student implements Serializable {
   int id; String name;
   Student(int i, String n) { id = i; name = n; }
}
```

9. JDBC (Java Database Connectivity)

Definition

JDBC is an API that enables Java applications to **connect to a database**, execute queries, and retrieve data.

JDBC Architecture

Java Application
↓
JDBC API
↓
JDBC Driver
↓
Database

JDBC Driver Types

Type	Description
Type 1	JDBC-ODBC Bridge
Type 2	Native-API Driver
Type 3	Network Protocol Driver
Type 4	Thin Driver (Pure Java)

Steps to Connect to Database

1. Load the Driver class
2. Establish Connection
3. Create Statement
4. Execute Query
5. Process Result
6. Close Connection

Example: JDBC Program

```
import java.sql.*;

public class JDBCExample {
   public static void main(String[] args) {
      try {
         // Step 1: Load driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // Step 2: Connect
         Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

         // Step 3: Create statement
         Statement stmt = con.createStatement();

         // Step 4: Execute query
         ResultSet rs = stmt.executeQuery("SELECT * FROM student");

         // Step 5: Process result
         while (rs.next()) {
            System.out.println(rs.getInt(1) + " " + rs.getString(2));
         }

         // Step 6: Close connection
         con.close();
      } catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

10. EXECUTING SQL QUERIES

Method	Used For
executeQuery()	SELECT statements
executeUpdate()	INSERT, UPDATE, DELETE
execute()	Any SQL statement

Example:

```
stmt.executeUpdate("INSERT INTO student VALUES (1, 'Ravi')");
```

SUMMARY

Concept	Key Points
Socket	End point for communication
TCP/UDP	TCP – reliable; UDP – fast
InetAddress	Handle IP addresses
URL	Represent web addresses
ServerSocket	For TCP server
Multi-threaded Server	Handles many clients
RMI	Remote method call
Serialization	Convert object to byte stream
JDBC	Database connectivity in Java
DriverManager	Establish connection
Statement	Execute SQL queries