Experiment - 6

Aim :- Implement Datagram UDP socket programming in java

Overview:

The provided Java code demonstrates Datagram UDP socket programming. The UDP server listens on port 9876, receiving messages from clients and optionally sending a response. The UDP client sends a "Hello" message to the server, prints the sent message, and optionally receives a response from the server. This implementation enables simple communication between a UDP server and client using datagrams in Java

Code:

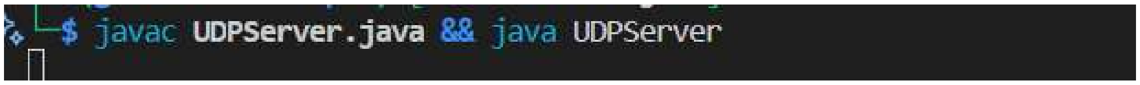
UDP Server

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

class UDPServer {
  public static void main(String args[]) throws Exception {
    DatagramSocket serverSocket = new DatagramSocket(9876);
    byte[] receiveData = new byte[1024];
    byte[] sendData;
    while (true) {
      DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
      serverSocket.receive(receivePacket);
      String sentence = new String(receivePacket.getData(), 0,
receivePacket.getLength());
      InetAddress IPAddress = receivePacket.getAddress();
      int port = receivePacket.getPort();
      String capitalizedSentence = sentence.toUpperCase();
      sendData = capitalizedSentence.getBytes();
      DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
          IPAddress, port);
      serverSocket.send(sendPacket);
    }
  }
}
```

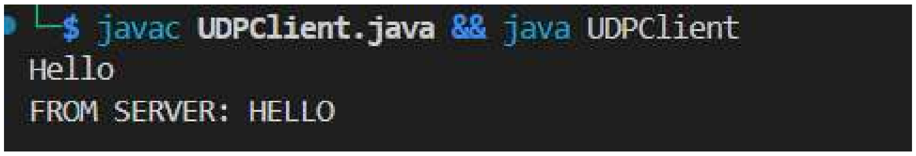
UDP Server

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

class UDPClient {
  public static void main(String args[]) throws Exception {
    BufferedReader inFromUser = new BufferedReader(
        new InputStreamReader(System.in));
    DatagramSocket clientSocket = new DatagramSocket();
    InetAddress IPAddress = InetAddress.getByName("localhost");
    byte[] sendData;
    byte[] receiveData = new byte[1024];
    String sentence = inFromUser.readLine();
    sendData = sentence.getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
        IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence = new String(receivePacket.getData(),
        0, receivePacket.getLength());
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
  }
}
```

Output:



```
└─$ javac UDPServer.java && java UDPServer
```



```
└─$ javac UDPClient.java && java UDPClient
Hello
FROM SERVER: HELLO
```

Experiment - 7

Aim :- Implement Socket programming for TCP in Java Server and Client Sockets.

Overview:

This Java code implements TCP socket programming with a server and client. The TCP server listens on a specified port, accepts client connections, and can handle multiple clients concurrently. The client connects to the server, sends a message ("Hello, TCP Server!"), and receives a response. This implementation facilitates communication between a TCP server and client through sockets in Java.

Code:

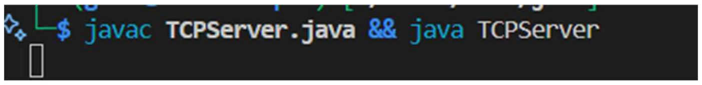
TCP Server

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

class TCPServer {

 public static void main(String argv[]) throws Exception {
  String clientSentence;
  String capitalizedSentence;
  ServerSocket welcomeSocket = new ServerSocket(6789);
  while (true) {
   Socket connectionSocket = welcomeSocket.accept();
   BufferedReader inFromClient = new BufferedReader(
    new InputStreamReader(connectionSocket.getInputStream())
   );
   DataOutputStream outToClient = new DataOutputStream(
    connectionSocket.getOutputStream()
   );
   clientSentence = inFromClient.readLine();
   capitalizedSentence = clientSentence.toUpperCase() + '\n';
   outToClient.writeBytes(capitalizedSentence);
  }
 }
}
```

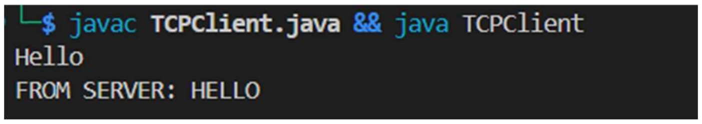
TCP Client

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

class TCPClient {

 public static void main(String argv[]) throws Exception {
    String sentence;
    String modifiedSentence;
    BufferedReader inFromUser = new BufferedReader(
      new InputStreamReader(System.in)
    );
    Socket clientSocket = new Socket("localhost", 6789);
    DataOutputStream outToServer = new DataOutputStream(
      clientSocket.getOutputStream()
    );
    BufferedReader inFromServer = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream())
    );
    sentence = inFromUser.readLine();
    outToServer.writeBytes(sentence + '\n');
    modifiedSentence = inFromServer.readLine();
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
  }
}
```

Output:



```
└─$ javac TCPServer.java && java TCPServer
```



```
└─$ javac TCPClient.java && java TCPClient
Hello
FROM SERVER: HELLO
```

Experiment - 8

Aim :- Socket Programming

Overview:

This Java code represents a basic implementation of TCP socket programming with a server and client. The server waits for a connection, accepts a client, and reads messages until the client sends "Over." The client connects to the server, sends messages from the terminal to the server until "Over" is entered, and then closes the connection. This practical demonstrates simple bidirectional communication between a TCP server and client, establishing a connection and exchanging messages over sockets in a networked environment.

Code:

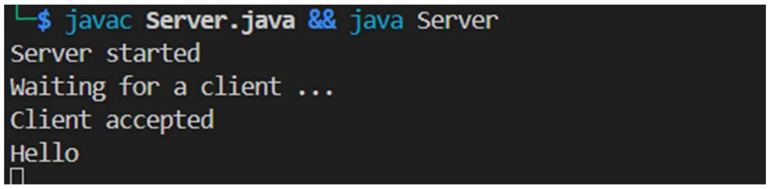
Server

```
// A Java program for a Server
import java.io.*;
import java.net.*;

public class Server {

  //initialize socket and input stream
  private Socket socket = null;
  private ServerSocket server = null;
  private DataInputStream in = null;

  // constructor with port
  public Server(int port) {
    // starts server and waits for a connection
    try {
      server = new ServerSocket(port);
      System.out.println("Server started");
      System.out.println("Waiting for a client ...");
      socket = server.accept();
      System.out.println("Client accepted");
      // takes input from the client socket
      in =
        new DataInputStream(new BufferedInputStream(socket.getInputStream()));
      String line = "";
      // reads message from client until "Over" is sent
      while (!line.equals("Over")) {
        try {
          line = in.readUTF();
          System.out.println(line);
        } catch (IOException i) {
          System.out.println(i);
      }
    }
    System.out.println("Closing connection");
    // close connection
    socket.close();
    in.close();
   } catch (IOException i) {
    System.out.println(i);
   }
  }

  public static void main(String args[]) {
   Server server = new Server(5000);
  }
}
```

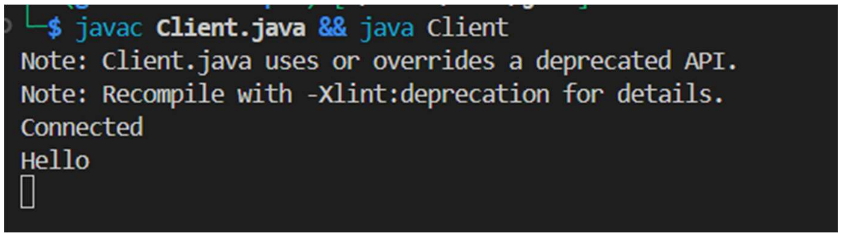
Client

```
// A Java program for a Client
import java.io.*;
import java.net.*;

public class Client {

  // initialize socket and input output streams
  private Socket socket = null;
  private DataInputStream input = null;
  private DataOutputStream out = null;

  // constructor to put ip address and port
  public Client(String address, int port) {
   // establish a connection
   try {
    socket = new Socket(address, port);
    System.out.println("Connected");
    // takes input from terminal
    input = new DataInputStream(System.in);
    // sends output to the socket
    out = new DataOutputStream(socket.getOutputStream());
   } catch (UnknownHostException u) {
    System.out.println(u);
    return;
   } catch (IOException i) {
    System.out.println(i);
    return;
   }
   // string to read message from input
   String line = "";
```

```
    // keep reading until "Over" is input
    while (!line.equals("Over")) {
      try {
        line = input.readLine();
        out.writeUTF(line);
      } catch (IOException i) {
        System.out.println(i);
      }
    }
    // close the connection
    try {
      input.close();
      out.close();
      socket.close();
    } catch (IOException i) {
      System.out.println(i);
    }
  }

  public static void main(String args[]) {
    Client client = new Client("127.0.0.1", 5000);
  }
}
```



```
└─$ javac Server.java && java Server
Server started
Waiting for a client ...
Client accepted
Hello
```



```
└─$ javac Client.java && java Client
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Connected
Hello
```

Experiment - 9

Aim :- Implement a basic Java client-server interaction using sockets, facilitating communication between a server and multiple clients for date and time queries.

Overview:

The practical consists of two Java programs: Server1 listens on port 5056, handling incoming client connections with a dedicated thread (ClientHandler). Clients connect to the server, request either the current date or time, and receive the corresponding information. The interaction continues until a client requests to exit, closing the connection. The implementation emphasizes socket programming, multi-threading, and communication between server and clients.

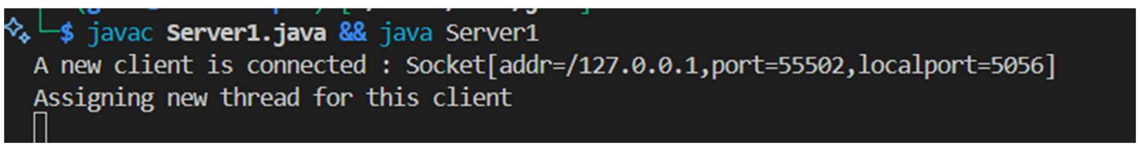
Code:

Server1

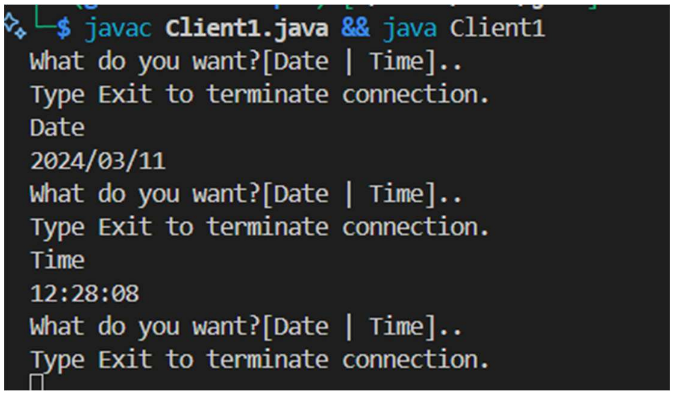
```
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;

// Server class
public class Server1 {

 public static void main(String[] args) throws IOException {
  // server is listening on port 5056
  ServerSocket ss = new ServerSocket(5056);

  // running infinite loop for getting
  // client request
  while (true) {
   Socket s = null;

   try {
    // socket object to receive incoming client requests
    s = ss.accept();

    System.out.println("A new client is connected : " + s);

    // obtaining input and out streams
    DataInputStream dis = new DataInputStream(s.getInputStream());
    DataOutputStream dos = new DataOutputStream(s.getOutputStream());

    System.out.println("Assigning new thread for this client");
    // create a new thread object
    Thread t = new ClientHandler(s, dis, dos);
    // Invoking the start() method
    t.start();
   } catch (Exception e) {
        s.close();
        e.printStackTrace();
      }
    }
  }
}

// ClientHandler class
class ClientHandler extends Thread {

  DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");
  DateFormat fortime = new SimpleDateFormat("hh:mm:ss");
  final DataInputStream dis;
  final DataOutputStream dos;
  final Socket s;

  // Constructor
  public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos) {
    this.s = s;
    this.dis = dis;
    this.dos = dos;
  }

  @Override
  public void run() {
    String received;
    String toreturn;
    while (true) {
      try {
        // Ask user what he wants
        dos.writeUTF(
          "What do you want?[Date | Time]..\n" +
          "Type Exit to terminate connection."
        );

        // receive the answer from client
        received = dis.readUTF();

        if (received.equals("Exit")) {
          System.out.println("Client " + this.s + " sends exit...");
          System.out.println("Closing this connection.");
          this.s.close();
          System.out.println("Connection closed");
          break;
        }

        // creating Date object
        Date date = new Date();
        // write on output stream based on the
        // answer from the client
        switch (received) {
          case "Date":
            toreturn = fordate.format(date);
            dos.writeUTF(toreturn);
            break;
          case "Time":
            toreturn = fortime.format(date);
            dos.writeUTF(toreturn);
            break;
          default:
            dos.writeUTF("Invalid input");
            break;
        }
      } catch (IOException e) {
        e.printStackTrace();
      }
    }

    try {
      // closing resources
      this.dis.close();
      this.dos.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Client

```
// Java implementation for a client
// Save file as Client.java
import java.io.*;
import java.net.*;
import java.util.Scanner;

// Client class
public class Client1 {

  public static void main(String[] args) throws IOException {
    try {
      Scanner scn = new Scanner(System.in);

      // getting localhost ip
      InetAddress ip = InetAddress.getByName("localhost");
      // establish the connection with server port 5056
      Socket s = new Socket(ip, 5056);

      // obtaining input and out streams
      DataInputStream dis = new DataInputStream(s.getInputStream());
      DataOutputStream dos = new DataOutputStream(s.getOutputStream());

      // the following loop performs the exchange of
      // information between client and client handler
      while (true) {
       System.out.println(dis.readUTF());
       String tosend = scn.nextLine();
       dos.writeUTF(tosend);

       // If client sends exit,close this connection
       // and then break from the while loop
       if (tosend.equals("Exit")) {
        System.out.println("Closing this connection : " + s);
        s.close();
        System.out.println("Connection closed");
        break;
       }

       // printing date or time as requested by client
       String received = dis.readUTF();
       System.out.println(received);
      }

      // closing resources
      scn.close();
      dis.close();
      dos.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```



```
└─$ javac Server1.java && java Server1
  A new client is connected : Socket[addr=/127.0.0.1,port=55502,localport=5056]
  Assigning new thread for this client
```



```
└─$ javac Client1.java && java Client1
What do you want?[Date | Time]..
Type Exit to terminate connection.
Date
2024/03/11
What do you want?[Date | Time]..
Type Exit to terminate connection.
Time
12:28:08
What do you want?[Date | Time]..
Type Exit to terminate connection.
```

Experiment - 10

Aim :- Demonstrate basic client-server communication using Java's ServerSocket and Socket classes.

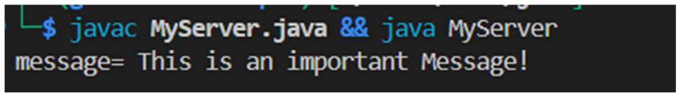
Overview:

The server program (MyServer) creates a ServerSocket on port 6666, accepts a client connection, reads a message using DataInputStream, and displays it. The client program (MyClient) connects to the server, sends a message using DataOutputStream, and closes the connection. This simple implementation showcases socket communication, with the server listening for client messages and the client sending a predefined message to the server. The programs illustrate the fundamental concepts of socket programming in Java.

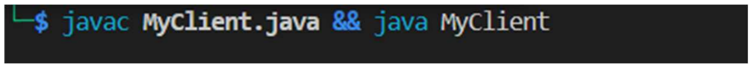
Code:

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

public class MyServer {

  public static void main(String[] args) {
    try {
      ServerSocket ss = new ServerSocket(6666);
      Socket soc = ss.accept();
      DataInputStream dis = new DataInputStream(soc.getInputStream());
      String str = (String) dis.readUTF();
      System.out.println("message= " + str);
      ss.close();
    catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

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

public class MyClient {

  public static void main(String[] args) {
    try {
      Socket soc = new Socket("localhost", 6666);
      DataOutputStream d = new DataOutputStream(soc.getOutputStream());
      d.writeUTF("This is an important Message!");
      d.flush();
    d.close();
    soc.close();
  catch (Exception e) {
    System.out.println(e);
  }
 }
}
```



```
└─$ javac MyServer.java && java MyServer
message= This is an important Message!
```



```
└─$ javac MyClient.java && java MyClient
```

Experiment - 11

Aim :- Illustrate bidirectional communication between a Java server and client using sockets.

Overview:

The server (MyServer1) establishes a ServerSocket on port 3333, accepts a client connection, and engages in a continuous dialogue. It reads messages from the client, prints them to the console, and responds with messages input by the server. The client (MyClient1) connects to the server, sends messages entered by the client, and prints server responses. The communication loop continues until either side inputs "stop," closing the connection gracefully. This example emphasizes the fundamentals of socket programming, showcasing a basic chat-like interaction between a server and client.

Code:

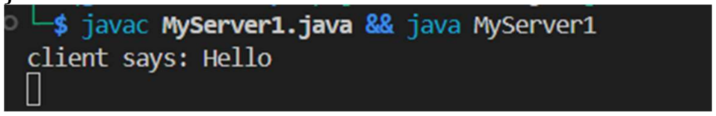
MyServer

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

class MyServer1 {

  public static void main(String args[]) throws Exception {
    ServerSocket ss = new ServerSocket(3333);
    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();
  }
}
```

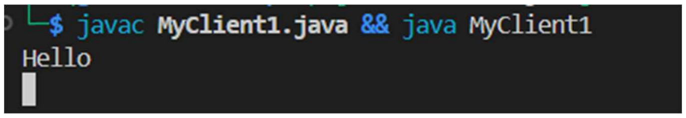
MyClient

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

```
class MyClient1 {

  public static void main(String args[]) throws Exception {
    Socket s = new Socket("localhost", 3333);
    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();
  }
}
```



```
└─$ javac MyServer1.java && java MyServer1
client says: Hello
```



```
└─$ javac MyClient1.java && java MyClient1
Hello
```