Network Programming

- A network is a collection of devices that share a common communication protocol and connected with some communication medium.
- A protocol defines a set of rules to which all the communicating parties adhere to.
- The communication medium may be guided medium like cables or unguided medium like wifi.
- Networks consist of connections different computing devices like computers, printers, routers, etc. Such connections carry data between one point in the network and another. This data is represented as bits i.e. a sequence of 0 and 1.
- The main uses of network can be described by the following keywords:
  - Resource Sharing,
  - High Reliability,
  - economical (saves money),
  - Communication Medium,
  - Access to remote information.
- Network Programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

- The good aspect of Java is that it includes a cross platform model for network communication.
- The java.net package was designed with a group of classes and interfaces which not only provide low level but also high level communication details.
- The java.net package provides support for the two common network protocols i.e. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) .
  - TCP allows for reliable communication between two applications.
  - UDP is a connection-less protocol that provides unreliable but best effort communication service.
- The java.net package provides Networking Classes. For example:
  - InetAddress Class
  - ServerSocket Class:
  - Socket Class
  - URL Class
  - URLConnection Class
  - DatagramSocket Class
  - DatagramPacket class
  - etc.

Port Number:

- The port number field of an IP packet is specified as a 16-bit unsigned integer.
- Each of the applications get at least one port number to uniquely identify them.
- Different kinds of ports:
  - well-known ports:
    - ports 0 to 1023 are reserved for well known services
    - A well-known service is widely implemented and a well known port is assigned to them. For e.g. 22 for Secure Shell (ssh) , 80 for HTTP, 21 for FTP, etc.
  - Ephemeral ports/Registered:
    - ports starting from 1024 can be custom used
    - ports can be used by ordinary users.
  - Dynamic / Private ports:
    - Used for private, or customized services or temporary purposes

Socket:

- A socket is one endpoint of a two-way communication link between two programs running on the network.
- Socket defines a network connection as a stream from which bytes can be read and similarly bytes can be send.
- Java Socket can be used to provide both connection-oriented as well as connectionless services.
- While creating a socket for communication the connection oriented system involves the four tuples:
  - client IP address.
  - Client Port Number,
  - server IP address and
  - server Port Number.
- But for connectionless service only two tuples will be sufficient i.e. server IP address and server Port Number.

Steps in Client Server communication

1. Sockets Program using UDP connection:

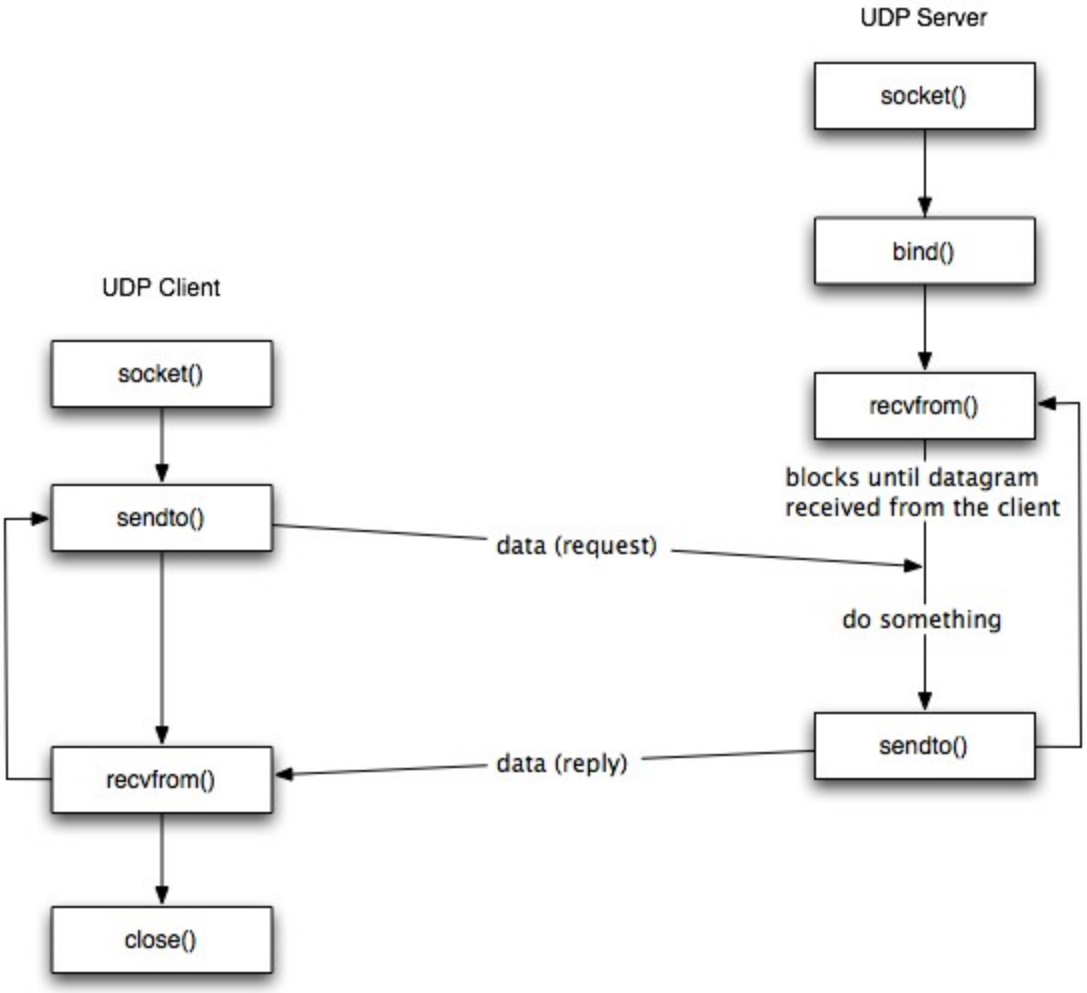

Figure : UDP client Server programming (https://bit.ly/3Zykfrr)

2. Socket Program using TCP connection:

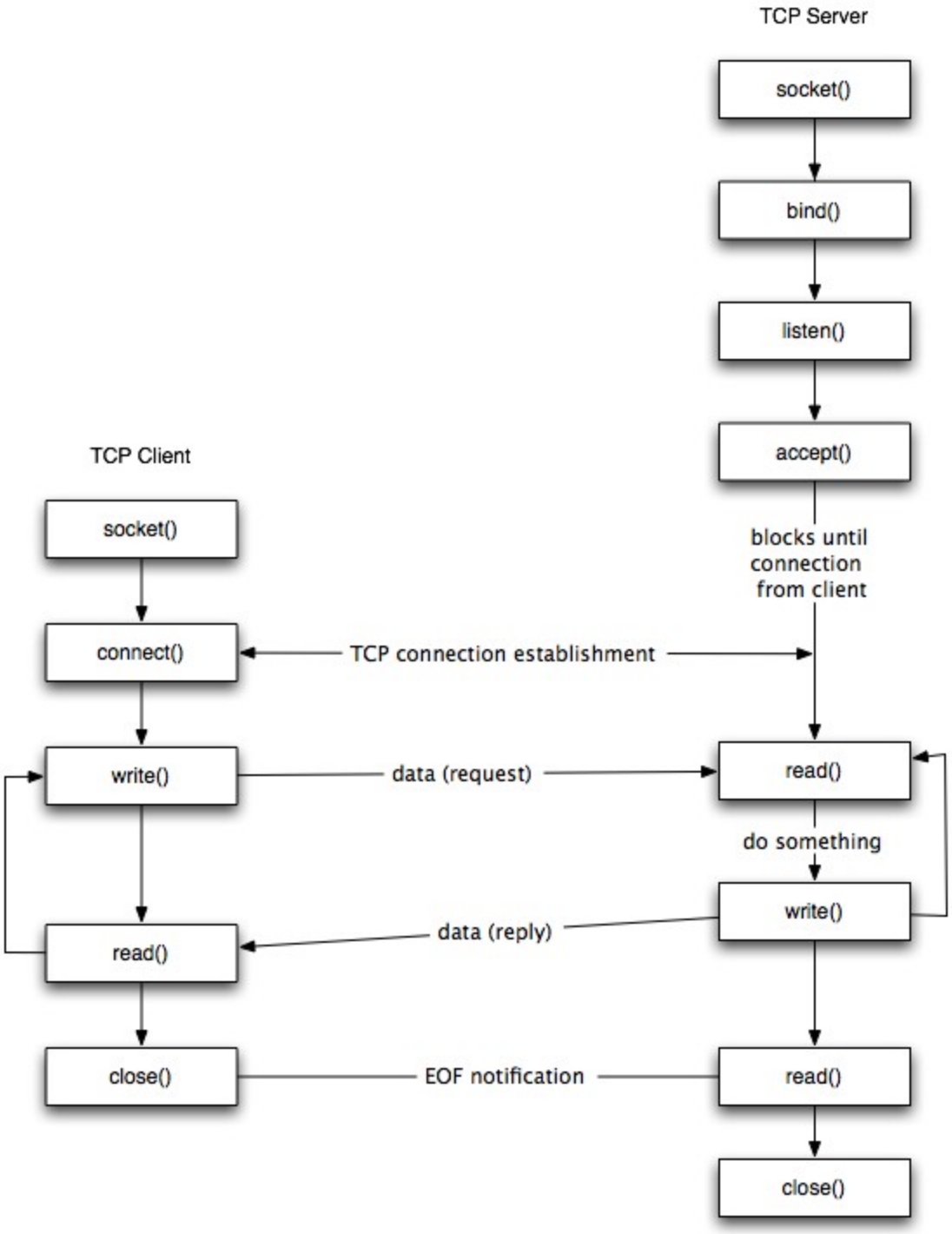

Figure : TCP client Server programming(https://bit.ly/3Zykfrr)

```java
1# Program to get the IP address of given host.

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

public class IP
{
  public static void main ( String[] args ) throws IOException
  {
    String hostname = args[0];

    try
    {
      InetAddress ipaddress = InetAddress.getByName(hostname);
      System.out.println("IP address: " + ipaddress.getHostAddress());
    }
    catch ( UnknownHostException e )
    {
      System.out.println("Could not find IP address for: " + hostname);
    }
  }
}

// compile: javac IP.java
//run: java IP google.com
```

```
2# The following URLDemo program demonstrates the various parts of a URL.
A URL is entered on the command line, and the URLDemo program outputs
each part of the given URL.

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

public class UrlDemoJava
{
   public static void main(String [] args)
   {
      try
      {
      //  URL url = new URL("https://www.google.com.np/?gws_rd=cr&ei=URmRU8y_EcfpkAWK4oDoCA");
          URL url = new URL("https://www.gces.edu.np");
         System.out.println("URL is " + url.toString());
         System.out.println("protocol is "
                                    + url.getProtocol());
         System.out.println("authority is "
                                    + url.getAuthority());
         System.out.println("file name is " + url.getFile());
         System.out.println("host is " + url.getHost());
         System.out.println("path is " + url.getPath());
         System.out.println("port is " + url.getPort());
         System.out.println("default port is "
                                   + url.getDefaultPort());
         System.out.println("query is " + url.getQuery());
         System.out.println("ref is " + url.getRef());
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

```
3# The Program to download the content of the given url.

import java.net.*;
import java.io.*;
public class UrlConnectionDemoJava
{
   public static void main(String [] args)
   {
      try
      {
         URL url = new URL("http://www.google.com");
         URLConnection urlConnection = url.openConnection();
         HttpURLConnection connection = null;
/*instanceof keyword can be used to test if an object is of a specified type.
if (objectReference instanceof type
*/
         if(urlConnection instanceof HttpURLConnection)
         {
            connection = (HttpURLConnection) urlConnection;
         }
         else
         {
            System.out.println("Please enter an HTTP URL.");
            return;
         }
         BufferedReader in = new BufferedReader(
         new InputStreamReader(connection.getInputStream()));
         String urlString = "";
         String current;
         while((current = in.readLine()) != null)
         {
            urlString += current;
         }
         System.out.println(urlString);
      }catch(IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

```
4# Program to demonstrate client server communication.

// SimpleServer.java: A simple server program.
import java.net.*;
import java.io.*;
public class SimpleServer {
    public static void main(String args[]) throws IOException {
        // Register service on port 1254
        ServerSocket s = new ServerSocket(1254);
        Socket s1=s.accept(); // Wait and accept a connection
        // Get a communication stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (s1out);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
        s1out.close();
        s1.close();
    }
}


// SimpleClient.java: A simple client program.
import java.net.*;
import java.io.*;
public class SimpleClient {
    public static void main(String args[]) throws IOException {
        // Open your connection to a server, at port 1254
        Socket s1 = new Socket("localhost",1254);
        // Get an input file handle from the socket and read the input
        InputStream s1In = s1.getInputStream();
        DataInputStream dis = new DataInputStream(s1In);
        String st = new String (dis.readUTF());
        System.out.println(st);
        // When done, just close the connection and exit
        dis.close();
        s1In.close();
        s1.close();
    }
}
```

5# Use UDP to write a program to demonstrate client server communication where a client program reads some message from the user and sends it to the server. Then the server sends acknowledge to the client.

6# Use TCP to write a program to demonstrate client server communication where a client program reads some message from the user and sends it to the server. Then the server sends acknowledge to the client.