# Network Programming

B.Tech. (IT), Sem-5,
Core Java Technology (CJT)

Dharmsinh Desai University
Prof. (Dr.) H B Prajapati



# TCP/IP

- TCP/IP is a de-facto standard of Internet communication
  - End-to-end transmission (TCP or UDP)
  - Packet routing (IP)
- TCP is a connection-oriented protocol, while UDP is a connectionless protocol.
- Connection-oriented protocols ensure (e.g., TCP)
  - guaranteed delivery,
  - sequence of data, and
  - acknowledgement for the data sent.
  - I.e., it is a reliable protocol
- connectionless protocols provide (e.g., UDP)
  - No guarantee for delivery
  - No sequencing and
  - No acknowledgement.
  - I.e., it is an unreliable protocol



# Port Numbers

- All standard applications run on standard port numbers
  - aka *well-known port numbers* (from 0–1023),
  - e.g. HTTP (80), FTP(20/21), DAYTIME (13), ECHO (7) etc
- Mostly server applications use well-known port numbers.
- The client applications use *ephemeral port numbers* (i.e. short-lived or temporary) starting from 1024 onwards.



# Socket

- A *socket* is an end point of communication. (analogy: electrical socket)
- Sockets are created at both ends of communication
- Socket is defined by three things:
  - *IP Address,*
  - *port*, and
  - the *protocol* to be used for communication.
- Sockets hold two streams:
  - An input stream and
  - An output stream.
- Each end of the socket has a pair of streams.
- Socket was initiated by University of California at Berkeley in Berkeley Software Distribution (BSD)



# Connection

- To address the connection, include the following:
  - The address or name of remote machine.
    - IP address (e.g., 10.11.12.13)
    - DNS name (e.g., www.ddu.ac.in)
  - A port number to identify the purpose at the server. Port numbers range from 0–65535.



# Sockets in Java

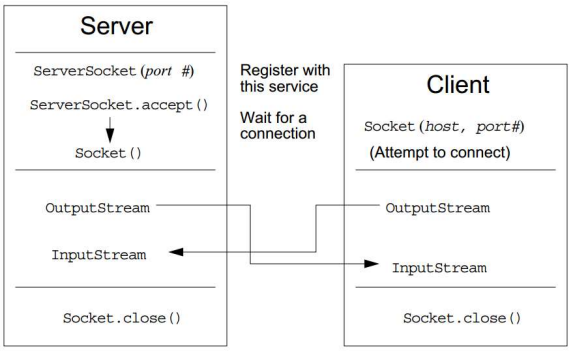
- There are two kinds of TCP sockets in Java.
  - One is for servers (ServerSocket)
  - The other is for clients (Socket).
- The ServerSocket is for servers.
  - It waits for clients to connect before doing anything.
  - It is also called listener.
  - For Information: UNIX Socket API allows making a server using the following functions
    - socket(), bind(), and listen()
    - On arrival of new connection accept() returns. The accept() is kept in an infinite while loop.

# Sockets in Java

- The Socket class is for clients
  - It connects to server sockets and initiate protocol exchanges.
  - Socket object implicitly establishes a connection between the client and server.
  - For Information: UNIX Socket API allows making a server using the following functions
    - socket()
    - Establish the connection to server using connect()



# Java Networking Model






# TCP client socket: Socket

- TCP clients are created with the help of Socket class.
- Client sockets need not be assigned a port number explicitly; they are automatically assigned a port number by the operating system and connected to the local IP address.
- Constructors:
  - Socket(String hostName, int port)
    - It can throw UnknownHostException and IOException
  - Socket(InetAddress, int port)
    - It can throw IOException



# Methods of Socket class

- Socket can be examined at any time for the address and port information associated with it.
- InetAddress getInetAddress()
  - It returns the InetAddress associated with the Socket object.
  - It returns null if the socket is not connected.
- int getPort()
  - It returns the remote port to which the invoking Socket object is connected.
  - It returns 0 if the socket is not connected
- int getLocalPort( )
  - It returns the local port (ephemeral or temporary or short-lived) to which the invoking Socket object is bound.
  - It returns –1 if the socket is not bound.



# Methods of Socket class

- Methods for Input from server and Output to server
  - InputStream getInputStream() throws IOException
    - It returns the InputStream associated with the invoking socket.
  - OutputStream getOutputStream() throws IOException
    - It returns the OutputStream associated with the invoking socket



# TCP server socket: ServerSocket

- Interest in client connections is indicated to the system using ServerSocket.
- The constructors for ServerSocket reflect the service port number, to which client will connect.
- Size of listen queue can also be specified
  - How many connections can be kept pending by server TCP, after which the server TCP will refuse connections
  - The default value is 50

# TCP server socket: ServerSocket

- Constructors:
  - ServerSocket(int port) throws IOException
    - Creates a server socket on the specified port with a queue length of 50.
  - ServerSocket(int port, int maxQueue) throws IOException
    - Creates a server socket on the specified port with the specified maximum queue length of maxQueue
  - ServerSocket(intpor t, int maxQueue, InetAddress localAddress) throws IOException
    - On a multihomed host (having more than one IP), specific IP address can be specified

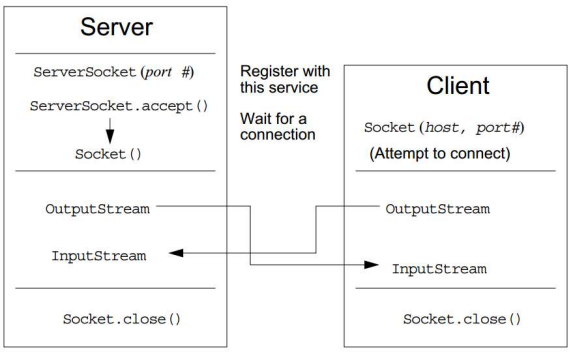


# TCP server socket: ServerSocket

- ServerSocket waits for a new client connection using accept()
  - public Socket accept() throws IOException
- The accept() is a blocking call that will wait for a client to initiate communications and then it will return with a Socket (connected with the client)
- The connected socket is then used for communication with the client.
- The ServerSocket accept new connection
- The connected Socket talks with the client

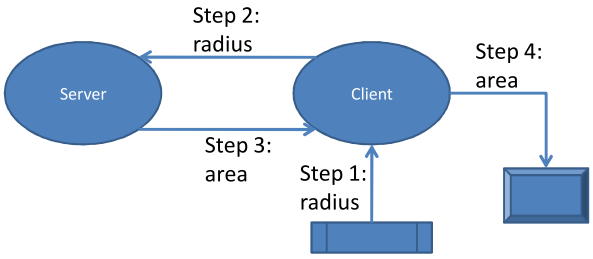


# Java Networking Model






# Program: TCP Server Client






# AreaServer.java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.StringTokenizer;

public class AreaServer{
   public static void main(String[] args){
        System.out.println("Press ^C to terminate the Server");
```



# AreaServer.java (cont…)

```
try{
        ServerSocket s=new ServerSocket(8000);
        while(true){
                Socket connectedSocket=s.accept();
                BufferedReader brFromClient=new
BufferedReader(new InputStreamReader(
connectedSocket.getInputStream()) );
                PrintWriter pwToClient=new PrintWriter(
                connectedSocket.getOutputStream(),
true);
```

# AreaServer.java (cont…)

```
                StringTokenizer st=new
StringTokenizer(brFromClient.readLine());
                double radius=new
Double(st.nextToken()).doubleValue();
                System.out.println("Server Received the
following from "+connectedSocket);
                System.out.println("\tRadius="+radius);
                double area= radius*radius*Math.PI;
                pwToClient.println(area);
                System.out.println("\tServer sent
Area="+area);
            }
        }
```



# AreaServer.java (cont…)

```
        catch(IOException e){
            System.out.println(e);
        }
    }
}
```



# AreaClient.java

```
import java.net.Socket;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.StringTokenizer;

public class AreaClient{
    public static void main(String[] args){
        try{
            Socket connectToServer = new
Socket("localhost",8000);
```



# AreaClient.java (cont…)

```
            BufferedReader brFromServer=new
BufferedReader(new
InputStreamReader(connectToServer.getInputStream()));
            PrintWriter pwToServer=new PrintWriter(
                connectToServer.getOutputStream(), true);
            BufferedReader brKeyboard=new
BufferedReader(new InputStreamReader( System.in));
```



# AreaClient.java (cont…)

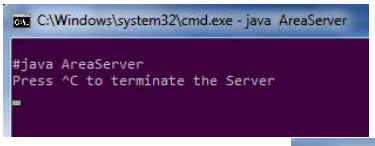
```
            System.out.print("Enter a radius = ");
            double
radius=Double.parseDouble(brKeyboard.readLine().trim());
            pwToServer.println(radius);
            StringTokenizer st=new
StringTokenizer(brFromServer.readLine());
            double area=new
Double(st.nextToken()).doubleValue();
            System.out.println("Area received from the server
is "+area);
        }
```

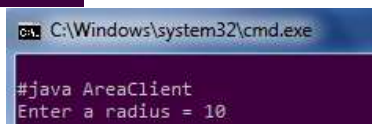


# AreaClient.java (cont…)

```
        catch(IOException e){
            System.out.println(e);
        }
    }
}
```

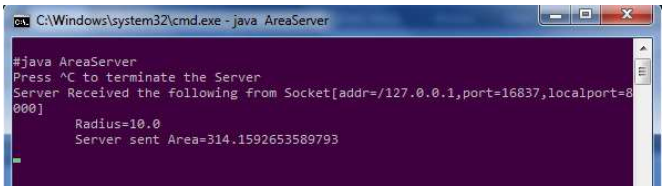
# Running the program

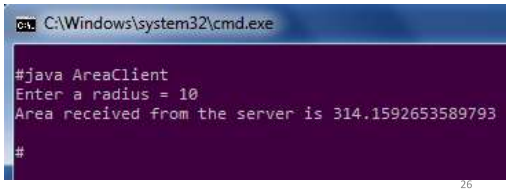





# Running the program






# Internet address: InetAddress class

- It has no accessible constructor
- It's object is created using factory methods.
  - static InternetAddress getLocalHost() throws UnknownHostException
    - It returns the InetAddress object that represents the local host
  - static InetAddress getByName(String hostName) throws UnknownHostException
    - The getByName( ) method returns an InetAddress for a host name passed to it.



# Internet address: InetAddress class

  - static InetAddress[] getAllByName(String hostName) throws UnknownHostException
    - On the Internet, it is common that several machines represent a single name (e.g., website).
    - The getAllByName( ) factory method returns an array of InetAddresses that represent all of the addresses that a particular name resolves to.
  - static InetAddress getByAddress(String ipAddress, byte[] addr)
    - Takes an IP address and returns an InetAddress object.
  - static InetAddress getByAddress(byte[] addr)



# Instance methods of InetAddress class

- Instance methods:
  - byte[ ] getAddress()
    - Provides address in a byte array in network byte order (as opposed to platform specific little-endian or big-endian order)
  - String getHostAddress()
    - IP address in string form
  - String getHostName()
    - Name of host associated with the address
  - boolean isMulticastAddress()
    - Whether multicast is supported?
    - Multicasting is supported on UDP socket.



# Inet4Address and Inet6Address

- Beginning with version 1.4, Java has included support for IPv6 addresses.
  - Two subclasses of InetAddress were created: Inet4Address and Inet6Address.
  - A reference of InetAddress can refer to an object of any of these two classes.

# Program: Printing addresses

```
import java.net.InetAddress;
import java.net.UnknownHostException;
class PrintAddresses{
    public static void main(String[] args){
        InetAddress[] addr=null;
        if(args.length!=1){
                System.out.println("Usage : GetNames
    Name_of_server");
        }
```

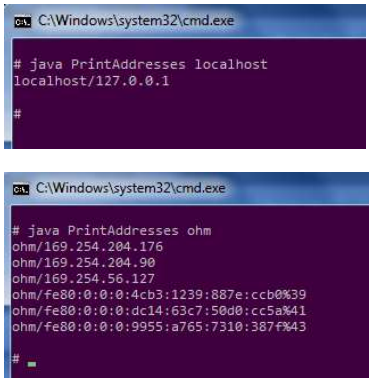


# Program: Printing addresses (cont…)

```
        try{
                addr=InetAddress.getAllByName(args[0]);
        }
        catch(UnknownHostException e){
                System.out.println(e);
        }
        for(int i=0;i<addr.length;i++){
                System.out.println(addr[i]);
        }
    }
}
```



# Running the Program: Printing addresses



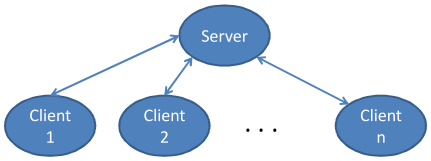






# Concurrent Server

- Concurrent servers are those that can process many clients at a time.
- Clients need not wait for other clients to finish their interaction with the server (parallel execution).
- A numbers of client's request arrives at the server, it is accepted and a thread is created for handling the client's request.
- The server then continues to listen requests from other clients.






# Concurrent Server

- The concurrent server can be coded using the following structure of the code:

```
while(true){
    Socket connectToClient = s.accept();
    Thread t=new ThreadClass(connectToClient);
    t.start();
}
```



# Concurrent Echo TCP Server

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.InputStream;
import java.io.IOException;

class EchoServer{
    public static void main(String[] args) throws
    IOException{
```

# Concurrent Echo TCP Server (cont…)

```
        int port=5555;
        Socket cs;
        ServerSocket ss=new ServerSocket(port);
        while(true){
                cs=ss.accept();
                System.out.println("One client
    connected:"+cs);
                new HandleRequest(cs);
        }
    }
}
```



# Concurrent Echo TCP Server (cont…)

```
class HandleRequest extends Thread{
    BufferedReader br;
    PrintWriter pw;
    HandleRequest(Socket s) throws IOException{
        InputStream is=s.getInputStream();
        br=new BufferedReader(new InputStreamReader(is));
        pw=new PrintWriter(s.getOutputStream(),true);
        System.out.println("Connection with Client is
    established");
        start();
    }
```



# Concurrent Echo TCP Server (cont…)

```
    public void run(){
        System.out.println("Interaction started");
        String s="";
        try{
                while((s=br.readLine())!=null){
                        System.out.println("Client Sent: "+s);
                        pw.write(s+"\n");
                        pw.flush();
                }
        }
```



# Concurrent Echo TCP Server (cont…)

```
        catch(Exception e){
                e.printStackTrace();
        }
    }
}
```



# Echo TCP Client

```
import java.net.Socket;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;

class EchoClient{
    public static void main(String[] args) throws IOException{
```



# Echo TCP Client (cont…)

```
        if(args.length!=2){
                System.out.println("Error pass name of server and
    port no");
                System.exit(0);
        }
        String s="";
        Socket cs=new Socket(args[0],Integer.parseInt(args[1]));
        InputStream is=cs.getInputStream();
        BufferedReader kbr=new BufferedReader(new
    InputStreamReader(System.in));
        BufferedReader br=new BufferedReader(new
    InputStreamReader(is));
```

# Echo TCP Client (cont…)

```
        PrintWriter pw=new
PrintWriter(cs.getOutputStream(),true);
        System.out.println("Enter $ to quit");
        System.out.print("Enter string: ");
        while((s=kbr.readLine())!=null){
                if(s.equals("$")){
                        return;
                }
                pw.write(s+"\n");
                pw.flush();
```

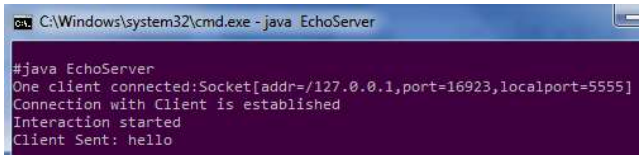


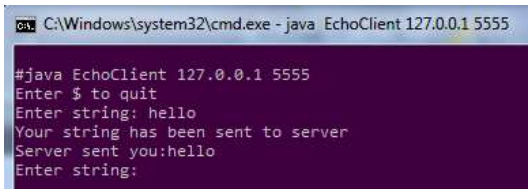
# Echo TCP Client (cont…)

```
                System.out.println("Your string has been sent to
    server");
                System.out.println("Server sent
    you:"+br.readLine());
                System.out.print("Enter string: ");
            }
        }
}
```



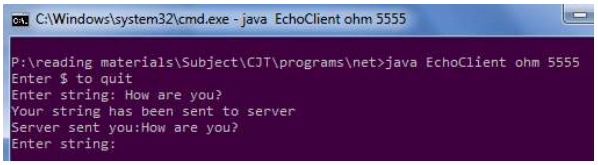
# Running the program

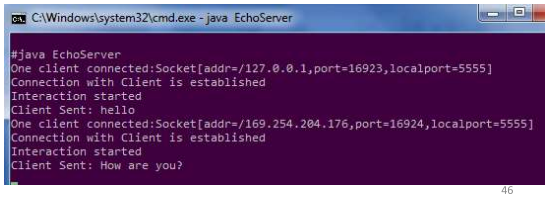





# Running the program






# URL Class

- URL stands for uniform resource locator.
- a standard way of locating resources on the Internet, e.g. http://www.yahoo.com.
- basic parts of a URL
  - protocol name: http, file, mailto, etc.
  - host: www.yahoo.com
  - port: this is an optional attribute specified after the host name, e.g. www.yahoo.com:80
  - file: name of the file to be accessed, e.g. www.yahoo.com/index.html
  - Reference: name of named reference within the page (i.e. cs), e.g. http://www.yahoo.com/index.html#cs



# URL Class: Constructors and methods

- Constructors:
  - URL(String urlSpecifier)
  - URL(String protocolName, String hostName, int port, String path)
  - URL(String protocolName, String hostName, String path)
- Methods:
  - int getPort()
  - String getHost()
  - String getFile()
  - String getRef()

# URLConnection

- It is used to access attributes of a remote resource.
- The attributes are exposed as per the HTTP protocol specification
- URLConnection is valid only for URL objects that use HTTP protocol
- Methods:
  - int getContentLength()
  - long getDate()
  - long getExpiration()
  - long getLastModified()
  - InputStream getInputStream() thorws IOException



# Reading from a URL

```
import java.net.*;
import java.io.*;
class URLExample{
    public static void main(String args[]) {
        try{
                URL u=new URL(args[0]);
    System.out.println("The Protocol used is: "+ u.getProtocol());
    System.out.println("The Host used is: "+u.getHost());
    System.out.println("The File used is: "+u getFile());
    System.out.println("The Port used is: "+u.getPort());
    System.out.println("The Reference in page is: "+u.getRef());
    URLConnection uc=u.openConnection();
    InputStream in= uc.getInputStream();
```



# Reading from a URL

```
                BufferedReader br=new BufferedReader(new
    InputStreamReader(in));
                String x=null;
                while((x=br.readLine())!=null)
                        System.out.println(x);
                br.close();
        }catch(MalformedURLException e){
                System.out.println(e);}
        catch(IOException e){
                System.out.println(e);}
        }
    }
}
```