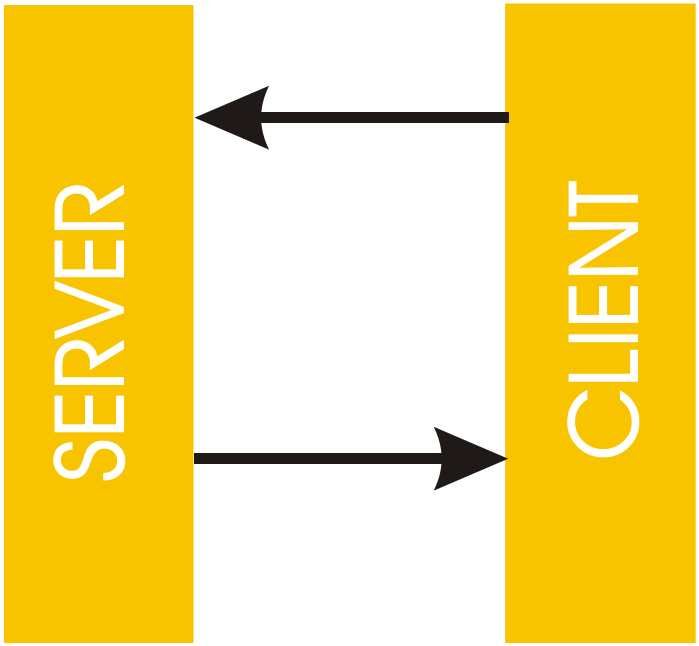
Socket Introduction

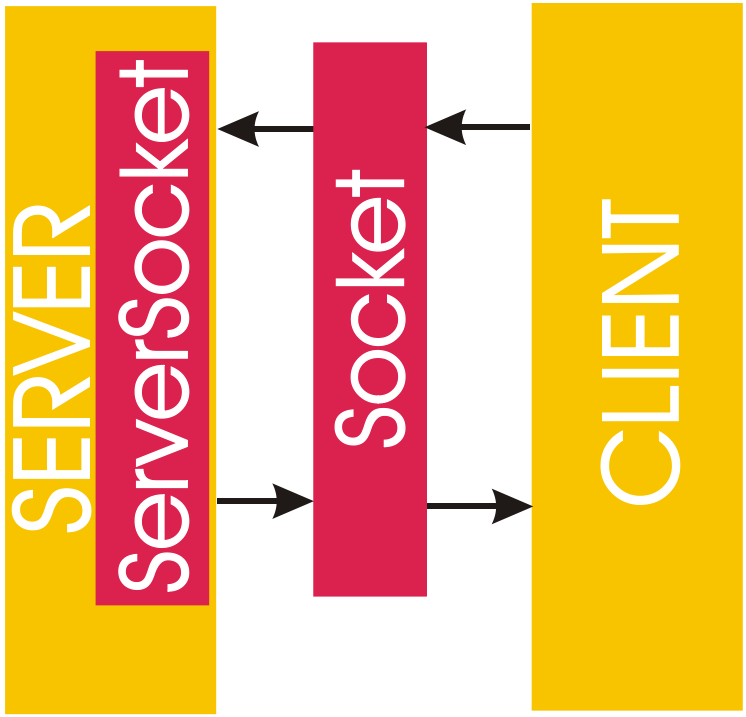
Client Server Programs

- Internet program will consist of two separate Java programs:
  - A server program , which provides information to anyone that wants it
  - A client program, which requests information from the server
- Communication goes back and forth between the server and client over the Internet (TCP/IP in our case)



Java Classes

- Internet Programming support classes are in the java.net package
- Classes of interest for our purposes are:
  - ServerSocket – handles listening for connection requests from a client on a port and provides a socket
  - Socket – provides the input and output streams for communication between the server and client
- These classes hide all the TCP/IP overhead



What is a socket?

- Socket
  - The combination of an IP address and a port number.
  - The name of the Berkeley-derived *application programming interfaces* (APIs) for applications using TCP/IP protocols.
  - Two types
    - Stream socket : reliable two-way connected communication streams
    - Datagram socket
- Socket pair
  - Specified the two end points that uniquely identifies each TCP connection in an internet.
  - 4-tuple: (client IP address, client port number, server IP address, server port number)

- Sockets simply open a connection between two network hosts to pass information between them.

Socket Basics

1)Open a socket.
2)Open an input stream and output stream to the socket.
3)Read from and write to the stream according to the server's protocol.
4)Close the streams&Close the socket.

1)How to Open a Socket?
-Client-

```
try {
    Socket clientSocket = new Socket("localhost", 6789);
}
  catch (IOException e) {
      System.out.println(e);
  }
```

//Socket("Machine name", PortNumber);

1)How to Open a Socket?
-Server-

```
ServerSocket  welcomeSocket  =
                                new   ServerSocket(6789);

Socket connectionSocket = welcomeSocket.accept();
```

When implementing a server you also need to create a socket object from the ServerSocket in order to listen for and accept connections from clients.

2)Open an input stream to the socket. -Client-

- On the client side, you can use the BufferedReader class to create an input stream to receive input from the user:

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

- On the client side, you can use the BufferedReader class to create an input stream to receive input from the user:

```
BufferedReader inFromServer = new BufferedReader(new
    InputStreamReader(clientSocket.getInputStream()));
```

2)Create an Output Stream -Client-

- create an output stream to send information to the server socket using DataOutputStream of java.io
- DataOutputStream outToClient = new DataOutputStream(clientSocket.getOutputStream());

3)Create an Output Stream for writing to server-Client-

- The class DataOutputStream allows you to write Java primitive data types;
- many of its methods write a single Java primitive type to the output stream.
- The method writeBytes is a useful one.

```
sentence = inFromUser.readLine();
outToServer.writeBytes(sentence + '\n');
```

2&3)Open an input stream and output stream writing to client. -server-

```
BufferedReader inFromClient =
new BufferedReader(new  InputStreamReader(
    connectionSocket.getInputStream()));

DataOutputStream outToClient = new
    DataOutputStream(connectionSocket.getOutputStr
    eam());

clientSentence = inFromClient.readLine();
```

4)Close Sockets?
-Client-

- clientSocket.close();
- inFromUser.close();
- outToClient

4)Close Sockets?
-server-

- inFromClient.close();
- outToClient.close();
- welcomeSocket.close();

Serving Multiple Clients

- You can use threads to handle the server's multiple clients simultaneously. Simply create a thread for each connection.

```
while (true)
{
Socket socket = serverSocket.accept();
Thread thread = new ThreadClass(socket);
    thread.start();
}
```