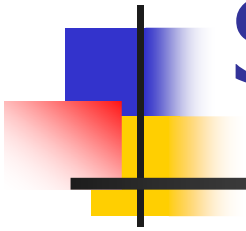
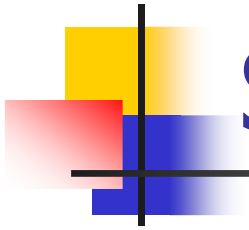


SOCKET

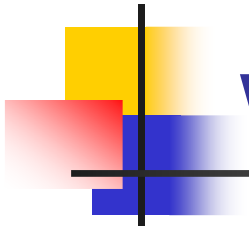
abdul sattar



Socket

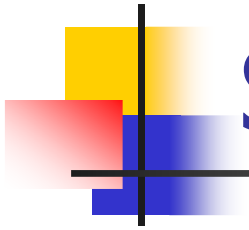
What is a socket?

A host-local, application-oriented, OS controlled interface into which an application process can both *send and receive* messages to and from another application process.



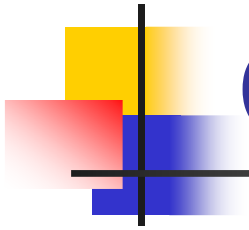
Working with Sockets

You can use sockets to transfer data between unrelated processes that can be running on the same workstation or on different hosts on a network



Socket Programming

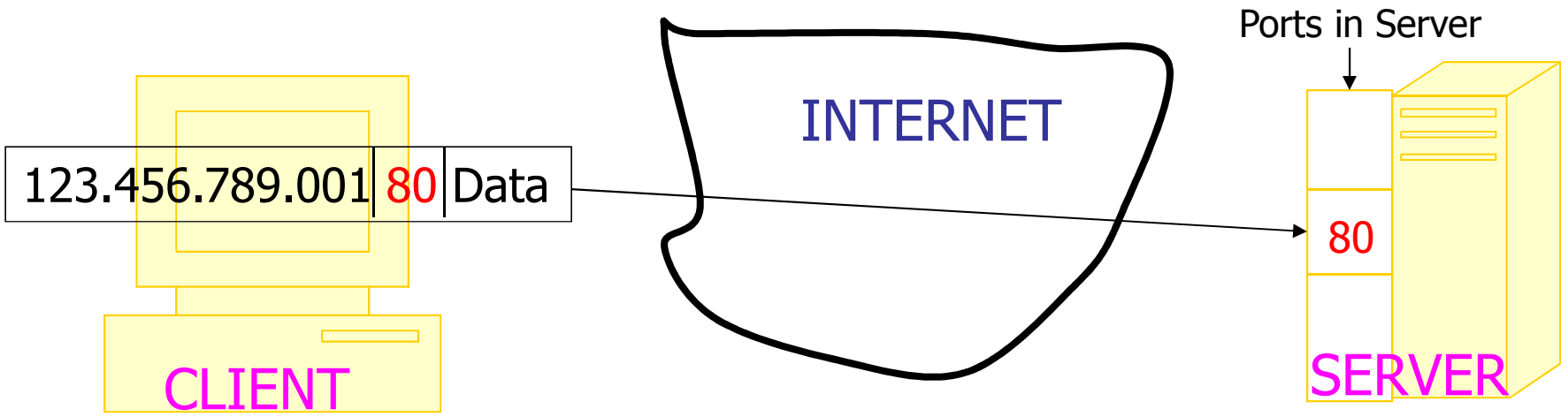
Build Client/Server application that communicate using socket



Client/Server programming

- Client connects to an address:port number
- Server is permanently running, listening to that port
  - server replies to the client
- Client receives the reply
- Differences between servers lie in the kind of request they expect (protocol) and reply they send

Client connects to Server

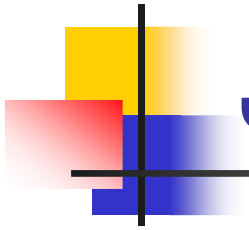


Well Known Ports

port numbers are 16 bit numbers, about 64,000 different ports.
ports 0-256 Internet services,
ports 256-1024 network services

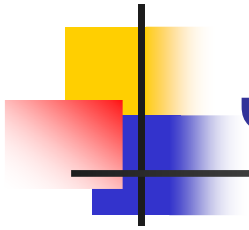
service-name	port/protocol aliases
echo	**7**/tcp
echo	**7**/udp
discard	**9**/tcp sink null
discard	**9**/udp sink null
systat	**11**/tcp users
ftp-data	**20**/tcp
ftp	**21**/tcp
telnet	**23**/tcp
smtp	**25**/tcp mail
www	**80**/tcp http
nntp	**119**/tcp usenet # Network News Transfer
ntp	**123**/tcp # Network Time Protocol

For complete list goto: http://www.networksorcery.com/enp/protocol/ip/ports00000.htm



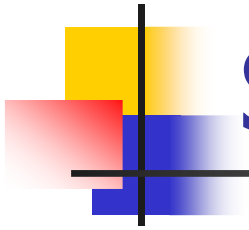
Java: InetAddress Class (1)

- Used to represent IP addresses
- Creators
  - static InetAddress[ ] getAllByName(String host)
    - Returns the list of all addresses for the specified host
  - Static InetAddress getByName(String host)
    - Returns an IP address for the specified host
  - Static InetAddress getLocalHost()
    - Returns an IP address for the local host
- Accessors
  - byte[ ] getAddress()
    - Returns 32-bit IP address



Java: InetAddress Class (2)

- **Accessors (continued)**
  - **String getHostAddress()**
    - **Returns IP address in dot-decimal notation**
  - **String getHostName()**
    - **Returns canonical name of the host**
  - **boolean isMulticastAddress()**
    - **Returns true if the address is a multicast**
    - **address**



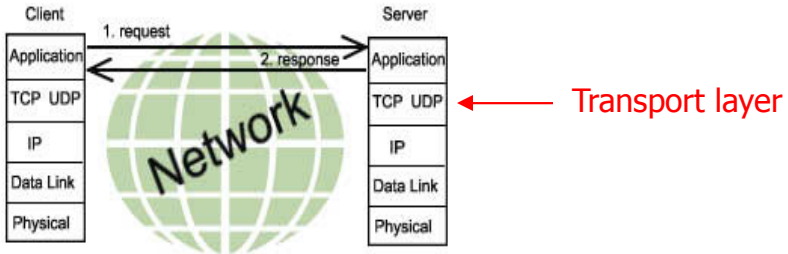
Socket API

- Introduced in BCD4.1 UNIX in 1981.
- Explicitly created, used and released by applications.

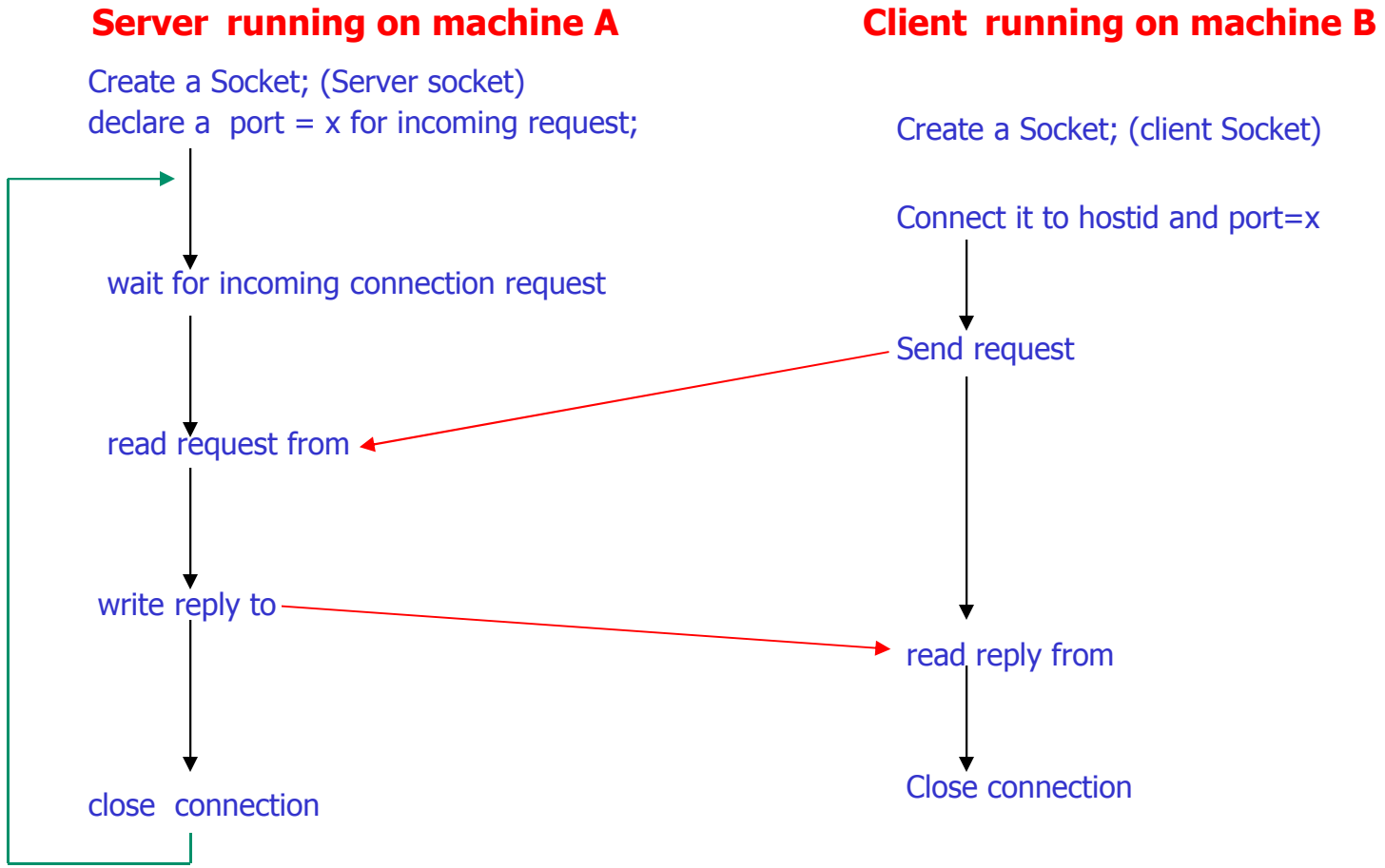
Socket services

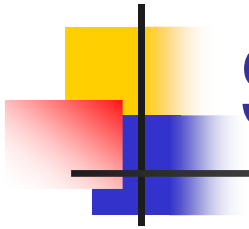
Two types of transport service via socket API

- Unreliable, connection-less datagram(UDP)
- Reliable, connection-oriented byte stream(TCP)



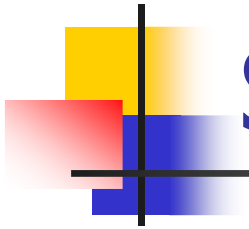
Socket-programming using TCP





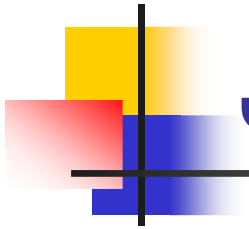
Sockets and OOP

- Motivation for object-oriented programming for network applications
  - Software engineering principles
  - Code reuse, especially through class libraries
  - Hiding programming details in objects
- Sockets classes available in …
  - Microsoft Foundation Classes (MFC)
  - Java class library
  - Other



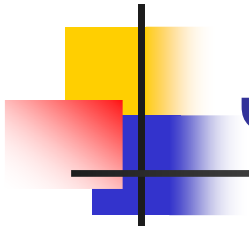
Sockets and Java

- **Java supports high-level abstractions for …**
  - **Network communication**
  - **Internet applications**
  - **Other functions (input/output, conversion, compression, user interface, etc.)**
- **Platform-independent, including operating system and hardware**
  - **Same client runs on multiple hosts using the JVM**
  - **Develop and support only one version**
  - **Client only needs a web browser with Java support**



Java: Socket Class (1)

- Used for TCP sockets
- Constructors
  - Socket(InetAddress remoteAddr, int remotePort)
  - Socket(String remoteHost, int remote Port)
  - Socket(InetAddress remoteAddr, int remotePort, InetAddress localAddr, intlocalPort)
  - Socket(String remoteHost, int remotePort, InetAddress localAddr, int localPort)
- Operators
  - void close()
  - void shutdownInput() – shutdown for receiving
  - void shutdownOutput() – shutdown for sending



Java: Socket Class (2)

- **Accessors/Mutators**
  - **int getPort()**
  - **InputStream getutStream()** // **Returns a stream for reading bytes from the socket**
  - **OutputStream getOutputStream()** // **Returns a stream for writing bytes to the socket**
- **getKeepAlive()**
- **void setKeepAlive(boolean on)**
- **InetAddress getLocalAddress()**
- **int getLocalPort()**

Java: Socket Class (3)

- **Creating a connected socket**

```
Socket socket = new Socket(server, servPort);
```

- **Sending data via a socket**

```
OutputStream out = socket.getOutputStream();
out.write(byteBuffer);
```

- **Receiving data via a socket**

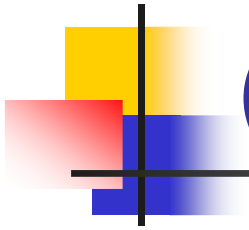
```
InputStream in = socket.getInputStream();

bytesRcvd = in.read(byteBuffer, totalBytesRcvd, byteBuffer.length - totalBytesRcvd
```

- **Closing a socket**

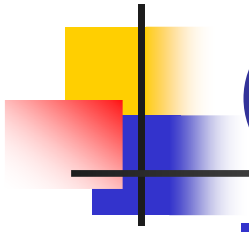
```
socket.close();
```

Java: ServerSocket Class (1)



- Used for (server) TCP sockets
- Constructors
  - ServerSocket(int localPort)
  - ServerSocket(int localPort, int queueLimit)
  - ServerSocket(int localPort, int queueLimit, InetAddress localAddr)
    - Can set the local IP address to limit to a particular interface
- Operators
  - Socket accept()
  - void close()

Java: ServerSocket Class (2)



- **Accessors/Mutators**
  - **InetAddress getInetAddress()**
  - **int getLocalPort()**
  - **int getSoTimeout()**
  - **void setSoTimeout(int timeout)**
- **Creating a socket listening at servPort**

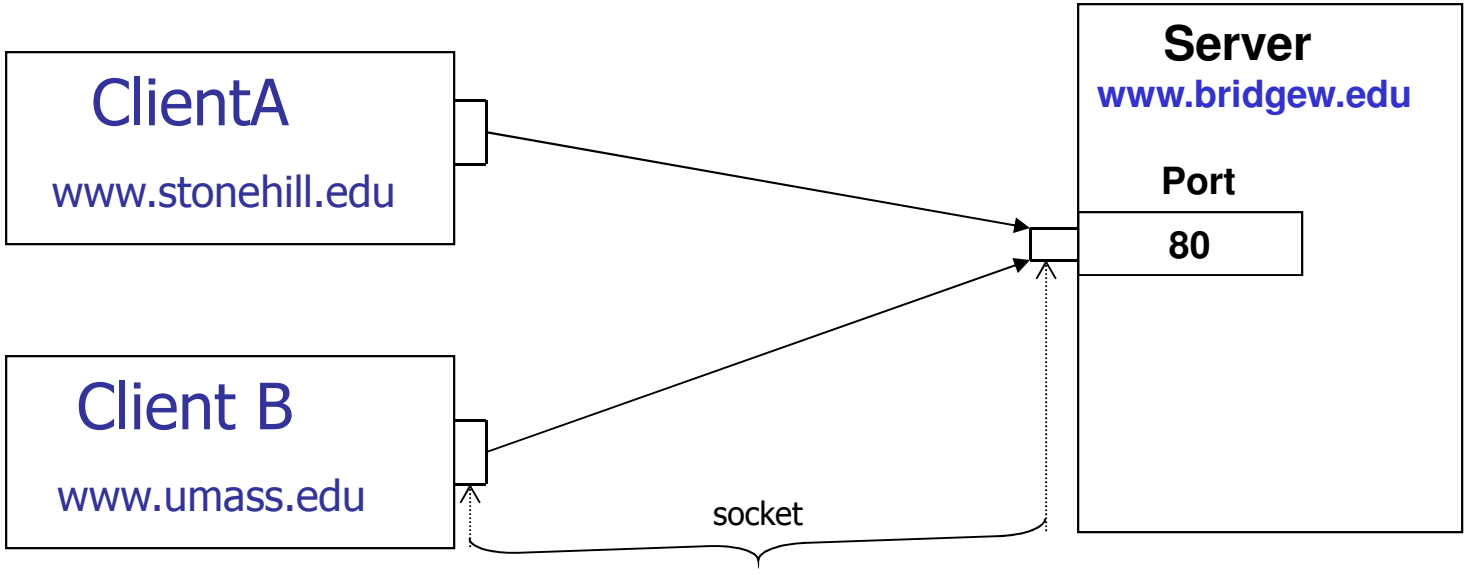
  ServerSocket servSock = new ServerSocket(servPort);
- **Accepting a connection**

  Socket clntSock = servSock.accept();

Java TCP Socket Example

A Server (web server) at *www.bridgew.edu*

- listens to port 80 for Socket Client Connection Requests
- Establish InputStream for sending data to client
- Establish OutputStream for receiving data from client

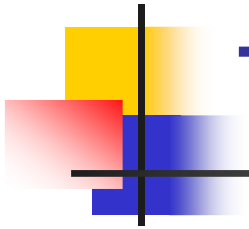


TCP connection example: (Server)

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class myserver {
            public static void main( String [] s) {
                        try {
                          ServerSocket s = new ServerSocket( 2003 );
                          While (true) {
                            // wait for a connection request from client
                            Socket clientConn = s.accept();
                            InputStream in = clientConn.getInputStream();
                            OutputStream out = clientConn.getOutputStream();
                            // communicate with client
                            // ..
                               clientConn.close(); // close client connection
                            }
                         }catch (Exception e){//do something about the exception}                }
            }
}
```

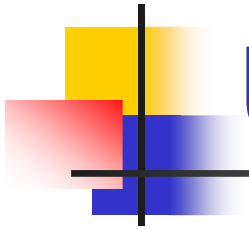
TCP connection example: (Client)

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class myclient {
            public static void main( String [] s) {
                        try {
                          InetAddress addr = InetAddress.getByName(
                                                              “www.bridgew.edu”);
                          Socket s = new Socket(addr, 2003);
                          InputStream in = s.getInputStream();
                          OutputStream out = s.getOutputStream();
                          // communicate with remote process
                          // e.g. GET document /sattar/index.html
                          s.close();
                        } catch(Exception e) {
                          System.out.println(“Exception”);
                          // do something about the Exception
                        }
            }
}
```



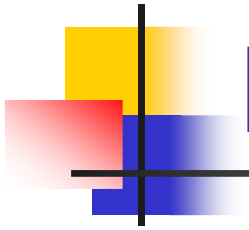
TCP examples

Demo



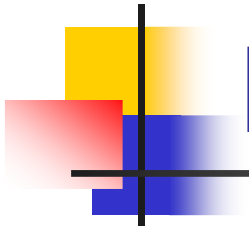
UDP (User Datagram Protocol)

- provides a connectionless service for the transfer of individual datagrams(packets)
- minimizes overhead since no network connection is established prior to a datagram being sent
- useful when application calls for small (~64 kbytes) independent messages
- significant differences
  - No ServerSocket
  - Explicit buffering



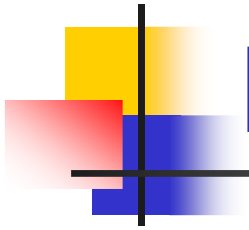
DatagramSocket

- used to both send and receive DatagramPackets
- as with TCP sockets, DatagramSockets must be bound to a particular port number
- Constructors
  - public static DatagramSocket()
  - public DatagramSocket(int port)
  - public DatagramSocket(int port,
    InetAddress iaddr)



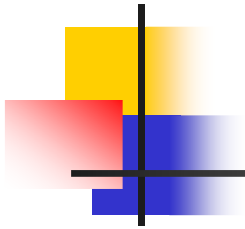
DatagramSocket methods

- void send(DatagramPacket p)
  - sends packet from this socket
  - throws IOException if i/o error occurs
- void receive(DatagramPacket p)
  - receives packet from this socket
  - throws IOException if i/o error occurs
- get and set methods for SoTimeout
  - used get/set socket timeout for receive operation



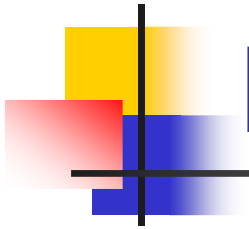
DatagramPacket

- used to implement a connectionless packet delivery service
- Each packet is routed from one machine to another based solely on information contained within that packet
- Multiple packets sent from one machine to another might be routed differently, and might arrive in any order



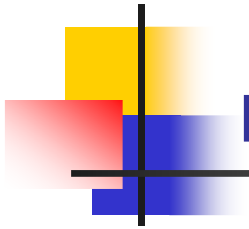
DatagramPacket

- Constructors
  - DatagramPacket(byte[] buf, int length)
    - Constructs a DatagramPacket for receiving packets of length length
  - DatagramPacket(byte[] buf, int length, InetAddress address, int port)
    - Constructs a datagram packet for sending packets of length length to the specified port number on the specified host



DatagramPacket methods

- InetAddress getAddress()
  - returns IP address of packet source (receive packet) or destination (send packet)
- int getPort()
  - returns port of packet source (receive packet) or destination (send packet)
- byte[] get data()
  - returns packet data
- int getLength()
  - returns length of data to be sent or data received
- corresponding set methods

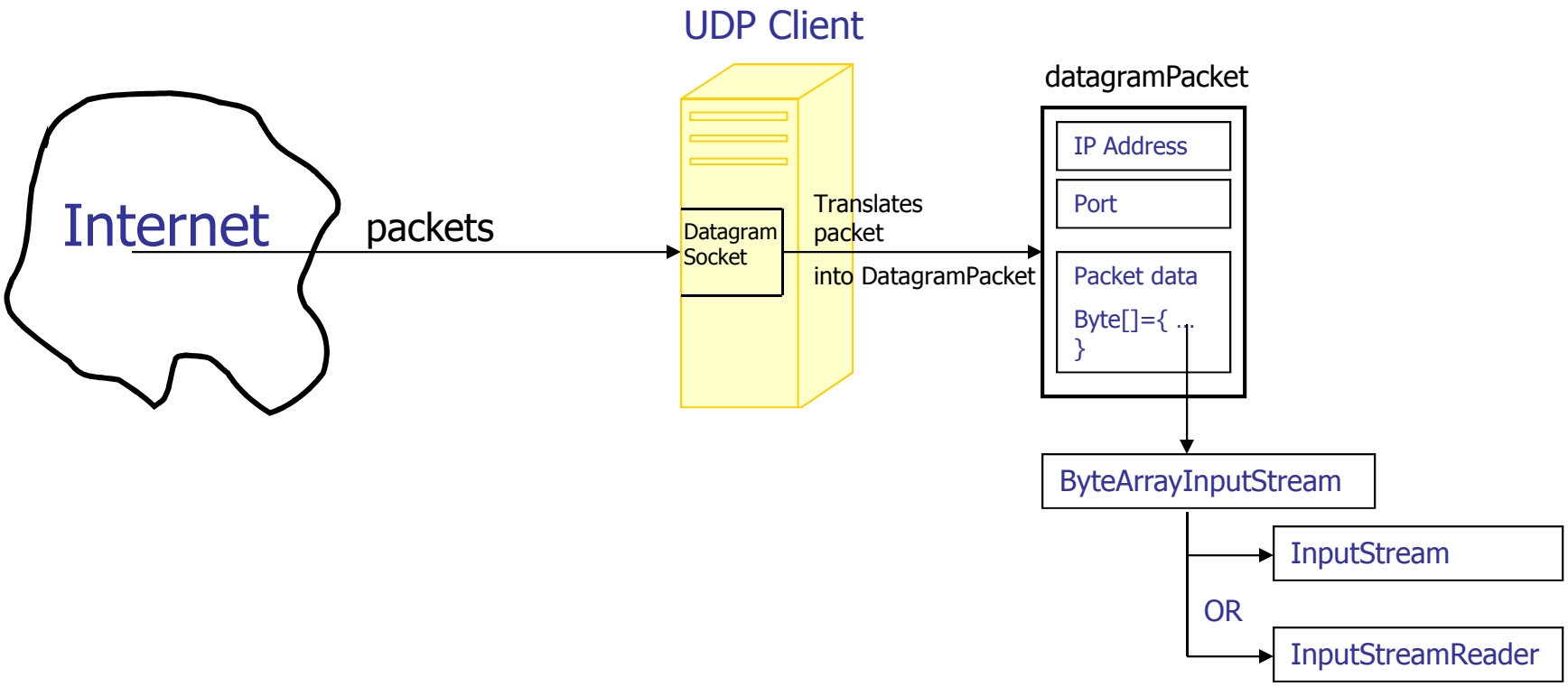


EX: Printing local host information

```
InetAddress address = InetAddress.getLocalHost();
System.out.println("Local Host:");
System.out.println("\t" + address.getHostName());
System.out.println("\t" + address.getHostAddress());
```

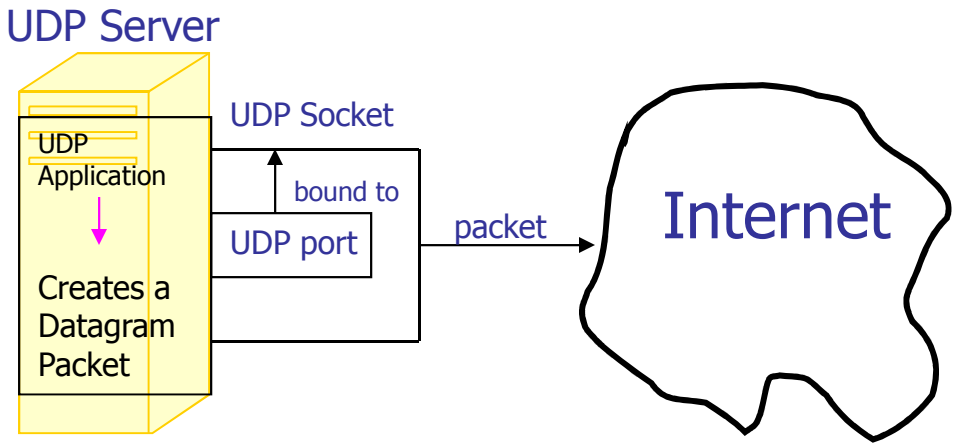
Working with UDP(1)

- Receiving data sent by a remote machine

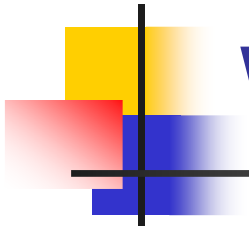


Working with UDP(2)

- Sending data to a remote machine




```
DatagramSocket socket = new DatagramSocket(2003);  //datagram socket bound to port 2003
DatagramPacket packet = new DatagramPacket(new byte[256], 256);
packet.setAddress(InetAddress.getByName( some remote UDP Client);
//write data to packet buffer
Socket.send(packet); //packet out to Internet
```



Working with UDP(3)

- Receiving UDP Packets
  - create DatagramSocket

```
DatagramSocket socket = new DatagramSocket(port);
```

  - construct reception packet

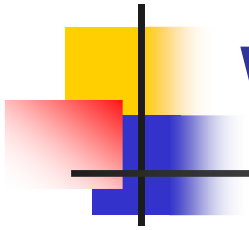
```
byte buffer[] = new byte[256];
DatagramPacket packet = new DatagramPacket(buffer,
                                           buffer.length);
```

  - wait for packet

```
socket.receive(packet);
```

  - close socket when done

```
socket.close();
```



Working with UDP(4)

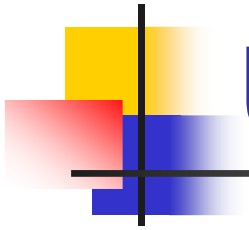
- Sending UDP packets
  - create DatagramSocket

    DatagramSocket socket = new DatagramSocket(2003);
  - construct transmission packet

    DatagramPacket packet =
    new DatagramPacket(buffer,buffer.length);
    packet.setAddress(InetAddress.getByName(somehost));
  - send packet

    socket.send(packet);
  - close socket when done

    socket.close();



UDP examples

Demo

URL class(1)

Let Java handle the details of the communications with a web server.

creating URLs

- **`URL(String spec)`**

  URL CS399 = new URL("http://webhost.bridgew.edu/sattar/");

- **`URL(URL context, String spec);`**

  URL CS399Lectures = new URL(CS399, "lecture.html");

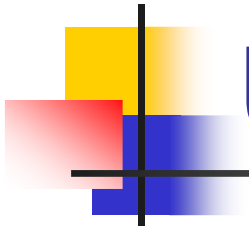
- **`URL(String protocol, String host, String file);`**

  URL CS399 = new URL("http","webhost.bridgew.edu", "/sattar/index.html");

- **`URL(String protocol, String host, int port, String file);`**

  URL CS399 = new URL("http","webhost.bridgew.edu", 80 , "/sattar/index.html");

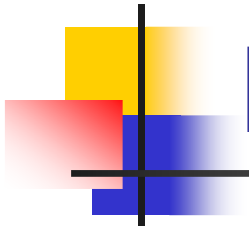
All URL constructors throw MalformedURLException



URL class(2)

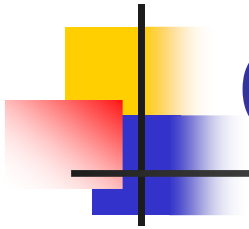
The URL class provides several methods that let you query URL objects

- String getProtocol() - returns protocol
- String getHost() - returns host
- int getPort() - returns port
- String getFile() - returns filename
- String getRef() - returns anchor



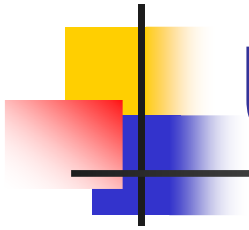
Reading from a URL

- You can call URL's openStream() method to get a stream from which you can read the contents of the `URL`
- The openStream() method returns a java.io.InputStream object, so reading from a URL is as easy as reading from an input stream



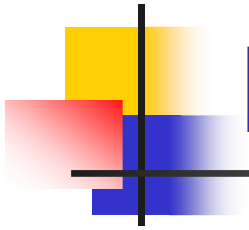
Connecting to a URL

- You can call URL's openConnection() method to open a TCP connection to the URL
- The openConnection() method returns a URLConnection object, which provides the ability to read from and write to a URL



URLConnection class

- abstract superclass of all classes that represent a TCP connection between an application and a URL
- Instances of this class can be used both to read from and to write to the resource referenced by the URL
- subclasses must implement connect() method



Building a simple Web server

- Handles one HTTP request
- Accepts the request
- Parses header
- Obtains requested file from server's file system
- Creates HTTP response message: (header lines + file)
- Sends response to client