Programming in Java

Dr. Nyein Aye Maung Maung
Dr. Eng (Ritsumeikan University, Japan)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Week	Topics
Week 1	Overview of JAVA, Data Types, Variables and Arrays
Week 2	Operators and Control Statements
Week 3	Classes and A closer look at Methods and Classes
Week 4	Inheritance, Polymorphism, Abstraction and Encapsulation
Week 5	Packages and Interfaces
Week 6	Exception Handling and Multi-threaded Programming
Week 7	String Handling
Week 8	Exploring java.lang and More utilities classes
Week 9	Java Collections Framework
Week 10	Java I/O
Week 11	Basic Graphical User Interface
Week 12	Event Handling
Week 13	Database Programming
Week 14	Applet and Networking

Lecture 14

Applet and Network Programming

Outline of Class

- Java Applet
- Networking / Socket Programming

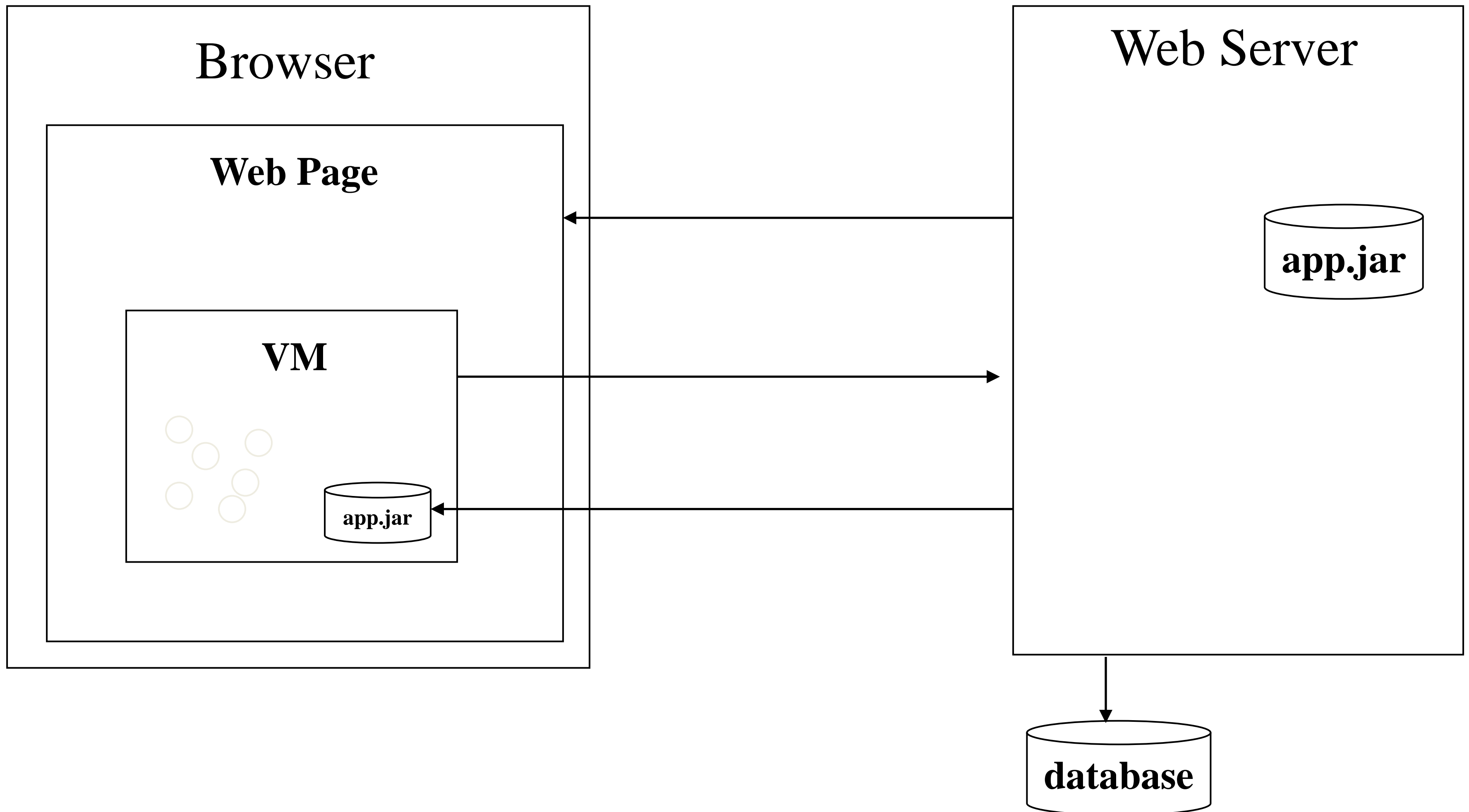
Topic 1

Applet

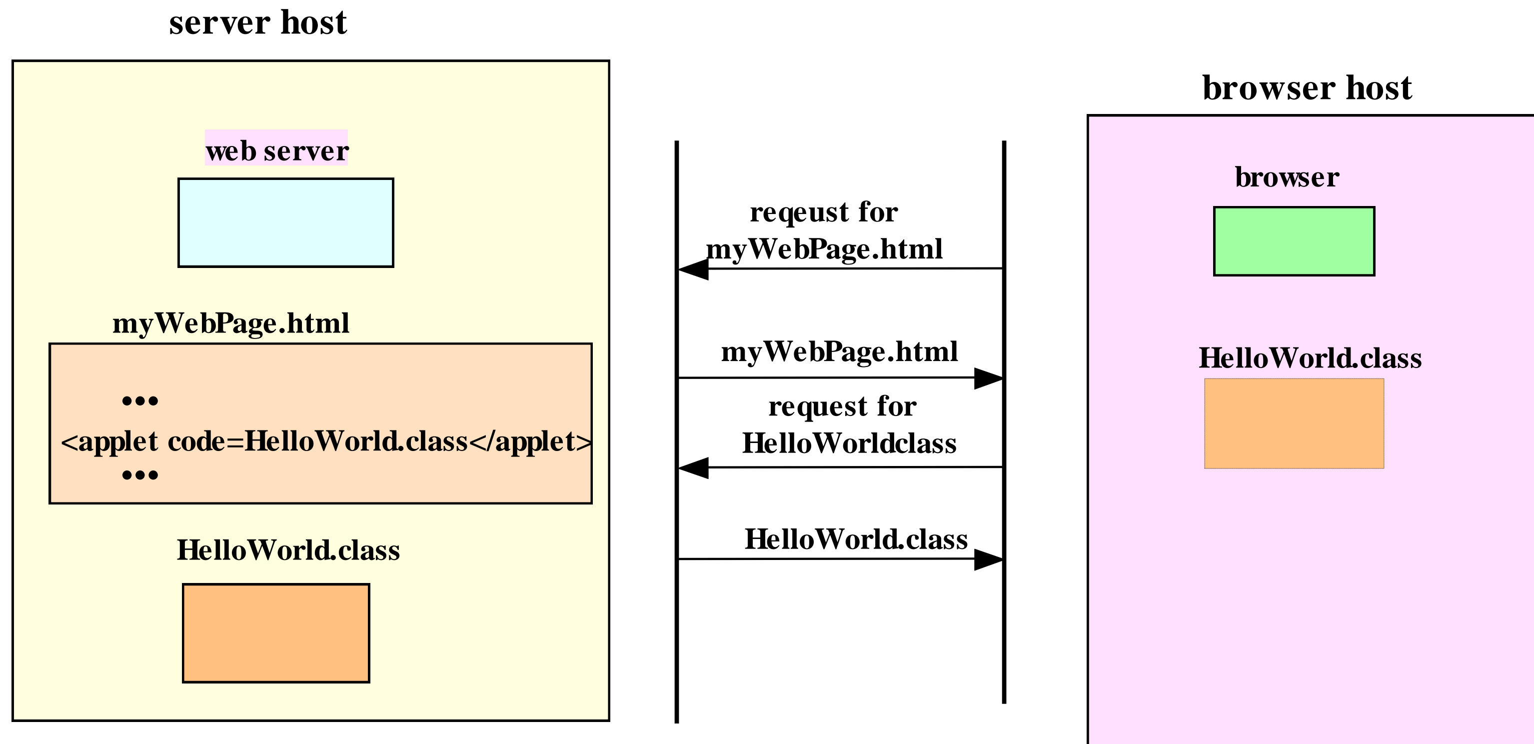
What is an Applet?

- Applet is a special Java program that can be embedded in HTML documents and run within a Web browser, or executed by applet viewer during the development.
- It is automatically executed by (applet-enabled) web browsers.
- In Java, non-applet programs are called a*pplications*.

Applet Architecture



- Applets are programs stored on a web server, similar to web pages.
- When an applet is referred to in a web page that has been fetched and processed by a browser, the browser generates a request to fetch (or download) the applet program, then executes the program in the browser's execution context, on the client host.



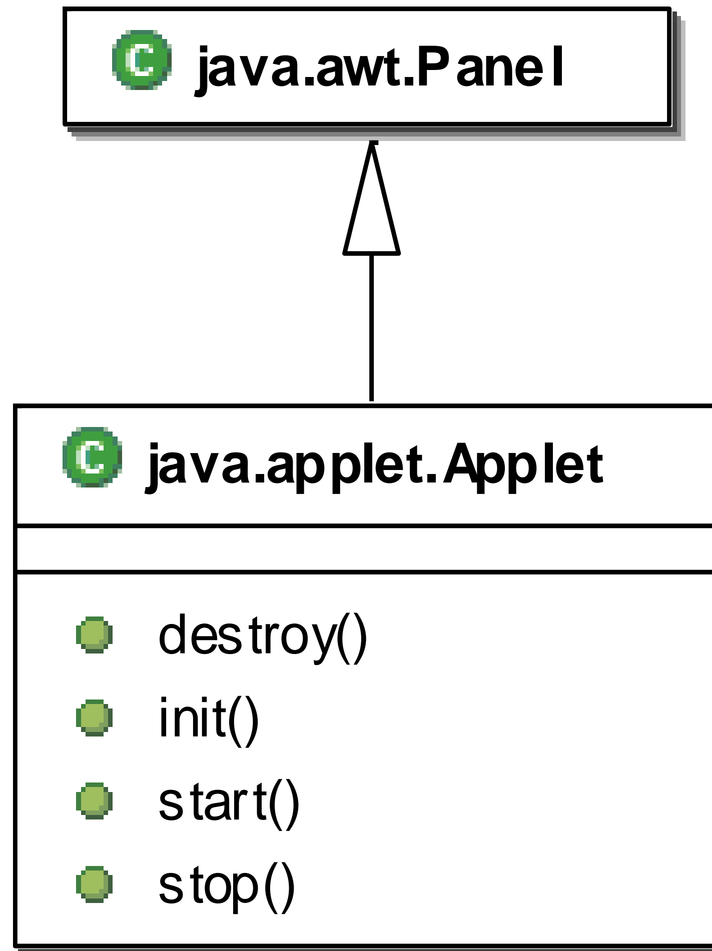
How Applets Differ from Applications (Application vs. Applet)

- Application
  - Trusted (i.e., has full access to system resources)
  - Invoked by Java Virtual Machine (JVM, `java`), e.g.,

    `java HelloWorld`
  - Should contain a main method, i.e.,

    `public static void main(String[])`
- Applet
  - Not trusted (i.e., has limited access to system resource to prevent security breaches)
  - Invoked automatically by the web browser
  - Should be a subclass of class `java.applet.Applet`

The Class Applet



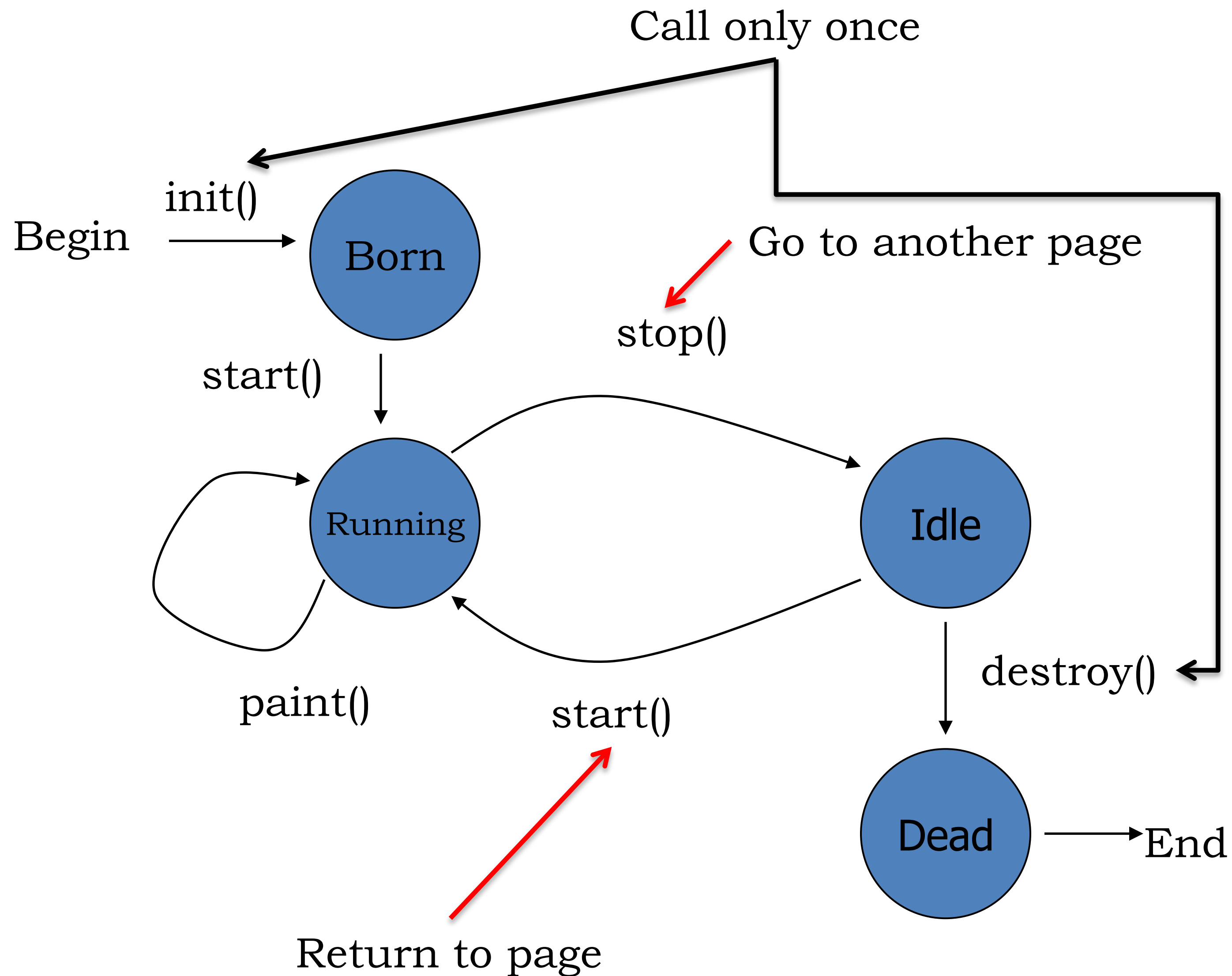
Applet methods

- public void init ()
- public void start ()
- public void stop ()
- public void destroy ()
- public void paint (Graphics)

Also:

- public void repaint()
- public void update (Graphics)
- public void showStatus(String)
- public String getParameter(String)

Applet Life Cycle Diagram

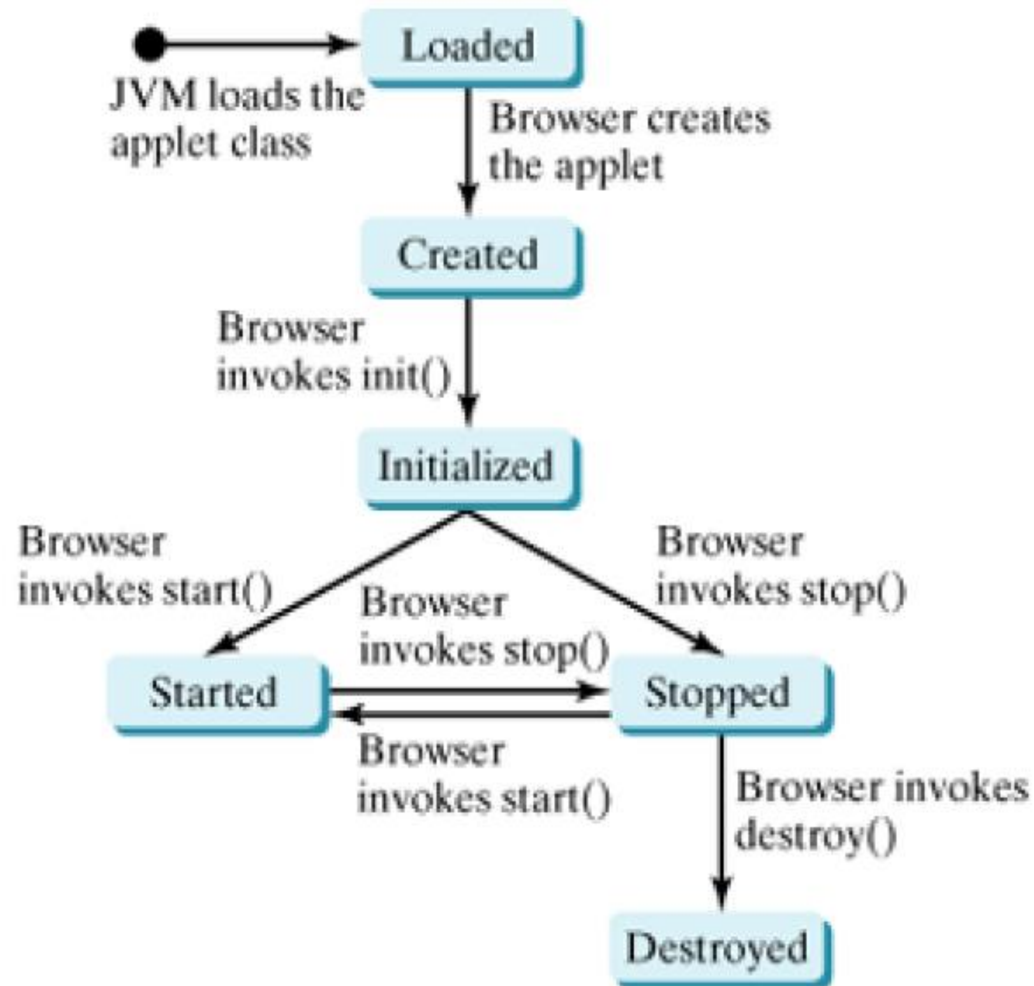


- **init()**
  - Called exactly once in an applet's life.
  - Called when applet is first loaded, which is after object creation, e.g., when the browser visits the web page for the first time.
  - Used to read applet parameters, start downloading any other images or media files, etc.
  - Is the best place to define the GUI Components (buttons, text fields, checkboxes, etc.), lay them out, and add listeners to them
- **start()**
  - Called at least once.
  - Called right after init( )
  - Called each time the page is loaded and restarted
  - Called ach time the user returns to the Web page containing the applet after surfing other pages

- **stop( )**
    - Called when the user leaves the page
- **destroy()**
  - Called exactly once.
  - Called when the browser unloads the applet.
  - Used to perform any final clean-up.

Browser Calling Applet Methods

- JVM loads the applet class
- Browser creates the applet.
- Browser invokes init()
- Browser invokes start() when a page is loaded and restarted.
- Browser invokes stop() when the browser released the page.
- Browser invokes destroy() to release system resources.



JApplet Class

- Necessary to create a Java applet that extends javax.swing.JApplet

```
// WelcomeApplet.java: Applet for displaying a message
   import javax.swing.*;
   public class WelcomeApplet extends JApplet {
   /** Initialize the applet */
   public void init() {
         add(new JLabel("Welcome to Java", JLabel.CENTER));
         }
   }
```

or public WelcomeApplet ()

Applets in HTML

- To put an applet into an HTML page, you use the <applet> tag, which has three required attributes, code (the class file) and width and height (in pixels)
- Example:

```
<applet  code="WelcomeApplet.class"  width=150  height=100>
 </applet>
```

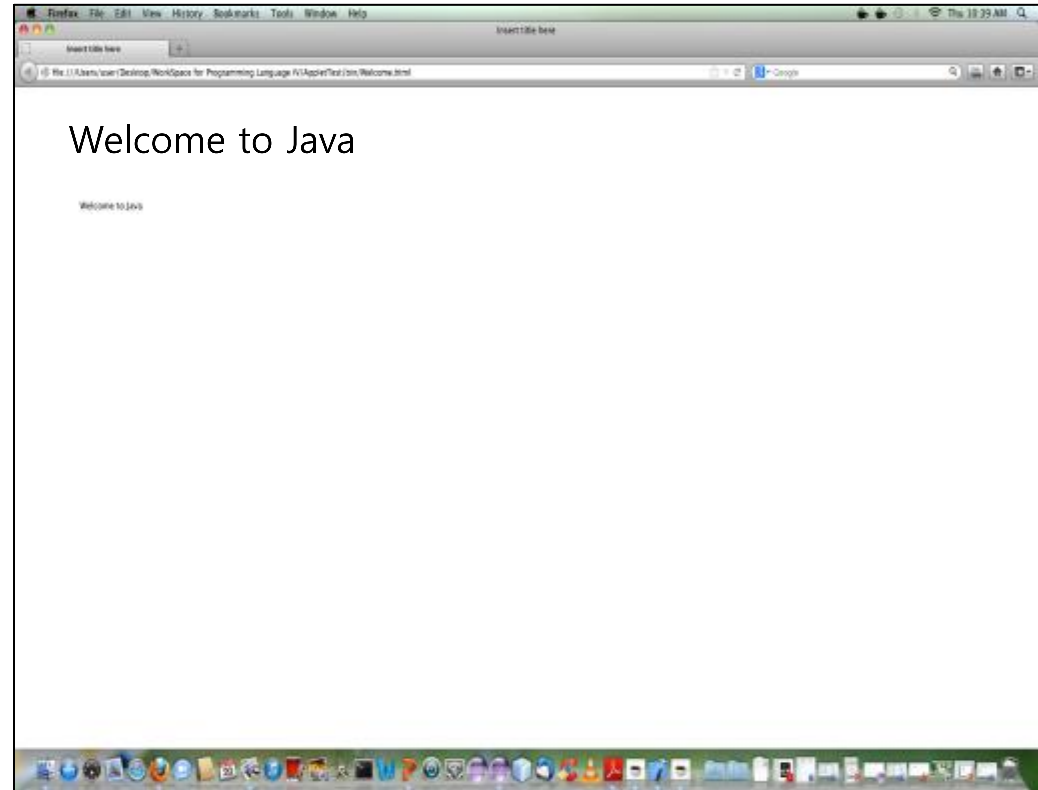
- If your applet contains several classes, you should jar them up; in this case you also need the archive attribute:

```
<applet  code="WelcomeApplet.class"
        archive="WelcomeApplet.jar"
          width=150   height=100>
</applet>
```

Applet in HTML

```
<html>
  <head>
    <title> Welcome Applet </title>
  </head>

  <body>
    <applet code="WelcomeApplet.class" width=300 height=150>
    </applet>
  </body>
</html>
```

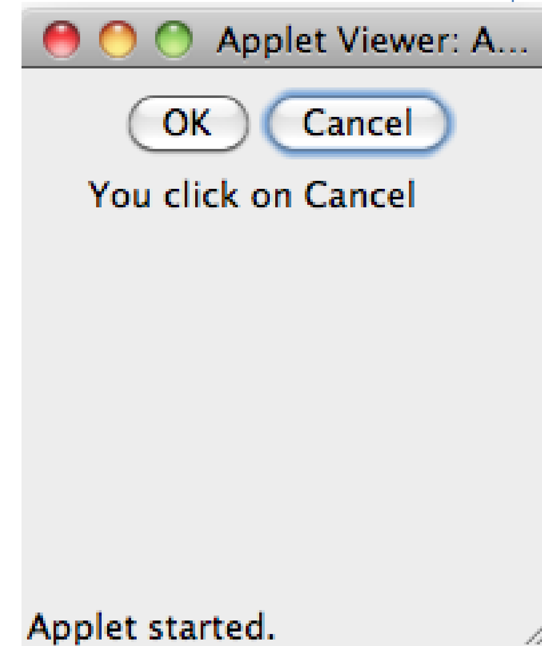
Test Applet Via Applet Viewer



Test Applet Via Web Browser

Example 1: Simple Applet with Events Demo.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class AwtAppletTest extends Applet implements ActionListener
{
        private Button okButton = new Button("OK");
        private Button cancelButton = new Button("Cancel");
        private Label status=new Label();
        public void init()
        {
                okButton.addActionListener(this);
                cancelButton.addActionListener(this);
                add(okButton);
                add(cancelButton);
                status.setText("Please click on a button");
                add(status);
        }
        public void actionPerformed(ActionEvent x)
        {
                String strcmd= x.getActionCommand();
                status.setText("You click on "+strcmd);
        }
}
```



Applets with parameters

- You can pass parameters to your applet from your HTML page:

```
<applet  code="MyApplet.class"   width=150   height=100>
    <param name="message"  value="Hello World">
    <param name="arraySize"  value="10">
</applet>
```

- The applet can retrieve the parameters like this:

```
String message = getParameter("message");
String sizeAsString = getParameter("arraySize");
```

- All parameters are passed as Strings, so you may need to do some conversions:

```
try { size = Integer.parseInt (sizeAsString) }
catch (NumberFormatException e) {…}
```

```
<html>
<head>
<title>Passing Strings to Java Applets</title>
</head>
<body>
<p>This applet gets a message from the HTML
page and displays it.</p>
<applet code = “WelcomeApplet.class” width = 200  height =
50>
<param name = MESSAGE value = "Welcome to Java" />
<param name = X value = 20 />
<param name = Y value = 30 />
</applet>
</body>
</html>
```

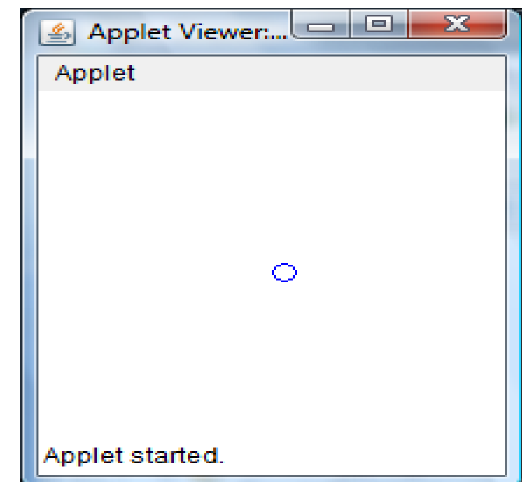
```
import javax.swing.*;
import java.applet.*;
public class WelcomeApplet2 extends JApplet {

public void init() {
        String message = getParameter("MESSAGE");
        int x = Integer.parseInt(getParameter("X"));
        int y = Integer.parseInt(getParameter("Y"));
        setSize(x, y);
        add(new JLabel(message));
}
}
```

Graphical Applet

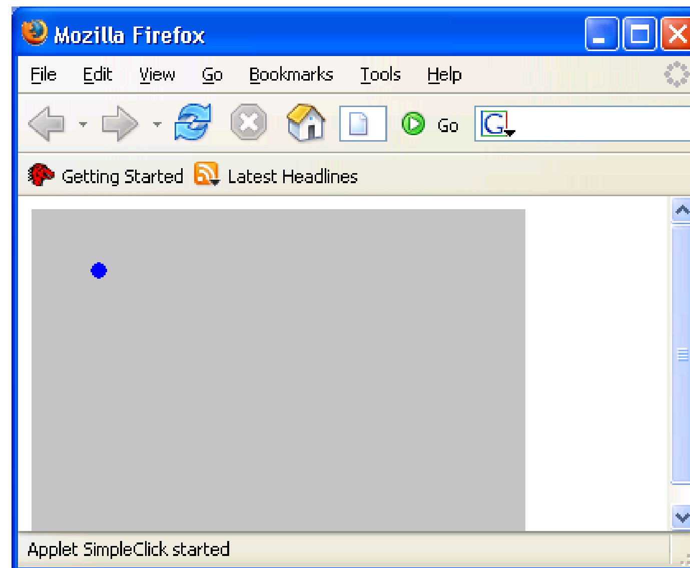
Use Graphics class to draw image

- Paint method draws image of Applet
  - Use graphics object to draw image
  - **g** parameter represents the applet image
  - Use commands on image to draw blue circle

```
public class SimpleClick extends Applet {
    public void paint(Graphics g) {
        g.setColor(Color.blue);
        g.drawOval(100, 100, 10, 10);
    }
}
```



```
public class SimpleClick extends Applet {
    public void paint(Graphics g) {
        g.setColor(Color.blue);
        g.fillOval(100, 100, 10, 10);
    }
}
```



repaint( )

- Call repaint( ) when you have changed something and want your changes to show up on the screen
  – You do *not* need to call repaint() when something in Java's own components (Buttons, TextFields, etc.)
  – You *do* need to call repaint() after drawing commands (drawRect(...), fillRect(...), drawString(...), etc.)
- repaint( ) is a *request*--it might not happen
- When you call repaint( ), Java schedules a call to update(Graphics *g*)

update( )

- When you call repaint( ), Java schedules a call to update(Graphics g)
- Here's what update does:

```
public void update(Graphics g) {
    // Fills applet with background color, then
    paint(g);
}
```

Example 2: Create an applet that displays a message twenty times at random places and with random color.

```
import java.awt.*;
import java.applet.*;
import java.util.Random;
public class HelloWorld2 extends Applet
{   private static final int NUM_WORDS=20;
    private static final Color[] colors =
    {   Color.black,Color.red,Color.blue,Color.green,Color.yellow};
    private static Random randy;
    private int randomInRange(int low, int high)
    {
            return Math.abs(randy.nextInt(high-low+1))+low;}
    public void init()
    {
            randy = new Random();
    }
```

```
public void paint(Graphics g)
    {    for (int i = 0; i < NUM_WORDS; i++) {
            int x = randomInRange(1, 200);
            int y = randomInRange(1, 200);
            g.setColor(colors[randomInRange(0,colors.length-1)]);
            g.drawString("Hello world!", x, y);
        }
    }
}
```

Developing Applets with GUI

- Every Java GUI program you have developed can be converted into an applet by replacing
  - **JFrame with JApplet** and deleting the main method.

```
import javax.swing.*;

public class DisplayLabel extends JFrame {
  public DisplayLabel() {
    add(new JLabel("Great!", JLabel.CENTER));
  }

  public static void main(String[] args) {
    JFrame frame = new DisplayLabel();
    frame.setTitle("DisplayLabel");
    frame.setSize(200, 100);
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

(a) GUI application

```
import javax.swing.*;
                                         JApplet
public class DisplayLabel extends JFrame {
  public DisplayLabel() {
    add(new JLabel("Great!", JLabel.CENTER));
  }

  public static void main(String[] args) {
    JFrame frame = new DisplayLabel();
    frame.setTitle("DisplayLabel");
    frame.setSize(200, 100);
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

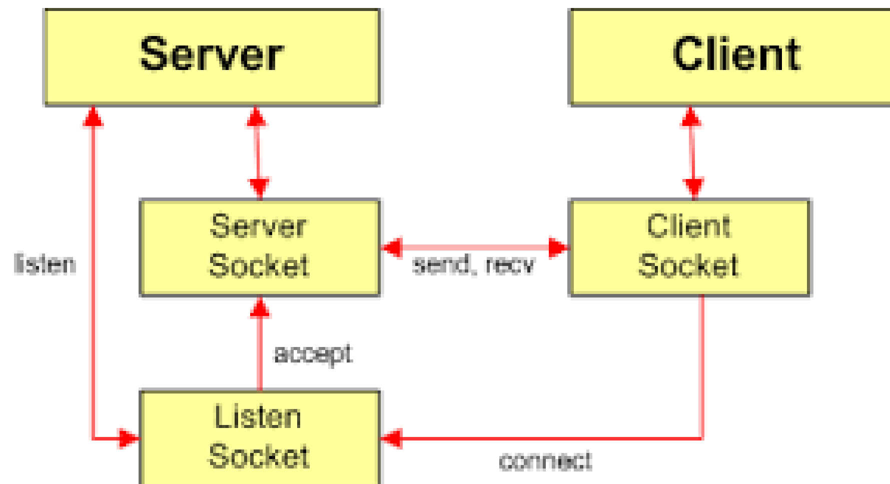
(b) Applet

Topic 2

Socket Programming

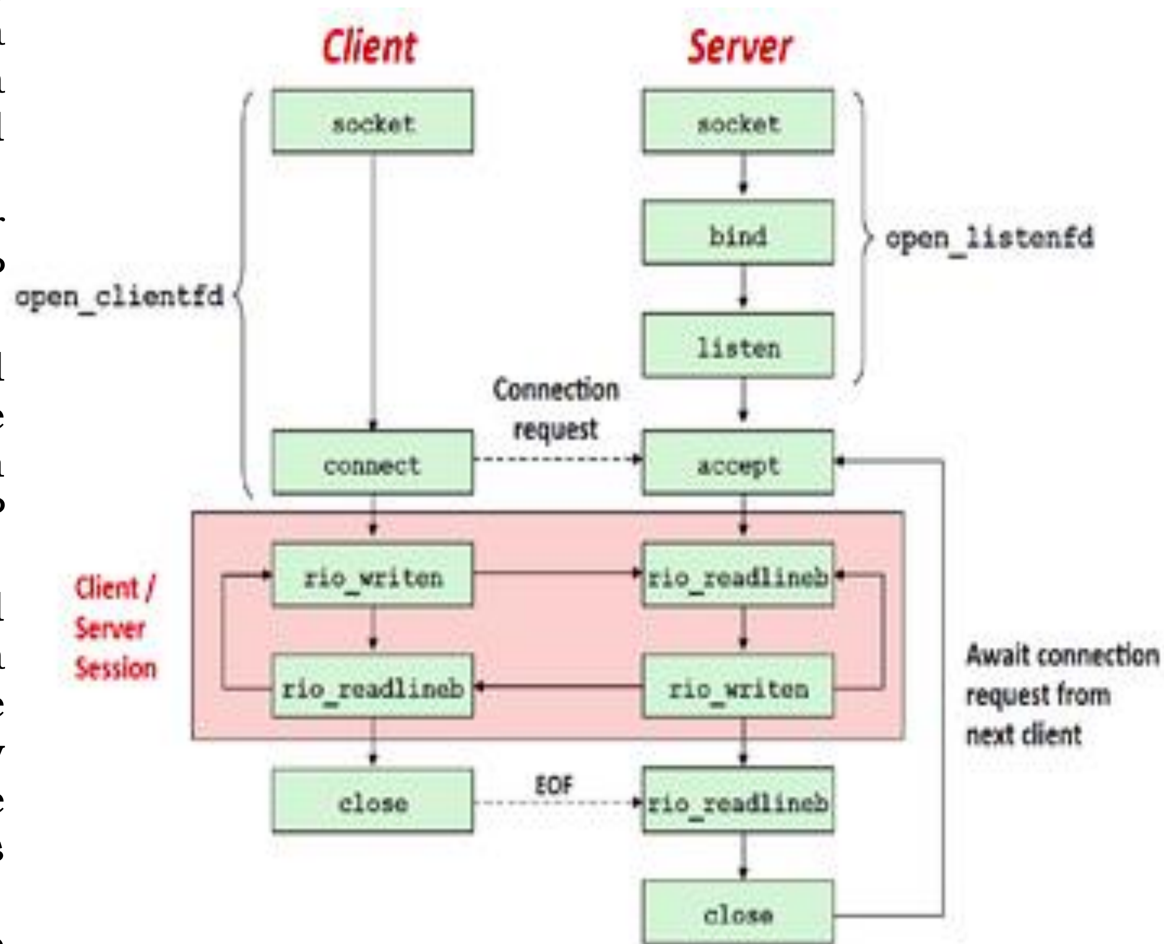
Socket

- Identify an endpoint in a network
- Allow a single computer to serve many different clients at once, as well as to serve many different types of information
- Client applications get a port and a socket on the client machine when they connect successfully with a server.
- Active vs Passive Sockets
  - A server uses a passive socket to wait the client connections.
  - A client uses an active socket to initiate a connection.



Socket Interface

- socket() creates a new socket
- bind() is used on the server side, and associates a socket with a socket address structure, i.e. a specified local port number and IP address.
- listen() is used on the server side, and causes a bound TCP socket to enter listening state.
- connect() assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.
- accept() accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection.
- close() causes the system to release resources allocated to a socket.



Port

- Numbered socket on a particular machine
- Associated with an IP address of a host and the protocol type of the communication
- Is a 16-bit unsigned integer, thus ranging from 0 to 65535.
  - 1~1024 are reserved by TCP/IP

    20 & 21: File Transfer Protocol (FTP)

    22: Secure Shell (SSH)

    23: Telnet remote login service

    25: Simple Mail Transfer Protocol (SMTP)

    53: Domain Name System (DNS) service

    80: Hypertext Transfer Protocol (HTTP) used in the World Wide Web

    110: Post Office Protocol (POP3)

    143: Internet Message Access Protocol (IMAP)

    161: Simple Network Management Protocol (SNMP)

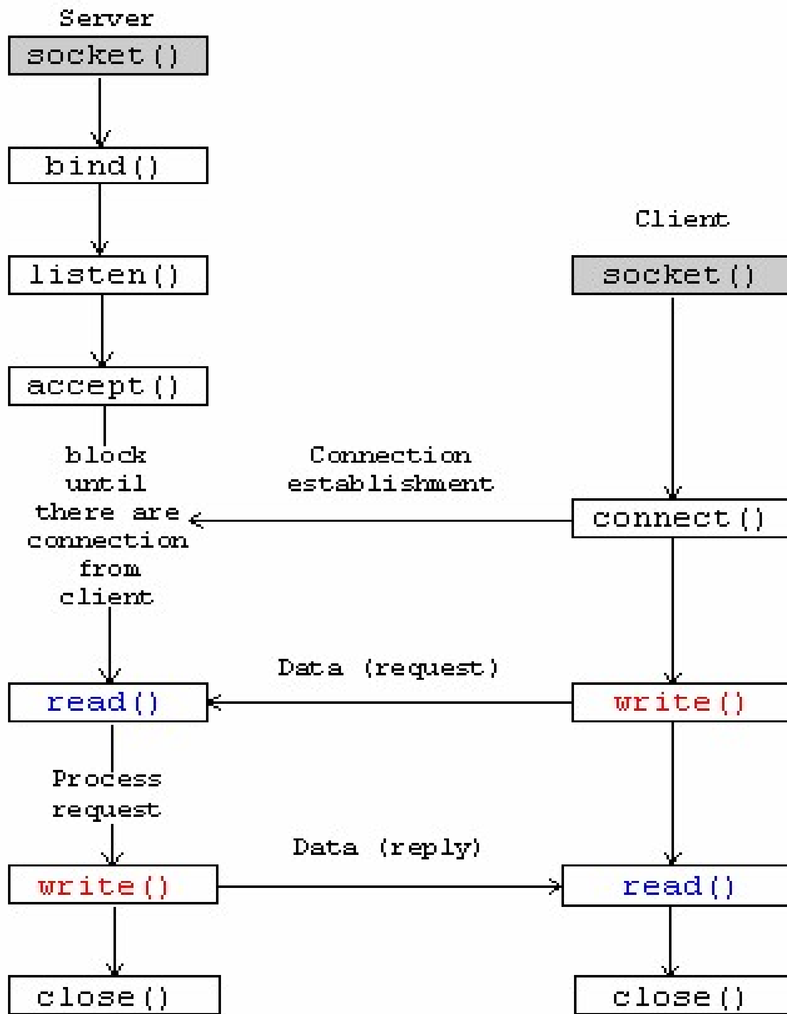
    194: Internet Relay Chat (IRC)

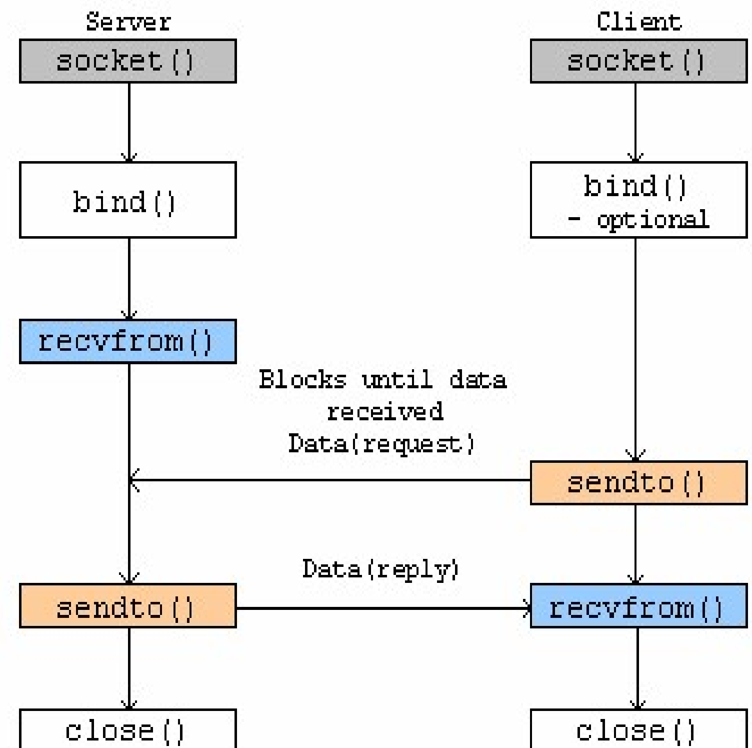
    443: HTTP Secure (HTTPS)
- Multiple clients can be communicating with a server on a given port. Each client connection is assigned a separate socket on that port.

Communication Protocols

- TCP/IP (Transfer Control Protocol)
  - Stream communication
  - A connection-oriented protocol
  - A connection must first be established between the pair of socket
  - Lost data is re-transmitted.
  - Data is delivered in-order.
  - While one of the sockets listens for a connection request (server), the other asks for a connection (client)
  - Once two sockets have been connected, they can be used to transmit data in both (or either one of the) directions.

- UDP (User Datagram Protocol)
  - Datagram communication
  - A connectionless protocol
  - Some packets may be lost and some packets may arrive out of order.
  - An unreliable protocol which is much faster and less overhead





(a)

(b)

Client/server relationship of the socket APIs for (a) connection-oriented protocol (TCP) and connectionless protocol (UDP)

Address

- Every computer on the Internet has Address
- Internet address uniquely identifies each computer on the Net
- IPv4 (Internet Protocol version 4)
  - 32 bits (four 8 bits values)
- IPv6 (Internet Protocol version 6)
  - 64 bits (four 16 bits values)
  - much larger address space
- The name of an Internet address is called its Domain Name.

java.net.InetAddress class

- Static construction using a factory method
  - `InetAddress` **`getByName`**`(String hostName)`
    - Determines the IP address of a host, given the host's name.
      - `hostName` can be “host.domain.com.au”
  - `InetAddress` **`getLocalHost()`**
    - Returns the address of the local host.
- Some useful methods:
  - `String` **`getHostName`**`()`
    - Gives you the host name (for example “www.sun.com”)
  - `String` **`getHostAddress`**`()`
    - Gives you the address (for example “192.18.97.241”)

TCP Sockets

- **Client Sockets (java.net.Socket)**
  - Implements **client sockets** (also called just "sockets").
  - An endpoint for communication between two machines.
  - Constructor and Methods

    Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.

    InputStream getInputStream()

    OutputStream getOutputStream()

    close()

- **Server Sockets (java.net.ServerSocket)**
  - Implements **server sockets**.
  - Waits for requests to come in over the network.
  - Performs some operation based on the request.
  - Constructor and Methods

    ServerSocket(int port)

    Socket Accept(): Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.

The java.net.Socket class

- The java.net.Socket class allows you to create socket objects that perform all four fundamental socket operations.
  - Can connect to remote machines; can send data; can receive data; can close the connection.
- Each Socket object is associated with **exactly one remote hos**t. To connect to a different host, a new Socket object must be create.
- Cannot just connect to any port on any host. The remote host must actually be listening for connections on that port.
- Can use the constructors to determine which ports on a host are listening for connections.

The java.net.ServerSocket Class

- A ServerSocket object is constructed on a particular local port. Then it calls **accept() to listen** for incoming connections.
- accept() blocks until a connection is detected. Then accept() returns a java.net.Socket object that performs the actual communication with the client.
- If another server socket is already listening to the port, then a java.net.BindException, a subclass of IOException, is thrown.
- No more than one process or thread can listen to a particular port at a time. This includes non-Java processes or threads.
- For example, if there's already an HTTP server running on port 80, you won't be able to bind to port 80.

Opening Socket

Client →

public Socket(String remotehost, int port) throws UnknownHostException, IOException

public Socket(InetAddress address, int port) throws IOException

Socket MyClient;

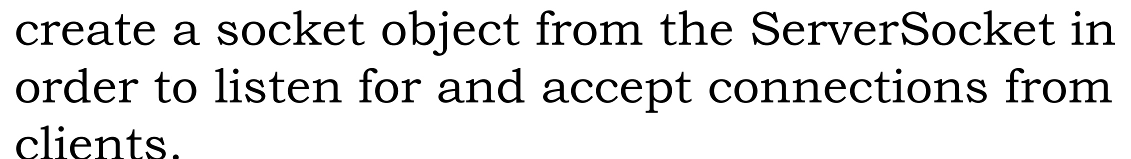
MyClient = new Socket(“hostname", PortNumber);

Server →

ServerSocket MyServer;

MyServer = new ServerSocket(PortNumber)

Socket clientSocket = MyServer.accept();



Sending and Receiving Data on Sockets

- At the server and client classes
- For receiving →

```
BufferedReader in =
        new BufferedReader(
          new InputStreamReader(
            socket.getInputStream()));
```

- For sending →

```
PrintWriter out =
        new PrintWriter(
          new BufferedWriter(
            new OutputStreamWriter(
              socket.getOutputStream())),true);
```

Sending and Receiving Data on Sockets

- From then on, it's like reading and writing any other I/O stream!

```
while (true) {
        String str = in.readLine();
        if (str.equals("END")) break;
        System.out.println("Echoing: " + str);
        out.println(str);
    }
```

Example 3: A TCP server is running on port 9090. When a client connects, it sends the client the current date and time; then closes the connection with that client. Write this simple server side program.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;

public class DateServer {
public static void main(String[] args) throws IOException {
        ServerSocket listener = new ServerSocket(9090);
        try {
            while (true) {
                Socket socket = listener.accept();
                try {
                    PrintWriter out =
                new PrintWriter(socket.getOutputStream(), true);
                    out.println(new Date().toString());
                } finally {
                    socket.close();
                }
            }
        }
        finally {
            listener.close();
        }
    }
}
```

Example 4: Write a TCP server program that reads data on port 10007 and capitalizes and sends received data back to the client until it gets 'Bye.' Message from client.

```
import java.net.*;
import java.io.*;
public class EchoServer
{
 public static void main(String[] args) throws IOException
   {
    ServerSocket serverSocket = null;
    try {
         serverSocket = new ServerSocket(10007);
        }
    catch (IOException e)
        {
         System.err.println("Could not listen on port: 10007.");
         System.exit(1);
        }
    Socket clientSocket = null;
    System.out.println ("Waiting for connection.....");
    try {
         clientSocket = serverSocket.accept();
        }
    catch (IOException e)
        {
         System.err.println("Accept failed.");
         System.exit(1);
        }
    System.out.println ("Connection successful");
    System.out.println ("Waiting for input.....");
```

```
    PrintWriter out = new
    PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(new
    InputStreamReader( clientSocket.getInputStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null)
        {
         System.out.println ("Server: " +
inputLine.toUpperCase());
         out.println(inputLine);
         if (inputLine.equals("Bye."))
             break;
        }
    out.close();


    in.close();
    clientSocket.close();
    serverSocket.close();
   }
}
```

Example 5: Write a client program that sends data to server port 10007 and displays the response back from the server.

```
import java.io.*;
import java.net.*;
public class echoClient {
    public static void main(String[] args) throws IOException
{
        String serverHostname = new String ("127.0.0.1");
        if (args.length > 0)
           serverHostname = args[0];
        System.out.println ("Attemping to connect to host " +
           serverHostname + " on port 10007.");
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            // echoSocket = new Socket("taranis", 7);
            echoSocket = new Socket(serverHostname, 10007);
            out = new PrintWriter(echoSocket.getOutputStream(),
true);
            in = new BufferedReader(new InputStreamReader(
```

```
echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: " +
serverHostname);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for "
                               + "the connection to: " +
serverHostname);
            System.exit(1);
        }
     BufferedReader stdIn = new BufferedReader(
                                   new
InputStreamReader(System.in));
     String userInput;

     System.out.print ("input: ");
   while ((userInput = stdIn.readLine()) != null) {
       out.println(userInput);
       System.out.println("echo: " + in.readLine());
         System.out.print ("input: ");
   }
   out.close();
   in.close();
   stdIn.close();
   echoSocket.close();
  }
}
```

The UDP Classes

- Java's support for UDP is contained in two classes:

  `java.net.DatagramSocket`

  `java.net.DatagramPacket`

- A datagram socket is used to send and receive datagram packets.
- A wrapper for an array of bytes from which data will be sent or into which data will be received.
- Also contains the address and port to which the packet will be sent.

java.net.DatagramPacket

- A wrapper for an array of bytes from which data will be sent or into which data will be received.
- Also contains the address and port to which the packet will be sent.
- A DatagramSocket object is a local connection to a port that does the sending and receiving.
- There is no distinction between a UDP socket and a UDP server socket.
- Also unlike TCP sockets, a DatagramSocket can send to multiple, different addresses.
- The address to which data goes is stored in the packet, not in the socket.

DatagramPacket Constructors

For receiving →

public DatagramPacket(byte[] data, int length)

For sending →

public DatagramPacket(byte[] data, int length,
InetAddress iaddr, int iport)

Sending UDP Datagrams

- To send data to a particular server
  - Convert the data into byte array.
  - Pass this byte array, the length of the data in the array (most of the time this will be the length of the array) and the InetAddress and port to which you wish to send it into the DatagramPacket() constructor.
  - Next create a DatagramSocket and pass the packet to its send() method

- For Example,

```
byte[] data = “This is the message”.getBytes();
DatagramPacket  packet =new DatagramPacket(data, data.length);

// Create an address
InetAddress destAddress
=InetAddress.getByName(“fred.domain.com”);
packet.setAddress(destAddress);
packet.setPort(9876);

DatagramSocket socket = new DatagramSocket();
socket.send(packet);
```

Receiving UDP Datagrams

- Construct a DatagramSocket object on the port on which you want to listen.

- Pass DatagramPacket object to the DatagramSocket's receive() method.
  - public synchronized void receive(DatagramPacket dp) throws IOException

- The calling thread blocks until a datagram is received.

example

```
try {
  byte buffer = new byte[65536];
  DatagramPacket incoming = new DatagramPacket(buffer,
  buffer.length);
  DatagramSocket ds = new DatagramSocket(2134);
  ds.receive(incoming);
  byte[] data = incoming.getData();
  String s = new String(data, 0, data.getLength());
  System.out.println("Port " + incoming.getPort() + " on " +
  incoming.getAddress() + " sent this message:");
  System.out.println(s);
}
catch (IOException e) {
  System.err.println(e);
}
```

Example 6: Write a UDP client program that sends data to server port 9876 and displays the response back from the server.

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

    class UDPClient {
        public static void main(String args[]) throws Exception
        {

          Scanner inFromUser=new Scanner(System.in);
        System.out.println("Enter msg to send to remote host");
        DatagramSocket clientSocket = new DatagramSocket();

          InetAddress IPAddress =
    InetAddress.getByName("hostname");

          byte[] sendData = new byte[1024];
          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());

 System.out.println("FROM SERVER:" + modifiedSentence);

 clientSocket.close();

    }
}
```

Example 6: Write a UDP srever program that listens on port 9876, capitalizes the incoming data and sends the modified packet back to the client .

```
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  = new byte[1024];

          while(true)
            {

              DatagramPacket receivePacket =
                 new DatagramPacket(receiveData, receiveData.length);

              serverSocket.receive(receivePacket);

              String sentence = new String(receivePacket.getData());
```

```
    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);

   }
  }
}
```

Thank you very much …