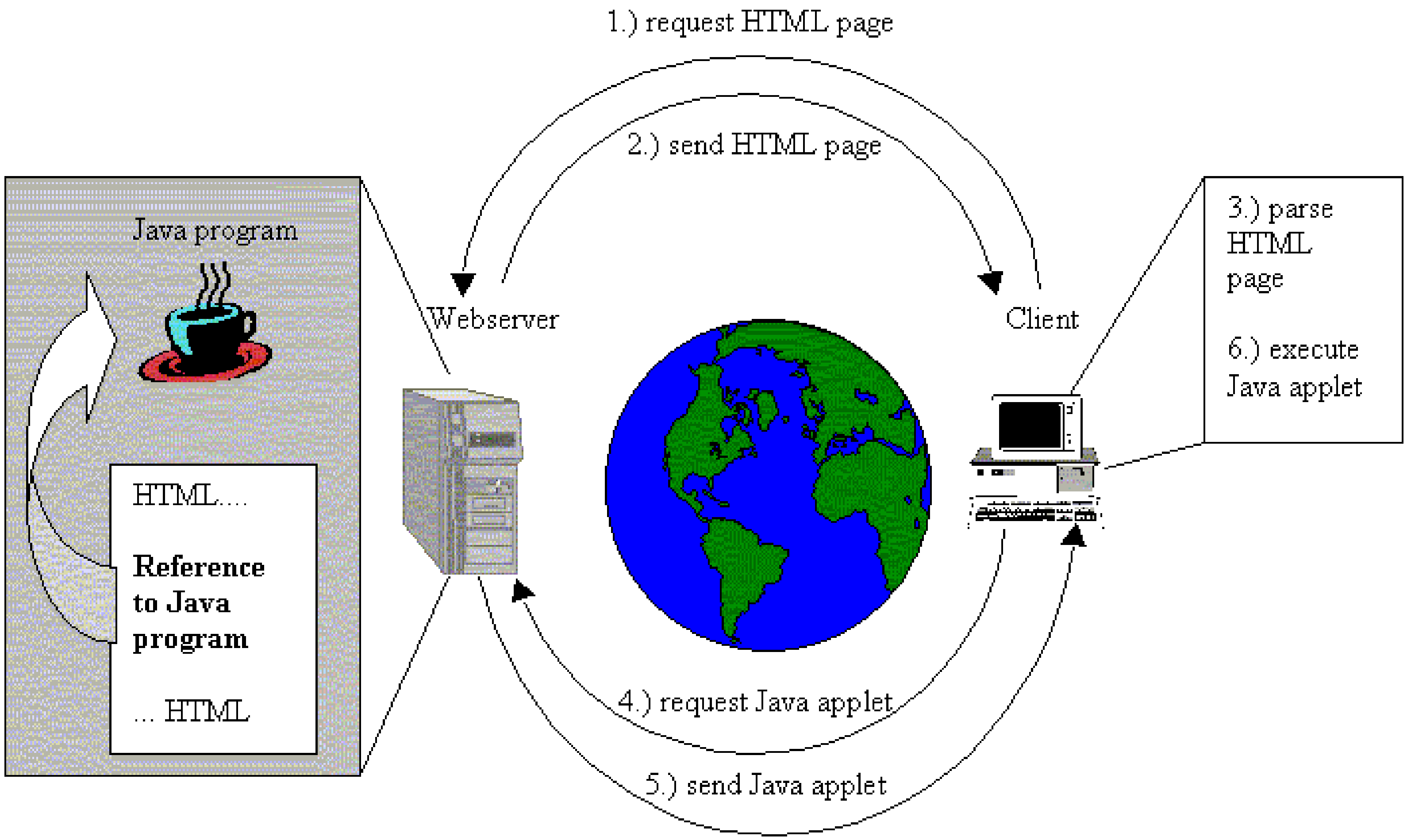
Unit-6

Applets

What is an applet?

- a small Java program that can be inserted into a web page and run by loading that page in a browser.
- An applet, like any application program, can do many things.
- It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and so on.
- This is the feature of Java that is primarily responsible for its initial popularity.
- Users can run applets simply by visiting a web page that contains an applet program.

Applet execution by a browser



Applet classes in Java

- Implementation
  - a top-level container, like a Frame.
  - behaves more like a Panel.
  - It does not contain a title bar, menu bar, or border.
  - This is why you don't see these items when an applet is run inside a browser.
  - When you run an applet using an applet viewer, the applet viewer provides the title and border.
  - `java.applet.Applet`

How Applets Differ from Applications

- Although both the Applets and stand-alone applications are Java programs, there are certain restrictions are imposed on Applets due to security concerns:
  - Applets don’t use the main() method, but when they are loaded, automatically call certain methods (init, start, paint, stop, destroy).
  - They are embedded inside a web page and executed in browsers.
  - Takes input through Graphical User Interface(GUI).
  - They cannot read from or write to the files on local computer.
  - They cannot run any programs from the local computer.
  - They are restricted from using libraries from other languages.
- The above restrictions ensures that an Applet cannot do any damage to the local system.

Building Applet Code: An Example

```
import java.awt.*;
import java.applet.Applet;
public class SimpleApplet extends Applet {
        public void paint(Graphics g) {
                g.drawString ("A Simple Applet",20, 20);
        }
}
```

- Begins with two import classes.
  - java.awt.* -- required for GUI
  - java.applet.* -- every applet you create must be a subclass of Applet, which is in java.applet package.
  - The class should start with public, because it is accessed from outside.

- Applets do not begin execution at **main().**

- An applet begins its execution when the name of its class is passed to an applet viewer or to a network browser.

- Compile the applet in the same way that we have been compiling programs.

- Running an applet involves a different process.

Running an Applet

1. Using web browser
2. Using appletviewer

Executing in a web browser.

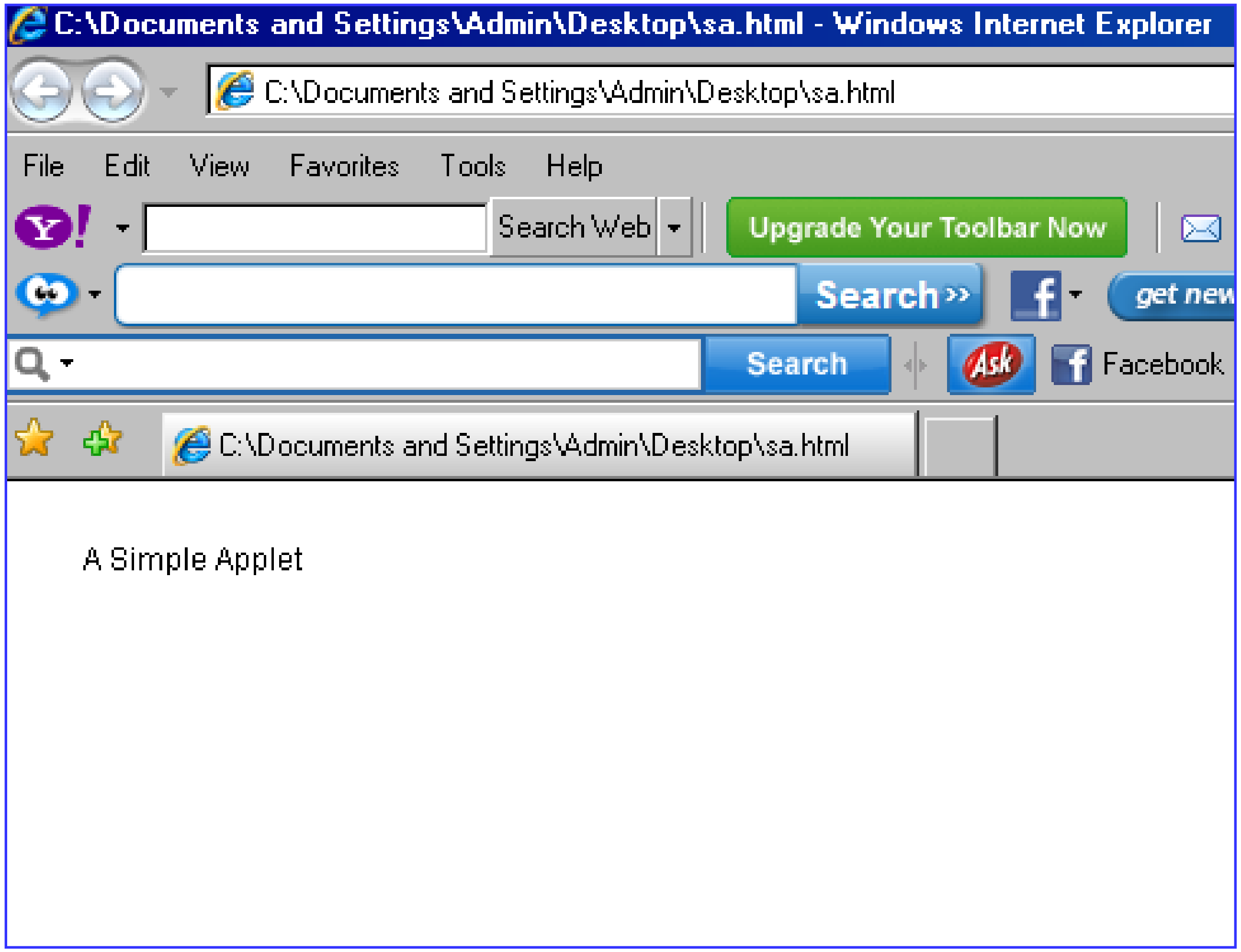
- To execute an applet in a web browser, you need to write a short HTML file that contains a tag (**Applet**) that loads the applet.

HTML file that contains a SimpleApplet

```
<APPLET code="SimpleApplet"
    width=400 height=300> </APPLET>
```

- Save this file with .html extension
- After you create this file, open your browser and then load this file, which causes SimpleApplet to be executed.
- Width and height specify the dimensions of the display used by the applet.

Output:


C:\Documents and Settings\Admin\Desktop\sa.html - Windows Internet Explorer
C:\Documents and Settings\Admin\Desktop\sa.html
File Edit View Favorites Tools Help
Search Web
Upgrade Your Toolbar Now
Search »
get new
Search
Facebook
C:\Documents and Settings\Admin\Desktop\sa.html

A Simple Applet

Executing by using appletviewer

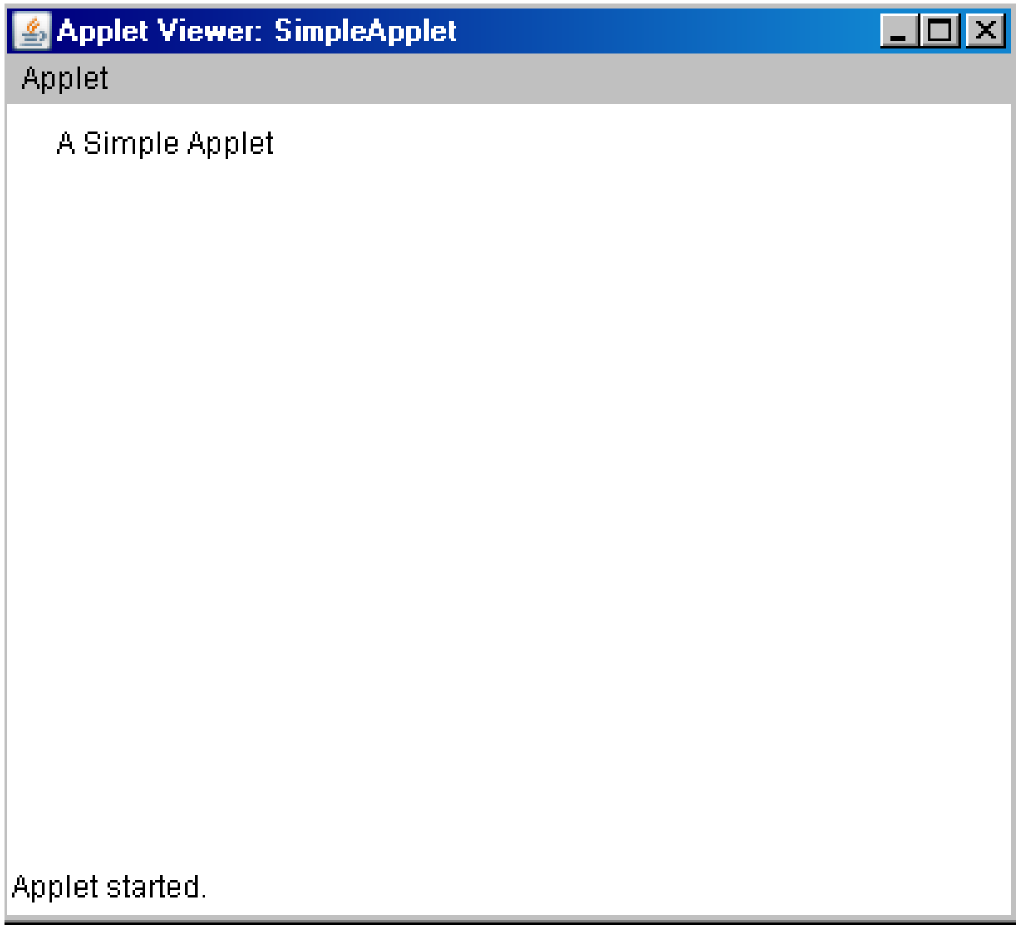
- appletviewer. An applet viewer executes your applet in a window.
- This is generally the fastest and easiest way to test your applet.

- There are two ways

1. Use earlier html page, which contains applet tag, then execute by using following command.

   - C:\>appletviewer SimpleApplet.html

Output:



```
A Simple Applet
```

```
Applet started.
```

2. Include a comment at the beginning of your source code file that contains the applet tag, then start applet viewer with your java source code file.

- C:\>appletviewer SimpleApplet.java

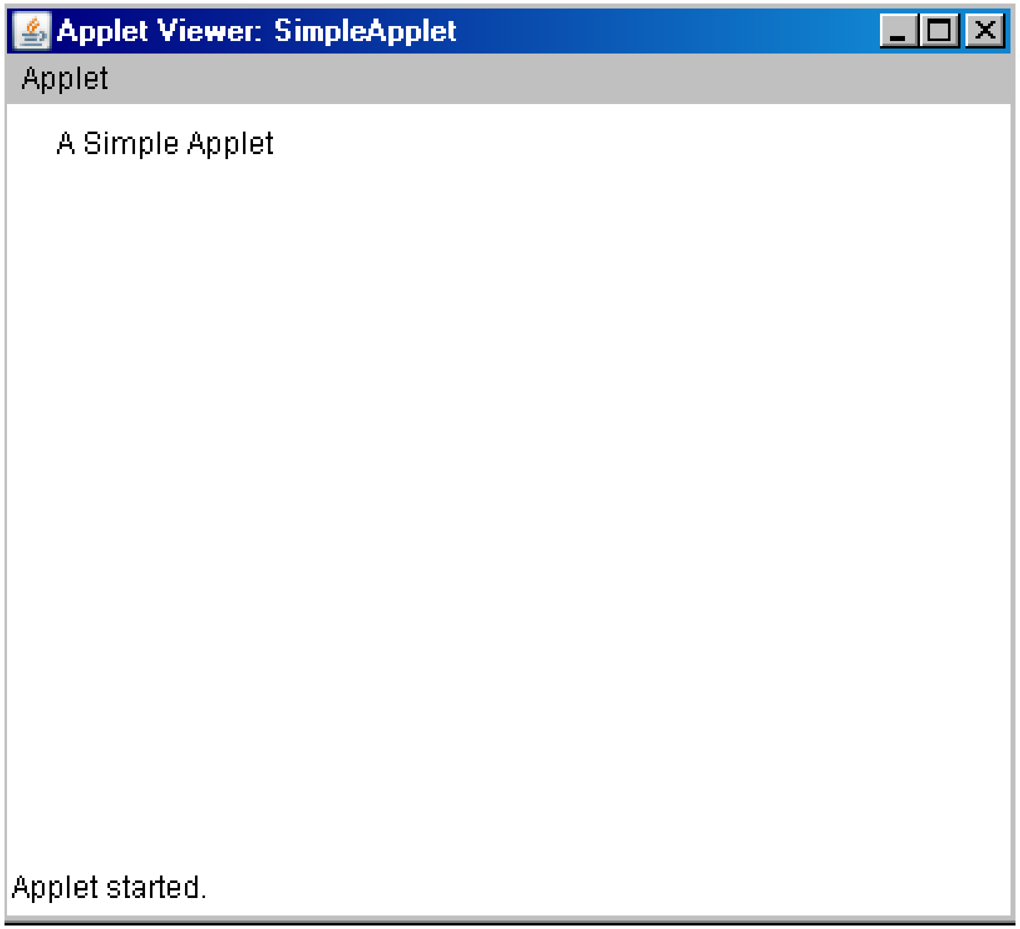
Ex:-

```
import java.awt.*;
import java.applet.Applet;
/*  <applet code="SimpleApplet"   width=400
   height=300 ></applet>    */
public class SimpleApplet extends Applet {
        public void paint(Graphics g) {
                g.drawString ("A Simple Applet",20, 20);
        }
}
```

Compile and then execute by using following command

```
C:\>appletviewer SimpleApplet.java
```

Output:



```
A Simple Applet
```

```
Applet started.
```

Structure of an applet

```
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletStructure" width=300 height=100>
</applet>
*/
public class AppletStructure extends Applet {
 // Called first.
 public void init() {
  // initialization
  }

 /* Called second, after init().  Also called whenever
    the applet is restarted. */
 public void start() {
  // start or resume execution
  }
// Called when the applet is stopped.
    public void stop() {
      // suspends execution
    }

    /* Called when applet is terminated.  This is the last
       method to be executed. */
    public void destroy() {
      // perform shutdown activities
   }

    // Called whenever an applet's output must be redisplayed.
    public void paint(Graphics g) {
      // redisplay contents of window
    }
   }
```

Life cycle of an Applet

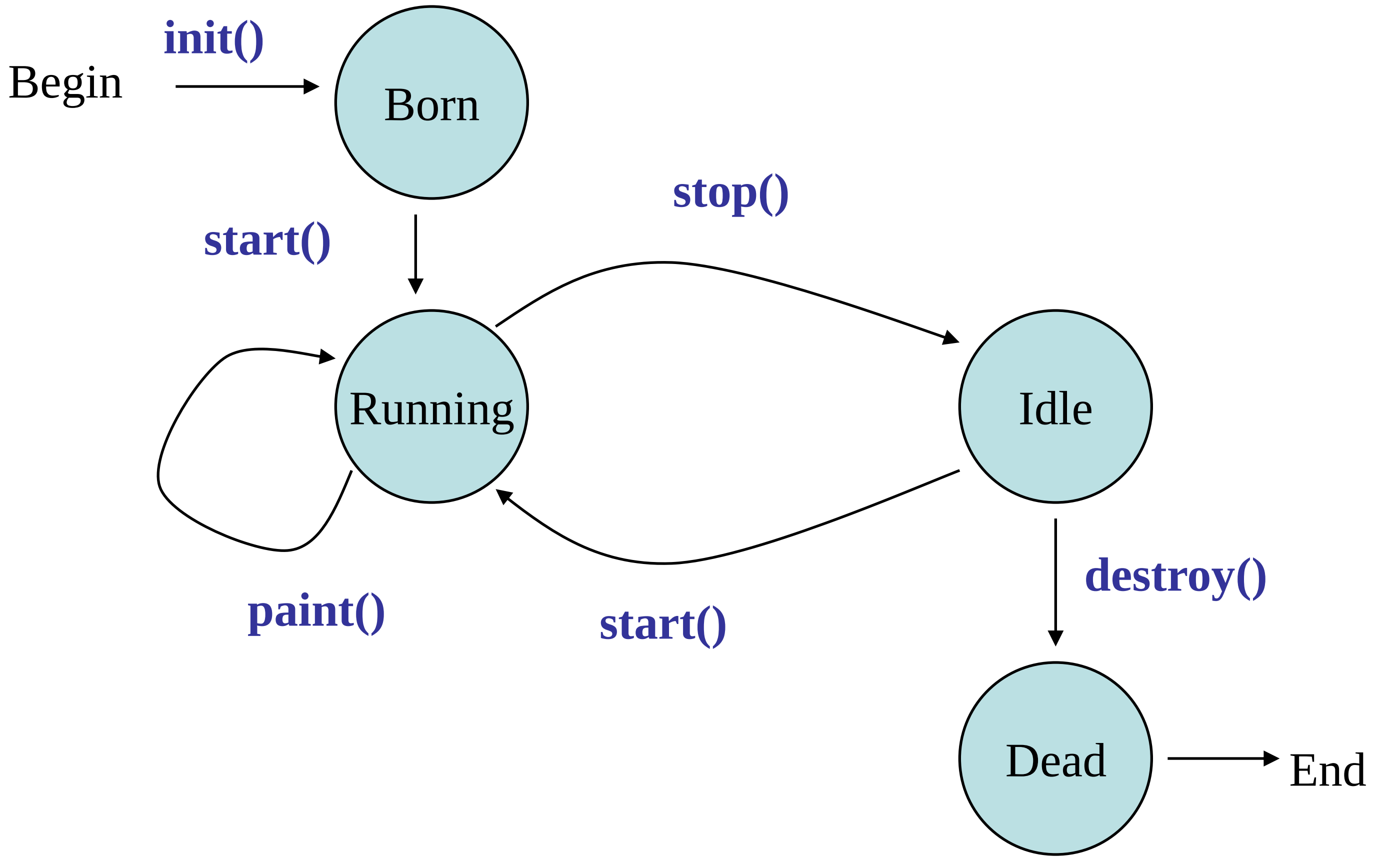


Fig: Applet Life Cycle Diagram

- When an applet is started , the following sequence of method calls takes place:
  1. **init( )**
  2. **start( )**
  3. **paint( )**
- When an applet is terminated, the following sequence of method calls takes place:
  1. **stop( )**
  2. **destroy( )**

Applet States

init()- called only once

- The **init( )** method is the first method to be called.
- This is where you should initialize variables.
- This method is called only once during the run time of your applet.

start()- called more than once

- The **start( )** method is called after **init( )**.
- **start( )** is called each time an applet's HTML document is displayed on screen.
- So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

paint( )- called more than once

- The **paint( )** method is called each time your applet's output must be redrawn.
- For example, when the applet window is minimized and then restored.
- **paint( )** is also called when the applet begins execution.
- The **paint( )** method has one parameter of type **Graphics**. This parameter is used to draw graphics.

stop( ) - called more than once

- The **stop( )** method is called when a web browser leaves the HTML document containing the applet.
- for example when it goes to another page.
- When **stop( )** is called, the applet is probably running.
- You should use **stop( )** to suspend threads that don’t need to run when the applet is not visible.
- You can restart them when **start( )** is called if the user returns to the page.

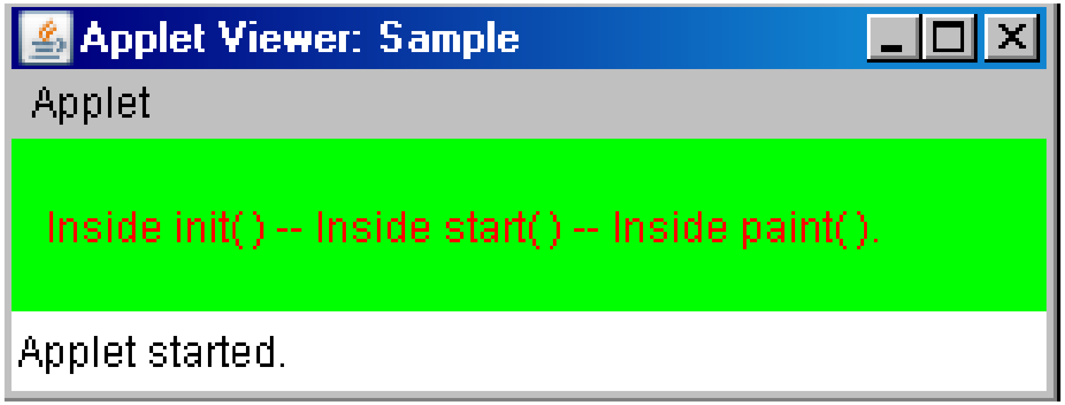
destroy( ) - called only once

- The **destroy( )** method is called whenever the browser is closed.
- Applet is removed completely from memory.
- At this point, you should free up any resources the applet may be using.
- The **stop( )** method is always called before **destroy( )**.

Creating Applets

```
/* A simple applet that sets the foreground and
background colors and outputs a string. */
import java.awt.*;
import java.applet.*;
/*
<applet code="Sample" width=300 height=50>
</applet>
*/
public class Sample extends Applet{
String msg;
// set the foreground and background colors.
public void init() {
setBackground(Color.green);
setForeground(Color.red);
msg = "Inside init( ) --";
}
// Initialize the string to be displayed.
public void start() {
msg += " Inside start( ) --";
}
// Display msg in applet window.
public void paint(Graphics g) {
msg += " Inside paint( ).";
g.drawString(msg, 10, 30);
}
public void stop()
{
 System.out.println("Inside stop( )");
}
public void destroy()
{
 System.out.println("Inside destroy( )");
}
}
```

Output:



```
Inside init( ) -- Inside start( ) -- Inside paint( ).
```

Applet started.

After closing appletviewer stop( ) and destroy( ) methods will be called.

Output in the command prompt:

```
Inside stop( )
Inside destroy( )
```

Passing Parameters to Applet

- The PARAM tag in HTML allows you to pass parameters to your applet.
- To retrieve a parameter, use the **getParameter( )** method.
- It returns the value of the specified parameter in the form of a **String** object.

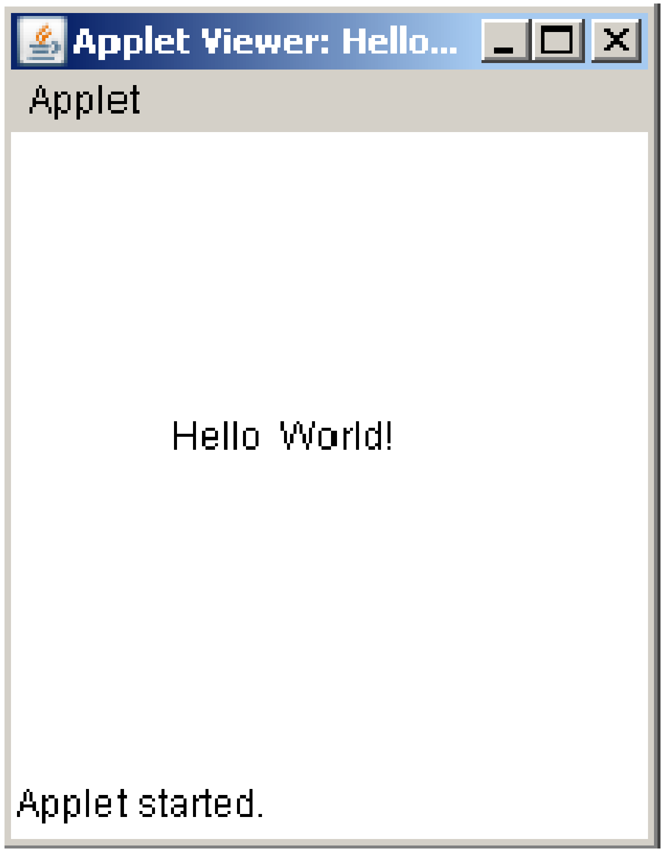
Applet Program Accepting Parameters

```
//HelloAppletMsg.java
import java.applet.Applet;
import java.awt.*;
/*  <APPLET
            CODE="HelloAppletMsg" width=200 height=200>
           <PARAM NAME="Greetings" VALUE="Hello  World!">
    </APPLET>  */

public class HelloAppletMsg extends Applet {

    String msg;

    public void init()
    {
           msg = getParameter("Greetings");
           if( msg == null)
                    msg = "Hello";
```

This is name of parameter specified in PARAM tag;
This method returns the value of paramter.

```
  }
    public void paint(Graphics g) {
           g.drawString (msg,50, 100);
    }
  }
```

Output:



```
Hello World!
```

```
Applet started.
```

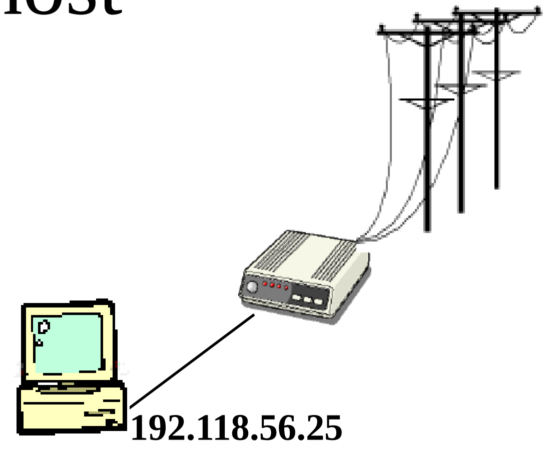
Types of applets

- There are two types of applets.
  - First is based on the Applet class
    - These applets use the AWT classes to provide the GUI.
    - This style of applet has been available since java was created.
    - It is used for simple GUI’s.
  - The second is based on the Swing class JApplet.
    - These applets use the Swing classes to provide the GUI.
    - Swing offers a richer and easy to use interface than AWT .
    - Swing based applets are more popular.

Basics of network programming

IP Address

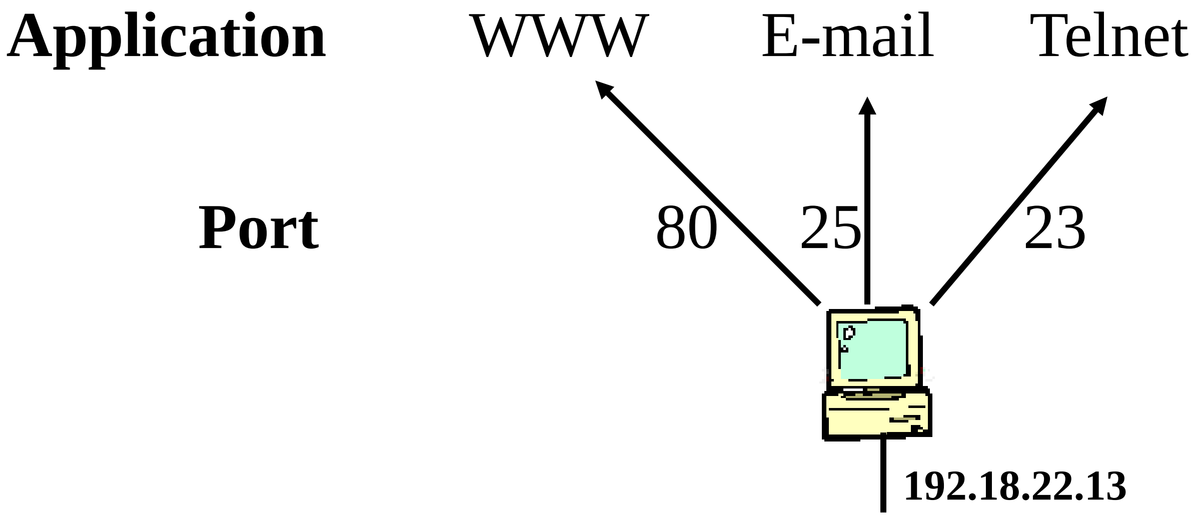
- 32-bit identifier
- Dotted-quad: 192.118.56.25
- Identifies a host



Ports

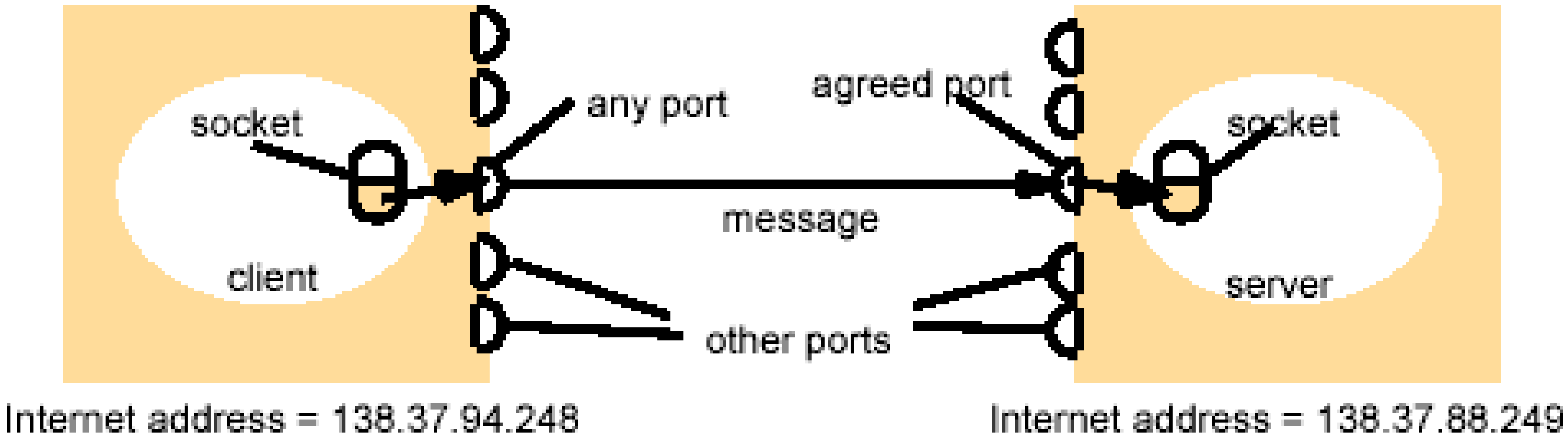
Identifying the specific application

- IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier)



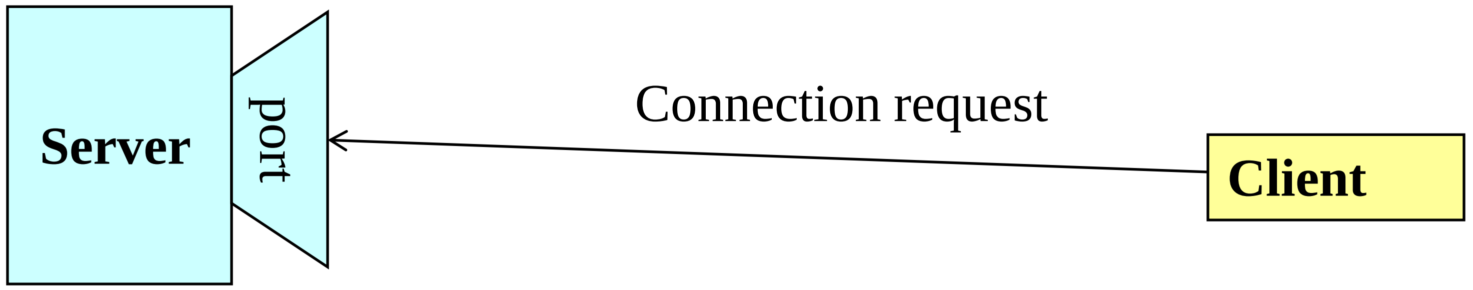
Sockets

- Socket is an object used for network programming.
- A socket is bound to a specific port number
- Network communication using Sockets is very much similar to performing file I/O

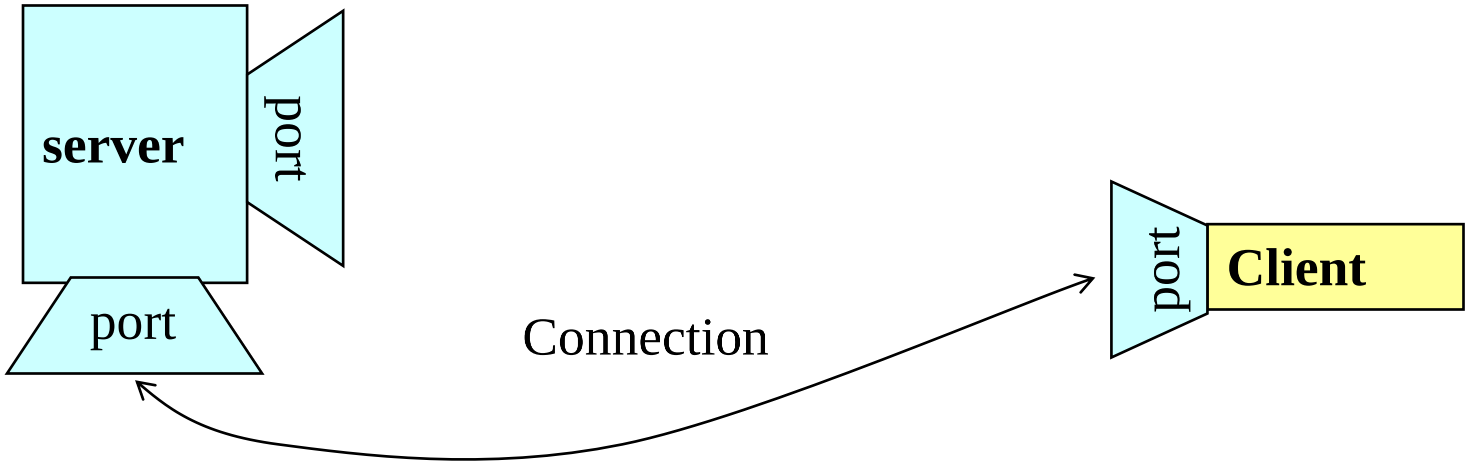

socket
client
any port
agreed port
message
other ports
socket
server
Internet address = 138.37.94.248
Internet address = 138.37.88.249

Socket Communication

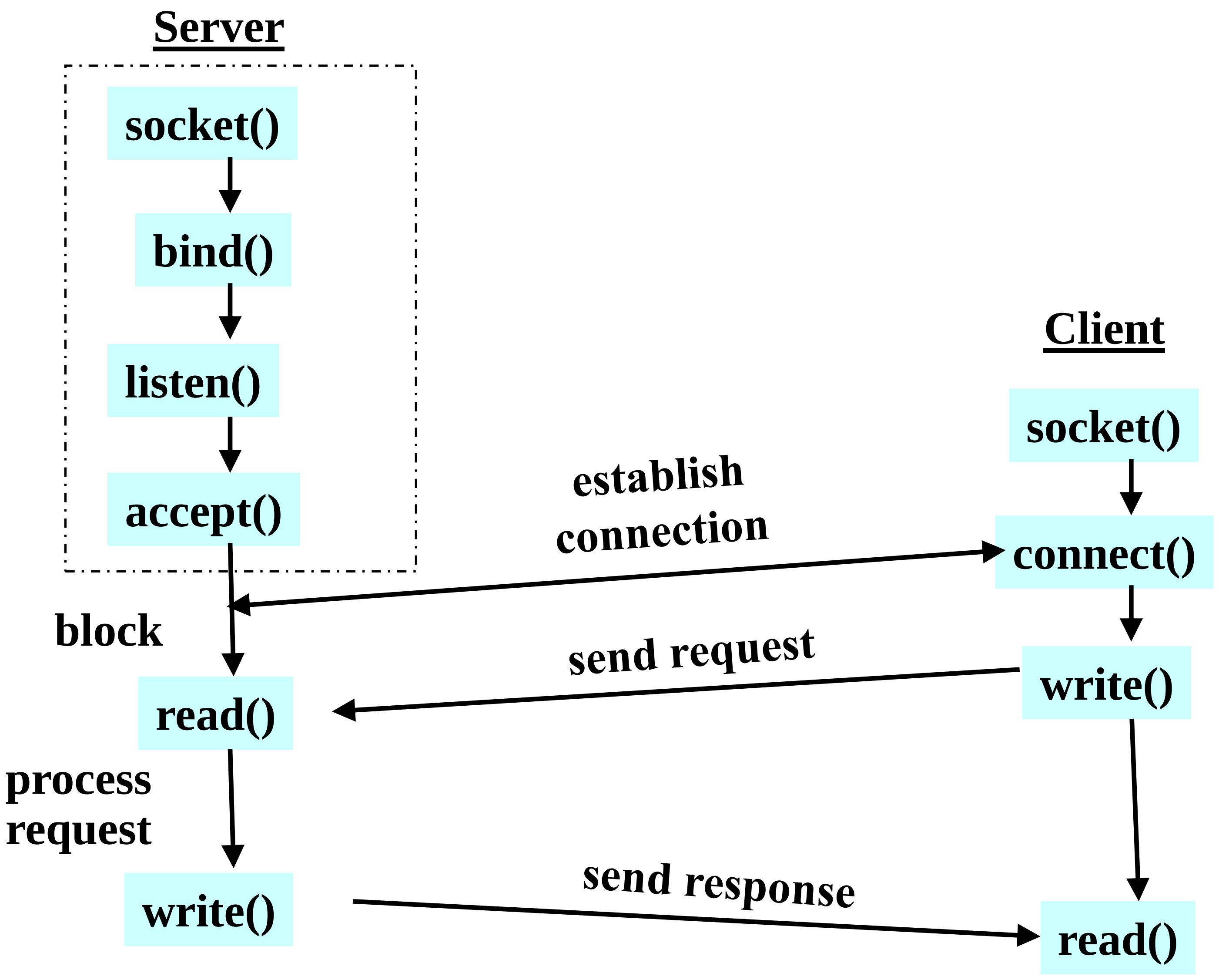
- A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request.



- If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bounds to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.



Client-Server communication



Java’s .net package

- Java’s .net package provides two classes:
  - Socket – for implementing a client
  - ServerSocket – for implementing a server

Implementing a Server

1. Open the Server Socket:

   ServerSocket server = new ServerSocket(PORT);

2. Wait for the Client Request:

   Socket s = server.accept();

3. Create I/O streams for communicating to the client

   DataInputStream dis = new DataInputStream(s.getInputStream());

   DataOutputStream dos = new DataOutputStream(s.getOutputStream());

4. Perform communication with client

   Receive data from client: String line = dis.readLine();

   Send data to the client: dos.write ("Hello\n");

5. Close sockets: s.close();

Implementing a Client

1. Create a Socket Object:

   Socket client = new Socket(server, port_id);

2. Create I/O streams for communicating with the server.

   DataInputStream dis = new DataInputStream(client.getInputStream());

   DataOutputStream dos = new DataOutputStream(client.getOutputStream());

3. Perform communication with the server:
   - Receive data from the server:

     String line = dis.readLine();

   - Send data to the server:

     dos.write ("Hello\n");

4. Close the socket when done:

   client.close();

Simple client-serever program

A simple server (simplified code)

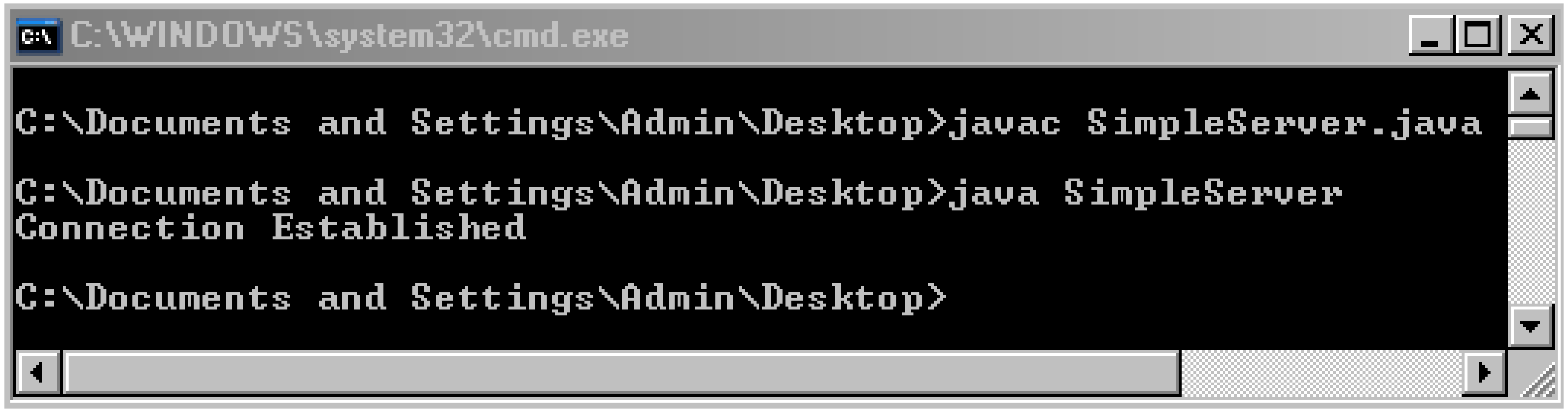
```
// SimpleServer.java: a simple server program
import java.net.*;
import java.io.*;
public class SimpleServer
{
   public static void main(String args[]) throws IOException
  {
    // Register service on port 1234
    ServerSocket server = new ServerSocket(1234);
    Socket s=server.accept(); // Wait and accept a connection
    System.out.println("Connection Established");
    // Get a communication stream associated with the socket
    DataOutputStream dos = new DataOutputStream(s.getOutputStream());
    // Send a string!
    dos.writeBytes("Hello Client");
    // Close the connection, but not the server socket
    dos.close();
    s.close();
  }
}
```

A simple client (simplified code)

```
// 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 1234
    Socket client = new Socket("localhost",1234);
    // Get an input file handle from the socket and read the input
    DataInputStream dis = new DataInputStream(client.getInputStream());
    String str = dis.readLine();
    System.out.println(str);
    // When done, just close the connection and exit
    dis.close();
    client.close();
  }
}
```

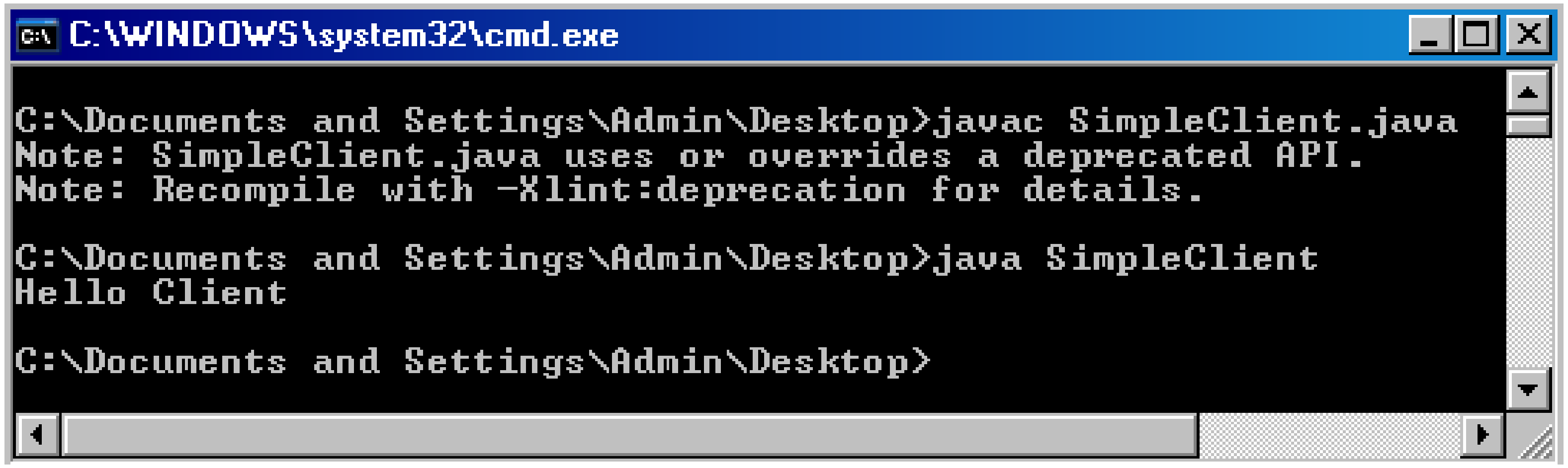
server_IP

Server side



```
C:\Documents and Settings\Admin\Desktop>javac SimpleServer.java

C:\Documents and Settings\Admin\Desktop>java SimpleServer
Connection Established

C:\Documents and Settings\Admin\Desktop>
```

Client side



```
C:\Documents and Settings\Admin\Desktop>javac SimpleClient.java
Note: SimpleClient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Documents and Settings\Admin\Desktop>java SimpleClient
Hello Client

C:\Documents and Settings\Admin\Desktop>
```

Serving Multiple Clients

- Multiple clients are quite often connected to a single server at the same time. Typically, a server runs constantly on a server computer, and clients from all over the Internet may want to connect to it. You can use threads to handle the server's multiple clients simultaneously. Simply create a thread for each connection. Here is how the server handles the establishment of a connection:

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

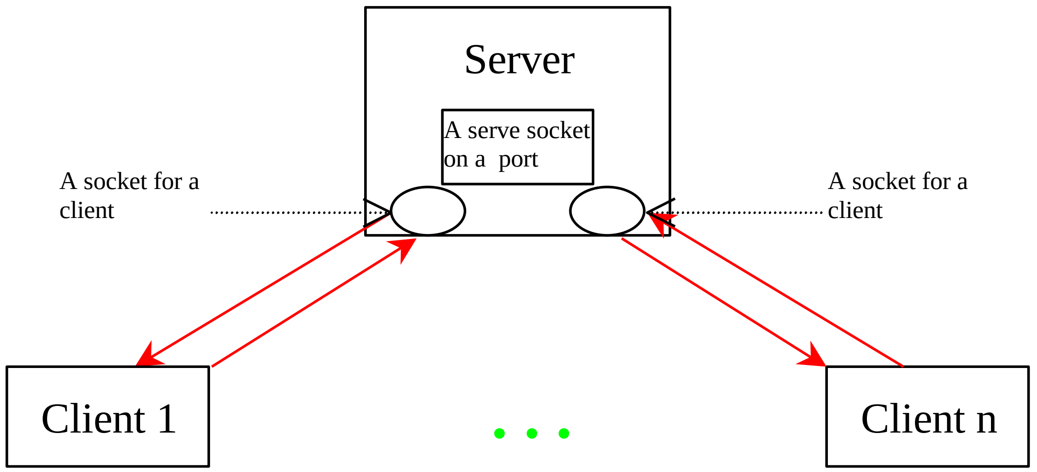
The server socket can have many connections. Each iteration of the while loop creates a new connection. Whenever a connection is established, a new thread is created to handle communication between the server and the new client; and this allows multiple connections to run at the same time.

Example: Serving Multiple Clients

Server for Multiple Clients

Start Server

Start Client



Note: Start the server first, then start multiple clients.

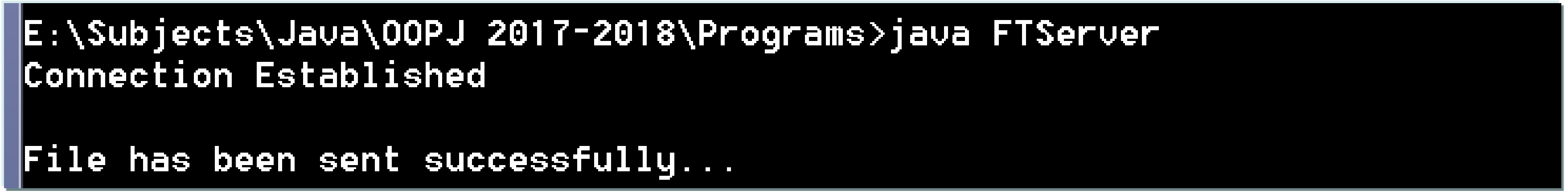
Sending file from server to client

Server program

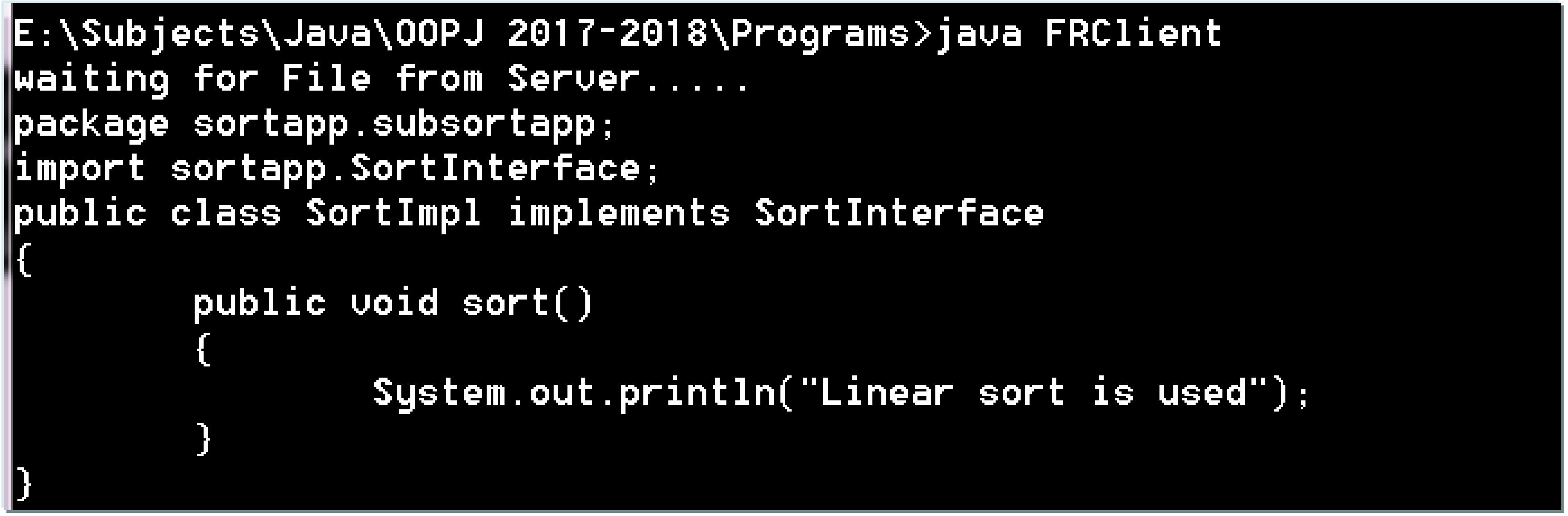
```
import java.io.*;
import java.net.*;
class FTServer{
        public static void main(String args[])throws Exception{
                ServerSocket ss=new ServerSocket(2424);
                Socket s=ss.accept();
                System.out.println("Connection Established\n");
                File myFile = new File (“E:/Programs/SortImpl.java");
                FileInputStream fis = new FileInputStream(myFile);
                DataInputStream dis=new DataInputStream(fis);
                byte [] mybytearray  = new byte [(int)myFile.length()];
                dis.read(mybytearray,0,mybytearray.length);
        DataOutputStream dos=new DataOutputStream(s.getOutputStream());
                dos.write(mybytearray,0,mybytearray.length);
                System.out.println("File has been sent successfully...");
                s.close();
                dos.close();
                fis.close();
                dis.close();
        }
}
```

Client program

```
import java.io.*;
import java.net.*;
class FRClient{
		public static void main(String args[])throws Exception{
				Socket s=new Socket("localhost",2424);
				System.out.println("waiting for File from Server.....");
		DataInputStream dis=new DataInputStream(s.getInputStream());

				String str;
				boolean b=true;

				while(b){
						str=dis.readLine();
						if(str==null)
								b=false;
						else
								System.out.println(str);
				}
				dis.close();
				s.close();
		}
}
```



```
E:\Subjects\Java\OOPJ 2017-2018\Programs>java FTServer
Connection Established

File has been sent successfully...
```



```
E:\Subjects\Java\OOPJ 2017-2018\Programs>java FRClient
waiting for File from Server.....
package sortapp.subsortapp;
import sortapp.SortInterface;
public class SortImpl implements SortInterface
{
        public void sort()
        {
                System.out.println("Linear sort is used");
        }
}
```