UNIT-V

File handling, Networking and Applets

Syllabus:
File Handling: Streams, Byte streams, Character streams and File streams.
Networking: TCP, UDP, URL Connection Class, Inet Address Class, Socket Programming.
Applets: Concepts of Applets, differences between applets and applications, life cycle of an applet, types of applets, creating applets, passing parameters to applets, Applet to applet communication.

File Handling:

Streams:

- Streams facilitate transporting data from one place to another.
- Different streams are needed to send or receive data through different sources, such as to receive data from keyboard, we need a stream and to send data to a file, we need another stream.
- Without streams, it is not possible to move data in Java.
- Streams can be categorized as **'input streams'** and **'output streams'**.
- **Input streams** are the streams which receive or read data while **output streams** are the streams which send or write data.
- All streams are represented by classes in **java.io (input- output) package**.
- input stream reads data. So, to read data from keyboard, we can attach the keyboard to an input stream, so that the stream can read the data typed on the keyboard.

```
DatalrtputStream dfs = new DataInputStream(System.in);
```

- In the above statement, we are attaching the keyboard to `DatalnputStream` object. The keyboard is represented by `System.in`.
- DatalnputStream object can read data coming from the keyboard.
- Here, System is a class and in is a field in System class. In fact, the System class has the following 3 fields:

  System.in: Represents **InputStream** object This object represents the standard input device, that is keyboard by default.
  System.out: Represents **PrintStream** object, This object by default represents the standard output device, that is monitor.
  System.err: Represents **PrintStream** object. This object by default represents the standard output device, that is monitor,

- Keyboard is an InputStream. Similarly, the monitor represented by PrintStream.

Classification of Streams:

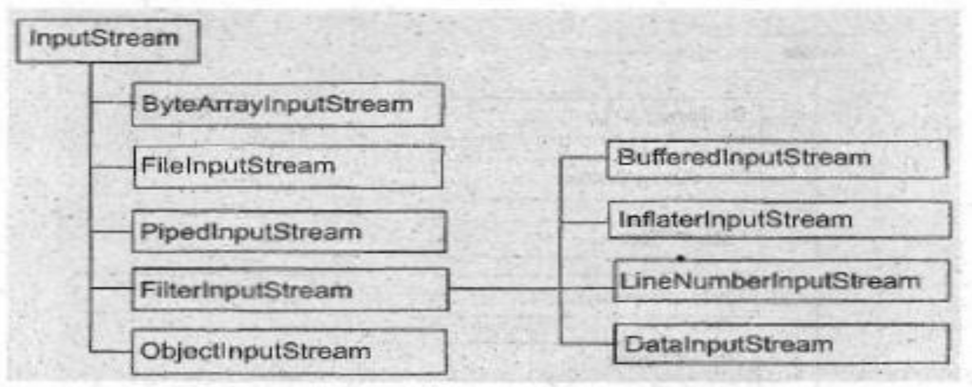
- Streams can be classified into types
    1. Byte streams
    2. Text streams (or) Character streams
- **Byte streams** represent data in the form of individual bytes.
- **Text streams** represent data as characters of each 2 bytes.
- If a class name ends with the word 'Stream', then it comes under byte streams, InputStream reads bytes and OutputStream writes bytes. For example;

```
FileInputStream
FileOutputStream
BufferedInputStream
BufferedOutputStream
```

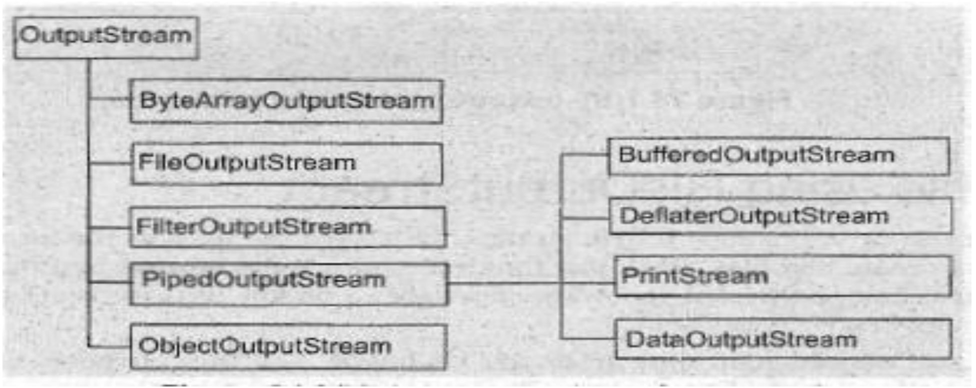
- If a class name ends with the word 'Reader or Writer' then it is taken as a text stream. Reader reads text and Writer writes text. For example,

```
FileReader
FileWriter
BufferedReader
BufferedWriter
```

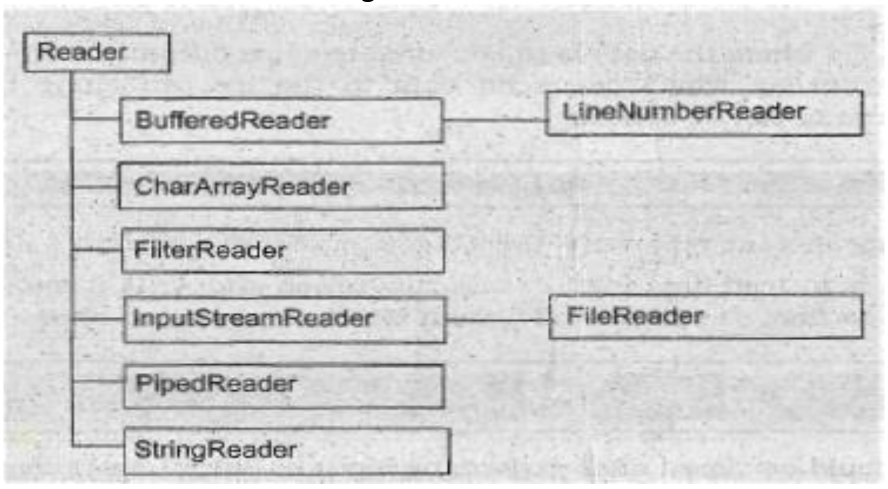
- **Byte streams** are used to handle any characters (text), images, audio, and video files.
- For example, to store an image file (.gif or \ jpg), we should go for a byte stream.
- Byte stream classes for reading the data



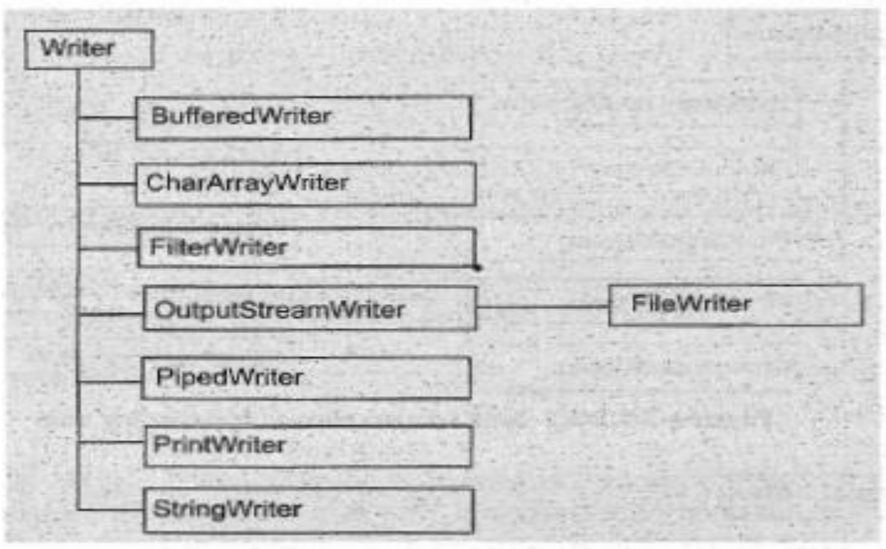
- Byte stream classes for writing the data



- **Character or text streams** can always store and retrieve data in the form of characters or text only.
- It means text streams are more suitable for handling text fijes like the ones we create in Notepad. They are not suitable to handle the images, audio, or video files.
- Character streams classes for reading data

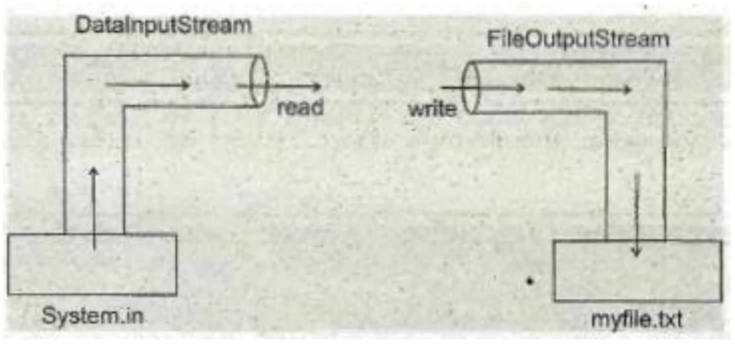


- Character streams classes for writing data



Creating a file using FileOutputStream:

- FileOutputStream class belongs to byte stream and stores the data in the form of individual bytes. DataInputStream is used to read data from keyboard.
- It can be used to create text files. We know that a file represents storage of data on a second storage media like a hard disk or CD.



Program:
```
import java.io.*;
class CreateFile
{
    public static void main(String args[])throws IOException
      {
           DataInputStream dis=new DataInputStream(System.in);
           FileOutputStream fout=new FileOutputStream("myfile.txt");
           System.out.println("enter text(give @ at end)");
           char ch;
           while((ch=(char)dis.read())!='@')
              fout.write(ch);
           fout.close();
      }
}
```
Compilation: `javac CreateFile.java`

Execution:
```
java CreateFile
enter text(give @ at end)
hello this is file handling concept.
@
```
To see file content: `type myfile.txt`

Ouput: `hello this is file handling concept.`

Appending text to already existing file:

- For appending text to already existing file we have use the following form

```
FileOutputStream fout=new FileOutputStream("myfile.txt",true);
```

Program:
```
import java.io.*;
class CreateFile1
{
  public static void main(String args[])throws IOException
  {
     DataInputStream dis=new DataInputStream(System.in);
     FileOutputStream fout=new FileOutputStream("myfile.txt",true);
     System.out.println("enter text(give @ at end)");
     char ch;
      while((ch=(char)dis.read())!='@')
         fout.write(ch);
      fout.close();
  }
}
```
Compilation: `javac CreateFile1.java`

Execution:
```
java CreateFile1
enter text(give @ at end)
 hello this is appended text
@
```
To see file content: `type myfile.txt`

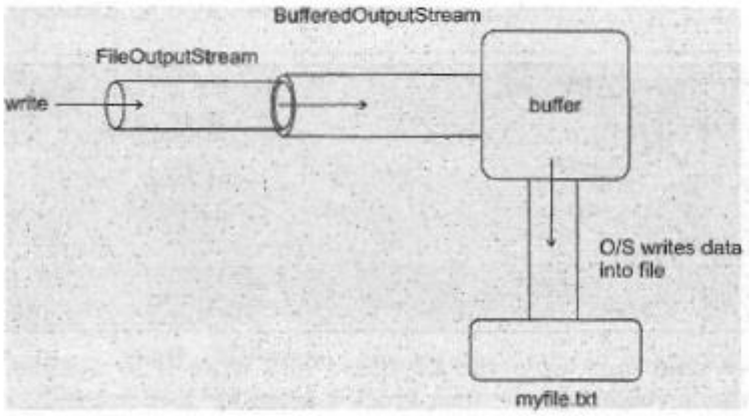
Ouput: `hello this is file handling concept.hello this is appended text`

Creating a file using BufferedOutputStream:

- While creating a file using BufferedOutputStream class the efficiency of the program is improved.
- BufferedOutputStream takes argument as FileOutputStream as reference for creating a Buffer.
- The default size of Buffer is 512 bytes.
- The general form of BufferedOutputStream class is

```
BufferedOutputStream bout=new BufferedOutputStream(fout,1024);
```

- In the above form "fout" means reference of FileOutputStream class and 1024 is bytes.



- **Example Program:**

```
import java.io.*;
class CreateFile2
{
  public static void main(String args[])throws IOException
  {
     DataInputStream dis=new DataInputStream(System.in);
     FileOutputStream fout=new FileOutputStream("myfile1.txt");
   BufferedOutputStream bout=new BufferedOutputStream(fout,1024);
     System.out.println("enter text(give @ at end)");
     char ch;
     while((ch=(char)dis.read())!='@')
        bout.write(ch);
     bout.close();
  }
}
```

Compilation: `javac CreateFile2.java`

Execution:

```
java CreateFile2
enter text(give @ at end)
hello this is BufferedOutputStream class.
@
```

To see file content: `type myfile1.txt`

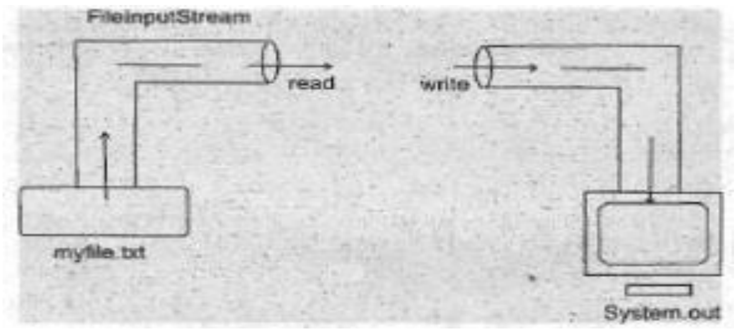
Ouput: `hello this is BufferedOutputStream class.`

****

Reading data from a file using FileInputStream:

- FileInputStream is useful to read data from a file in the form of sequence of bytes.
- It is possible read data from a text file using FileInputStream.

```
FileInputStream fin=new FileInputStream("myfile.txt")
```



Program:

```
import java.io.*;
class ReadFile
{
  public static void main(String args[])throws IOException
  {
     FileInputStream fout=new FileInputStream("myfile.txt");
     System.out.println("File contents:");
     int ch;
      while((ch=fin.read())!=-1)
             System.out.print((char)ch);
      fin.close();
  }
}
```

Compilation: `javac CreateFile1.java`

Execution: `java CreateFile1`

Ouput:

```
File contents:
hello this is file handling concept.hello this is appended
text.
```

Creating a file using FileWriter:

- FileWriter is useful to create a file by writing characters into it.
- The following program how to create a text file using FileWriter.
- **Program:**

```
import java.io.*;
class CreatingFile
{
    public static void main(String args[])throws IOException
     {
           DataInputStream dis=new DataInputStream(System.in);
           FileWriter fw=new FileWriter("myfile2.txt");
           System.out.println("enter text(give @ at end)");
           char ch;
           while((ch=(char)dis.read())!='@')
              fw.write(ch);
           fw.close();
     }
}
```

Compilation : `javac CreatingFile.java`

Execution: `java CreatingFile`

```
enter text(give @ at end)
hello we are CSE students
@
```

To see file content: `type myfile2.txt`

```
hello we are CSE students
```

Reading data from a file using FileReader:

- FileReader is useful to read data in the form of characters from a 'text' file.
- **Program:**

```
import java.io.*;
class ReadingFile
{
  public static void main(String args[])throws IOException
  {
     FileReader fout=new FileReader("myfile2.txt");
     System.out.println("File contents:");
     int ch;
     while((ch=fr.read())!=-1)
            System.out.print((char)ch);
     fr.close();
  }
}
```

Compilation : `javac ReadingFile.java`

Execution: `java ReadingFile`

Output:

```
hello we are CSE students
```

****

File class:

- File class of java.io package provides some methods to know the properties of a file or a directory.
- Creating the File object by passing the filename or directory name to it.

```
File obj=new File(“filename”);
Fiel obj=new File(“directory name”);
File obj=new File(“path”,”filename”);
File obj=new File(“path”,”directoryname”);
```

- We can create text files using File class by replacing “FileOutputStream” and “FileWriter” class with “File” class.

Creating file using File class:

```
   import java.io.*;
   class CreateFile2
   {
     public static void main(String args[])throws IOException
     {
        DataInputStream dis=new DataInputStream(System.in);
        File f=new File("myfile1.txt");
        System.out.println("enter text(give @ at end)");
        char ch;
        while((ch=(char)dis.read())!='@')
           f.write(ch);
        f.close();
     }
   }
```

****

Creating directories using File class:

```
import java.io.File;
class CreateDirectoryExample
{
    public static void main(String[] args)
    {
      // creating a directory in current working directory
      File file = new File("mydir");
       // exists() returns Boolean value true or false
      if (!file.exists())
      {
         // mkdir() returns Boolean value true or false
         if (file.mkdir())
           System.out.println("Directory is created!");
         else
           System.out.println("Failed to create directory!");
      }

     // creating multiple directories in specified path
     File files = new File("C:\\Directory2\\Sub2\\Sub-Sub2");
     if (!files.exists())
     {
        if (files.mkdirs())
           System.out.println("Multiple directories are created!");
        else
         System.out.println("Failed to create multi directories!");
     }
   }
}
```

****

Counting number of lines, words and characters in a text file:

```
import java.io.*;
class WordCountInFile
{
    public static void main(String[] args)throws IOException
    {
        BufferedReader reader = null;
        int charCount = 0;
        int wordCount = 0;
        int lineCount = 0;
        try
        {
            FileReader fr=new FileReader("myfile.txt");
            reader = new BufferedReader(fr);
            String currentLine = reader.readLine();
            while (currentLine != null)
            {
                lineCount++;
                String[] words = currentLine.split(" ");
                wordCount = wordCount + words.length;
                for (String word : words)
                {
                    charCount = charCount + word.length();
                }
                currentLine = reader.readLine();
            }
            System.out.println("Chars In a File: "+charCount);
            System.out.println("Words In a File: "+wordCount);
            System.out.println("Lines In a File: "+lineCount);
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
        finally
        {
           reader.close();
        }
    }
}
```

Compilation: `javac WordCountInFile.java`
Execution : `java WordCountInFile`
Output :

```
//Based on your file data displays output.
```

****

Reading data from console using BufferedReader:

```
import java.io.*;
class ReadFromConsole
{
   public static void main(String args[])throws Exception
   {

          DataInputStream dis=new DataInputStream(System.in);
          BufferedReader br=new BufferedReader(dis);
          System.out.println("Enter your name");
          String name=br.readLine();
          System.out.println("Welcome "+name);
   }
}
```

Compilation: `javac  ReadFromConsole.java`
Execution: `java ReadFromConsole`
Output:

```
Enter your name
Sachin Tendulkar
Welcome Sachin Tendulkar
```

****

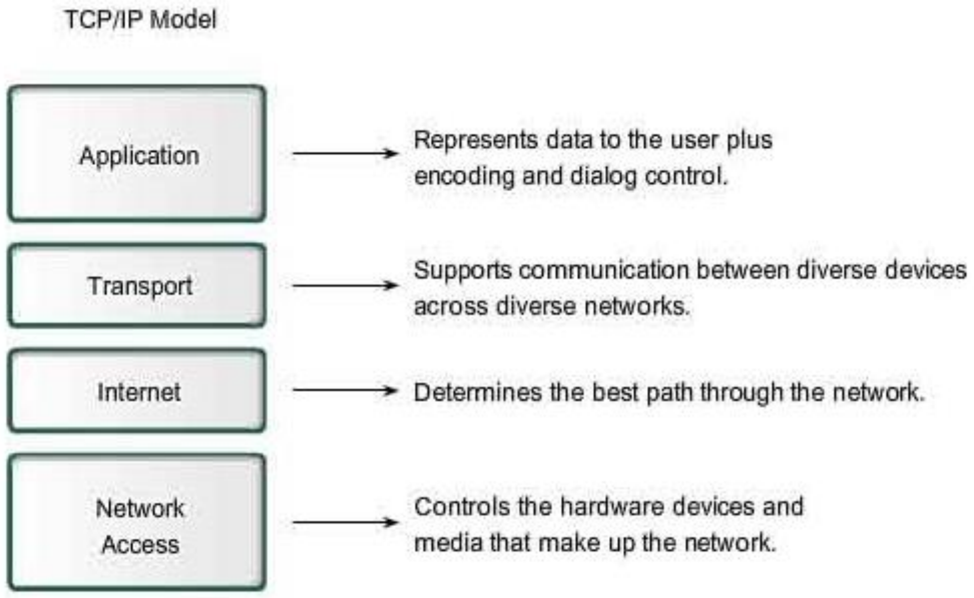
Networking:

Network:

- Two or more systems that are inter connected to each other is called a network.
- To establish a network we have 3 requirements
    - **Hardware**: includes computers, cables, modems, hubs etc.
    - **Software:** includes programs provide communication between client and server
    - **Protocol**: includes set of rules for sending and receiving data.

TCP/IP Protocol

- A protocol represents a set of rules to be followed by every computer on the network, Protocol is useful to physically move data from one place to another place on a network.
- TCP (Transmission Control Protocol) / IP (Internet Protocol) is the standard protocol model used on any network, including Internet.
- Before TCP/IP, we have OSI (Open Systems Interconnection) reference model. Based on OSI model TCP/IP Layer model is established.


TCP/IP Model
Application
Represents data to the user plus encoding and dialog control.
Transport
Supports communication between diverse devices across diverse networks.
Internet
Determines the best path through the network.
Network Access
Controls the hardware devices and media that make up the network.

- ***Application layer*** provides applications the ability to access the services of the other layers and defines the protocols that applications use to exchange data. There are many Application layer protocols and new protocols are always being developed.
  The most widely-known Application layer protocols are those used for the exchange of user information:
    - **Hypertext Transfer Protocol (HTTP)** is used to transfer files that make up the Web pages of the World Wide Web.
    - **File Transfer Protocol (FTP)** is used for interactive file transfer.
    - **Simple Mail Transfer Protocol (SMTP)** is used for the transfer of mail messages and attachments.
    - **Domain Name System (DNS)** is used to resolve a host name to an IP address.

- ***Transport layer*** (also known as the Host-to-Host Transport layer) is responsible for providing the Application layer with session and datagram communication services. The core protocols of the Transport layer are *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP).

  TCP(Transmission Control protocol): provides a one-to-one, connection-oriented, reliable communications service. TCP is responsible for the establishment of a TCP connection, the sequencing and acknowledgment of packets sent, and the recovery of packets lost during transmission.

  UDP(User Datagram Protocol): provides a one-to-one or one-to-many, connectionless, unreliable communications service. UDP is used when the amount of data to be transferred is small (such as the data that would fit into a single packet), when the overhead of establishing a TCP connection is not desired or when the applications or upper layer protocols provide reliable delivery.

- ***Internet layer or Network layer*** is responsible for addressing, packaging, and routing functions. The core protocol of the Internet layer is IP.
  ***Internet Protocol* (IP)** is a routable protocol responsible for IP addressing, routing, and the fragmentation and reassembly of packets.
- ***Network Interface layer*** (also called the Network Access layer) is responsible for placing TCP/IP packets on the network medium and receiving TCP/IP packets off the network medium.

****

<u>InetAddress class:</u>

- InetAddress class represents an IP address.
- The java.net.InetAddress class provides methods to get the IP of any host name.
- *for example* www.google.com, www.facebook.com etc.

methods of InetAddress class:

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	it returns the instance of InetAdddress containing local host name and address.
public String getHostName()	it returns the host name of the IP address.
public String getHostAddress()	it returns the IP address in string format.

Example :

```
import java.io.*;
import java.net.*;
public class InetDemo
{
   public static void main(String[] args)
   {
      try
      {
         InetAddress ip=InetAddress.getByName("www.google.com");

         System.out.println("Host Name: "+ip.getHostName());
         System.out.println("IP Address: "+ip.getHostAddress());
      }
      catch(Exception e)
      {
         System.out.println(e);
      }
   }
}
```

Output:

```
Host Name: www.google.com
IP Address: 216.58.220.36
```

****

URL class:

- The **URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web.
- For example
  http://www.dreamtechpress.com:80/indjex.html

1. **Protocol:** In this case, http is the protocol.
2. **Host name :** In this case, www.dreamtechpress.com is the Host name.
3. **Port Number:** It is an optional attribute. Here 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.html is the file name.

Methods of URL class:

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example:

```
import java.io.*;
import java.net.*;
public class URLDemo
{
	public static void main(String[] args)
	{
		try
		{
		   URL url=new URL("http://www.dreamtechpress.com:80/indjex.html");
			System.out.println("Protocol: "+url.getProtocol());
			System.out.println("Host Name: "+url.getHost());
			System.out.println("Port Number: "+url.getPort());
			System.out.println("File Name: "+url.getFile());
		}
		catch(Exception e)
		{
			System.out.println(e);
		}
	}
}
```

Output:

```
Protocol: http
Host Name: www.dreamtechpress.com
Port Number: 80
File Name: /indjex.html
```

URLConnection class:

- The **URLConnection** class represents a communication link between the URL and the application.
- This class can be used to read and write data to the specified resource referred by the URL.
- The openConnection() method of URL class returns the object of URLConnection class.
  Syntax:

  ```
  public URLConnection openConnection()throws IOException{}
  ```

- The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method.
- The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.
- Example

  ```
  import java.io.*;
  import java.net.*;
  public class URLConnectionExample
  {
  	public static void main(String[] args)
  	{
  ```

```
            try
            {
              URL url=new URL(“https://www.tutorialspoint.com/");
              URLConnection urlcon=url.openConnection();
              InputStream stream=urlcon.getInputStream();
              int i;
              while((i=stream.read())!=-1)
              {
                  System.out.print((char)i);
              }
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
```

Output:

Source code of webpage is displayed

Socket Programming:

- A socket is a point of connection between a client and server on a network.
- Each socket is given an identification number, which is called ‘port number'. Port number takes 2 bytes and can be from 0 to 65,535.
- Establishing communication between a server and a client using sockets is called 'socket programming'.
- Socket programming can be connection-oriented or connection-less.
- **Socket** and **ServerSocket** classes are used for connection-oriented socket programming.
- The client in socket programming must know two information: IP Address of Server, and Port number.

Socket class

- A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

- The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example:

MyServer.java

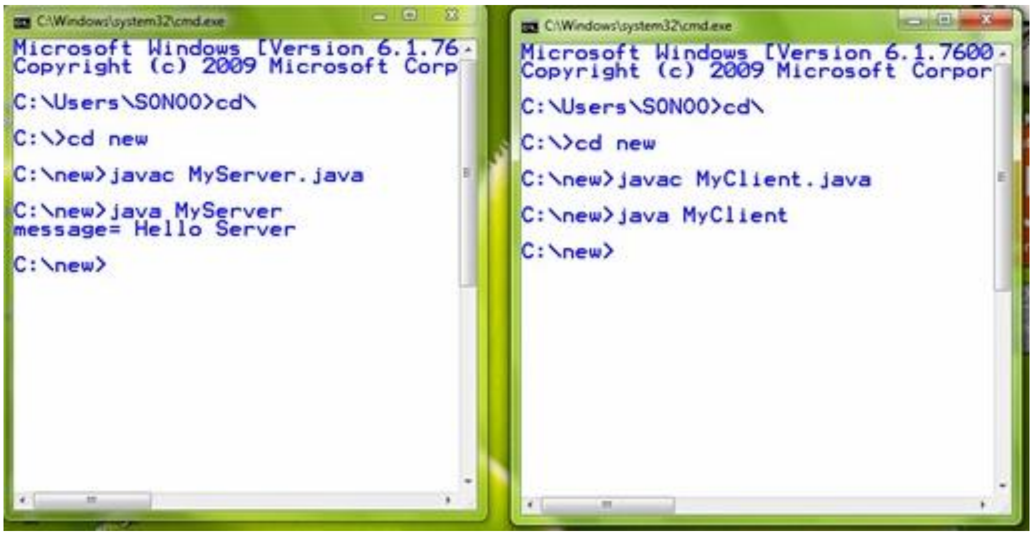
```
import java.io.*;
import java.net.*;
public class MyServer
{
      public static void main(String[] args)
      {
            try
            {
                  ServerSocket ss=new ServerSocket(6666);
                  Socket s=ss.accept();//establishes connection
                  DataInputStream dis=new DataInputStream(s.getInputStream());
                  // UTF: Unicode Transformation Format
                  String  str=(String)dis.readUTF();
                  System.out.println("message= "+str);
                  ss.close();
            }
            catch(Exception e)
            {
                  System.out.println(e);
            }
      }
}
```

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient
{
    public static void main(String[] args)
     {
        try
        {
            Socket s=new Socket("localhost",6666);
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());
             dout.writeUTF("Hello Server");
             dout.close();
             s.close();
        }
       catch(Exception e)
       {
            System.out.println(e);
        }
    }
}
```

Execution process:

- To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.
- After running the client application, a message will be displayed on the server console.



```
Microsoft Windows [Version 6.1.76
Copyright (c) 2009 Microsoft Corp

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyServer.java

C:\new>java MyServer
message= Hello Server

C:\new>
```

```
Microsoft Windows [Version 6.1.7600
Copyright (c) 2009 Microsoft Corpor

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyClient.java

C:\new>java MyClient

C:\new>
```

****

Applets:

- An applet is a small and portable application that runs under the restricted scope provided by JRE.
- It has dual compatibility as it can be executed on a web browser and also by applet viewer.
- JVM is required on the client machine to run an applet program.
- Applet represents Java byte code embedded in a web page.

Applet= java byte code+ html page

Differences between Applet and Application:

Applet	**Application**
Small Program	Large Program
Used to run a program on client Browser	Can be executed on stand alone computer system
Applet is portable and can be executed by any JAVA supported browser.	Need JDK, JRE, JVM installed on client machine.
Applet applications are executed in a Restricted Environment	Application can access all the resources of the computer
Applets are created by extending the java.applet.Applet	Applications are created by writing public static void main(String[] s) method.
Applet application has 5 methods which will be automatically invoked on occurance of specific event	Application has a single start point which is main method

Example:
`import java.awt.*;`
`import java.applet.*;`
`public class Myclass extends Applet`
`{`
`   public void init() { }`
`   public void start() { }`
`   public void stop() {}`
`   public void destroy() {}`
`   public void paint(Graphics g) {}`
`}`
	`public class MyClass`
	`{`
	`public static void main(String args[])`
	`{`
	`    //implementation`
	`}`
	`}`

Types of Applets:

- Web pages can contain two types of applets which are named after the location at which they are stored.
    1. Local Applet
    2. Remote Applet

Local Applets:

- A local applet is the one that is stored on our own computer system. When the Web-page has to find a local applet, it doesn't need to retrieve information from the Internet.
- A local applet is specified by a path name and a file name as shown below in which the codebase attribute specifies a path name, whereas the code attribute specifies the name of the byte-code file that contains the applet's code.

  <applet codebase="MyAppPath" code="MyApp.class" width=200 height=200>
  </applet>

Remote Applets:

- A remote applet is the one that is located on a remote computer system . This computer system may be located in the building next door or it may be on the other side of the world.
- No matter where the remote applet is located, it's downloaded onto our computer via the Internet. The browser must be connected to the Internet at the time it needs to display the remote applet.
- To reference a remote applet in Web page, we must know the applet's URL (where it's located on the Web) and any attributes and parameters that we need to supply.

<applet codebase="http://www.apoorvacollege.com" code="MyApp.class" width=200 height=200>
</applet>

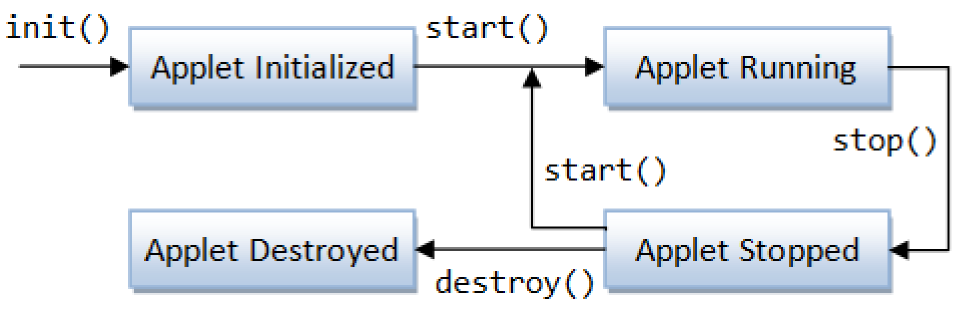
****

Applet life cycle:

- consists of 5 methods which will be automatically invoked by JRE when some action is done on the application/browser.
- List of Methods:

```
// exists insise “java.applet.Applet”
    public void init()
    public void start()
    public void stop()
    public void destroy()

//exist inside “java.awt.Component”
    public void paint(Graphics g)
```



init():

- Invoked only once when the application is launched using appletviewer or browser
- Should include code for component definition, Object creation, Layout settings, Basic look and feel

start()

- Invoked when application is maximized, Default application state is Maximized window so this will also be invoked on launching.
- Should include code for starting/resuming thread, starting/resuming Graphical User Interface, etc…

stop()

- Invoked when application is minimized, invoked also when the window is terminated while its in maximizes state.
- Should include code for pausing/stopping thread, pausing/stopping Graphical User Interface, etc…

destroy()

- Invoked when application is about to terminate (or) invoked when we close the application.
- Should include code for connection closing, file closing, etc…

paint()

- Invoked when application is to be refreshed in terms of GUI (or) invoked when we start, move or resize applet.
- Should include code for GUI design

****

Creating an Applet:

- Five steps to be kept in mind when writing an applet program:
  1. import java.applet.* (Need Applet class from java.applet package)
  2. create a public class
  3. extend the class from the base class Applet
  4. do not write main method
  5. write the applet life cycle methods (i.e. init, start, stop, destroy, paint)
- **File name:** abc.java

```
import java.applet.*;
public class abc extends Applet
{
    //methods of applet life cycle
}
```

- Steps to compile and Run Applet program:
  - Compile your java code using javac.exe on console/command prompt.
  - To run the generated class file, we have to write <applet> tag in an additional file(e.g. abc.html). Later on that file name is to be passed as an argument to appletviewer.exe as shown below
- **File name:** abc.html

```
<applet code="abc" width="500" height="500"></applet>
```

- **Commands to be executed:**

```
 c:> javac abc.java
c:> appletviewer abc.html
```

- **<u>Example:</u>**

<u>MyApp.java</u>

```
import java.applet.*;
import java.awt.*;
public class MyApp extends Applet
{
    public void init()
     {
         setBackground(Color.yellow);
     }
   public void paint(Graphics g)
    {
       g.drawString("hello applet",100,100);
    }
}
```

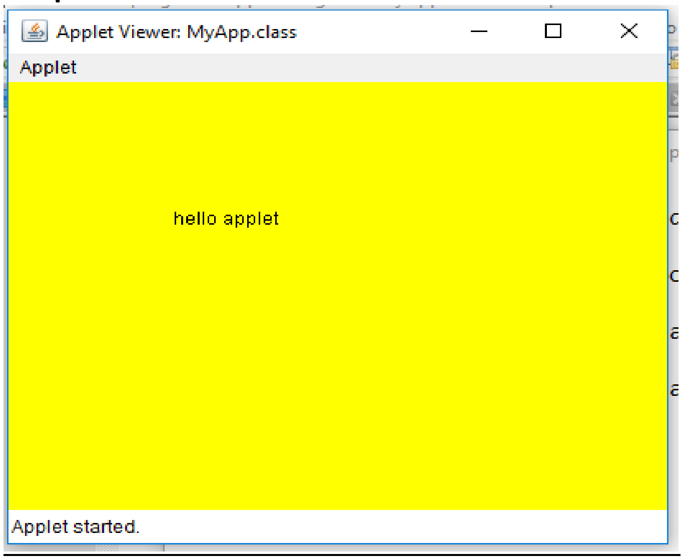
<u>MyApp.html</u>

```
<html>
  <applet code="MyApp.class" height="300" width="400">
  </applet>
</html>
```

Compilation: `javac MyApp.java`

Execution: `appletviewer MyApp.html`

Output:



- we can also execute the above applet program without html file by including applet tag inside a program within comment section.

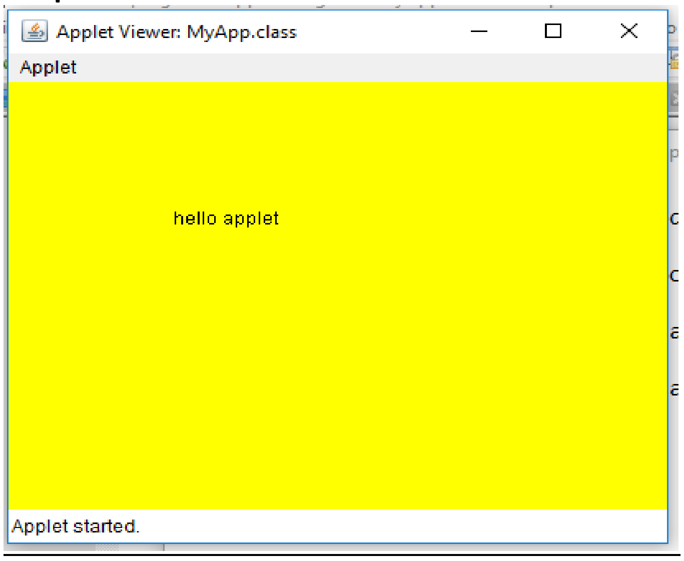
  <u>**MyApp.java**</u>

```
import java.applet.*;
import java.awt.*;
public class MyApp extends Applet
{
    public void init()
     {
         setBackground(Color.yellow);
     }
   public void paint(Graphics g)
    {
       g.drawString("hello applet",100,100);
    }
}
/*
<applet code="MyApp.class" height="300" width="300">
  </applet>
*/
```

  Compilation: `javac MyApp.java`

  Execution: `appletviewer MyApp.java`

  Output:



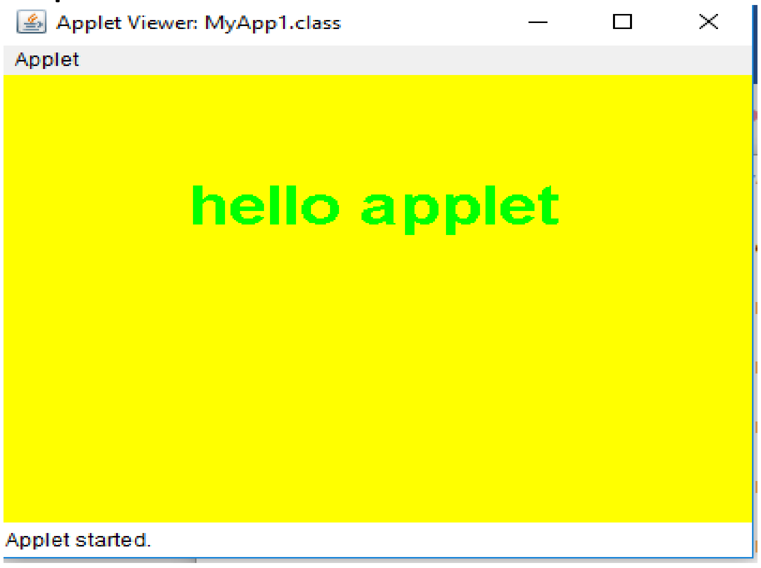
Example2:

```
import java.applet.*;
import java.awt.*;
public class MyApp1 extends Applet
{
    public void init()
     {
         setBackground(Color.yellow);
     }
    public void paint(Graphics g)
    {
        Font f=new Font("Arial",Font.BOLD,36);
        g.setFont(f);
```

```
            g.setColor(Color.green);
            g.drawString("hello applet",100,100);
        }
}
/*
<applet code="MyApp1.class" width="400" height="300"></applet>
*/
```
Compilation: `javac MyApp.java`

Execution: `appletviewer MyApp.java`

Output:



hello applet

- java.awt.Graphics class provides many methods for graphics programming.

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
9. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Passing parameters to applet:

- We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter().

```
public String getParameter(String parameterName)
```

Example:

UseParam.java:

```
import java.applet.Applet;
import java.awt.Graphics;
public class UseParam extends Applet
{
      public void paint(Graphics g)
      {
            String str=getParameter("msg");
            g.drawString(str,50, 50);
      }
}
```

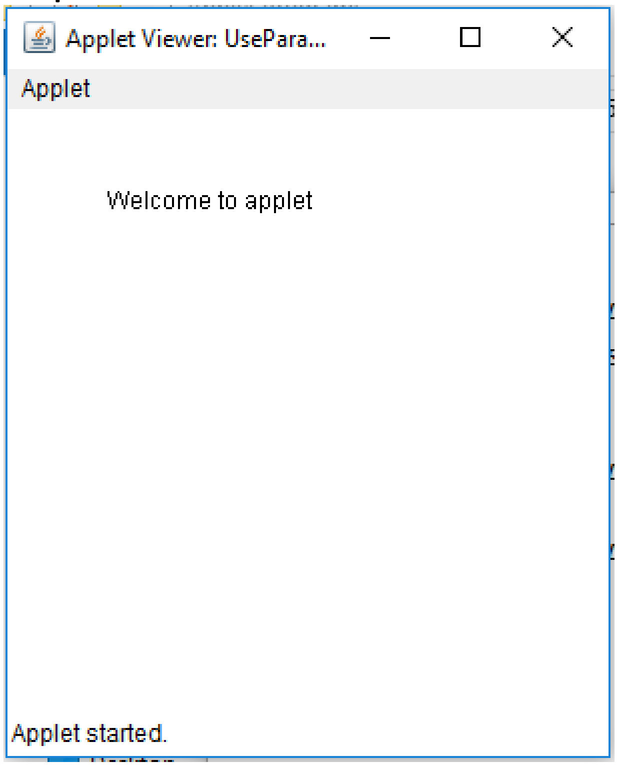
UseParam.html

```
<html>
      <body>
            <applet code="UseParam.class" width="300" height="300">
            <param name="msg" value="Welcome to applet">
            </applet>
      </body>
</html>
```

Compilation: `javac MyApp.java`

Execution: `appletviewer MyApp.java`

Output:



```
Welcome to applet
```

```
Applet started.
```

Applet to applet Communication:

java.applet.AppletContext class provides the facility of communication between applets. We provide the name of applet through the HTML file. It provides getApplet() method that returns the object of Applet.

```
      public Applet getApplet(String name){}
```

Example:

RequestApp.java:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class RequestApp extends Applet implements ActionListener
{
      public void init()
      {
            Button b=new Button("Click");
            b.setBounds(50,50,50,50);
            add(b);
            b.addActionListener(this);
      }
      public void actionPerformed(ActionEvent e)
      {
            AppletContext ctx=getAppletContext();
            Applet a=ctx.getApplet("response");
            Font f=new Font("Arial",Font.BOLD,36);
            a.setFont(f);
            a.setBackground(Color.green);
      }
}
```

ResponseApp.java:

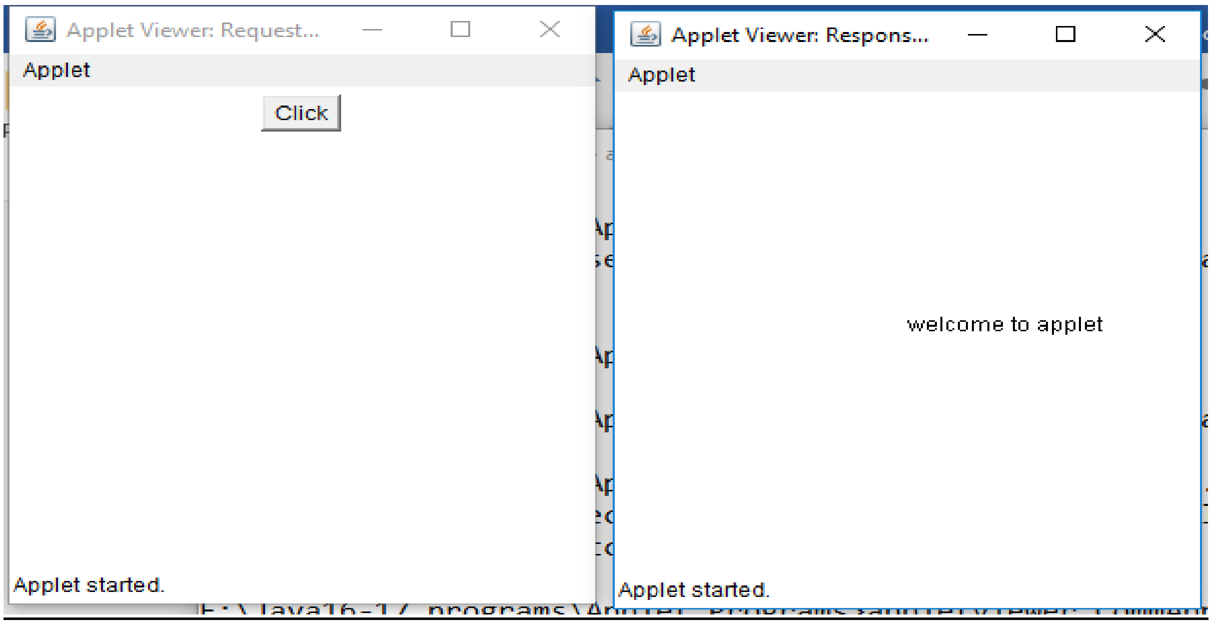
```
import java.applet.*;
import java.awt.*;
public class ResponseApp extends Applet
{
    public void paint(Graphics g)
      {
          g.drawString("welcome to applet",150,150);
      }
}
```

CommApp.html:

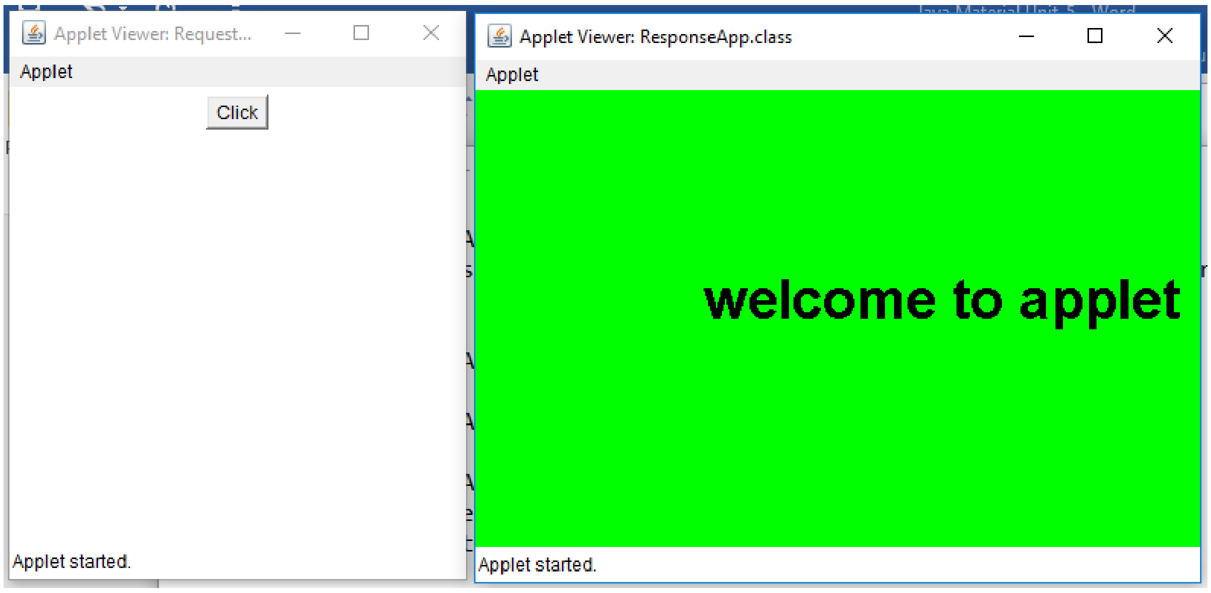
```
<html>
<body>
<applet code="RequestApp.class" width="300" height="300"
name="request">  </applet>

<applet code="ResponseApp.class" width="300" height="300"
name="response">  </applet>
</body>
</html>
```

Compilation: `javac RequestApp.java`
`javac ResponseApp.java`
Execution: `appletviewer CommApp.html`
Output:
Before button click:



After button click:



****