Unit-4

❖ Exception Handling :

➢ Exceptions:-

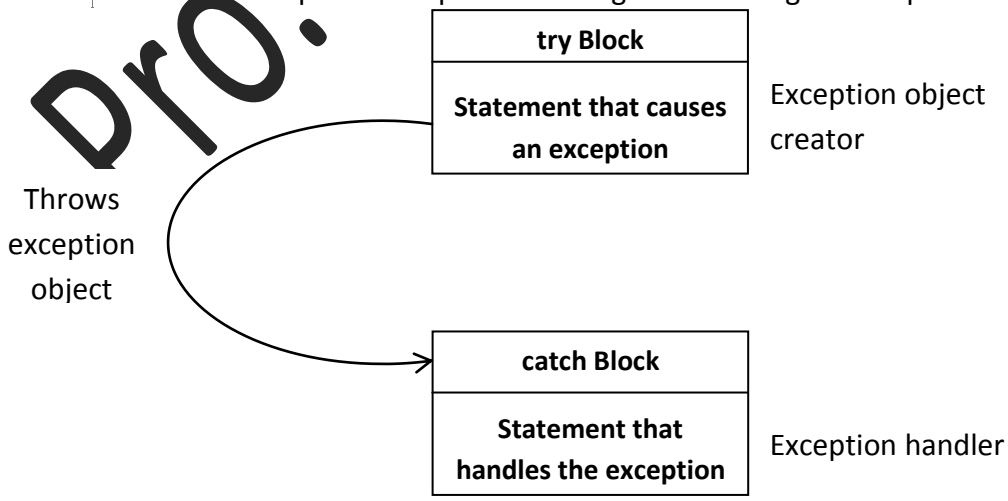
- ✓ An exception is a condition that is caused by a run-time error in the program.
- ✓ When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it(informs us that an error has occurred).
- ✓ If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program.
- ✓ If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object through by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.
- ✓ Error handling code that performs the following tasks:
    1. Find the problem(Hit the exception)
    2. Inform that an error has occurred(Throw the exception)
    3. Receive the error information(Catch the exception)
    4. Take corrective actions(Handle the exception)
- ✓ The error handling code basically consists of two segments, one to detect and to throw exceptions and the other to catch exceptions and to take appropriate actions.
- ✓ When writing programs, we must always be on the lookout for places in the program where an exception could be generated.
- ✓ Exceptions in java can be categorized into two types:
    1. **Checked Exceptions :**
        - ✓ These exceptions are explicitly handled in the code itself with the help of try catch blocks.
        - ✓ Checked exceptions are extended from the java.lang.Exception class.
    2. **Unchecked Exceptions :**
        - ✓ These exceptions are not essentially handled in the program code, instead the JVM handles such exceptions.
        - ✓ Unchecked exceptions are extended from the java.lang.RuntimeException class.
- ✓ It is important to note that checked and unchecked exceptions are absolutely similar as far as their functionality is concerned, the difference is only in the way they are handled.

- ✓ Common Java Exceptions

Exception Type	Cause of Exception
ArithmaticException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundsException	Caused by bad array indexes
ArrayStoreException	Caused when a program tries to store the wrong type of data in an array
FileNotFoundException	Caused by an attempt to access a nonexistent file
IOException	Caused by general I/O failures, such as inability to read from a file
NullPointerException	Caused by referencing a null object
NumberFormatException	Caused when a conversion between strings and number fails
OutOfMemoryException	Caused when there's not enough memory to allocate a new object
SecurityException	Caused when an applet tries to perform an action not allowed by the browser's security setting
StackOverFlowException	Caused when the system runs out of stack space
StringIndexOutOfBoundsException	Caused when a program attempts to access a nonexistent character position in a string

➢ Syntax of Exception Handling Code:-

- ✓ The basic concepts of exception handling are throwing an exception and catching it.



Exception handling mechanism

- ✓ Java uses a keyword try to preface a block of code that is likely to cause an error condition and "throw" an exception.
- ✓ A catch block defined by the keyword catch "catches" the exception "thrown" by the try block and handles it appropriately.
- ✓ The catch block is added immediately after the try block.
- ✓ Example of simple try and catch statements:

```
……….
……….
try
{
                statement;          //generates an exception
}
catch(Exception-type e)
{
                Statement;          //processes the exception
}
……….
……….
```

- ✓ The try block can have one or more statements that could generate an exception.
- ✓ If any one statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block that is placed next to the try block.
- ✓ The catch block too can have one or more statements that are necessary to process the exception.
- ✓ Remember that every try statement should be followed by at least one catch statement; otherwise compilation error will occur.
- ✓ Note that the catch statement works like a method definition.
- ✓ The catch statement is passed a single parameter, which is reference to the exception object thrown by the try block.
- ✓ **Program: use of try and catch blocks to handle an arithmetic exception**

```
class Error3
{
                public static void main(String args[])
                {
                    int a=10;
                    int b=5;
                    int c=5;
                    int x,y;
                    try
                    {
                            x=a/(b-c);        //Exception here
```

```
            }
            catch(ArithmaticException e)
            {
                    System.out.println("Division by Zero");
            }
            y=a/(b+c);
            System.out.println("Y="+y);
        }
}
```

Output :

```
Division by zero
Y=1
```

- ✓ There could be situations where there is a possibility of generation of multiple exceptions of different types within a particular block of the program code.
- ✓ We can use nested try statements in such situations.
- ✓ The execution of the corresponding catch blocks of nested try statements is done using a stack.
- ✓ **Program: Nested try statements**

```
class Nestedtry

{

        public static void main(string args[])

        {

                try

                {

                        int a=2,b=4,c=2,x=7,z;

                        int p[]={2};

                        p[3]=33;

                        try

                        {

                                z=x/((b*b)-(4*a*c));
```

```
                System.out.println("The value of z is="+z);
            }
            catch(ArithmaticException e)
            {
                System.out.println("Division by zero Exception");
            }
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Array index out of bounds");
        }
    }
}
```

Output :

```
Array index out of bounds
```

➢ Multiple Catch Statements:-

✓ It is possible to have more than one catch statement in the catch blocks as shown below:

```
.....
try
{
```

```
        Statement;      //generates an exception
    }
    catch(Exception-Type-1 e)
    {
        Statement;      //processes exception type 1
    }
    catch(Exception-Type-2 e)
    {
        Statement;      //processes exception type 2
    }
    .
    .
    catch(Exception-Type-N e)
    {
        Statement;      //processes exception type N
    }
    ……..
    ……..
```

- When an exception in a try block is generated, the java treats the multiple catch statements like cases in a switch statement.
- ✓ The first statement whose parameter matches with the exception object will be executed, and the remaining statements will be skipped.
- ✓ Note that java does not require any processing of the exception at all.
- ✓ We can simply have a catch statement with an empty block to avoid program abortion.
- ✓ **Example :**

```
catch(Exception e);
```

- ✓ The catch statement simply ends with a semicolon, which does nothing. This statement will catch an exception and then ignore it.

- ✓ **Program : Using multiple catch blocks**

```
class MultipleCatch
{
        public static void main(string args[])
        {
                int a[]={5,10}
                int b=10;
                try
                {
                        int x=a[2]/b-a[1];
                }
                catch(ArithmaticException e)
                {
                        System.out.println("Division by Zero…");
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                        System.out.println("Array Index Error…");
                }
                catch(ArrayStoreException e)
                {
                        System.out.println("Wrong Data Type…");
                }
                int y=a[1]/a[0];
```

```
            System.out.println("Y="+y);
        }
    }
```

Output:

```
Array Index Error…
Y=2
```

➢ Using Finally Statement:-

- ✓ Java supports another statement known as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements, finally block can be used to handle any exception generated within a try block.
- ✓ It may be added immediately after the try block or after the last catch block shown as below:

```
try
{
        …….
        …….
}
finally
{
        …….
        …….
}
```

```
try
{
        …….
        …….
}
catch(…..)
{
        …….
        …….
}

finally
{
        …….
        …….
}
```

- ✓ When a finally block is defined, this is guaranteed to execute, regardless of whether or not in exception is thrown.
- ✓ We can use it to perform certain house-keeping operations such as closing files and releasing system resources.

➢ Throwing Our Own Exceptions:-

- ✓ There may be times when we would like to throw our own exceptions.
- ✓ We can do this by using the keyword throw as below:

  throw new Throwable subclass;

- ✓ **Example :**

```
throw new ArithmaticException();
throw new NumberFormatException();
```

- ✓ **Program : Throwing Our Own Exception**

```
import java.lang.Exception;
class MyException extends Exception
{
        MyException(String message)
        {
                super(message);
        }
}
class TestMyException
{
        public static void main(Strings args[])
        {
                int x=5,y=1000;
                try
                {
                        float z=(float)x/(float) y;
                        if(z<0.01)
                        {
                                throw new MyException("Number is too small");
                        }
                }
                catch(MyException e)
                {
                        System.out.println(Caught My Exception");
```

```
                System.out.println(e.getMessage());
            }
            finally
            {
                System.out.println("I am always here");
            }
        }
}
```

Output :

Caught My Exception

Number is too small

I am always here

- ✓ In program Exception is a subclass of Throwable and therefore MyException is a subclass of Throwable class.
- ✓ An object of a class that extends Throwable can be thrown and caught.

- ✓ There could be situations where there is a possibility that a method might throw certain kinds of exceptions but there is no exception handling mechanism common within the method.
- ✓ The throws clause is used in such a situation.
- ✓ It is specified immediately after the method declaration statement and just before the opening brace.
- ✓ **Program: Use of Throws**

```
class Examplethrows
{
    static void divide_m() throws ArithmaticExxception
    {
        int x=22,y=0,z;
        z=x/y;
    }
    public static void mai(String args[])
    {
        try
        {
```

```
                Divide_m();
            }
            catch(ArithmaticException e)
            {
                System.out.println("Caught the exception"+e);
            }
        }
}
```

Output :

Caught the exception java.lang.ArithmaticException: / by zero

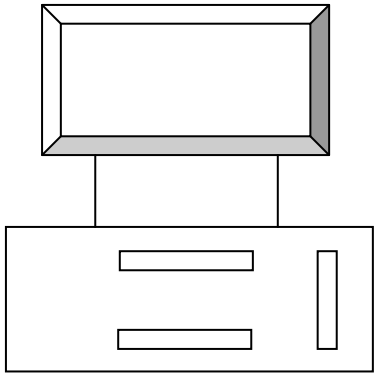
❖ Applets :

➢ Introduction:-

- ✓ Applets are small java programs that are primarily used in Internet computing.
- ✓ They can be transported over the Internet from one computer to another and run using the Applet Viewer or any Web browser that supports Java.
- ✓ Applet can perform arithmetic operations, display graphics, play sounds, accept user input, create animation and play games.
- ✓ Java has enabled them to create and use fully interactive multimedia web documents.
- ✓ Java applet make a significant impact on the world wide web.

➢ Local and Remote Applets:-

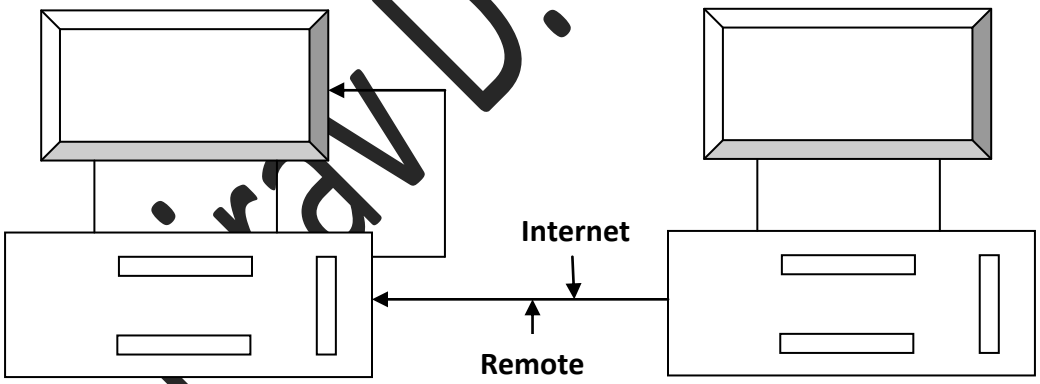
- ✓ Applets are small java programs that are primarily used in Internet computing.
- ✓ We can place applets into Web pages in two ways.
- ✓ One, we can write our own applets and embed them into Web pages.
- ✓ Second, we can download an applet from a remote computer system and then place it into a Web page.
- ✓ An applet developed locally and stored in a local system is known as a **local applet.**
- ✓ When a web page is trying to find a local applet, it does not need to use the internet and therefore the local system does not require the internet connection.
- ✓ It simply searches the directories in the local system and locates and loads the specified applet.



Local Computer

[Loading Local Applets]

- ✓ A **remote applet** is that which is developed by someone else and store on a remote computer connected to the internet.
- ✓ If our system is connected to the Internet, we can download to the Internet, we can download the remote applet onto our system via at the Internet and run it.



Local Computer

(Client)

Remote Computer

(Server)

- ✓ In order to locate and load a remote applet, we must know the applet's address on the web.
- ✓ This address is known as Uniform Resource Locator(URL) and must be specified in the applet's HTML document as the value of the CODEBASE attribute.
- ✓ **Example :**

  CODEBASE=http://www.netserve.com/applets
- ✓ In the case of local applets, CODEBASE may be absent or may specify a local directory.

➢ Preparing To Write Applets:-

- ✓ Before we try to write applets, we must make sure that java is installed properly and also ensure that either the java appletviewer or a java-enabled browser is available.
- ✓ The steps involved in developing and testing in applet are:
    1. Building an applet code(.java file)
    2. Creating an executable applet(.class file)
    3. Designing a Web page using HTML tags
    4. Preparing <APPLET> tag
    5. Incorporating <APPLET> tag into the Web page
    6. Creating HTML file
    7. Testing the applet code

➢ Building Applet Code:-

- ✓ It is essential that our applet code uses the services of two classes, namely, **Applet** and **Graphics** from the Java class library.
- ✓ The **Applet** class which is contained in the **java.applet** package provides life and behavior to the applet through its methods such as **init(), start()** and **paint().**
- ✓ The **paint()** method of the **Applet** class, when it is called actually displays the result of the applet code on the screen.
- ✓ The output may be text, graphics or sound.
- ✓ The **paint()** method, which requires a Graphics object as an argument, is defined as follows:

```
public void paint(Graphics g)
```

- ✓ This requires that the applet code imports the **java.awt** package that contains the **Graphics** class.
- ✓ All output operations of an applet are performed using the methods defined in the **Graphics** class.
- ✓ Applet code will have a general format as shown below:

```
import java.awt.*;
import java.applet.*;
……
……
public class appletclassname extends Applet
{
        ……
        ……
        public void paint(Graphics g)
        {
                ……
                ……    //Applet Operation Code
```

```
        }
    }
```

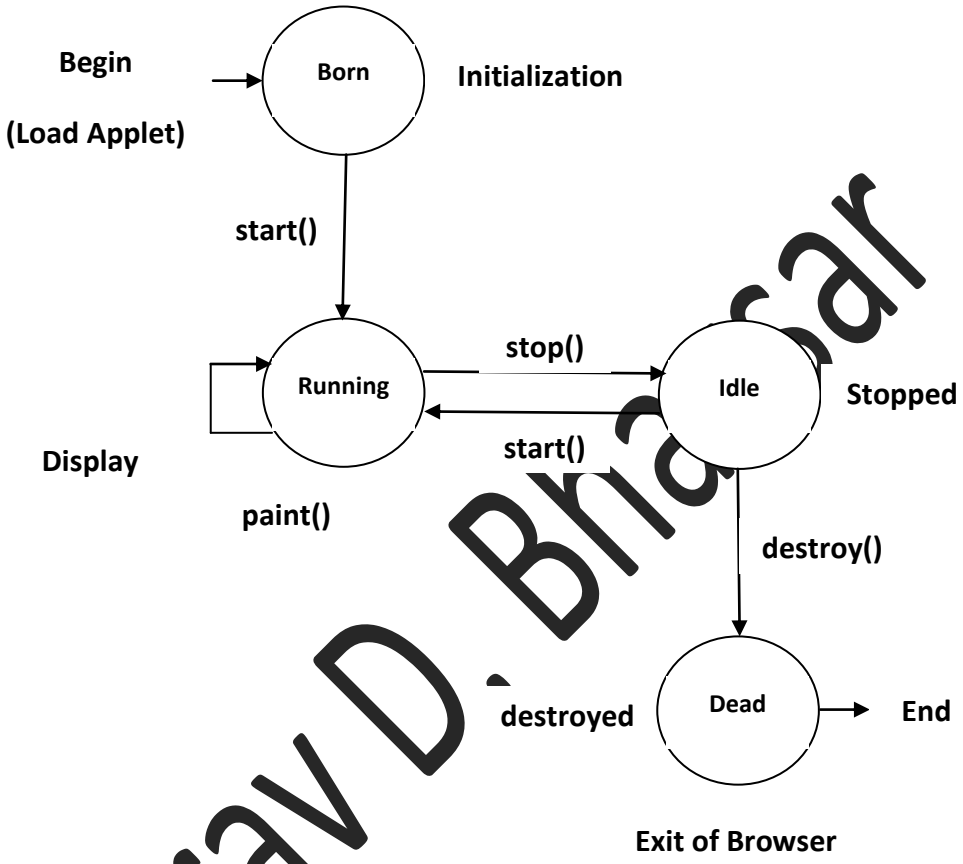
- ✓ The appletclassname is the main class for the applet.
- ✓ When the applet is loaded, Java creates an instance of this class, and then a series of **Applet** class methods are called on that instance to execute the code.
- ✓ **Example:**

```
import java.awt.*;
import java.applet.*;
public class HelloJava extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello Java…", 10, 100);
    }
}
```

- ✓ Remember that the applet code in above example should be saved with the file name HelloJava.java, in a java subdirectory.
- ✓ Note that public keyword for the class HelloJava.
- ✓ Java requires that the main applet class be declared public.

➢ Applet Life Cycle:-

- ✓ The Applet States include:
  - Born on initialization state
  - Running State
  - Idle State
  - Dead or destroyed state



[An Applet's State Transition Diagram]

- **Initialization State:-**

✓ Applet enters the initialization state when it is first loaded.
✓ This is achieved by calling the init() method of Applet Class.
✓ The Applet is born.
✓ At this stage, we may do the following:
  - Create Objects needed by the applet
  - Set up initial values
  - Load images or fonts
  - Set up colors
✓ The initialization occurs only once in the applet's life cycle.
✓ We must override the init() method:

```
public void init()
{
        .....
        .....(Action)
}
```

- **Running State:-**

✓ Applet enters the running state when the system calls the start() method of Applet class.
✓ This occurs automatically after the applet is initialized.
✓ Starting can also occur if the applet is already in 'stopped' (idle) state.
✓ **Example :** We may leave the Web page containing the applet temporarily to another page and return back to the page.
✓ This again starts the applet running.
✓ The start() method may be called more than once.
✓ We may override the start() method to create a thread to control the applet.

```
public void start()

{

        .....

        .....(Action)

}
```

- **Idle or Stopped State:-**

✓ An Applet becomes idle when it is stopped from running.
✓ Stopping occurs automatically when we leave the page containing the currently running applet.
✓ We can also do so by calling the stop() method explicitly.
✓ If we use a thread to run the applet, then we must use stop() method to terminate the thread.
✓ We can achieve this by overriding the stop() method:

```
public void stop()
{
        .....
```

```
        …..(Action)
    }
```

○ Dead State:-

- ✓ An applet is said to be dead when it is removed from memory.
- ✓ This occurs automatically by invoking the destroy() method when we quit the browser.
- ✓ Like initialization, destroying stage occurs only one in the applet's life cycle.
- ✓ If the applet has created any resources, like threads, we may override the destroy() method to clean up these resources.

```
    public void destroy()
    {
        …..
        …..(Action)
    }
```

○ Display State:-

- ✓ Applet moves to the display state whenever it has to perform some output operation on the screen.
- ✓ This happens immediately after the applet enters into the running state.
- ✓ The paint() method is called to accomplish this task.
- ✓ Almost every applet will have a paint() method.
- ✓ Like other methods in the life cycle, the default version of paint() method does absolutely nothing.
- ✓ We must therefore override this method if we want anything to be displayed on the screen.

```
    public void paint (Graphic g)
    {
        …..
        …..(Display Statements)
    }
```

- ✓ It is to be noted that the display state is not considered as a part of the applet's life cycle.
- ✓ In fact, the paint() method is defined in the Applet class.
- ✓ It is inherited from the Component class, a super class of Applet.

➢ Creating An Executable Applet:-

- ✓ Executable applet is nothing but the .class file of the applet, which is obtained by compiling the source code of applet.
- ✓ Compiling an applet is exactly the same as compiling an application.
- ✓ Therefore, we can use the java compiler to compile the applet.

- ✓ **Example :**

```
import java.awt.*;
import java.applet.*;
public class HelloJava extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("Hello Java..", 10, 100);
        }
}
```

- ✓ Let us consider the HelloJava applet.
- ✓ This applet has been stored in a file called HelloJava.java.
- ✓ Here are the steps required for compiling the HelloJava applet.
    1. Move to the directory containing the source code and type the following command:
        javac HelloJava.java
    2. The compiled output file called HelloJava.class is placed in the same directory as the source.
    3. If any error message is received, then we must check for errors, correct them and compile the applet again.

➢ Applet Tag:-

- ✓ We have included a pair of <APPLET…> and </APPLET> tags in the body section.
- ✓ The <APPLET…> tag supplies the name of the applet to be loaded and tells the browser how much space the applet requires.
- ✓ The <APPLET> tag given below specifies the minimum requirements to place the HelloJava applet on a Web page:

```
<APPLET
        CODE=HelloJava.class
```

```
        WIDTH=400
        HEIGHT=200>
    </APPLET>
```

- ✓ This HTML code tells the browser to load the compiled Java applet HelloJava.class, which is in the same directory as the HTML file.
- ✓ And also specifies the display area for the applet output as 400 pixels width and 200 pixels height.
- ✓ We can make this display area appear in the centre of the screen by using the CENTER tags shown as follows:

```
<CENTER>

    <APPLET

        .....

        .....

    </APPLET>

</CENTER>
```

➢ Adding Applet To HTML File:-

- ✓ Now we can out together the various components of the Web page and create a file known as HTML file.
- ✓ Insert the <APPLET> tag in the page at the place where the output must appear.
- ✓ Following is the content of the HTML file that is embedded with the <APPLET> tag of our HelloJava applet.

```
<HTML>
    <HEAD>
        <TITLE>
            Welcome to Java Applets
        </TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>Welcome to the world of Applets</H1>
        </CENTER>
        <BR>
        <CENTER>
            <APPLET
                CODE=HelloJava.class
                WIDTH=400
```

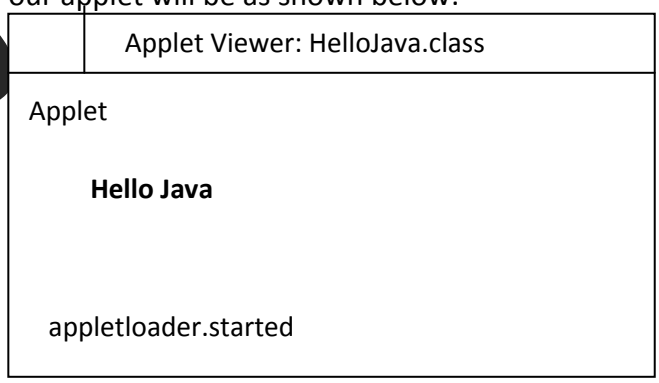
```
                    HEIGHT=200>
                </APPLET>
            </CENTER>
        </BODY>

    </HTML>
```

- ✓ We must name this file as HelloJava.html and save it in the same directory as the compiled applet.

➢ Running The Applet:-

- ✓ Now that we have created applet files as well as the HTML file containing the applet, we must have the following files in our current directory:
  - HelloJava.java
  - HelloJava.class
  - HelloJava.html
- ✓ To run an applet , we require one of the following tools:
  1. Java-enabled Web browser(Such as HotJava or Netscape)
  2. Java appletviewer
- ✓ If we use a Java-enabled Web browser, we will be able to use the entire Web page containing the applet.
- ✓ If we use the appletviewer tool, we will only see the applet output.
- ✓ Remember that the appletviewer is not a full-fledged Web browser and therefore it ignores all of the HTML tags except the part pertaining to the running of the applet.
- ✓ The appletviewer is available as a part of the Java Development Kit.
- ✓ We can use it to run our applet as follows:
  - appletviewer HelloJava.html
- ✓ Notice that the argument of the appletviewer is not the .java file or the .class file, but rather .html file.
- ✓ The output of our applet will be as shown below:



➢ Passing Parameters To Applets:-

- ✓ We can supply user-defined parameters to an applet using <PARAM…> tags.
- ✓ Each <PARAM> tag has a **name** attribute such as **color**, and a **value** attribute such as **red**.
- ✓ **Example:** We can change the colour of the text displayed to red by an applet by using a <PARAM…> tag as follows:

```
<APPLET>
<PARAM=color VALUE="red">
</APPLET>
```

- ✓ Similarly, we can change the text to be displayed by an applet by supplying new text to the applet through a <PARAM…> tag as shown below:

```
<PARAM NAME=text VALUE="I Love Java">
```

- ✓ Passing parameters to an applet code using <PARAM> tag is something similar to passing parameters to the main() method using command line arguments.
- ✓ To set up and handle parameters, we need to do two things:
  1. Include appropriate <PARAM…> tags in the HTML document.
  2. Provide Code in the applet to parse these parameters.
- ✓ Parameters are passed on an applet when it is loaded.
- ✓ We can define the init() method in the applet to get hold of the parameters defined in the <PARAM> tags.
- ✓ This is done using the getParameter() method, Which takes one string argument representing the name of the parameter and returns a string containing the value of that parameter.
- ✓ **Program:**

Applet HelloJavaParam

```
import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
{
    String str;
    public void init()
    {
        str=getParameter("string");
        if(str==null)
            str="Java";
        str="Hello"+str;
    }
    public void paint(Graphics g)
    {
        g.drawString(str, 10,100);
    }
}
```

- ✓ Now, let us create HTML file that contains this applet.
- ✓ Below Program shows a Web page that passes a parameter whose NAME is "string" and whose VALUE is "APPLET!" to the applet HelloJavaParam.

The HTML file for HelloJavaParam applet

```
<HTML>
    <HEAD>
        <TITLE>Welcome to Java Applets</TITLE>
    </HEAD>
    <BODY>
        <APPLET CODE=HelloJavaParam.class
            WIDTH=400
            HEIGHT=200>
            <PARAM NAME="string"
                VALUE="Applet!">
        </APPLET>
    </BODY>
</HTML>
```

- ✓ Save this file as HelloJavaParam.html and then run the applet using the applet viewer as follows:

  Appletviewer HelloJavaParam.html

❖ Graphics Class :

➢ Introduction:-

- ✓ One of the most important features of Java is its ability to draw graphics.
- ✓ We can write Java applets that draw lines, figures of different shapes, images, and text in different fonts and styles.
- ✓ We can also add in different colours in display.
- ✓ Every applet has its own area of the screen known as canvas, where it creates its display.

- ✓ The size of an applet's space is decided by the attributes of the <APPLET…> tag.
- ✓ A java applet draws graphical image inside its space using the coordinate system.
- ✓ Java's coordinate system has the origin (0,0) in the upper-left corner.
- ✓ Positive x values are to the right, and positive y values are to the bottom.
- ✓ The values of coordinates x and y are in pixels.

➢ The Graphics Class:-

- ✓ Java's Graphics class includes methods for drawing many different types of shapes, from simple lines to polygons to text in a variety of fonts.
- ✓ To draw a shape on the screen, we may call one of the methods available in the Graphics class.
- ✓ All the drawing methods have arguments representing end points, corners, or starting locations of a shape as values in the applet's coordinate system.
- ✓ To draw a shape, we only need to use the appropriate method with the required arguments.
- ✓ **Drawing Methods of the Graphics Class**

Method	Description
clearRect()	Erase a rectangular area of the canvas.
copyArea()	Copies a rectangular area of the canvas to another area.
drawArc()	Draws a hollow arc.
drawLine()	Draws a straight line.
drawOval	Draws a hollow oval.
drawPolygon()	Draws a hollow polygon.
drawRect()	Draws a hollow rectangle.
drawRoundRect()	Draws a hollow rectangle with rounded corners.
drawString()	Displays a text string.
fillArc()	Draws a filled arc.
fillOval()	Draws a filled oval.
fillPolygon()	Draws a filled polygon.
fillRect()	Draws a filled rectangle.
fillRoundRect()	Draws a filled rectangle with rounded corners.
getColor()	Retrieves the current drawing colour.

getFont()	Retrieves the currently used font.
getFontMetrics()	Retrieves information about the current font.
setColor()	Sets the drawing colour.
setFont()	Sets the font.

- ✓ **Program: Using Methods of Graphics Class**

```
<html>
    <body>
        <applet code=graphics_methods.class width=200 height=200>
        </applet>
    </body>
</html>

import java.awt.*;
import java.applet.*;
public class graphics_methods extends Applet
{
    String s=new String();
    String s1=new String();
    String s2=new String();

    Font f1=new Font("Courier New",Font.BOLD,20);

    public void paint(Graphic g)
    {
        g.setFont(f1);
        g.setColor(Color.blue);
        g.drawString("Illustration of methods of Graphics class",200,520);
        Font f2=g.getFont();
        s=f2.toString();
        g.drawString(s,5,540);
        g.setColor(Color.green);
        Color col=g.getColor();
        s2=col.toString();
        g.drawString(s2,5,560);
        g.fillRect(500,15,70,90);
        g.drawRect(160,5,60,60);
        g.drawOval(10,120,155,95);
        g.setColor(Color.yellow);
```

```
                g.fillOval(700,140,50,150);
                g.setColor(Color.black);
                g.drawLine(380,100,200,180);
                g.drawArc(400,150,180,280,90,70);
                int x2[]={200,120,280,240};
                int z2=4,y2[]={260,370,370,270};
                g.setColor(Color.blue);
                g.fillPolygon(x2,y2,z2);
                g.setColor(Color.red);
                g.drawRect(15,15,30,50);
                FontMetrics f3=g.getFontMetrics();
                S1=f3.toString();
                g.drawString();
                g.setColor(Color.magenta);
                g.fillRoundRect(510,400,90,80,20,20);

        }

    }
```