UNIT IV

MANAGING EXCEPTIONS, APPLETPROGRAMMING

ERROR - The mistakes which we have done while developing our program, those mistakes are called errors **[5 Marks]**

There are two types of error as follows:

1. Compile time errors
2. Run time errors

1 COMPILE TIME ERRORS

Compile time errors Compiler time error means Java compiler identify the syntax error at the time of compilation. And without successfully compilation, compiler does not create .class file. That means we have to compile the program which should be error free and then compiler creates .class file of the program and then we can run the program.

The common problems are:

- Missing braces
- Missing semicolon
- Missing double quote in string
- = instead of == operator
- And so on.

For example:

```
class Try1
{
        public static void main(String args[])
        {
                int a=12;
                int b=0;
                int c=a/b
                System.out.println("Division is+c);
        }
}
```

Output:

```
C:\cc>javac  Try1.java
Try1.java:8: ';' expected
System.out.println("Division is+c);
^
Try1.java:8: unclosed string literal
System.out.println("Division is+c);
^
2 errors
```

2 RUN TIME ERRORS

Several time program may compile successfully and compiler creates the .class file of the program but when the time of running the program, it shows the error and that type of error called run time error.

The common problems are:

- Divide by zero
- Conversion of invalid string to number
- access the element that is out of bound of an array
- Passing the parameters with invalid range.
- And so on.

For example:

write a program to find out division of two numbers.

```
class Try1
{
	public static void main(String args[])
	{
		int a=12;
		int b=0;
		int c=a/b;
		System.out.println("Division is"+c);
	}
}
```

Output:

```
C:\cc>javac Try1.java
C:\cc>java Try1
Exception in thread "main" java.lang.ArithmeticException: / by
zero at Try1.main(Try1.java:7)
```

EXCEPTION – A run time errors are called exception. Or abnormal termination of the program is called exception. **[2 Marks]**

EXCEPTION HANDLING - Handling run time errors is called exception handling.

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained. **[2 Marks]**

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

TYPES OF EXCEPTION **[2 Marks]**

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun micro system says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

TO HANDLE EXCEPTION IN JAVA WE USE 5 KEYWORD [5 Marks]

1. **try**
2. **catch**
3. **finally**
4. **throw**
5. **throws**

1 Java try block [2 Marks]

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch [2 Marks]

```
try
{
//code that may throw exception
}
catch(Exception_class_Name ref)
{
}
```

Syntax of try-finally block

```
try
{
//code that may throw exception
}
finally
{
}
```

2 catch block **[2 Marks]**

Java catch block is used to handle the Exception. It must be used after the try block only. You

can use multiple catch block with a single try.

Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1
{
        public static void main(String args[])
        {
                int data=50/0;//may throw exception
                 System.out.println("rest of the code...");
        }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed. Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

Let's see a simple example of java multi-catch block.

```
public class TestMultipleCatchBlock
{
        public static void main(String args[])
        {
                Try
                {
                        int a[]=new int[5]; a[5]=30/0;

                }

                catch(ArithmeticException e)
                {
                        System.out.println("task1 is completed");
                }
                 catch(ArrayIndexOutOfBoundsException e)
```

```
                {
                        System.out.println("task 2 completed");
                }
                catch(Exception e)
                {
                        System.out.println("common task completed");
                }

                System.out.println("rest of the code...");
        }
}
```

Output:task1 completed
rest of the code...

3 The finally Block **[2 Marks]**

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax −

Syntax

```
try {
  // Protected code
}catch(ExceptionType1 e1)
 {
  // Catch block
}catch(ExceptionType2 e2)
 {
  // Catch block
}catch(ExceptionType3 e3)
 {
  // Catch block
}
finally
{
  // The finally block always executes.
}
```

Example

```
public class ExcepTest
{
        public static void main(String args[])
        {
                int a[] = new int[2];
                try
                {
                        System.out.println("Access element three :" + a[3]);
                }

                }catch(ArrayIndexOutOfBoundsException e)
                {
                        System.out.println("Exception thrown :" + e);
                }
                finally
                {
                        a[0] = 6;
                        System.out.println("First element value: " + a[0]); System.out.println("The
                        finally statement is executed");
                }
         }
}
```

This will produce the following result −

Output

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

4 throw keyword **[2 Marks]**

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throws keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
throw exception;
```

Let's see the example of throw IOException.

```
throw new IOException("sorry device error); java
```

throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
public class TestThrow1
{
        void validate(int age)
        {
                if(age<18)
                        throw new ArithmeticException("not valid");
                 else
                        System.out.println("welcome to vote");
        }
}
class Demo
{
         public static void main(String args[])
        {
                TestThrow1 obj=new TestThrow1();
                Obj.validate(13); System.out.println("rest
                of the code...");
        }
}
```

Output:

Exception in thread main java.lang.ArithmeticException:not valid

5 throws keyword **[2 Marks]**

The Java throws keyword is used to declare an exception. It gives information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

```
return_type method_name() throws exception_class_name
{
//method code
}
```

Example

```
class Error
{
  void divide() throws ArithmeticException
```

```
	{
		int c=10/0;
	}
}

class ExceptionHandling
{
	public static void main(String args[])
	{
		System.out.println("Connection Established");
		Error obj=new Error();
		try
		{
			obj.divide();
		}
		catch(ArithmeticException ae)
		{
			System.out.println("Divide by zero error");
		}
		finally
		{
			System.out.println("Connection Terminated");
		}
	}
}
```

OUTPUT

```
C:\Program Files\Java\jdk1.7.0\bin>javac ExceptionHandling.java
C:\Program Files\Java\jdk1.7.0\bin>java ExceptionHandling
Connection Established
Divide by zero error
Connection Terminated
```

APPLET PROGRAMMING

Applet - 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 Applet Viewer or any web browser that supports java. **[2 Marks]**

Applets are small applications that are accessed from an internet server, transported over the Internet, automatically installed and run as part of a web document. After an applet arrives on the client it has limited access to resources, so that it can produce an arbitrary multimedia user interface and run complex computations without introducing the role of viruses or breaching data integrity.

LOCAL AND REMOTE APPLETS

Java provides us the facility for both creating CUI and GUI Programs All the Previous Topics are related with the CUI but Applets Provides the ability to user for creating Graphical Programs Like Creating Buttons, Creating Events such that Executing the Code when a user Clicks on Button Etc. There are two type of Applets like Stand Alone or either Local Applets The Applets those are Created on Client Machine are Called as Local Applets and the Applets Those are located at Remote Computer but we are using them then they are Called as Remote Applets. Applets are used for creating Graphical Programs.

HOW APPLETS DIFFER FROM APPLICATIONS [2 Marks]

There are significant differences between applets and stand alone java applications. Applets are not full featured application programs. Applets are usually written to accomplish a small task or a component of a task. They are designed for use on the Internet, they impose certain limitations and restrictions in their design.

1. main() method is not used in any Applet as it is not required for initiation the execution of the code, where as , Applet when loaded, automatically call certain methods of Applet class to start and execute the applet code.
2. Applets are run from inside a web page using a special feature known as HTML tag. Where as application run stand-alone and independently on web.
3. Files that are located in the local computer cannot be read or write by the applets.
4. Applets cannot communicate with other servers on the network.
5. Applets cannot run any program from the local computer.
6. Applets are restricted from using libraries from other languages such as C or C++.

Application	Applet
Applications are larger programs	Applets are the small programs
Application execution starts with the main method	Applets don't have the main method
while the java applications are designed to work with the client as well as server	Applets are designed just for handling the client site problems
Application can access all the resources of the computer	Applet applications are executed in a Restricted Environment
Whereas application run stand-alone and independently on web.	Applets are run from inside a web page using a special feature known as HTML tag.

PREPARING TO WRITE APPLETS

Until now we have been creating simple Java application program with a single main() method that created objects, set instance variables and ran methods. To write any applet, we will need to know:

1. When to use applets.
2. How an applet works,
3. What sort of features an applet has, and
4. Where to start, when we first create our own applet.

The following are the steps that are involved in developing and testing and applet.

1. Buliding an applet code(.java file)
2. Creating an executable applet(.class file)
3. Designing a web page using HTML
4. Preparing <Applet Tag>
5. Incorporating <Applet> tag into the web page.
6. Creating HTMl file.
7. Testing the applet code.

BULIDING APPLET CODE

To building the applet code two classes of java library are essential namely Applet and Graphics. The Applet class is contained in java.applet package provides life and beehaviour to the applet through its methods such as int(), start() and paint(). Unlike with applications, where java calls the main() method directly to initiate the execution of the program, when an applet is loaded java automatically calls a series of Applet class methods for starting running and stopping the applet code. The Applet class therefore maintains the life cycle of an applet.

To display the result of the applet code, the paint() method of the Applet class is called up. The output may be test, graphics, or sound. The syntax of paint() method which requires a Graphic 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 contain the Graphic class. All output operations of an applet are performed using the methods defined in the graphics class.
The general format of applet code is as following

```
import java.awt.*;
import java.applet.*;
_____
_____
public class applet classname extends Applet
{
  .....
  .....statements
  .....
  .....
 public void paint(Graphics g)
 {
```

```
    .....
    .....//Applet operations code
    .....
  }
  .....
  .....
}
```

First Applet program

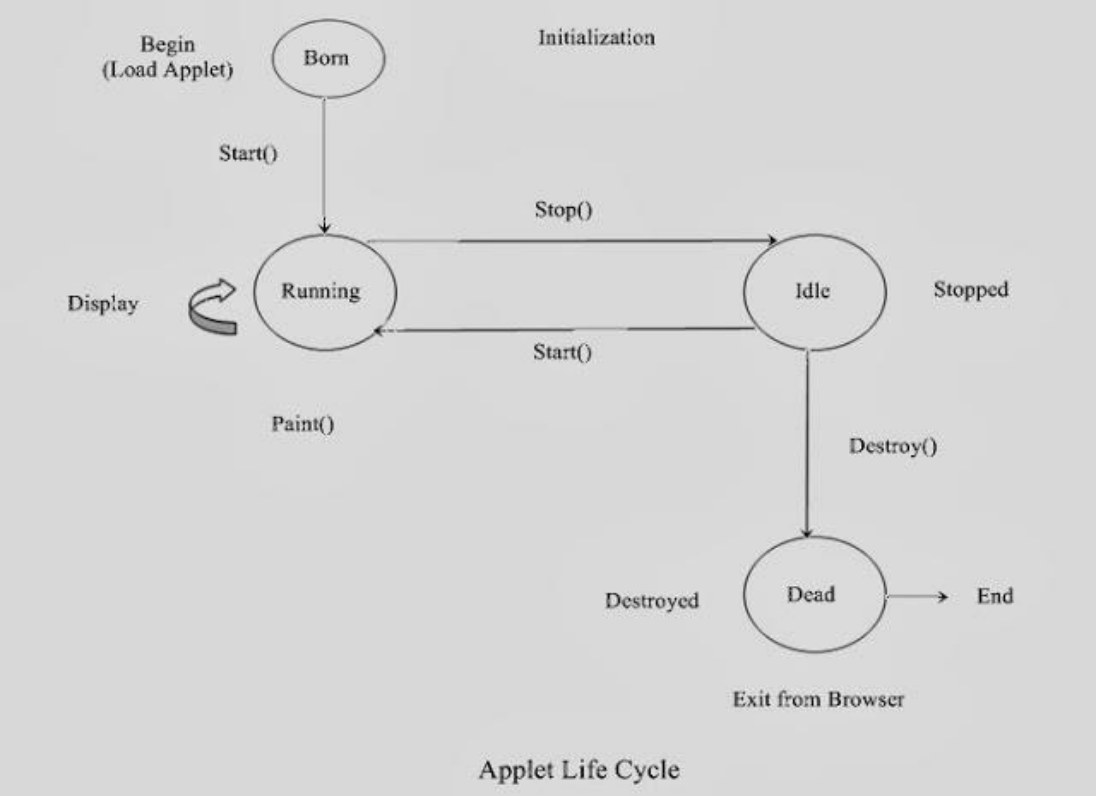
```
import java.awt.*;
import java.applet.*;
public class FirstJavaApplet extends Applet
{
      public void paint(Graphics g)
          {
            g.drawString("My First Java Applet Program",100,100);
          }
}
```

```
<HTML>
<HEAD>
 <TITLE>My First Java Applet Program</TITLE>
</HEAD>
<BODY>
 <APPLET Code="FirstJavaApplet.class" Width=300 Height=200>
 </APPLET>
</BODY>
</HTML>
```

"Running Applet"

"Output of FirstJavaApplet.class"

APPLET LIFE CYCLE **[5 Marks]**



Applet Life Cycle

1 Initialization State

when the browser downloads an HTML page containing applets, it creates an instance for each of the Applet classes, using the no arg constructor. Applet must have a no argument constructor otherwise it cannot be loaded. Initialization can be done through init(). The init() method is the first method to be called. It is used to initialize the applet each time it is reloaded. Applets can be used for setting up an initial state, loading images or fonts, or setting parameters. For example:

```
public void init()
{
  //code here
}
```

2 Running State

Immediately after calling init(), the browser calls the start() method. start() is also called when user returns to an HTML page that contains the applet. So, it ht user leaves a web page and come back, the applet resumes execution at start(). So, when the applet called the start() Method it is called its running state. Staring can also occur if the applet is already in "stopped" (idle) state.

```
public void start()
{
_____
_____
(Action) _____
}
```

3 Idle or Stopped State

When we leave the page containing the currently running applet, then it stop running and becomes idle. We can also do so by calling stop() Method explicitly.

public void stop()
{
_____
_____
(Action) _____
}

4 Dead State
When an applet is completely removed from the Memory, it is called dead. This occurs automatically by invoking the destroy() method when we quit the browser Like initialization, dead state occurs only once in the applet's life cycle.

public void destroy()
{
_____
_____
(Action) _____
}

5 Display State
Painting is how an applet displays something on screen-be it text, a line, a colored background, or an image. The paint() method is used for displaying anything on the applet paint() method takes an argument, an instance of class graphics. The code given can be as follows:

public void paint(Graphics g)
{
}

where Graphics class contains the member functions that can be used to display the output to the browser.

text String, the drawString() method is used.
g.drawString(String s, int x, int y)

where x and y are the coordinate positions on the screen and s is the string to be output.

CREATING AN EXECUTABLE APPLET

Executable applet is nothing but the .class file of applet, which is obtained by compiling the source code of the applet. Compiling the applet is exactly the smae as compiling an application using following command.

javac appletname.java

The compiled output file called appletname.class should be placed in the smae directory as the source file.

DESIGNING A WEB PAGE

Java appleta are programs that reside on web page. A web page is basically made up of text and HTML tags that can be interpreted by a web browser or an applet viewer. Like Java source code, it can be prepared using any ASCII text editor. A web page is also called HTML page or HTML document. Web pages are stored using a file extension .html such as my Applet.html. Such files are referred to as HTML files. HTML files should be stored in the same directory as the compiled code of the applets.

A web page is marked by an opening HTML tag <HTML> and closing HTML tag </HTML> and is divided into the following three major parts:

1) Comment Section
2) Head Section
3) Body Section

Comment Section
This is very first section of any web page containing the comments about the web page functionality. It is important to include comments that tell us what is going on in the web page. A comment line begins with <! And ends with a > and the web browsers will ignore the text enclosed between them. The comments are optional and can be included anywhere in the web page.

Head Section
This section contains title, heading and sub heading of the web page. The head section is defined with a starting <Head> tag and closing </Head> tag.

```
<Head>
<Title> Welcome to Java Applet </Title>
</Head>
```

Body Section
After the Head Section the body section comes. This is called as body section because the entire information and behavior of the web page is contained in it, It defines what the data placed and where on to the screen. It describes the color, location, sound etc. of the data or information that is going to be placed in the web page.

```
<Body>
<Center>
<H1>
Welcome to Java >
</H1>
</Center>
<BR>
<APPLET - - ->
</APPLET>
</Body>
```

APPLET TAG **[2 Marks]**

The <Applet...< tag supplies the name of the applet to be loaded and tells the browser how much space the applet requires. The ellipsis in the tag <Applet...> indicates that it contains certain attributes that must specified. 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 this HTML file. It 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 center of screen by using the CENTER tags as showsn below:

```
<CENTER>
<Applet>
---------------
---------------
---------------
</Applet>
</CENTER>
```

The applet tag discussed above specified the three things:

1) Name of the applet
2) Width of the applet (in pixels)
3) Height of the applet (in pixels)

ADDING APPLET TO HTML FILE

To execute an applet in a web browser, you need to write a short HTML text file that contains the appropriate APPLET tag. Here is the HTML file that executes **SimpleApplet**:

```
<Applet code = Hellojava.class width = 400 Height = 200 >
</Applet>
```

The width and height attributes specify the dimensions of the display area used by the applet. After you create this file, you can execute the HTML file called RunApp.html (say) on the command line.

```
c:\>appletviewer RunApp.html
```

Running the Applet

To execute an applet with an applet viewer, you may also execute the HTML file in which it is enclosed, eg.

c:\>appletviewer RunApp.html

Execute the applet the applet viewer, specifying the name of your applet's source file. The applet viewer will encounter the applet tage within the comment and execute your applet.

What are the attributes of Applet tags?

1. height : Defines height of applet
2. width: Defines width of applet
3. align: Defines the text alignment around the applet
4. alt: An alternate text to be displayed if the browser support applets but cannot run this applet
5. archive: A URL to the applet when it is stored in a Java Archive or ZIP file
6. code: A URL that points to the class of the applet
7. codebase: Indicates the base URL of the applet if the code attribute is relative
8. hspace: Defines the horizontal spacing around the applet
9. vspace: Defines the vertical spacing around the applet
10. name: Defines a name for an applet
11. object: Defines the resource name that contains a serialized representation of the applet
12. title: Display information in tool tip

PASSING PARAMETERS TO APPLETS

Parameters are passed to applets in NAME=VALUE pairs in `<PARAM>` tags between the opening and closing `APPLET` tags. Inside the applet, you read the values passed through the `PARAM` tags with the `getParameter()` method of the `java.applet.Applet` class.

The program below demonstrates this with a generic string drawing applet. The applet parameter "Message" is the string to be drawn.

```
import java.applet.*;
import java.awt.*;

public class DrawStringApplet extends Applet {

  private String defaultMessage = "Hello!";

  public void paint(Graphics g) {

    String str = this.getParameter("Message");
    if (str == null) str = defaultMessage;
    g.drawString(str, 50, 25);

  }

}
```

You also need an HTML file that references your applet. The following simple HTML file will do:

```
<HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>

<BODY>
This is the applet:<P>
<APPLET code="DrawStringApplet" width="300" height="50">
<PARAM name="Message" value="Welcome to Applet!">
This page will be very boring if your
browser doesn't understand Java.
</APPLET>
</BODY>
</HTML>
```

Of course you are free to change "Howdy, there!" to a "message" of your choice. You only need to change the HTML, not the Java source code. PARAMs let you customize applets without changing or recompiling the code.

This applet is very similar to the HelloWorldApplet. However rather than hardcoding the message to be printed it's read into the variable `str` from a `PARAM` element in the HTML.

You pass `getParameter()` a string that names the parameter you want. This string should match the name of a `PARAM` element in the HTML page. `getParameter()` returns the value of the parameter. All values are passed as strings. If you want to get another type like an integer, then you'll need to pass it as a string and convert it to the type you really want.

The `PARAM` element is also straightforward. It occurs between `<APPLET>` and `</APPLET>`. It has two attributes of its own, `NAME` and `VALUE`. `NAME` identifies which `PARAM` this is. `VALUE` is the string value of the `PARAM`. Both should be enclosed in double quote marks if they contain white space.

An applet is not limited to one `PARAM`. You can pass as many named PARAMs to an applet as you like. An applet does not necessarily need to use all the PARAMs that are in the HTML. Additional PARAMs can be safely ignored.

HTML TAGS **[5 Marks]**

Tag	*Function*
<HTML>....</HTML>	Signifies the beginning and end of a HTML file.
<HEAD>....</HEAD>	This tag may include details about the Web page. Usually contains <TITLE> tag within it.
<TITLE>....</TITLE>	The text contained in it will appear in the title bar of the browser.
<BODY>....</BODY>	This tag contains the main text of the Web page. It is the place where the <APPLET> tag is declared.
<H1>....</H1> <H6>....<H/6>	Header tags. Used to display headings. <H1> creates the largest font header, while <H6> creates the smallest one.
<CENTER> ....<CENTER>	Places the text contained in it at the centre of the page.
<APPLET ...>	<APPLET ...> tag declares the applet details as its attributes.
<APPLET>....</APPLET)	May hold optionally user-defined parameters using <PARAM> Tags.
<PARAM....>	Supplies user-defined parameters. The <PARAM> tag needs to be placed between the <APPLET> and </APPLET> tags. We can use as many different <PARAM> tags as we wish for each applet.
<B>....<B>	Text between these tags will be displayed in bold type.
 	Line break tag. This will skip a line. Does not have an end tag.
<P>	Para tag. This tag moves us to the next line and starts a paragraph of text. No end tag is necessary.
<IMG...>	This tag declares attributes of an image to be displayed.
<HR>	Draws a horizontal rule.
<A....> </A>	Anchor tag used to add hyperlinks.
<FONT ...> ... </FONT>	We can change the colour and size of the text that lies in between <FONT> and </FONT> tags using COLOR and SIZE attributes in thetag<FONT...>.
<! . . . .>	Any text starting with a < ! mark and ending with a > mark is ignored by the Web browser. We may add comments here. A comment tag may be placed anywhere in a Web page.

DISPLAY NUMERICAL VALUES IN APPLET

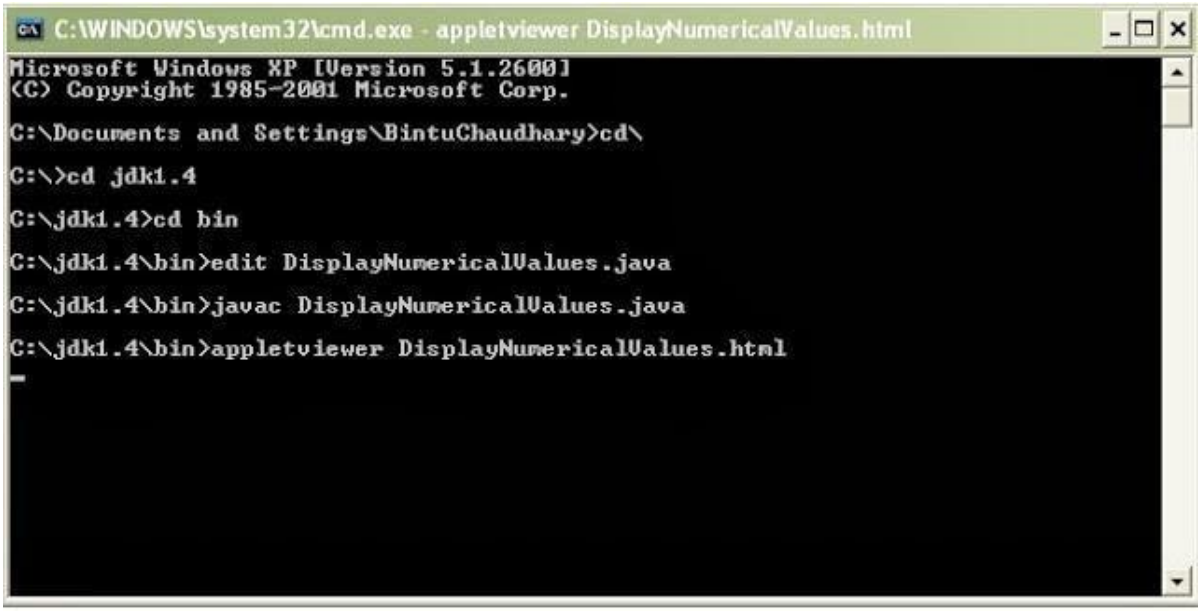
In applets, we can display numerical values by first converting them into strings and then using the drawstring() method of Graphics class we can do this easily by calling the valueOf() Method of String class. Let us consider an example:

```
import java.awt.*;
import java.applet.*;
public class DisplayNumericalValues extends Applet
{
        public void paint(Graphics g)
        {
                int val1 = 10;
                int val2 = 20;
                int sum = val1 + val2;
                String str_sum = "Sum="+String.valueOf(sum);
                g.drawString(str_sum,100,200);
        }
}
```

This applet runs using the following HTML file:

```
<HTML>
<HEAD>
        <TITLE>Display Numerical Values</TITLE>
</HEAD>
<BODY>
        <APPLET Code="DisplayNumericalValues.class" Width=400 Height=300>
        </APPLET>
</BODY>
</HTML>
```

After this you can comiple your java applet program as shown below:



"Output of DisplayNumericalValues.class"

GETTING INPUT FROM THE USER IN JAVA APPLET

Applets work in graphical environment. Therefore applets treat input as text strings. We must first create an area of the screen in which user can type and edit input items. We can do this by using the TextField class of the applet package. The values of the fields can be given even editer after the creation of input fields. Next step is to retrieve the items from the fields for display of calculations.

For any kinds of computation on the input field, we must convert it to the right form and the resuls again converted back to strings for display.

Let us consider an example

```
import java.awt.*;
import java.applet.*;
public class GettingInputfromtheUser extends Applet
{
	TextField t1, t2;
	public void init()
	{
		t1 = new TextField(10);
		t2 = new TextField(10);

		add(t1);
		add(t2);

		t1.setText("0");
		t2.setText("0");
	}
	public void paint(Graphics g)
	{
		int a=0,b=0,c=0;
		String str1,str2,str;

		g.drawString("Enter the number in each box",10,50);

		try
		{
			str1=t1.getText();
			a=Integer.parseInt(str1);

			str2=t2.getText();
			b=Integer.parseInt(str2);
		}
		catch(Exception e)
		{
		}
		c=a+b;
		str=String.valueOf(c);
```

```
            g.drawString("Sum is",10,15);
            g.drawString(str,100,75);
        }
}
```

This applet runs using the following HTML file:

```
<HTML>
<HEAD>
        <TITLE>Getting Input from the User</TITLE>
</HEAD>
<BODY>
        <APPLET Code="GettingInputfromtheUser.class" Width=400 Height=300>
        </APPLET>
</BODY>
</HTML>
```