UNIT-1
Chapter-1
Introducing Applet

1.1 Applet Basics

Applets are small programs that are designed for transmission over the Internet and run within a browser. Because Java's virtual machine is in charge of executing all Java programs, including applets, applets offer a secure way to dynamically download and execute programs over the Web.

There are two general varieties of applets: those based on the Abstract Window Toolkit (AWT) and those based on Swing. Both the AWT and Swing support the creation of a graphical user interface (GUI). The AWT is the original GUI toolkit and Swing is Java's lightweight alternative. It is important to understand, however, that Swing-based applets are built upon the same basic architecture as AWT-based applets. Furthermore, Swing is built on top of the AWT. Therefore, the information and techniques presented here describe the foundation of applet programming and most of it applies to both types of applets.

/ / A Minimal AWT-based applet.

```
import     java.awt.*;
import java.applet.*;
public class SimpleApplet extends Applet {
    public void paint (Graphics g) {
        g.drawString (—Java makes applets easy. —, 20, 20);
    }
}
```

This applet begins with two import statements. The first imports the Abstract Window Toolkit classes. Applets interact with the user (either directly or indirectly) through the AWT, not through the console-based I/O classes. The AWT contains support for a window- based, graphical user interface. The next import statement imports the applet package. This package contains the class Applet. Every applet that you create must be a subclass (either directly or indirectly) of Applet. The next line in the program declares the class SimpleApplet.
This class must be declared as public because it will be accessed by outside code. Inside SimpleApplet, paint( ) is declared. This method is defined by the AWT Component class (which is a superclass of Applet) and is overridden by the applet. paint( ) is called each time the applet must redisplay its output. This can occur for several reasons. For example, the window in which the applet is running can be overwritten by another window and then uncovered. Or the applet window can be minimized and then restored. paint( ) is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, paint( ) is called. The paint( ) method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

Inside paint( ), there is a call to drawString( ), which is a member of the Graphics class.
This
method outputs a string beginning at the specified X, Y location. It has the following general form:

void drawString(String message, int x, int y)
Here, message is the string to be output beginning at x, y. In a Java window, the upper-left corner is location 0,0. The call to drawString( ) in the applet causes the message to be displayed beginning at location 20,20.

The applet does not have a main( ) method. Unlike the standalone application program, applets do not begin execution at main( ). In fact, most applets don't even have a main( ) method. Instead, an applet begins execution when the name of its class is passed to a browser or other applet-enabled program.

Compiling applet program is similar to stand alone application program. However, running SimpleApplet involves a different process. There are two ways in which you can run an applet: inside a browser or with a special development tool that displays applets. The tool provided with the standard Java JDK is called appletviewer use it to run the applets. We can also run them in your browser, but the appletviewer is much easier to use during development.

One way to execute an applet (in either a Web browser or the appletviewer) is to write a short HTML text file that contains a tag that loads the applet. Currently, Oracle recommends using the APPLET tag for this purpose.
<applet code = ―simple applet ― width=200 height=60>
</applet>
The width and height statements specify the dimensions of the display area used by the applet. To execute SimpleApplet with an applet viewer, you will execute this HTML file. For example, if the preceding HTML file is called StartApp.html, then the following command line will run SimpleApplet:
C:\> appletviewer StartApp.html

1.1.2 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

1.2 Building Applet Code

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 behaviour to the applet through its methods such as init(), start() and point(). 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 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 arquement, 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.

Syntax:

```
Import java.awt.*;
Import java.applet.*;
 ……..
 ……..

Public class appletclassname extends Applet
{
 ……
  Public void paint(Graphics g)
  {
   ……..
   ……..
  }
 …….
 …….
}
```

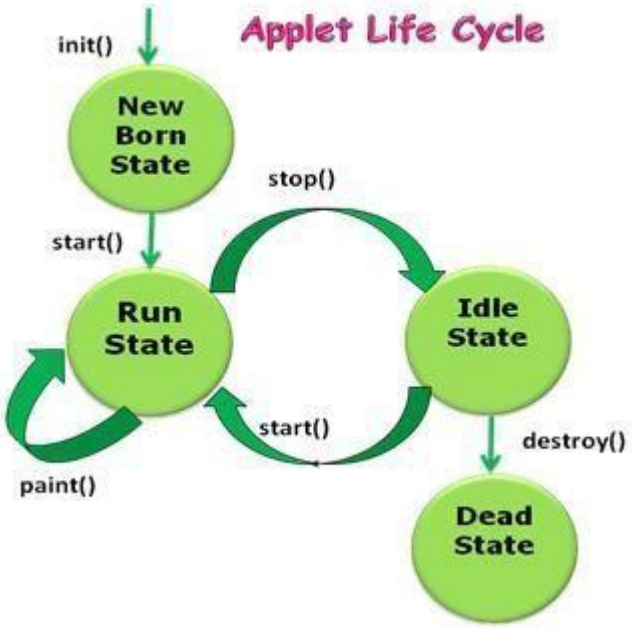
The appletclassname is the main class for the applet. When the applet is loaded, Java creates an instance of this class, and then a series of Applet class methods are called on that instance to execute the code.

Example:

```
Import java.awt.*;
Import java.applet.*;
Public class HelloJava extends Applet
{
      Public void paint(Graphics g)
       {
         g.drawString(―Hello Java‖, 10,100);
       }
}
```

1.3 Applet Life Cycle

When an applet is loaded, it undergoes a series of changes in its state as shown in fig.



- Born on initialization state
- Running state
- Idle state
- Dead or destroyed state

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

```
public void init()
{
  ……
  ……
}
```

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.

```
public voids start()
{
……
……
}
```

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.

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

……
……..
}

Dead State

An applet is said to be dead when it is removed from memory. This occurs by invoking the destroy() method when we quit the browser. Destroying stage occurs only once in the applet's life cycle.

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

Display State

Applet moves to the display state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The paint() method is called to accomplish this task.

```
public void paint(Graphics g)
{
……
……..(Display statements)
}
```

The **paint()** method is defined in the Applet class.

1.4 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 ellipsis in the tag<APPLET..> indicates that it contains certain attributes that must specified.

```
<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>
```

Note that <APPLET> tag discussed above specifies three things.

1. Name of the applet
2. Width of the applet(in pixels)

3. Height of the applet(in pixels)

1.5 Adding Applet to HTML File

We can put 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 of the applet must appear. Following is the content of the HTML file that is embedded with the <APPLET> tag of our HelloJava applet.

```
<HTML>
<! It specifies the applet to be loaded and executed.
>
<HEAD>
        <TITLE>
              Welcome to Java
        </TITLE>
</HEAD>
<BODY>
       <CENTER>
             <H1> Welcome to the world of Applet </H1>
       </CENTER>
       <BR>
       <CENTER>
```

1.6 Passing parameters to Applet

We can supply user-defined parameters to an applet using <PARAM ... > tags. Each <PARM… > tag has a name attribute such as color, and a value attribute such as red. Inside the applet code, the applet can refer to that parameter by name to find its value. For 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 to an applet when it is loaded. We can define the init( ) method in the applet to get hold of the parameters defined in the <PARM> 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.

Applet HellojavaParm

```
import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
{
String str;
public void init( )
{
str = getParameter ("string—);       //Receiving parameter value
if (str = = nu1l)
        str=‖Java‖;
                str= —Hello‖ + str                    //Using the value
          }
public void paint (Graphics g)
{
g.drawString(str, 10, 100);
}
}
```

Now we have to 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>
<! Parameterized HTML file>
<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>
```

This will produce output as Hello Applet. if we remove <param> tag from HTML file will produce output as Hello Java.

CHAPTER 2
MANAGING INPUT/OUTPUT FILES IN JAVA

2.1 Introduction

A file is a collection of related records placed in a particular area on the disk. A record is composed of several fields and a field is a group of characters. Characters in java are Unicode characters composed of two bytes, each byte containing eight binary digits, 1 or 0.
Storing and managing data using files is known as file processing which includes tasks such as creating files, updating files and manipulation of data.

2.2 Concept of Streams

In file processing, input refers to the flow of data into a program and output means the flow of data out of a program. Input to a program may come from the keyboard, the mouse, the memory, the disk, a network, or another program. Similarly, output from a program may go to the screen, the printer, the memory, the disk, a network, or another program.

The Stream Classes

Java‘s stream-based I/O is built upon four abstract classes: **InputStream**, **OutputStream**, **Reader**, and **Writer**. These classes were briefly discussed in Chapter 13. They are used to create several concrete stream subclasses. Although your programs perform their I/O operations through concrete subclasses, the top-level classes define the basic functionality common to all stream classes.

InputStream and **OutputStream** are designed for byte streams. **Reader** and **Writer** are designed for character streams. The byte stream classes and the character stream classes form separate hierarchies. In general, you should use the character stream classes when working with characters or strings, and use the byte stream classes when working with bytes or other binary objects.

The Byte Streams

The byte stream classes provide a rich environment for handling byte-oriented I/O. A byte stream can be used with any type of object, including binary data. This versatility makes byte streams important to many types of programs. Since the byte stream classes are topped by **InputStream** and **OutputStream**, our discussion will begin with them.

InputStream

InputStream is an abstract class that defines Java‘s model of streaming byte input. It implements the **Closeable** interface. Most of the methods in this class will throw an **IOException** on error conditions. (The exceptions are **mark( )** and **markSupported( )**.) Table 19-1 shows the methods in **InputStream**.

OutputStream

OutputStream is an abstract class that defines streaming byte output. It implements the **Closeable** and **Flushable** interfaces. Most of the methods in this class return **void** and throw an **IOException** in the case of errors. (The exceptions are **mark( )** and **markSupported( )**.)

The below table gives a brief description of all the methods provided by the **InputStream** class

Method	Description
1. Read()	Reads a byte from the input stream
2. Read(byte b[])	Reads an array of bytes into b
3. Read(byte b[], int n, int m)	Reads m bytes into b starting from nth byte
4. Available()	Gives number of bytes availale in the input
5. Skip(n)	skips over n bytes from the input stream
6. Reset()	Goes back to the beginning of the stream
7. Close	closes the input stream

Table gives a brief description of all the methods of outputstream class

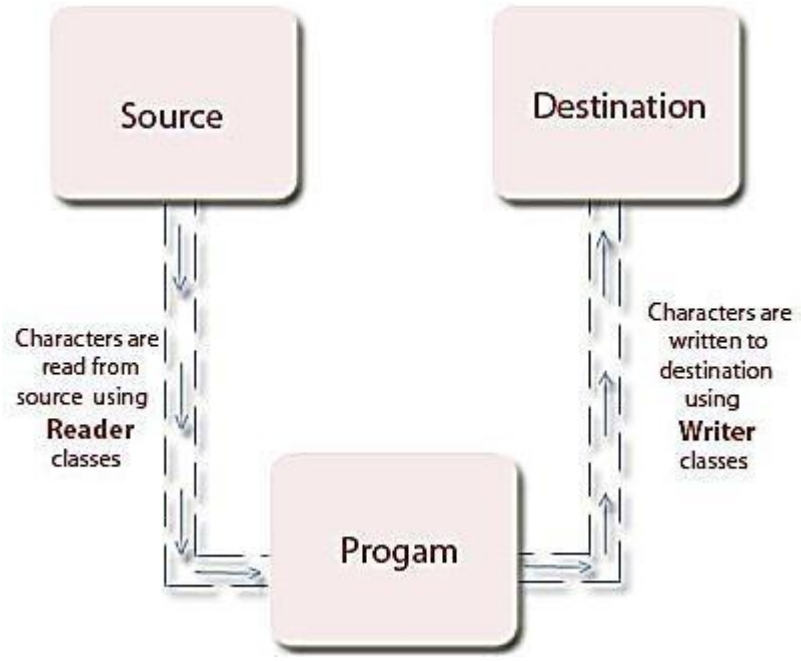
Method	Description
1. Write()	Writes a byte to the output stream
2. write(byte[] b)	Writes m bytes in the array b to the output stream
3. write(byte b[], int n, int m)	Writes m bytes from array b starting from nth byte
4. close()	closes the output stream
5. flush()	flushes the output stream

2.3 Character Stream Classes

Character Stream Classes are used to read characters from the source and write **characters** to destination.

There are two kinds of Character Stream classes - Reader classes and Writer classes.

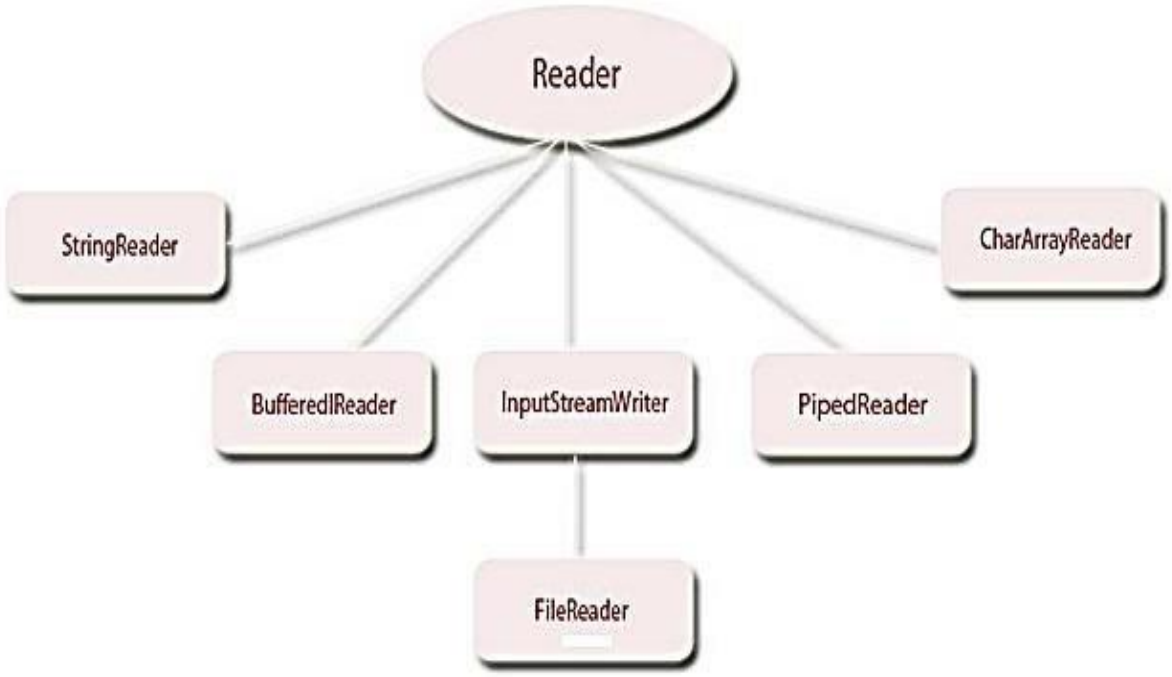
- **Reader Classes** - These classes are subclasses of an abstract class, Reader and they are used to read characters from a source(file, memory or console).
- **Writer Classes** - These classes are subclasses of an abstract class, Writer and they used to write characters to a destination(file, memory or console).



Reader

Reader class and its subclasses are used to read characters from source. Reader class is a base class of all the classes that are used to read characters from a file,

memory or console. Reader is an abstract class and hence we can't instantiate it but we can use its subclasses for reading characters from the input stream. We will discuss subclasses of Reader class with their code, in the next few articles.



Hierarchy of reader stream class

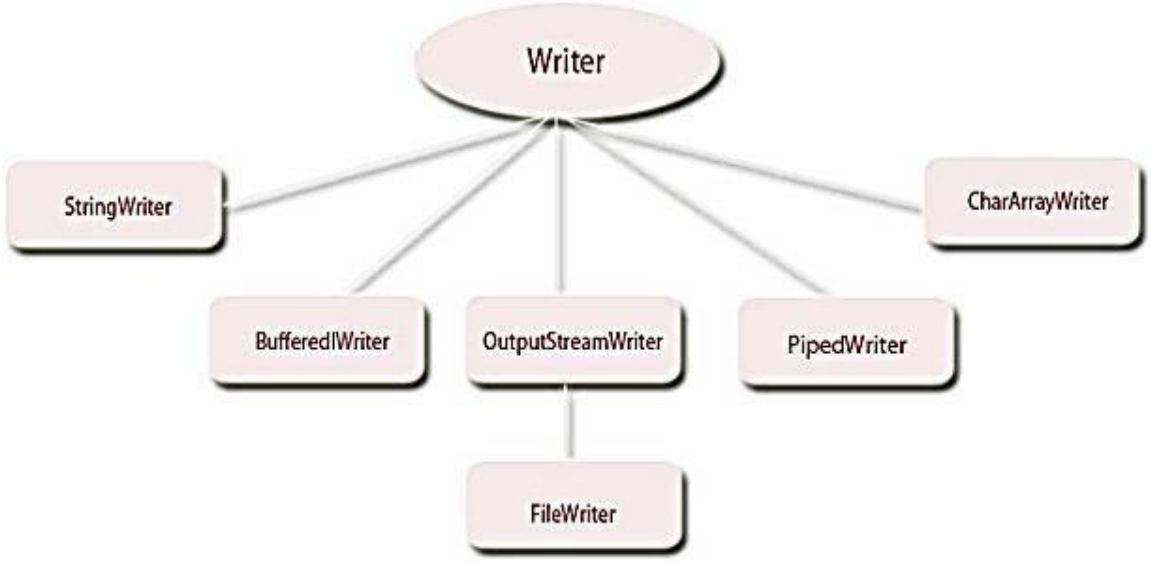
Methods of Reader classes

These methods are of Reader class.

Methods	Description
Int read()	This method reads a character from the input stream
Int read(char[], ch)	This method reads a chunk of characters from the input stream and store them in its char array, ch.
close()	This method closes this output stream and also frees any system resources connected with it.

Writer

Writer class and its subclasses are used to write characters to a file, memory or console. Writer is an abstract class and hence we can't create its object but we can use its subclasses for writing characters to the output stream. We will discuss subclasses of Writer with their code in the next few articles.



Hierarchy of Writer classes

Methods of writer classes

Methods of Writer class provide support for writing characters to the output stream. As this is an abstract class. Hence, some undefined abstract methods are defined in ths subclasses of Output Stream.

Methods	Description
abstract void flush()	This method flushes the output steam by forcing out buffered bytes to be written out.
void write(int c)	This method writes a characters(contained in an int) to the output stream.
void write(char[] arr)	This method writes a whole char array(arr) to the output stream.
abstract void close()	This method closes this output stream and also frees any resources connected with this output stream.

2.4 Using the File Class

The java.io package in cludes a class known as the **File** class that provides support for creating files and directories. The class includes several constructing for instantiating the File objects. This class also contains several methods for supporting the operations such as

- Creating a file
- Opening a file
- Closing a file
- Deleting a file
- Getting the name of a file
- Getting the size of a file
- Checking the existence of a file
- Renaming a file
- Checking whether the file is writable
- Checking whether the file is readable

2.5 Input/output Exceptions

When creating files and performing i/o operations on them, the system may generate i/o related exception.
The basic i/o related exception classes and their functions are given below.

EOFException	Signals that an end of the file or end of stream has been reached unexpectedly during input.
FileNotFoundException	Informs that a file could not be found
InterruptedIOException	Warns that an I/O operations has been interrupted
IOException	Signals that an I/O exception of some sort has occurred

Each i/o statement or group of i/o statements must have an exception handler around it as shown below.

```
try
{
……
……//I/O statements
}
Catch(IOException e)
{
…….
……//message output statement
}
```

2.6 Creation of Files

If we want to create and use a disk file,we need to decide the following about the file and its intended purpose.

- Suitable name for the file
- Data type to be stored
- Purpose(reading,writing, or updating)
- Method of creating the file

A filename is a unique string of characters that helps identify a file on the disk.
Example: input.data or input.txt

Data type is to decide the type of file stream classes to be used for handling the data. We should decide whether the data to be handled is in the form of characters, bytes or primitive type.

The purpose of using a file must also be decided before using it. For example, we should know whether the file is created for reading only, or writing only, or both the operations.

For using a file, it must be opened first. This is done by creating a file stream and then linking it to the filename. A file stream can be defined using the classes of **Reader/InputStream** for reading data and **Writer/OutputStream** for writing data.

Common Stream Classes used for I/O Operations

Characters

Read	Write
CharArrayReader	CharArrayWriter
FileReader	FileWriter
PipedReader	PipedWriter

Bytes

Read	Write
ByteArrayInputstream	ByteArrayOutputStream
FileInputStream	FileOutputStream
PipedInputStream	PipedOutputStream

Initializing using a file object

There are two ways of initializing the file stream objects. All of the constructors require that we provide the name of the file either directly, or indirectly by giving a file object that has already been assigned a filename.

The following code segment illustrates the use of **direct** approach.

```
FileInputStream fis;
try
{
// Assign the filename to the file stream object
fis=new FileInputStream(―test.dat‖);
……
}
Catch(IOException e){}
```

The **indirect** approach uses a file object that has been initialized with the desired filename. This is illustrated by the following code.

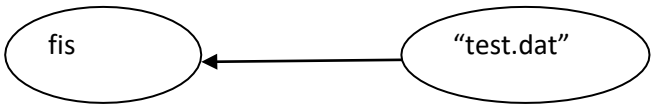
```
File infile;                    //Declare a file object
Infile=new File(―test.dat‖);    //Assign the filename to the file object
FileInputStream fis;
try
{

//Give the value of the file object
//to the file stream object
fis=new FileInputStream(infile);
}
Catch(…..)
{ }
```

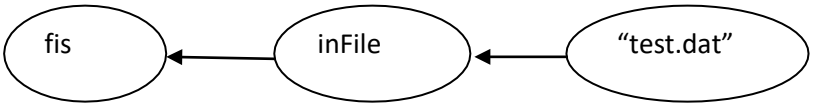
The code above includes five tasks:

- Select a filename
- Declare a file object

- Give the selected name to the file object declared
- Declare a file name object
- Connect the file to the file stream object



Stream object Filename



Stream object File object Filename

2.7 Reading/Writing Characters

The two subclasses used for handling characters in files are FileReader(for reading characters) and FileWriter(for writing characters).
The below program uses these two file stream classes to copy the contents of a file named ―input.dat‖ into a file called ―output.dat‖.

```
Import java.io.*;
Class CopyCharacters
{
    Public static void main(String args[])
    {
        //Declare and create input and output files
        File infile=new File(―input.dat‖);
        File outfile=new File(―output.dat‖);
        FileReader ins=null; //Creates file stream ins
        FileWriter outs=null; //Creates file stream outs
        try
        {
            ins=new FileReader(infile);    //opens infile
            outs=new FileWriter(outFile);          //Opens outfile
            //Read and write till the end
            int ch;
            While((ch=ins.read())!=-1)
            {
                Outs.write(ch);
            }
        }
Catch(IOException e)
{
System.out.println(e);
System.exit(-1);
}
Finally      //close files
```

```
{
	try
	{
				Ins.close();
				Outs.close();
			}
			Catch(IOException e){}
		}
	}
}
```

2.8 Reading or writing Bytes

FileReader and File Writer classes to read and write 16-bit characters. Most file systems use only 8-bit bytes.
Two commonly used classes for handling bytes are FileInputStream and FileOutputStream classes.

Writing Bytes to a file

```
import java. io. *;
class WriteBytes
{
Public static void main(String args[])
{
Byte cities []={_D‘,‘E‘,‘L‘,‘H‘,‘I‘};
FileOutputStream outfile=null;
try
{
outfile=new FileOutputStream(−city.txt‖);
outfile.write(cities);
outfile.close();
}
catch(IOException e)
{
System.out.println(e);
System.exit(-1);
}
}
}
```

Reading bytes from file

```
import java.io.*;
class ReadBytes
{
	Public static void main(String args[])
	{
		FileInputStream infile=null;
		int b;
		try
```

```
            {
                infile=new FileInputStream(args[0]);
                while((b=infile.read())!=-1)
                {
                    System.out.println((char)b);

                }
                infile.close();
            }
            catch(IOException e)
            {
            System.out.println(e);
            }
    }
}
```

2.9 Random Access File

RandomAcessFile class supported by the java.io package allows us to create files that can be used for reading and writing data with random access.
Ie, we can ‐jump around‖ in the file while using the file. Such files are known as random access files.
A file can be created and opened for random access by giving a mode string as a parameter to the constructor when we open the file.
We can use one of the following two mode strings.

- ‐r‖ for reading only
- ―rw‖ for both reading and writing

An existing file can be updated using the ―rw‖ mode.
Random access files support a pointer known as file pointer that can be moved to arbitrary positions in the file prior to reading or writing.
The file pointer is moved using the method **seek()** in the RandomAccessFile class.
When the file is opened by the statement
File =new RandomAccessFile(“rand.dat”);
The file pointer is automatically positioned at the beginning of the file.

Reading/Writing using a random access file

```
Import java.io.*;
Class Random
{
    Public static void main(String args[])
    {
        RandomAccessFIle file=null;
        try
        {
            file=new RandomAccessFile(―rand.dat‖, ―rw‖);
            file.writeChar(_X‘);
            file.writeInt(555);
            file.writeDouble(3.1412);
            file.seek(0);
            System.out.println(file.readChar());
```

```
                    System.out.println(file.readInt());
                    System.out.println(file.readDouble());
                    File.seek(2);
                    System.out.println(file.readInt());
                    file.seek(file.length());
                    filewriteBoolean(false);
                    file.seek(4);
                    System.out.println(file.readBoolean());
                    file.close();
            }
            Catch(IOException e)
            {
            System.out.println(e);
            }
    }
}
```

Assignment 1

1. Which of these functions is called to display the output of an applet?

A. display()
B. paint()
C. displayApplet()
D. PrintApplet()
Answer: Option B

2. Which of these methods can be used to output a sting in an applet?

A. display()
B. print()
C. drawString()
D. transient()
Answer: Option C

3. What does AWT stands for?

A. All Window Tools
B. All Writing Tools
C. Abstract Window Toolkit
D. Abstract Writing Toolkit
Answer: Option C

4. Which of these is used to perform all input & output operations in Java?

A. streams
B. Variables
C. classes
D. Methods
Answer: Option A

5. Which of these is a type of stream in Java?

A. Integer stream
B. Short stream
C. Byte stream
D. Long stream
Answer: Option C

6. Which of these classes are used by Byte streams for input and output operation?

A. InputStream
B. InputOutputStream
C. Reader
D. All of the mentioned

Answer: Option A

7. Which of these classes are used by character streams for input and output operations?

A. InputStream
B. Writer
C. ReadStream
D. InputOutputStream
Answer: Option B

8. Which exception is thrown by read() method?

A. IOException
B. InterruptedException
C. SystemException
D. SystemInputException
Answer: Option A

9. Which of these packages contain classes and interfaces used for input & output operations of a program?

A. java.util
B. java.lang
C. java.io
D. All of the mentioned
Answer: Option C

10. Which of these class is not a member class of java.io package?

A. String
B. StringReader
C. Writer
D. File
Answer: Option A

11. Which of these class is not related to input and output stream in terms of functioning?

A. File
B. Writer
C. InputStream
D. Reader
Answer: Option A

12. The .....package contains a large number of stream classes that provide capabilities for processing all types of data.
A) java.awt
B) java.io
C) java.util
D) java.net

Answer: Option B

13. Which of the following method(s) not included in InputStream class.
A) available( )
B) reset( )
C) flush( )
D) close( )
Answer: Option C

14. Which of the following methods not included in OutputStream class.
A) write( )
B) skip( )
C) close( )
D) flush( )
Answer: Option B

15. The DataInputStream and DataOutputStream classes are.....streams that allow the reading and writing of java primitive data types.
A) file
B) sequence
C) object
D) filter
Answer: Option D

16. The .....class provides the capacity to read primitive data types from an input stream.
A) pushbackInputStream
B) DataInputStream
C) BufferedInputStream
D) PipeInputStream
Answer: Option B

17. DataInput is
A) an abstract class defined in java.io
B) a class we can use to read primitive data types
C) an interface that defines methods to open files
D) an interface that defines methods to read primitives data types
Answer: Option D

18. The following example shows the creation of a

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

public class Main extends Applet{
public void paint(Graphics g){
g.drawString("Welcome in Java Applet.",40,20);
}
}
```

A. Banner Using Applet

B. Basic Applet
C. Display Clock
D. None Of The Above
Answer: Option B

19. An applet can play an audio file represented by the AudioClip interface in the java, applet package Causes the audio clip to replay continually in which method?
A. Public Void Play()
B. Public Void Loop()
C. Public Void Stop()
D. None Of The Above
Answer: Option B

20. What invokes immediately after the start() method and also any time the applet needs to repaint itself in the browser?
A. Stop()
B. Init()
C. Paint()
D. Destroy()
Answer: Option C

21. Which method is called only once during the run time of your applet?
A. stop()
B. paint()
C. init()
D. destroy()
Answer: Option C

22. When an applet is terminated which of the following sequence of methods calls take place?
A. stop(),paint(),destroy()
B. destroy(),stop(),paint()
C. destroy(),stop()
D. stop(),destroy()
Answer: Option D

23. Which is a special type of program that is embedded in the webpage to generate the dynamic content?
A. Package
B. Applet
C. Browser
D. None Of The Above
Answer: Option B

24. What is used to run an Applet?
A. An Html File
B. An Appletviewer Tool(For Testing Purpose)
C. Both A & B
D. None Of The Above
Answer: Option C

25. Which is the correct order of lifecycle in an applet?
A. Applet Is Started,Initialized,Painted,Destroyed,Stopped
B. Applet Is Painted,Started,Stopped,Initilaized,Destroyed
C. Applet Is Initialized,Started,Painted,Stopped,Destroyed
D. None Of The Above
Answer: Option C

Long Answer questions

1. Explain the life cycle of Applet with neat diagram.
2. How to build the applet code along with the syntax.
3. How to pass the parameters to Applet. Explain with example.
4. Explain about Input/output Exception.
5. Explain about Random Access File along with example.
6. Write a note on reading and writing bytes.
7. Write a note on reading and writing characters.
8. What are the procedures to create the file.
9. Explain about character stream classes.
10. Write all the methods of InputStream and OutputStream classes