**1. Stream**

**2. Serialization & Deserialization.**

**3. Sequence Stream**

**4.Char Stream**

**5. File**

**STREAMS**

1. **Persistence Media:-**  The environment that allows to store data permanently is called persistent media. We can store the data permanently in three places.

**a.**File.

**b.**Database.

**c.** Remote computer(socket)

1. **Persistence:-**  The process of storing data permantently in a persistence media is called persistence.
2. **Persistence Logic:-**  The logic that persist data in a persistence media is called persistence logic.

**Ex:**- IO stream based logic, JDBC based logic, Networking based logic.

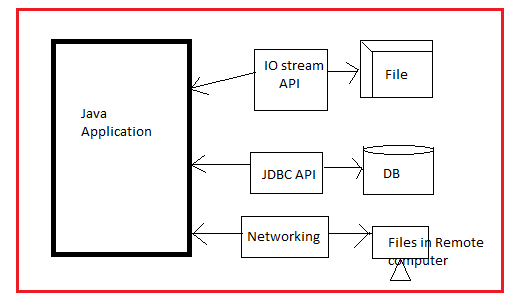
1. **Persistence Technologies:-**  The technology that rovides API to develop persistence logic is called persistence technology.

Persistence technologies are

1. IO Stream:- to persist data in flat files.

2. JDBC , HIBERNATE:- to persist data in DB.

3. Networking:- to persist data in Remote computer.

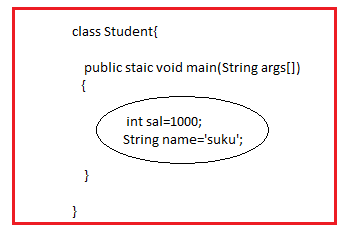


This chapter is also known as ‘file handling/file management system’. Java.io.\* package provided classes

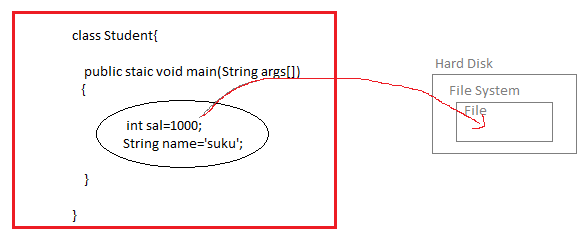
1. To read the data from key board and file.
2. To write the data to file and console.

Thas’ why It is better to call this chapter as IO stream instead of file handling or file management system.

Need of IO Streams:



The values which is stored in either primitive variable and referenced variable is available during the program execution. After program execution these values are’t avaiilable because these are stored in RAM. This is a problem. This problem is solved by storing values in permanent place(Disk).



File Management system has following disadvantages:

1. Security problem.
2. There are no searching algorithms to search the file for data.
3. Data organizaiton in file is difficult.
4. …etc.

By above problems ,In industry file management system is not being used.

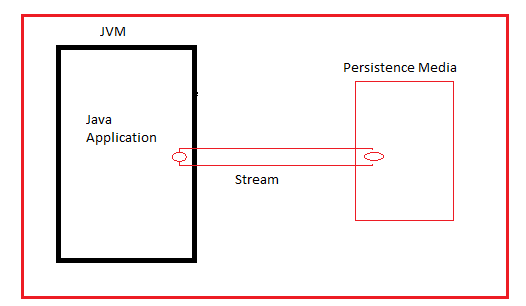
But when we want to store small amout of data and we don’t want to security to that data, we should choose file system. When want to store huge amount of date, it is better to select the DB.

**5.Stream**:- Stream is a logical connection between java program and persistence media /IO device.

Ex:- There is logical connection between wifi-router and mobile. This connection can’ t be visible. Therefore It is also called a logical connection.

The stream can be defined as “ It is continuous flow of data between java program and persistence media”.

Technically, stream is java object that allows reading and writing data to persistence media.

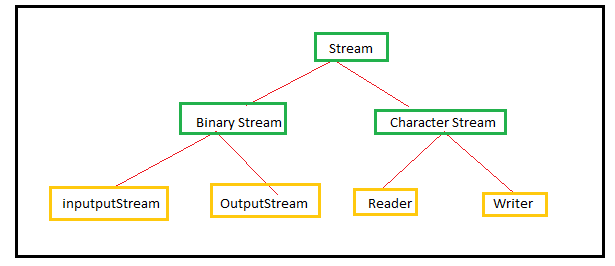


Generally based on **data flow direction** in stream, streams are divided into two types.

1. Input Stream: The stream that allows data to come into java application from persistent media is called input stream. Basically input stream is used to read the data from persistence media.
2. Output Stream: The stream that allows data to send out from java application to persistent media is called output stream. Basically input stream is used to write data to persistence media from java application.

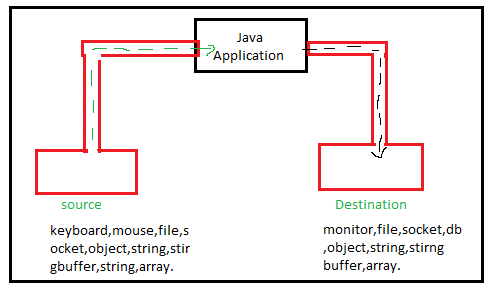
Based on format of data flow in stream, streams are divided into two types:

1. Binary Stream:- The streams which allows data in the format of byte is called binary streams.
2. Character stream:- The streams which allows data in the format of characters is called character streams.



The data can be read from several sources like keyboard, mouse, file, database ,socket, oject,array,string ,stringBuffer.

The data is written to several destinations like monitor,f ile, database ,socket, object, Array, String, StringBuffer.



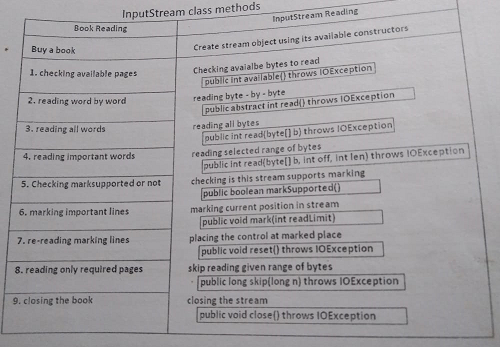
The InputStream is the abstract super class for all binary input stream classes.

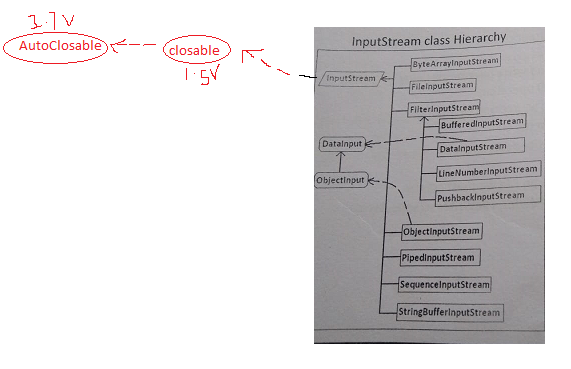
The OutputStream is abstract super class for all binary output stream classes.

Reader is abstract super class for all character input stream classes.

Writer is abstract super class for all character output stream classes.

**6. InputStreamClass:-** It is abstract class. It is super class to all binary input stream classes. This class totally 12 methods. One of the 12 methods is read(). The abstract method is ‘read()’ method. The way of reading data from multiple sources is different . That’s why it is defined as ‘abstract’ method.

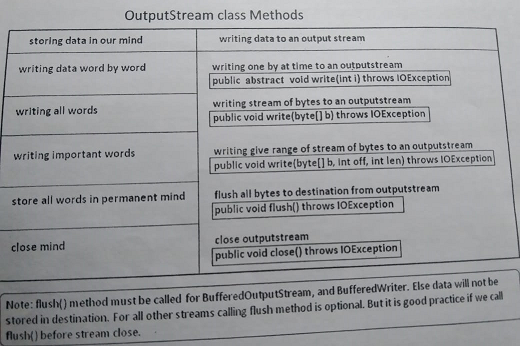


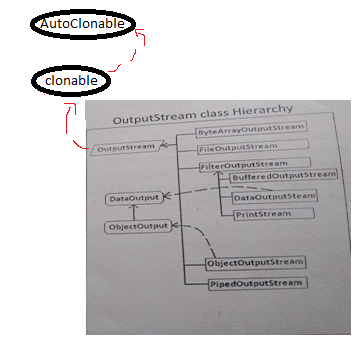


The LineNumberInputStream and PushbackInputStream classes are deprecated classes.

Upto java 1.4v , the closing method was in InputStream class. Java 1.5v onwards, It is placed in closable interface. The autoclosable is also interface. It is superclass to closable interface. It was introduced in java 1.7v.

**7.OutputStream:-** The OuputStream is abstract class.The OutputStream is super class for all binary output stream classes. This class has 8 methods. Among 5 methods , 4 methods are concrete methods and 1 method is abstract method. The abstract method is ‘write()’ method. The way of writing data from multiple sources is different . That’s why it is defined as ‘abstract’ method.





**Steps to write the data from file**.

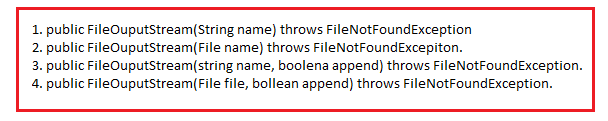
1. Add import java.io.\* statement to program.
2. Create a FileOutputStream class object.
3. Invoke write() method to store data in stream.
4. Invoke the flush() method to write data in file from stream.
5. Close the connection.
6. Handle the exception.

**Steps to Read the data from File.**

1. Add import java.io.\* statement to program.
2. Create a FileInputStream class object.
3. Invoke the read() method to reading data.
4. Close the connection.
5. Handle the Exception.
6. **FileOutputStream:**- It is ouput stream type object/ It is subclass of OutputStream. It is used to write data into a file in binary format one byte at a time. If we want to write number ,text ,images in binary format then we should choose FileOutputStream.

Limitation: Writing only one byte data at atime.

Syntax:-



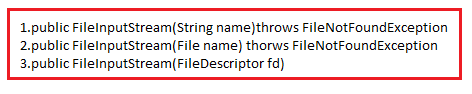
In following cases , this constructor throws FileNotFoundException.

* File Does not exist but can’t be created.
* The passed files name is directory rather than a regular file.
* The passed file is a Read Only File.

1. **FileInputStream:-** It is subclass of InputStream class.If we want to read data (number,text,images,object)from file in binary format then we should choose FileInputStream.

Limitation: reading the data one byte at time.

Syntax:



In following cases, FilenotFoundException will be raised.

* If the file is not existed with the passed name.
* Passed file name is directory rather than regular file.
* If file does not have reading permissions.

Application1: Writing data to file.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException

{

FileOutputStream fos=new FileOutputStream("E:/abc.txt");

fos.write('a');

fos.write('b');

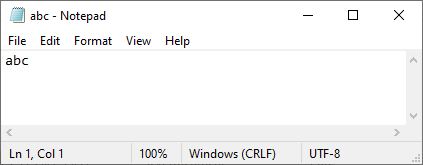
fos.write(99);

fos.close();

}

}

Case:1 execute above program without creating abc.txt file. The FileOuputStream class creates tis file when its object is created.



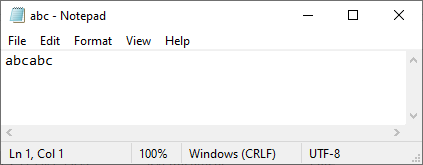
Case2: In above program, replace

FileOutputStream fos=new FileOutputStream("E:/abc.txt");

With

FileOutputStream fos=new FileOutputStream("E:/abc.txt",true);

Execute same program again and observe abc.txt file daata. Find out whether it contains previous data or not.



Case3: change abc.txt file permission to read-only.

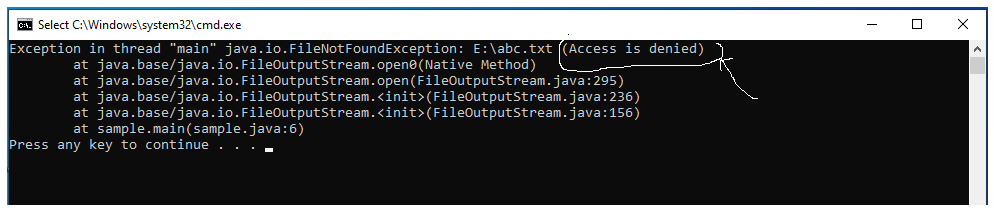
Setting read-only permission.

1. right click on file.

2. click properties.

3. check read-only checkbox.

4.click ok.



Application:2 Read the data from file and display on console.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException

{

FileInputStream fis=new FileInputStream("E:/abc.txt");

int i;

while((i=fis.read())!=-1)

{

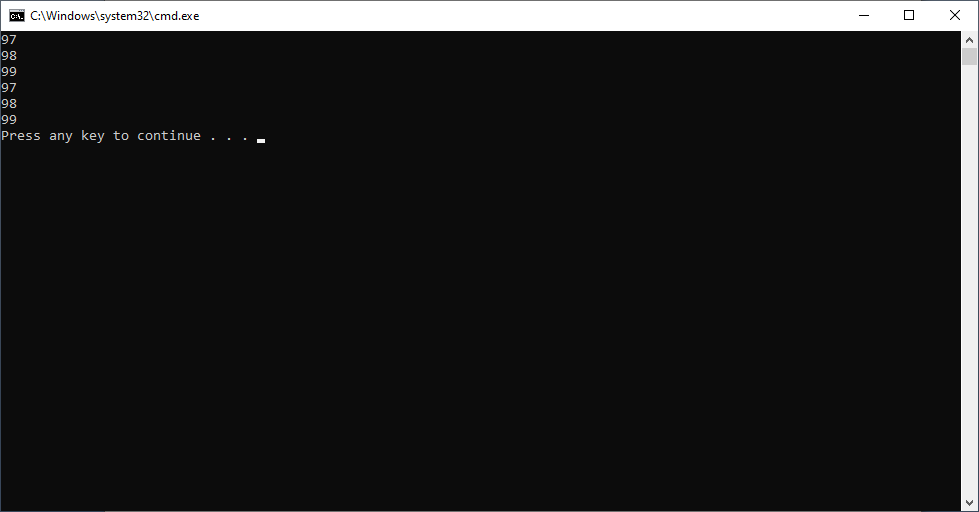
System.out.println(i);

}

fis.close();

}

}



Example: Writing Data into and Reading Data from file.

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.util.Scanner;

**public** **class** Demo {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

File nf=**new** File("D:/suku.txt");

nf.createNewFile();

**byte** b[]=**new** **byte**[10];

**byte** d[]=**new** **byte**[10];

**byte** e[]=**new** **byte**[10];

**byte** f[]=**new** **byte**[]{'\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_'};

String name,qual,age;

FileInputStream f1=**new** FileInputStream(nf);

FileOutputStream f2=**new** FileOutputStream(nf);

Scanner s1=**new** Scanner(System.***in***);

System.***out***.print("Enter the Number of persons:");

**int** n=s1.nextInt();

s1.nextLine();

**int** i=0,j=0;

**while**(j<n)

{

System.***out***.print("Enter the name:");

name=s1.nextLine();

System.***out***.print("Enter the age:");

age=s1.nextLine();

System.***out***.print("enter the qualification:");

qual=s1.nextLine();

**for**( i=0;i<name.length();i++)

{

b[i]=(**byte**)name.charAt(i);

}

**for**( i=0;i<qual.length();i++)

{

d[i]=(**byte**)qual.charAt(i);

}

**for**( i=0;i<age.length();i++)

{

e[i]=(**byte**)age.charAt(i);

}

f2.write(b,0,b.length);

f2.write('\t');

f2.write(e,0,e.length);

f2.write('\t');

f2.write(d,0,d.length);

f2.write('\n');

f2.write(f);

f2.write('\n');

j++;

}

**int** x;

System.***out***.println("==================Data======================== ");

**while**((x=f1.read())!=-1)

{

System.***out***.print((**char**)x);

}

}

}

**Output:**

Enter the Number of persons:2

Enter the name:A.sukumar

Enter the age:41

enter the qualification:mca

Enter the name:A.veena

Enter the age:38

enter the qualification:B.com

==================Data========================

A.sukumar 41 mca

­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A.Veena 38 b.com

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

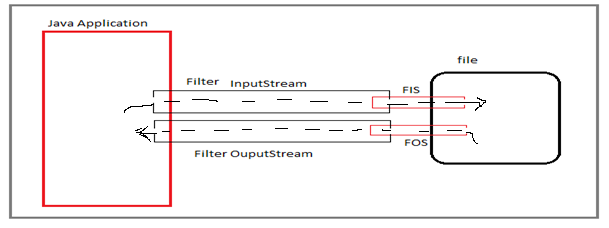
**Limitation of FIS and FOS:**

1. FIS and FOS allow us to read and write data only in the format of bytes.
2. It is not possible to read or write data in the format of primitive data(except byte) or objects(array,collection,string,…etc).
3. FIS read data only one byte from file at once and FOS writes data only one byte to at once.

Solution is: FilterInputStream and FilterOuputStream.

**8. FilterInputStream and FilterOuputStream:**

* Filters can not connect to source or destination directly, instead they can only be connected to another inputStream or OutputStream.
* The filter connected t inputStream is called FilterInputStream.
* The filter connected to OuutStream is called FilterOuputStream.
* So, all FilterInputStream and FilterOuputStream classes contain constructor to take other inputStream and OutputStream object as argument which it uses as its basic source of data for reading and writing.



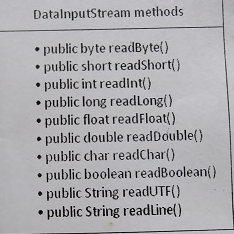
**8.1.DataInputStream and DataOuputStream**:- These classes are used to read and write data in primitive type size bytes from the underlying inputStreams and OuputStreams.

**a. DataInputStream:-** It is subclass of FilterInputStream and DataInput Interface.

Syntax:



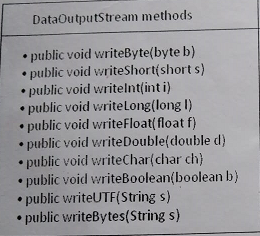
It implements below methods from DataInput interface for reading bytes from a binary input stream and convert them in to corresponding java primitive type.



**b. DataOutputStream:-** It is subclass of FilterOuputStream and DataOuput interface.



It implements below methods from DataOuput interface for converting data from any of the java primitive types to a stream of bytes and writing these bytes to binary output stream.



Public writeChars(String/char[] ab);

Application:1 Write a application demonstrates writing data in primitive type size bytes using Data Ouput stream class and reading data in primitive type size using DataInputStream class.

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.util.Scanner;

**public** **class** Demo {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

File nf=**new** File("D:/suku.txt");

nf.createNewFile();

String f="\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

String name,qual;

String age;

FileInputStream f1=**new** FileInputStream(nf);

FileOutputStream f2=**new** FileOutputStream(nf);

DataInputStream di=**new** DataInputStream(f1);

DataOutputStream d1=**new** DataOutputStream(f2);

Scanner s1=**new** Scanner(System.***in***);

System.***out***.print("Enter the Number of persons:");

**int** n=s1.nextInt();

s1.nextLine();

**int** i=0,j=0;

**while**(j<n)

{

System.***out***.print("Enter the name:");

name=s1.nextLine();

System.***out***.print("Enter the age:");

age=s1.nextLine();

System.***out***.print("enter the qualification:");

qual=s1.nextLine();

d1.writeUTF(name+" "+age+" "+qual);

d1.writeChar('\n');

d1.writeUTF(f);

d1.writeChar('\n');

/\*

\* d1.writeChar(); d1.writeChars(); d1.writeChar(); d1.writeChars(); d1.write();

\* d1.writeChars(); d1.write();

\*/

d1.flush();

j=j+1;

}

d1.close();

f2.close();

**int** x=0;

System.***out***.println("==================Data======================== ");

**while**(x<n)

{

System.***out***.print(di.readUTF());

System.***out***.print(di.readChar());

System.***out***.print(di.readUTF());

System.***out***.print(di.readChar());

++x;

}

}

}

Output:

Enter the Number of persons:2

Enter the name:A.sukumar

Enter the age:42

enter the qualification:mca

Enter the name:A.veena

Enter the age:38

enter the qualification:mba

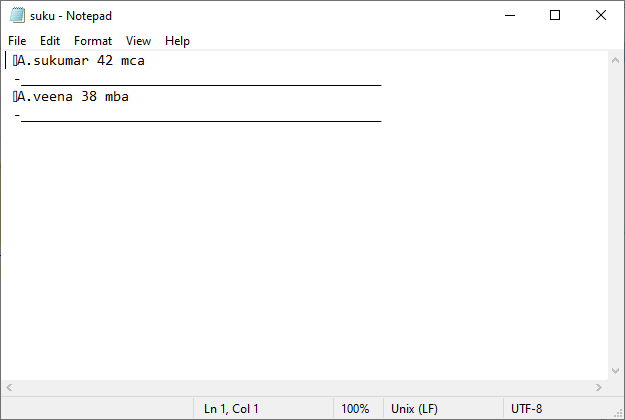
==================Data========================

A.sukumar 42 mca

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A.veena 38 mba

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Limitation: DIS class can’t read objects from persistence media and DOS class can’t write object to persistence media.

Solution: ObjectInputStream and ObjectOuputStream.

**SERIALIZATION & DESERIALIZATION**

**1. Marker Interface/Tag interface:-** The interface which does not have fields,methods and constants is known as ‘marker interface’. It delivers run-time type information about object.

The Empty interface is said to be merker interface.

Example to Built-in interface:

1. Serializable.
2. Clonable.
3. Remote Interface.
4. EventListener.
5. RandomAccess.

**2.Serialization**:- Serialization is process of converting objects into stream of bytes to transfer the object code from one JVM to another JVM and to send to OuputStream.

To send or to Write java object to external world/Output Stream , The object must be of type **java.io.Serializable** interface. It means this object class must be a subclass of java.io.Serializable interface. The Serializable is marker interface.

The Serialization operation is performed by **writeObject()** method of ObjectOuputStream .

The writeObject() method stores object state in file with below information.

1. Its classname.
2. SerialVersion UID. It is unique identifier of class.
3. Non-static, non-transient variable names and their data type.
4. Those variables current modified values.

**3.Deserialization**:- The Deserialization is the process of converting stream of bytes into original object. The Deserialization operation is performed by readObject() method of ObjectInputStream.

While Deserialization, the readObject() does following activites.

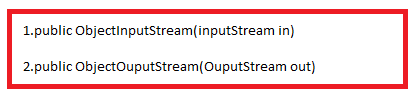
1. readObject() method first reads class name from serialized file and loads that class by using Class.forName(). While loading class static variables get memory with originally assigned values and SBs are executed.
2. Then it compares serialVersionUID of current loaded class with the serialVersionUID that is stored in serialized file.
3. If both are not same it terminates deserialization process by throwing exception **java.io.invalidClassException**.
4. If both values are same, it creates object with the current loaded class non-static variables without using new keyword and constructor.
5. Then readObject() method populates values

c.1. from serialized file for the non-transient variables and

c.2. default values for transient variables and also for the non-static variables those are not available in serialized file.

**4.ObjectInputStream & ObjectOuputStream classes:**

Syntax:



Application:1 The application demonstrating writing object to file.

import java.io.\*;

class Student implements Serializable

{

int sid=1;

String sname="suku";

float per=45.6f;

}

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException

{

ObjectOutputStream o1=new ObjectOutputStream(new FileOutputStream("E:/abc.txt"));

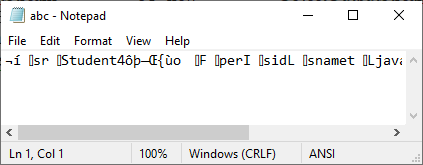
Student s1=new Student();

o1.writeObject(s1);

o1.close();

}

}



Application:2 The Application Demonstrates reading object from file and display object state on console.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

ObjectInputStream o1=new ObjectInputStream(new FileInputStream("E:/abc.txt"));

Student s1=(Student)o1.readObject();

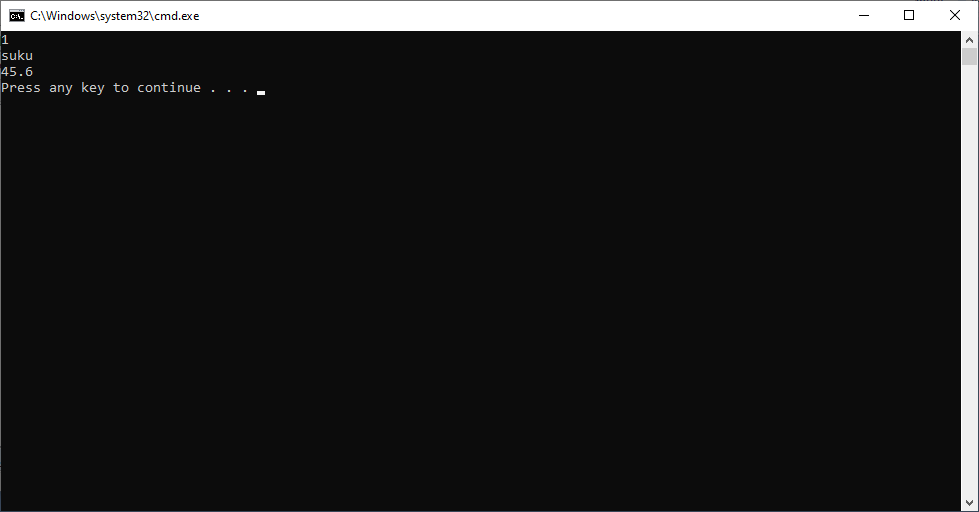
System.out.println(s1.sid);

System.out.println(s1.sname);

System.out.println(s1.per);

}

}



Example:3 Write a Java application to write 2 students into file and read 2 students from file and display on console.

**package** raos.sukumar;

**import** java.io.EOFException;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.Serializable;

**import** java.util.Scanner;

**class** Student **implements** Serializable{

**int** sno;

String sname;

**int** age;

**float** []marks=**new** **float**[5];

**public** Student(**int** a,String b, **int** c , **float** []d)

{

sno=a;

sname=b;

age=c;

marks=d;

}

}

**public** **class** Demo {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException,IOException,EOFException {

// **TODO** Auto-generated method stub

FileInputStream f1=**new** FileInputStream("D:/suku.txt");

FileOutputStream f2=**new** FileOutputStream("D:/suku.txt");

ObjectOutputStream o2=**new** ObjectOutputStream(f2);

Scanner s1=**new** Scanner(System.***in***);

Student s2=**new** Student(1,"sukumar",41,**new** **float**[] {2.3f,4.5f,7.6f});

Student s3=**new** Student(2,"veena",38,**new** **float**[] {2.5f,5.5f,9.6f});

o2.writeObject(s2);

o2.writeObject('\n');

o2.writeObject(s3);

o2.close();

ObjectInputStream o1=**new** ObjectInputStream(f1);

System.***out***.println("======================Data===================");

Student s4=(Student)o1.readObject();

o1.readObject();

Student s5=(Student)o1.readObject();

System.***out***.println(s4.sno+" "+s4.sname+" "+s4.age);

System.***out***.println(s5.sno+" "+s5.sname+" "+s5.age);

o1.close();

}

}

Output:-

======================Data===================

1 sukumar 41

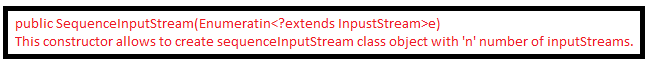
2 veena 38

**SEQUENCE STREAM**

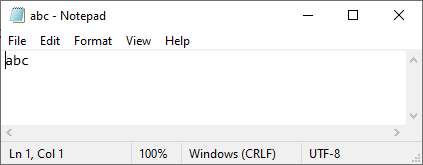
**1.SequenceInputStream:-**This class is used to read data from multiple input streams in sequence from the first one until end of file is reached, where upon it reads from the second one, and so on, until end of file is reached on the last of the contained input Streams.

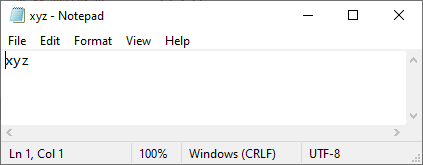
Syntax:





Application: This Program Demonstrates SequenceInputStream.





import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

int data;

FileInputStream f1=new FileInputStream("E:/abc.txt");

FileInputStream f2=new FileInputStream("E:/xyz.txt");

SequenceInputStream s=new SequenceInputStream(f1,f2);

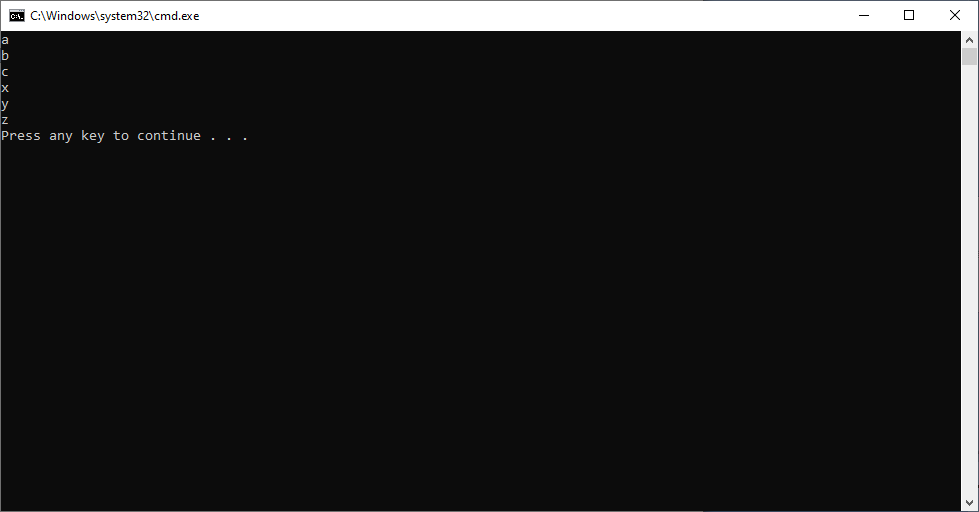
while((data=s.read())!=-1){

System.out.println((char)data);

}

}

}

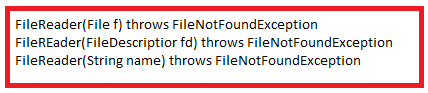


**CHARACTER STREAMS**

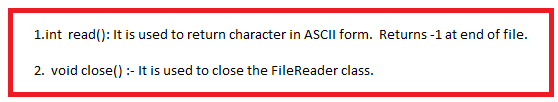
* The character stream classes are introduced in jdk1.1 version to read and write daa in terms of characters.
* For all character input streams Reader is super class.
* For all character output streams Writer is the super class.

1. **FileReader:-** This class is used to read data from **text** file as characters.

Syntax:

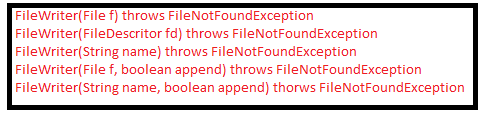


Methods;

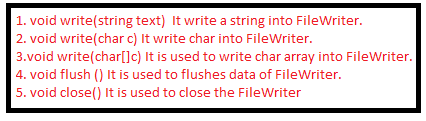


1. **FileWriter:**- This class is used to write data to **text** file as characters.

Syntax:-



Methods:



Application: This program demonstrates FileReader and FileWriter classes.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

FileReader f=new FileReader("E:/abc.txt");

FileWriter f1=new FileWriter("E:/abc.txt");

f1.write(97);

f1.write(98);

f1.close();

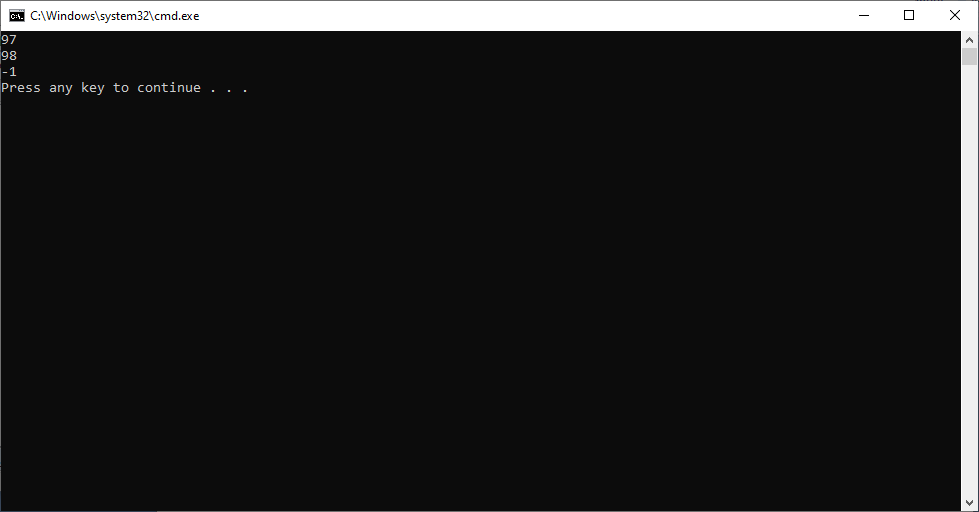
System.out.println(f.read());

System.out.println(f.read());

System.out.println(f.read());

}

}



Limitations :

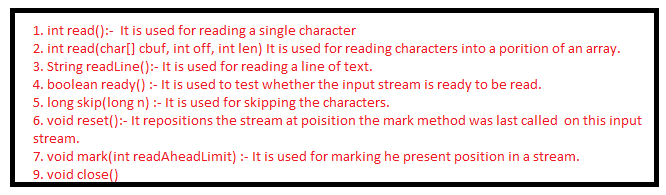
1. It is not recommended to use the reading and writing image files.
2. Using fileReader, we can only read one character at time. It reduces the application performance.

**3.BufferedReader:-** Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits [Reader](https://www.javatpoint.com/java-reader-class) [class](https://www.javatpoint.com/object-and-class-in-java).

Syntax:

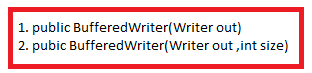


Methods:

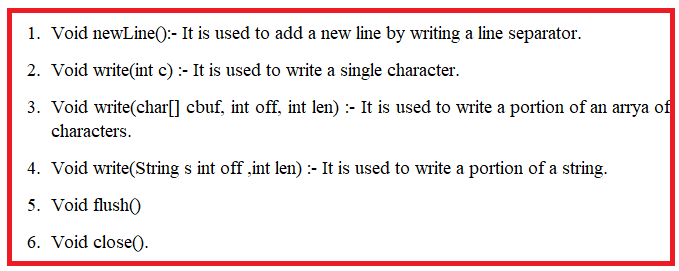


4.**BufferedWriter:-** Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits [Writer](https://www.javatpoint.com/java-writer-class) class. The buffering characters are used for providing the efficient writing of single [arrays](https://www.javatpoint.com/array-in-java), characters, and [strings](https://www.javatpoint.com/java-string).

Syntax:



Methods:



Application:- This application demonstrates BufferedInputStream and BufferedOuputStream.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

FileReader f=new FileReader("E:/abc.txt");

FileWriter f1=new FileWriter("E:/abc.txt");

BufferedReader bf=new BufferedReader(f);

BufferedWriter bf1=new BufferedWriter(f1);

bf1.write("This is sukumar");

bf1.newLine();

bf1.write("This is veena");

bf1.close();

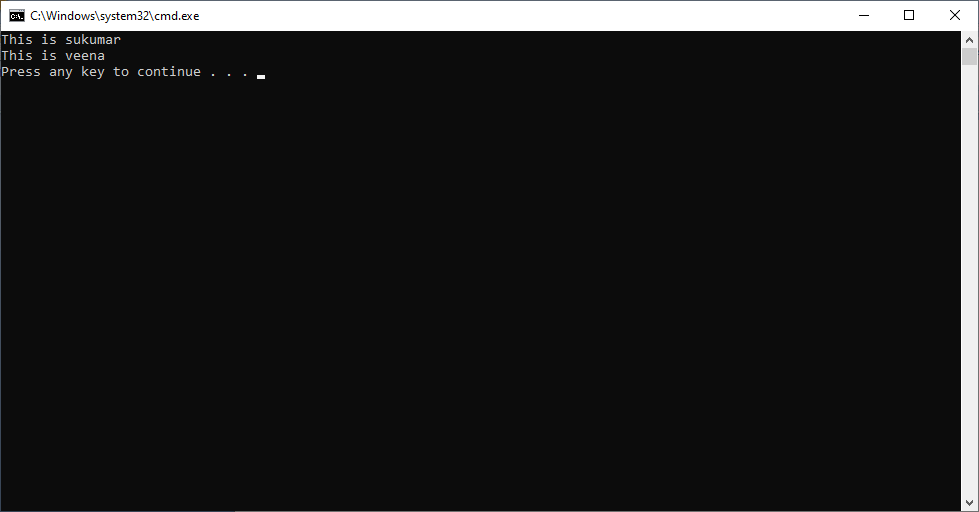
System.out.println(bf.readLine());

System.out.println(bf.readLine());

bf.close();

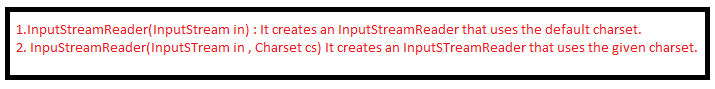
}

}

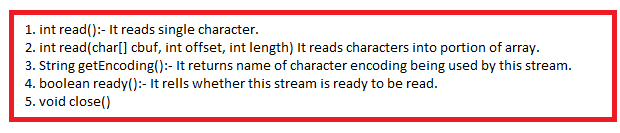


**5. InputStreamReader:-** An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Syntax:

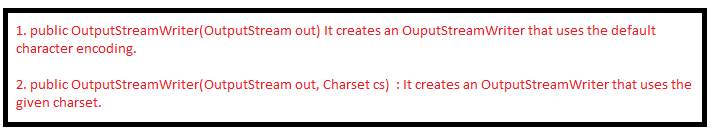


Methods:

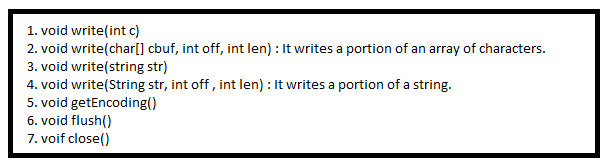


**6. OutputStreamWriter**:- OutputStreamWriter is a [class](https://www.javatpoint.com/object-and-class-in-java) which is used to convert character stream to byte stream, the characters are encoded into byte using a specified charset. write() method calls the encoding converter which converts the character into bytes. The resulting bytes are then accumulated in a buffer before being written into the underlying output stream.

Syntax:



Methods:



Application: This application demonstrates the InputStreamReader and OuputStreamWriter class.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

char data[]=new char[20];

FileInputStream f=new FileInputStream("E:/abc.txt");

FileOutputStream f1= new FileOutputStream("E:/abc.txt");

InputStreamReader f2=new InputStreamReader(f);

OutputStreamWriter f3=new OutputStreamWriter(f1);

f3.write("This is sukumar");

f3.write("This is veena");

f3.close();

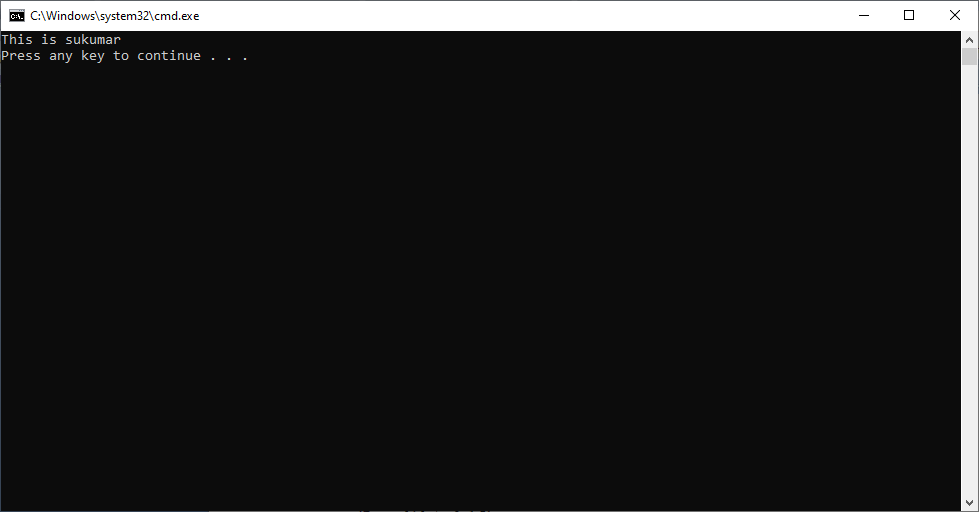
f2.read(data,0,15);

System.out.println(data);

f2.close();

}

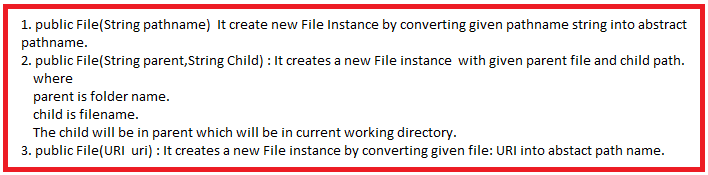
}



**FILE**

**1.File:-**  This class is used to create, delete ,rename files and directories and also used to know about the files and directory information like file-name, permisions on file …etc.

Syntax:



**1.1Methods**: This class has below methods to perform different operations on regular files and directories.



It checks whether file or direcotry is existed with given name or nor.

Return true, if fileis existed else returns false.



It creates an empty normal file with given name in given path.

Returns true, if the file is created with given file name.



It creates a directory with given name in the given path.

Returns true, if directory is created with given file name.



It cretes a both parent and child directories with gien names in the given path. It creates child directory inside a parent direcotry.

Returns true, if directory is created with given file.



It returns true, if given file name denoted a normal file.



It returns true, if given file name denoted directory.



Renames the current file with given name. It returns true, after renaming.



It deletes file immediately. It returns true, after deleting.



It deletes file after JVM terminates.



Returns true, if file is readable.



Returns true, if file is writable.



Returns true, if file is hidden.



It sets file is readable.



It returns last modified time in milliseconds.



It returns the length of file. The return value is unspecified if this pathname denotes a directory.



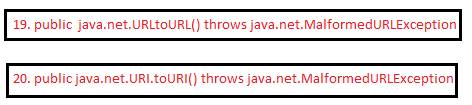
It returns all file and subdirectory names as String objects using String Array.



It returns all file and subdirectory names as File objects using Filearray.



It returns File object state in String format.



The above two functions convert file normal path to URL to URI.

**1.2. Fields:** File separators are platform dependent . In windows we must use “\\” and in Linux or solaries we must use “/” . so when we are moving project from windows to linux or solaries or vice versa in all java files we must change file separators to platform specific separator. Since it is manual task, It requires lot of testing so it lead lot of maintaince cost.

To solve this problem we must have way to retrieve file separator dynamically specific to current platform.

1. Public static final char separatorChar.
2. Public static final String separator.

Application#1:

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

File f1= new File("x.txt");

File f2= new File("rock", "y.txt");

File f3= new File("rock1");

File f4= new File(f3.toURI());

System.out.println(f1);

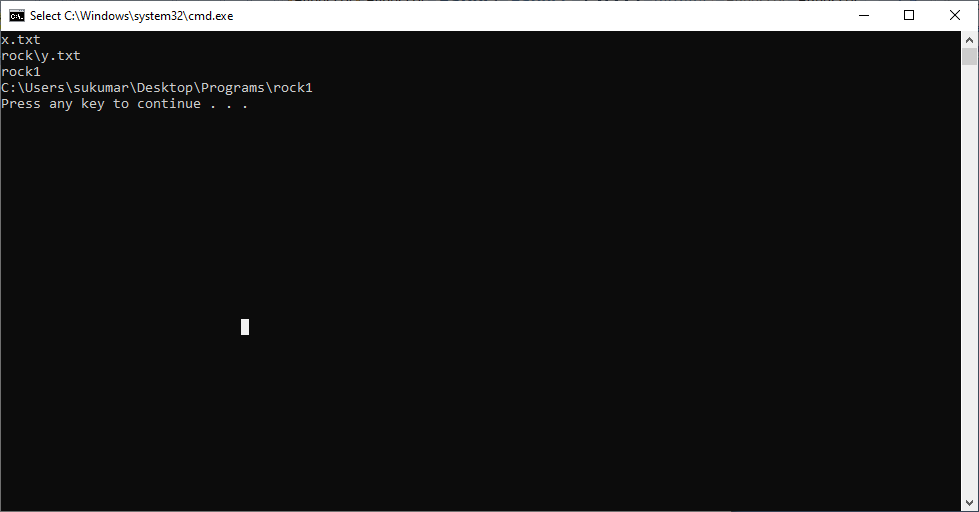
System.out.println(f2);

System.out.println(f3);

System.out.println(f4);

}

}

****

**Note:-** while executingabove program, the files x.txt,y.txt and rock1 will not be created in current working directory. JVM just creates file class object with given file name as that object state.

Application#2: This program demonstrates how to create file and displaying file information.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

File f1= new File("x.txt");

System.out.println("is File Created:"+ f1.createNewFile());

System.out.println("Is it File:"+ f1.isFile());

System.out.println("can we write in file?:"+f1.canWrite());

System.out.println("Is it readable file?:"+f1.canRead());

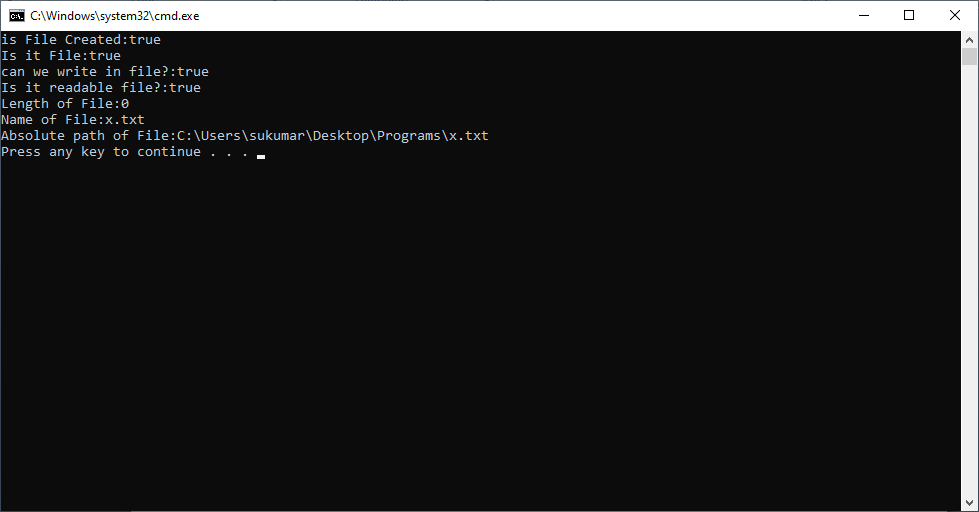
System.out.println("Length of File:"+ f1.length());

System.out.println("Name of File:"+f1.getName());

System.out.println("Absolute path of File:"+ f1.getAbsolutePath());

}

}



Application#2: This program demonstrates the “How to create directory” and display the directory information.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

File f1= new File("y.txt");

System.out.println("is File Created:"+ f1.mkdir());

System.out.println("Is it File:"+ f1.isFile());

System.out.println("can we write in file?:"+f1.canWrite());

System.out.println("Is it readable file?:"+f1.canRead());

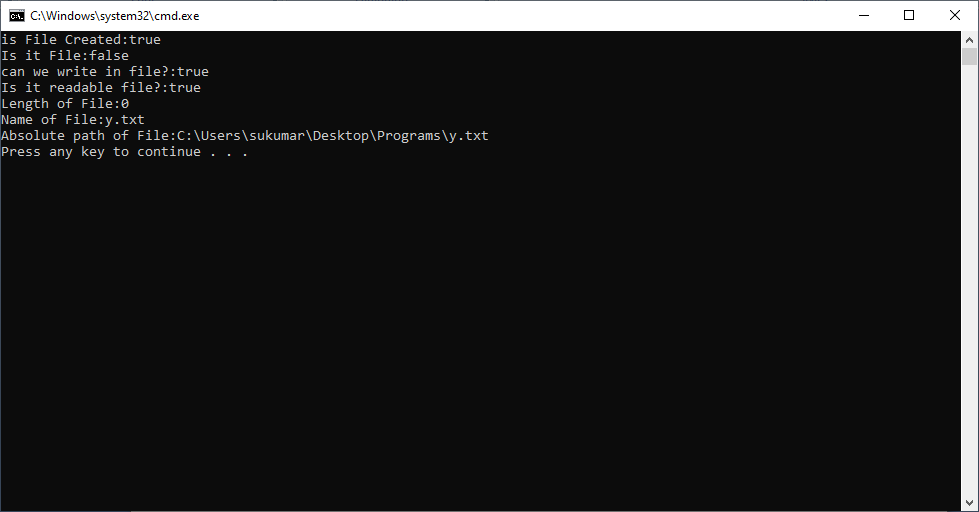
System.out.println("Length of File:"+ f1.length());

System.out.println("Name of File:"+f1.getName());

System.out.println("Absolute path of File:"+ f1.getAbsolutePath());

}

}



Application#4: This program demonstrates the file separator.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

File f1= new File("rock","z.txt");

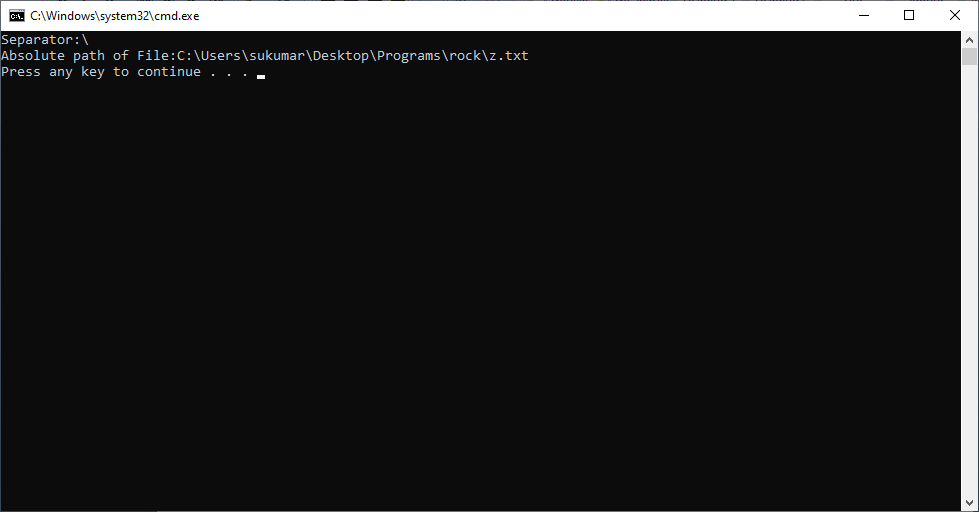
boolean x=f1.mkdir();

System.out.println("Separator:"+File.separator);

System.out.println("Absolute path of File:"+ f1.getAbsolutePath());

}

}



Application#5: This program demonstrates the how to delete a file.

import java.io.\*;

class sample

{

public static void main(String[] args) throws FileNotFoundException,IOException,ClassNotFoundException

{

File f1= new File("z1.txt");

boolean x=f1.createNewFile();

System.out.println("Delete a File:"+ f1.delete());

}

}

