JAVA I/O (Input / Output) Streams:

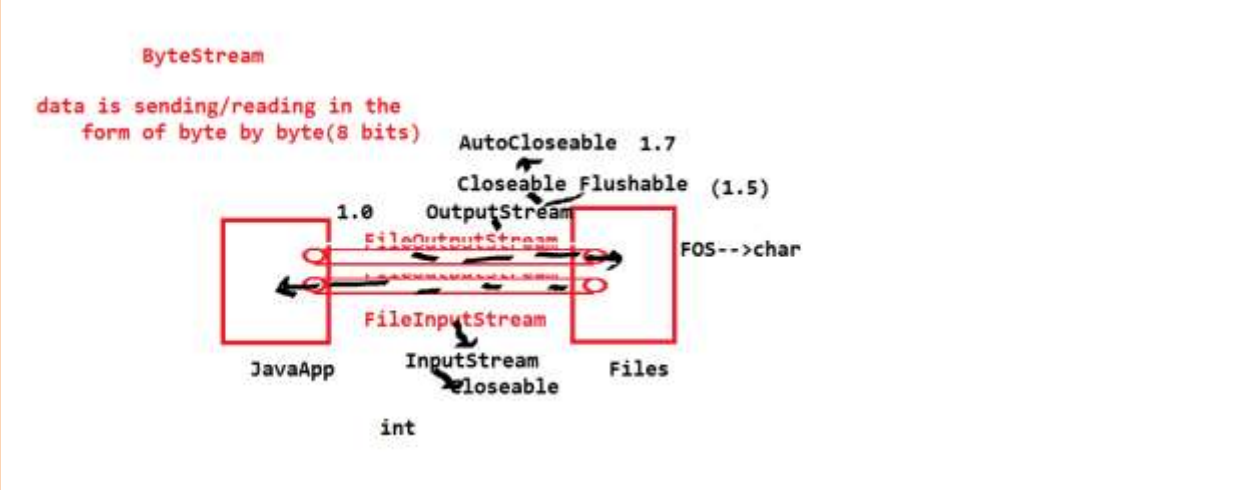
Using Java application, we can store the data permanently in a specific place either in file or in database. If java application wants to communicate with file and database we need some low level predefine logic, that given by java software in two packages.
java.io

java.sql

java. io package facilitates communication between java and files only. java.sql package facilitates communication between java and different databases(oracle, my-sql, db2, mysql, sqlserver, tera-data etc) .The logic used to store data permanently in one place is called Persistent Logic and the location is called Persistent Store. or database.
Stream: The two-way data flow between java application and files is called stream.

In java, there are two types of streams.
1) Byte stream.
2) Character stream.

Data flow between java and files in form of byte by byte constitutes the Byte stream. Data flow between java and files in form of character by character constitutes the Character stream.



java.io.OutputStream:

java.io.OutputStream is the super class for all output stream classes under the byte stream for writing the data into files. It is an abstract class. This abstract class implements two interface those are:

1) java.io.Closeable.

2)java.io.Flushable.

In this class we have one abstract method that is write(int). The various subclasses of this class they are:

ByteArrayOutputStream,FileOutputStream,FilterOutputStream,ObjectOutputStream, PipedOutputStream.
With the help of FileOutputStream, we can write data in byte format only. The data is stored in the file in the form character.

FileOutputStream uses either existing file or new file.

If file is not exists FOS created a new file with help given file name.

If file is existed FOS uses existed file name.

write() method and read() method always throws IOException. All io package classes throwing one more compile time exception that is java.io.FileNotFoundException.

FileInputStream is the basic class for reading the data from file.

This class always needs existed file name.

FIS class read the data from the file in the form of byte by byte.

The data is reading in the form int format.

```
import java.io.*;

public class FOSFISDemo{

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

		FileOutputStream fos = new FileOutputStream("vaibhav.text");

		fos.write('v');

		fos.write(65);

		fos.write('Z');

		System.out.println("data was entered in the form of character");

		FileInputStream fis = new FileInputStream("vaibhav.text");

		int i= 0;

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

			System.out.println(i+"..."+(char)i);

		}
```

```
            System.out.println("data was sucessfully read");
        }
}
```

Note: FileOutputStream fos = new FileOutputStream("vaibhav.text", true);

The above syntax allows new data to be appended to existing data, while keeping the old data intact.

java.io. InputStream:

java.io.InputStream is the super class for all the input stream classes under byte stream for reading data from file.

The various subclasses of this class are: AudioInputStream, ByteArrayInputStream, FileInputStream, ObjectInputStream, PipedInputStream, SequenceInputStrea StringBufferInputStream.

FileOutputStream and FileInputStream having the drawback like storing the data in the form of character and reading the data in the form of integer.

We cannot read the data in the different data type format.

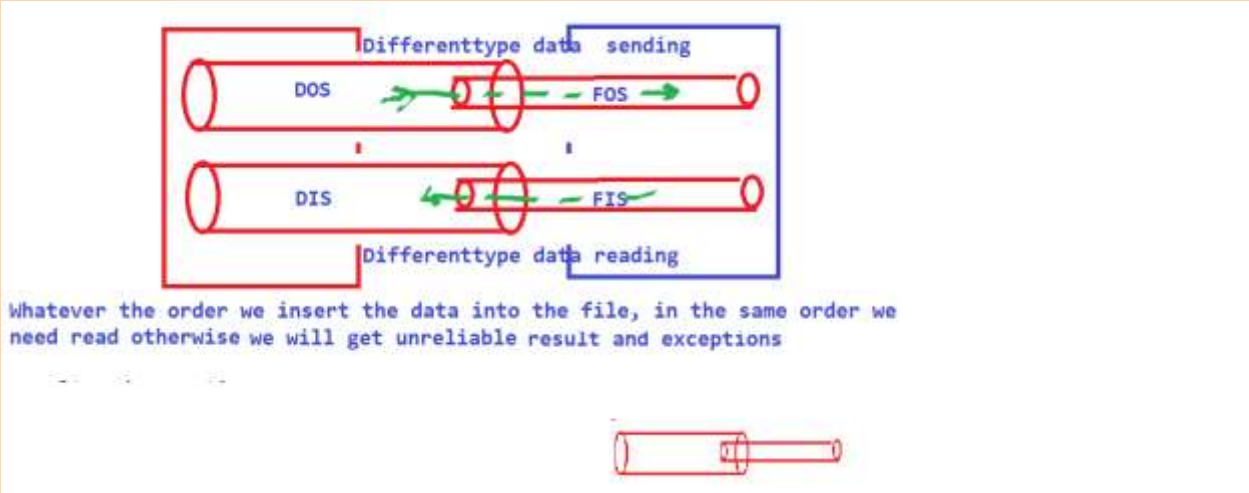
To resolve this problem we can go for DataInputStream and DataOutputStream.

FOS,FIS are basic classes for writing and reading the data to and from the files.

All classes are using these two basic classes for communicating with the files.

```
package streams;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class StreamDemo {
     public static void main(String[] args) throws
FileNotFoundException, IOException{
          try(FileOutputStream fos = new
FileOutputStream("nit1");){
               byte b[]= {65,66,67,68,69};
               fos.write(b);
          }

          /*FileOutputStream fos = null;
          try{
          fos = new FileOutputStream("nit");
          byte[] b = {100,97,100};
          fos.write(b);

          }
          catch(FileNotFoundException e){
               System.out.println("File is not available ");
          }
          catch(IOException e){
               System.out.println("writig and reading
problem");
          }
          finally{
               try {
                    if(fos !=null){
                         fos.close();
                    }
                    else{
                         System.out.println("connection is not
established");
                    }
               } catch (IOException e) {
                    e.printStackTrace();
```

```
            }
        }*/

    }
}
```




```
import java.io.*;

public class DOSDISDemo {

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

    FileOutputStream fos = new
    FileOutputStream("Vaibhav1.text");
    DataOutputStream dos = new DataOutputStream(fos);
    dos.writeByte(100);

    dos.writeChar('a');

    dos.writeInt(120);

    dos.writeBoolean(false);

    FileInputStream fis = new
    FileInputStream("Vaibhav1.text"); DataInputStream dis =
    new DataInputStream(fis);
    System.out.println(dis.readByte());
```

```
System.out.println(dis.readChar());

System.out.println(dis.readInt());

System.out.println(dis.readBoolean());
}
}
```

Note: DataOutputStream and DataInputStream classes are not directly communicate with files, first these two classes are communicate with FileOutputStream and FileInputStream respectively, by using FOS and FIS functionalities DOS and DIS are communicating with files for writing and reading the data in the form of different datatypes.

ByteArrayOutputStream :-

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

    FileOutputStream fos1 = new
FileOutputStream("siddharth1.text");

    FileOutputStream fos2 = new
FileOutputStream("siddharth2.text");

    ByteArrayOutputStream bos = new
ByteArrayOutputStream();

    bos.write(100);

    bos.writeto(fos1);

    bos.writeto(fos2);

    System.out.println("Success");
```

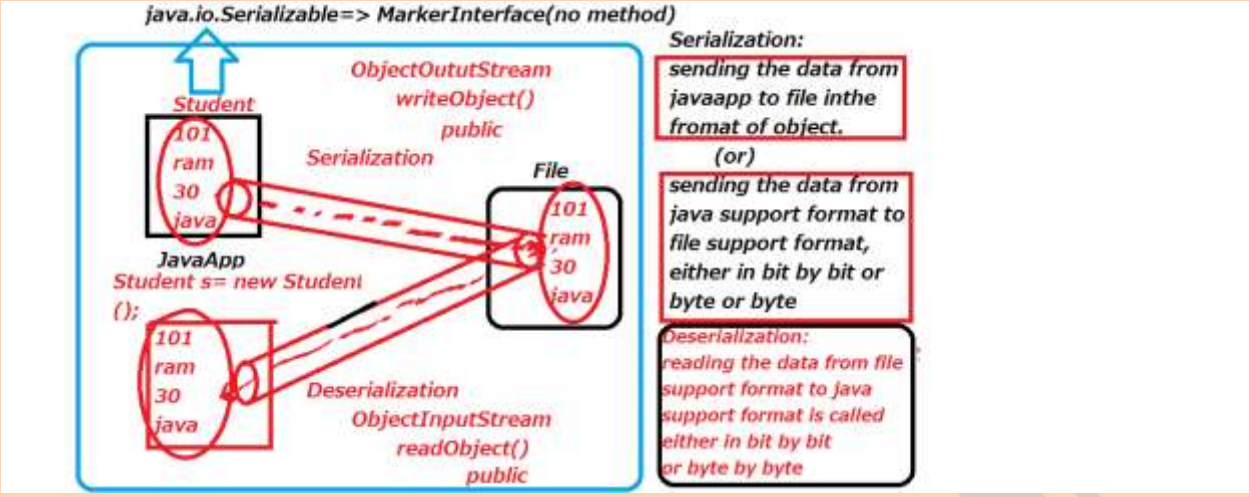
```
        FileInputStream fis1 = new
FileInputStream("Siddharth1.text");

        FileInputStream fis2 = new
FileInputStream("Siddharth2.text");

                int i = 0;

                while((i=sis.read()) != -1) {

                        System.out.println(i+".....”+(char)i);

                }

        }

}
```

Serialization:

Converting/ translating data in the form of object from java supported architecture to file support architecture is called serialization.In the process of serialization, the data is encrypted according to java internal algorithms.

To support serialization, java provides pre-defined class - java.io.ObjectOutputStream(writeObject(obj)).

It is mandatory that the object that is being serialized implements java.io.Serializable interface (if not - NotSerializableException).



De-Serialization:

To translate the data from file support format to java support format is called de-serialization.

De-serialization is supported by pre-defined class java.io.ObjectInputStream(readObject(obj)).

readObject() method will throws one compile time exception that is ClassNotFoundException.

In the deserialization jvm will create one new object, without executing the constructor.

java.io.serializable:

It is a marker interface i.e it does not contain any methods.Any class implementing Serializable, JVM will treat that class as special object and thereby participates in serialization.

```
import.java.io*;

class Student implements serializable{

      int sid = 111;  String sname = "siddharth";   int fee = 1000;      String password = "kick";
```

```
}
public class SerializationDemo {
    public static void main(String []args) throws IO Exception,
ClassNotFoundException
    FileOutputStream fos = new
FileOutputStream("suresh.ser");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    Student s = new Student();oos.writeObject(s);
    System.out.println("serialization success");
    FileInputStream fis = new FileInputStream("suresh.ser");
    ObjectInputStream ois = new ObjectInputStream(fis);
    object obj = ois.readObject();
    Student s1 = (Student)obj;
    System.out.println(s1.sid);
    System.out.println(s1.sname);
    System.out.println(s1.fee);
    System.out.println(s1.password);
    }
}
}
```

Note:

To stop an object from participating in serialization, we should mention data as 'transient' with the keyword transient String

password = “kick”; output -> NULL Static data is also does not participate in serialization.

static int sid = 111;

output -> 0

During Execution and in the process of deserialization, password is returned as NULL, but static value which is initially) is replaced by original value - 111.

Customized Serialization & De-serialization:-

```
import java.io.*;

class Student implements Serializable {

    int sid = 111;

    String sname = “siddharth”;

    int fee = 1000;

    transient String password = “ramchandra”;

private void writeObject(Object output stream oos) throws Exception {

    oos.default.writeObject();password = “123ramchandrahellohowru”;

    oos.writeObject(password);

}

private void readObject(Object Input Stream ois) throws Exception {

    ois.default readObject();

    String password1 = (String)ois.readObject();
```

```
        password = password1.substring(3,13);
}
public class SerializationDemo {
        public static void main(String []args) throws IO Exception,
ClassNotFoundException
        FileOutputStream fos = new
FileOutputStream("suresh.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        Student s = new Student();
        oos.writeObject(s);
        System.out.println("serialization success");
        FileInputStream fis = new FileInputStream("suresh.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        object obj = ois.readObject();
        Student s1 = (Student)obj;
        System.out.println(s1.sid);
        System.out.println(s1.sname);
        System.out.println(s1.fee);
        System.out.println(s1.password);
        }
        }
}
```

```
package com.io;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
class Student implements Serializable{
     int sid=101;
     String sname = "varun";
     transient String scourse = "java";
     transient String spassword = "varun123";
     int sage = 25;
     char sgender = 'm';
     int sfee = 5000;
     private void readObject(ObjectInputStream ois )throws
Exception{
          System.out.println("readObject method executing");
          ois.defaultReadObject();
          String dummypassword = (String)ois.readObject();
          String dummycourse = (String)ois.readObject();

          spassword =
dummypassword.substring(20,
25).concat(dummypassword.substring(10, 13));
          scourse = dummycourse.substring(7, 11);
     }
     private void writeObject(ObjectOutputStream oos)throws
Exception{
          System.out.println("writeObject method executing");
          oos.defaultWriteObject();
oos.writeObject("habeebsbii123ciciramvarunkiranmaheshnitjava
.net");
oos.writeObject(".netphpjavaandroidoraclephython");
     }
}
public class Serialization {
```

```
    public static void main(String[] args)throws
IOException,
    ClassNotFoundException{
        File f = new File("std.ser");
        FileOutputStream fos = new FileOutputStream(f);
        ObjectOutputStream oos = new
ObjectOutputStream(fos);
        Student s = new Student();
        oos.writeObject(s);
        System.out.println("serialization completed");
        FileInputStream fis = new FileInputStream(f);
        ObjectInputStream ois = new ObjectInputStream(fis);
        Object o = ois.readObject();
        Student s1 = (Student)o;
        System.out.println(s1.sid);
        System.out.println(s1.sname);
        System.out.println(s1.sgender);
        System.out.println(s1.sage);
        System.out.println(s1.scourse);
        System.out.println(s1.spassword);
        System.out.println(s1.sfee);
    }
}
```

Object Graph Serialization:-

```
class Dog implements Serializable {
    Dog(){
        System.out.println("zero argument constructor");
    }
    Cat c = new Cat();
}
Class Cat implements Serializable {
    Rat r = new Rat ();
```

```
}
Class Rat implements Serializable {
    int i = 111;
}
public class SerializationDemo {
public static void main(String []args) throws IO Exception,
FilenotFound Exception {
/* instead of student object creation in the above main method,
we create dog object and pass it to write object */
    Dog d = new Dog ();
    oos.writeObject(d);
    /*instead of casting to Student, we should cast into Dog*/
    Dog d1 = (Dog)obj;
    System.out.println(d1.c.r.i);
}
}
}
```

Note:

In the above program, if any class (Dog/Cat/Rat) does not implement serializable interface, then we get a java.io.NotSerializable Exception.

Importance of order:

```
class Dog1 implements Serializable {
        int i = 111;
}
Class Cat1 implements Serializable {
int j = 222;
}
Class Rat1 implements Serializable {
int k = 333;
}
public class SerializationDemo {
public static void main(String []args) throws IO Exception,
FilenotFound Exception {
FileOutputStream fos = new FileOutputStream("balaji.text");
            ObjectOutputStream oos = new
ObjectOutputStream(fos);
            Dog1 d = new Dog1();
            Cat1 c = new Cat1();
            Rat1 r = new Rat1();
            oos.writeObject(d);
            oos.writeObject(c);
            oos.writeObject(r);
            System.out.println("Success");
```

```
        FileInputStream fis = new
FileInputStream("balaji.text");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Rat1 r1 = (Rat1)ois.readObject();
        Dog1 d1 = (Dog1)ois.readObject();
        Cat1 c1 = (Cat1)ois.readObject();
/* the above order generates an error. Order is very important
and order given in serialization should be maintained for
deserialzation also. */
```

Serialization with Inheritance:

•If any sub-class undergoes serialization, then parent class is serialized automatically.

```
    class Animal implements Serializable{
        int i = 111;
    }
    class Cow extends Animal implements Serializable {
        int j = 222;
    }
            (program as above)
        Cow c = new Cow();
        oos.writeObject(c);
            (program as above)
        Cow c1 = (cow)ois.readObject();
```

```
System.out.println(c1.i);

System.out.println(c1.j);
```

Note:

•If parent class implements Serializable then the sub class need not always implement Serializable.

```
class Animal implements Serializable{

    int i = 111;

}

class Cow extends Animal {

        int j = 222;

}
```

•If sub-class implements Serializable and the parent class does not implement Serializable, then the super class data does not participate in serialization.

```
    class Animal {

    int i = 111;

}

class Cow extends Animal implements Serializable {

        int j = 222;

}

        Cow c = new Cow();

        c.j = 333;

        c.i = 444
```

output -> j = 333

i = 111 (not 444 because it did not participate in serialization)

Externalization:

```
class Student implements Externalizable {
    int sid;
    String sname;
    int sage;
    public Student (){
        System.out.println("student constructor");
    }
    public Student (int sid, String sname, int sage){
        super();
        this.sid = sid;
        this.sname = sname;
        this.sage = sage;
}
public void readExternal(Object input ois) throws IO Exception,
ClassNotFoundException{
    String sname1 = (String)ois.readObject();sname = sname1;
public void writeExternal(object output oos) throws IO Exception,
ClassNotFoundException{
    sname = "krishna";
```

```
        oos.writeObject(sname);
        }
}
public class Serialization Demo {
public static void main(String [] args) throws
FileNotFoundException{
        Student s = new Student(101, "peri", 24);
        oos.writeObject(s);
        object obj = ois.readObject();
        Student s1 = (Student)obj;
        System.out.println(s1);
        }
}
```

•When reading data from file in the process of de-externalization, public zero argument constructor is mandatory. If not available - invalid class Exception.

Difference betweenSerialization&Externalization:

In Serialization, everything is taken care of by JVM (no user input) whereas in Externalization, everything is taken care of by programmer only.

Performance of serialization is low as compared to Externalization.

Serialization needs Serializable interface (marker interface - no method).

Externalization needs Externalizable interface.

Externalizable interface - needs two methods writeExternal() & readExternal().

Serialization does not require any default constructors whereas Externalization needs public zero arguemnt constructor.

In Serialization there is a use of transient keyword but in externalization no use.

serialVersionUID:

It is one private static final long type variable, creating by the JVM for every .class file.

This serialVersionUID is changes from one os to another os and one version of jvm to another version of jvm, one bit to another bit, and based content.

If we are not writing any serialVersionUID in our .java file by default JVM will gives one uniquire identification number, if we are mention serialVersionUID in our program then JVM not giving its own unique identification number, if will uses our own identification number only.

create one folder like USA and create the following files and saved in USA folder.

Student.java

```
public class Student implements java.io.Serializable{

    //private static final long serialVersionUID=600l;

    int sid = 101;

    String sname = "Reddy";

    int sage = 24;
```

```
    double sfee = 5100.00;   String scourse = "java";
    String saddr="LBNagar";
}
```

SDemo.java:

```
import java.io.*;
public class SDemo{
    public static void main (String[] ram)
    throws IOException,ClassNotFoundException{
        FileOutputStream fos = new
FileOutputStream("std.ser");
        ObjectOutputStream oos = new
ObjectOutputStream(fos);
        Student s1 = new Student();
        oos.writeObject(s1);//upcasting
        System.out.println("Serialization completed");
    }
}
```

Open command prompt and goto USA folder and do the following things.

Javac Student.java

Javac SDemo.java

Java SDEmo

Create one folder give name as INIDA and do the following things(creating classes separately).

Student.java:

Student.java

```
public class Student implements java.io.Serializable{
    //private static final long serialVersionUID=600l;
    int sid = 101;
    String sname = "Reddy";
    int sage = 24;
    double sfee = 5100.00;      String scourse = "java";
    String saddr="LBNagar";
}
```

DSDemo.java:

```
import java.io.*;
public class DSDemo{
    public static void main(String[] s)throws IOException,
    ClassNotFoundException{
        FileInputStream fis = new
FileInputStream("std.ser");
        ObjectInputStream ois = new
ObjectInputStream(fis);
        Object o = ois.readObject();//UC
        Student s2 = (Student)o;//DC
    System.out.println(s2.sid);
```

```
        System.out.println(s2.sname);
        System.out.println(s2.sage);
        System.out.println(s2.scourse);
        System.out.println(s2.sfee);
        System.out.println(s2.saddr);

        }

}
```

COPY THE STD.SER FILE FROM USA FOLDER TO INIDA FOLDER THEN DO THE FOLLOWING THINGS.

Javac Student.java

Java DSDemo.java

Java DSDemo

We can able to get appropriate data (output).

Open INDIA folder Student.java and add bellow code into that .java file like bellow.

Student.java

```
public class Student implements java.io.Serializable{

        //private static final long serialVersionUID=600l;

        int sid = 101;

        String sname = "Reddy";

        int sage = 24;

        double sfee = 5100.00;

        String scourse = "java";

        String saddr="LBNagar";
```

String sfaculty=”ramchandar”;

String scountry=”india”;

}

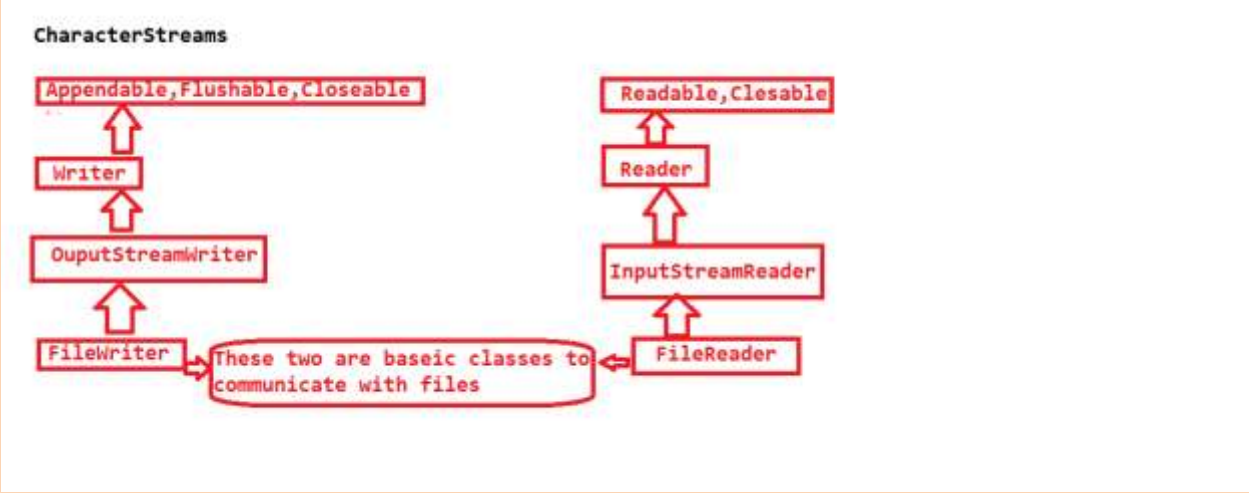
Then compile again the Student.java and execute DSDemo.class file like bellow.

Javac Student.java

Java DSDemo

Output: we will get one exception like java.io.InvalidClassException.

To avoiding this problem in both folder(USA/INDIA) un comment the serialVersionUID statement in Student.java file. Then do the things,which we were done from starting onwards.



import java.io.*;

public class FWAFRDemo {

```
	public static void main(String[] args)throws
FileNotFoundException,IOException {
		FileWriter fw = new FileWriter("sony.text");
		fw.write(100);
		fw.write('a');
		fw.write('z');
		fw.flush();
		fw.close();
		FileReader fr = new FileReader("sony.text");
		int i=0;
		while((i=fr.read())!=-1){
			System.out.println(i+"...."+(char)i);
		}
	}
}
```

BufferedReader:

It is having two methods.

1. **readLine():**
   Will read the data in the form of String.
2. **read():**
   Will read only one single character ascii value. With the help of BufferedReader we can read the data from keyboard and also from files.

```
import java.io.*;

import java.util.StringTokenizer;

public class FWAFRDemo {

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

		InputStreamReader isr = new
InputStreamReader(System.in);

		BufferedReader br = new BufferedReader(isr);

		System.out.println("enter some data");

		String s = br.readLine();

		System.out.println(s);

		System.out.println("enter some data");

		int i = br.read();

		System.out.println(i);

		FileReader fr = new FileReader("sony.text");

		BufferedReader br1 = new BufferedReader(fr);

		String s1 = br1.readLine();

		System.out.println(s1);

		int total = 0;

		StringTokenizer st = new StringTokenizer(s1," ");

		while(st.hasMoreTokens()){

			total = total+Integer.parseInt(st.nextToken());

		}
```

```
            System.out.println(total);
        }
}
```

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
public class Demo {
        public static void main(String[] args) throws IOException{
        //1.  writing the data on keyboard
        OutputStreamWriter osw = new
OutputStreamWriter(System.out);
                BufferedWriter bw = new BufferedWriter(osw);
                bw.write("kiran");
                bw.flush();

        //2.  writing the data in file

                File f1 = new File("sbaj");
                FileWriter fw = new FileWriter(f1);
                BufferedWriter bw = new BufferedWriter(fw);
                bw.write("nareshit");
```

```
        bw.close();
        System.out.println("data placed into file");

//3.  reading the data from file

        File f1 = new File("sbaj");
        FileReader fr = new FileReader(f1);
        BufferedReader br = new BufferedReader(fr);
        String s = br.readLine();
        System.out.println("s: "+s);

//4.        reading the multiple lines from file
        File f2 = new File("sbaj1");
        FileReader fr2 = new FileReader(f2);
        BufferedReader br2 = new BufferedReader(fr2);
        for(int i=1;i<=5;i++){
             String s1 = br2.readLine();
             System.out.println("s1: "+s1);
        }

//5.  reading the data from keyboard
```

```
        InputStreamReader isr = new
InputStreamReader(System.in);
                BufferedReader br3 = new BufferedReader(isr);
                System.out.println("enter some data");
                String s3 = br3.readLine();
                System.out.println("s3: "+s3);
                System.out.println("enter some data");
                int a = br3.read();
                System.out.println("a: "+a);

        //6   counting number of line existed in file

                File f  = new File("sbaj1");
                System.out.println(f.getAbsolutePath());

                System.out.println(f.getPath());
                FileReader fr = new FileReader(f);
                BufferedReader br = new BufferedReader(fr);
                 int lineNumber = 0;
              String data = "";
              do {
                 data = br.readLine();
```

```
            if (data != null) {
                lineNumber++;
            } // End of the if //
        } while(data != null);
        System.out.println("lineNumber: "+lineNumber);

    }
}
```

java.io.Console:

```
import java.io.*;
class ConsoleDemo {
public static void main(String[] args)
throws FileNotFoundException,IOException {
    Console con = System.console();
    System.out.println("enter some data");
    String s = con.readLine();
    System.out.println(s);
    System.out.println("enter some data");
    char c[] = con.readPassword();
```

```
        System.out.println(c);     String s1 = new String(c);
        System.out.println(s1);
        }
}
class Check {
        public static void main(String...ram)throws Exception{
                System.out.println("check class");
                String s = System.getProperty("loadingclass");
                Class.forName(s);
        }
}
java -Dloadingclass=Student Check
```

System.out.println():

out is a static field in System class. So we are called out filed by using class name(System).

But println() is not available in System class, this println() available in java.io.PrintStream. It is a non-static method.

If we want call println() we need PrintStream class reference.

If we are creating reference PrintStream and calling println() the data is not print on the console, the data will be print on the file.

If we print the data on the console, we System.out. This System.out will represents output device like console.

If we are using System.out.println() then only the data will print on the console.

```
import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.PrintStream;

public class FWAFRDemo {

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

	FileOutputStream fos = new FileOutputStream("mona.text");

	PrintStream ps = new PrintStream(fos);

	ps.println("data is not print on console");

	System.out.println("data is print on console");

	System.err.println("this is also represents console");

	System.setOut(ps);

	System.setErr(ps);

	System.out.println("data is print on console");

	System.err.println("this is also represents console");

	}

}
```

```
class A implements Serializable{
	String s1 = "ram";
}
```

```
class B implements Serializable{
     String s2 = "suji";
}
class C implements Serializable{
     String s3 = "vani";
}
 public class StreamDemo {
     public static void main(String[] args)
               throws FileNotFoundException, IOException,
               ClassNotFoundException{
          FileOutputStream fos = new
FileOutputStream("file.text");
          ObjectOutputStream oos = new
ObjectOutputStream(fos);
          A obj1 = new A();
          B obj2 = new B();
          C obj3 = new C();
          obj1.s1 = "java";
          obj2.s2 = "science";
          obj3.s3 = "SE";
          oos.writeObject(obj1);
          oos.writeObject(obj2);
          oos.writeObject(obj3);
          FileInputStream fis = new
FileInputStream("file.text");
          ObjectInputStream ois = new ObjectInputStream(fis);

          for(int i=0;i<3;i++){
               Object o = ois.readObject();
               if(o instanceof A){
                    A obj4 = (A)o;
                    System.out.println(obj4.s1);
               }
               else if(o instanceof B){
                    B obj5 = (B)o;
                    System.out.println(obj5.s2);
               }
                    else{
                         C obj6 = (C)o;
```

```
                    System.out.println(obj6.s3);
                }
            }
        }
}
```

SERIALIZATION WITH IS-A RELATION

super class	sub class	class which is partispated in Serialization	superclassimplements Serializable	subclassimplements serializable	Result
A	B	B	Yes	Yes	Both
A	B	B	NO	Yes	only subclas
A	B	B	Yes	No	Both
A	B	A	Yes	No	only super
A	B	A	Yes	Yes	only super

Program on create temparory file and getting paths:

```
import java.io.File;
import java.io.IOException;


public class B {
    public static void main(String[] args)throws Exception{
    try{
        File temp = File.createTempFile("i-am-a-temp-file",
".tmp" );

        String absolutePath = temp.getAbsolutePath();

System.out.println(absolutePath.lastIndexOf(File.separator))
;
        System.out.println("File path : " + absolutePath);
```

```
            String filePath = absolutePath.

substring(0,absolutePath.lastIndexOf(File.separator));
            System.out.println("File path : " + filePath);
        }catch(IOException e){
            e.printStackTrace();
        }
       }
}
```

Try with resource:

Syntax:

```
try(--){

}
```

The main intension of try with resource blocks is closing connections between java applications to files, placing data into files automatically (without using flush()) and handle the exceptions.

In resource place we can write only class which are implements java.io.AutoCloseable interface.

In resource place we can write more than one statements.

Try with resource block no need to ended with catch or finally block.

If we want to handle then write catch blocks otherwise just use throws keyword on top of the method.

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class Demo {
```

```
     public static void main(String[] args) throws
IOException{
          File f = new File("trywithresource.txt");
          try(FileWriter fw = new FileWriter(f);){
               char c[] = {'m','o','m'};
               fw.write(c);
               fw.flush();
          }
          /*catch(Exception e){

          }*/
          System.out.println("-----------");
     }

}
```

How to read image data and creating new image in JSE:

```
package com.io;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;

public class Demo {
     public static void main(String[] args) throws
IOException{
          File f = new
File("C:\\Users\\Ramchandar\\Desktop\\Koala.jpg");
          FileInputStream fis = new FileInputStream(f);

          File f1 = new
File("C:\\Users\\Ramchandar\\Desktop\\Annie.jpg");
          FileOutputStream fos = new FileOutputStream(f1);
          int i=0;
          while((i=fis.read())!=-1){
               fos.write(i);
          }
     System.out.println("image created");
```

```
    }

}
```

Try with resource with multiple statements:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class Demo {
    public static void main(String[] args) throws
IOException{
        File f = new File("trywithresource.txt");
        try(FileReader fr = new FileReader(f);
            BufferedReader br = new BufferedReader(fr); ){
        String s = br.readLine();
        System.out.println("s: "+s);
        }
    }

}
```