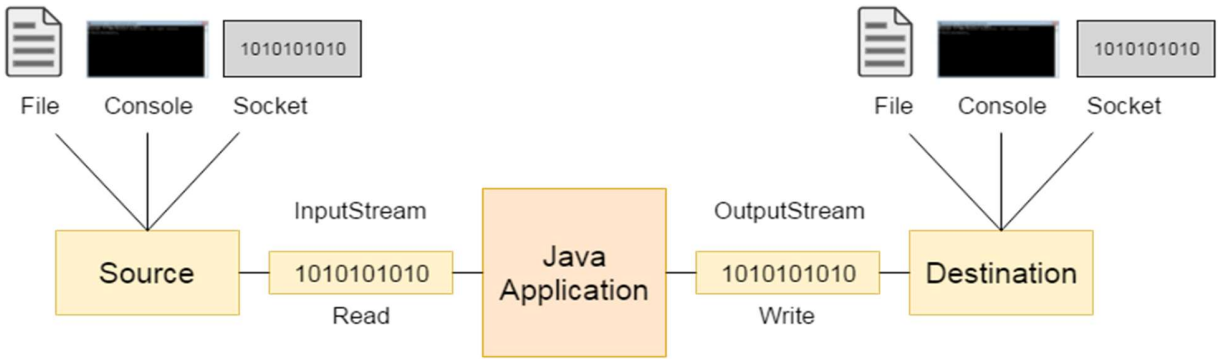
JAVA
Input/Output

By :

Subodh Sharma



Java Input/Output

Java.io package contains all the classes required for input and output operations. Java provides strong, flexible support for I/O as it relates to files, console and networks.
Java uses the concept of a stream to make I/O operation fast. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.

Stream : A stream can be defined as a sequence of data. A stream is linked to a physical device (keyboard, monitor, hard-disk, network socket etc.) by the java to produce or consume information.

In Java, three standard streams are created for us automatically. All these streams are attached with the console. **1) System.out:** standard output stream **2) System.in:** standard input stream **3) System.err:** standard error stream

There are two kinds of Streams –

Byte Streams – provide a convenient means for handling input/output of bytes. It is used when reading or writing binary data.

Byte streams are defined by using two class hierarchies. At the top two abstract classes: InputStream and OutputStream. Each of these abstract classes has several concrete subclasses, that handle various devices, such as disk files, network connection, memory buffer etc. . The two most important abstract methods are read() and write(). They are overridden by derived stream classes.

Character Streams – provide a convenient means for handling input/output of characters. They use Unicode and, therefore, can internationalized.

Character streams are defined by using two class hierarchies. At the top two abstract classes: Reader and Writer. Each of these abstract classes has several concrete subclasses. The most important methods are read() and write() which read and write characters of data. These methods are overridden by derived stream classes.

- **Byte Streams** − These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.
- **Character Streams** − These handle data in 16 bit Unicode. Using these you can read and write text data only.

Byte Streams:

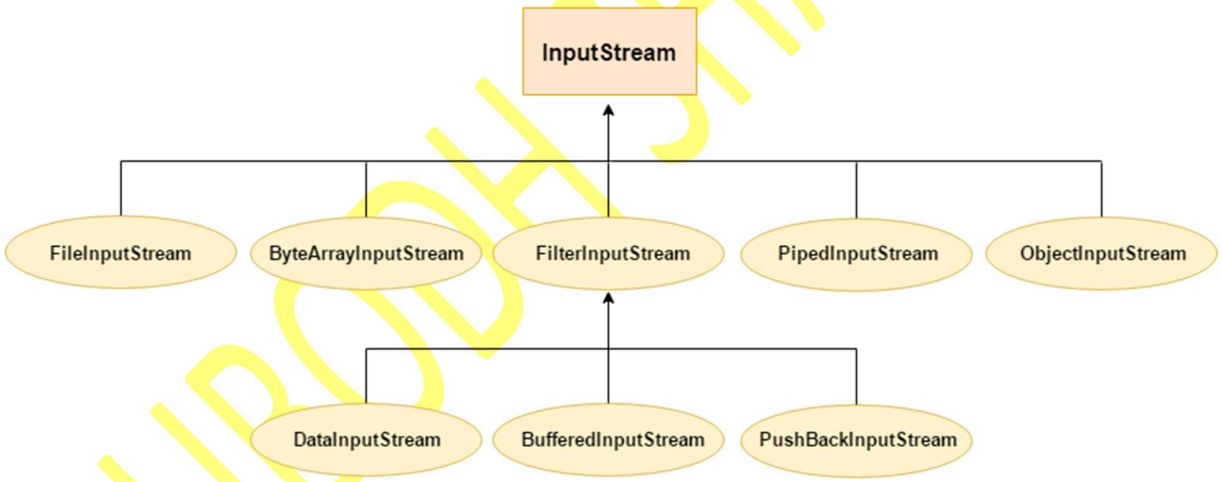
InputStream :

InputStream class is an abstract class. It is the super class of all classes representing an input stream of bytes.

InputStream is used to read data that must be taken as an input from a source array or file or any peripheral device. Few important methods are :

Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

InputStream Hierarchy



Concrete Subclasses of InputStream :

BufferedInputStream	Buffered input stream
ByteArrayInputStream	Reads from a byte array
DataInputStream	Contains methods for reading the java standard data types
FileInputStream	Read from file
FilterInputStream	Implements InputStream
ObjectInputStream	
PipedInputStream	Input pipe
PushbackInputStream	Which returns a byte to the input stream
SequenceInputStream	Combination of two or more inputs

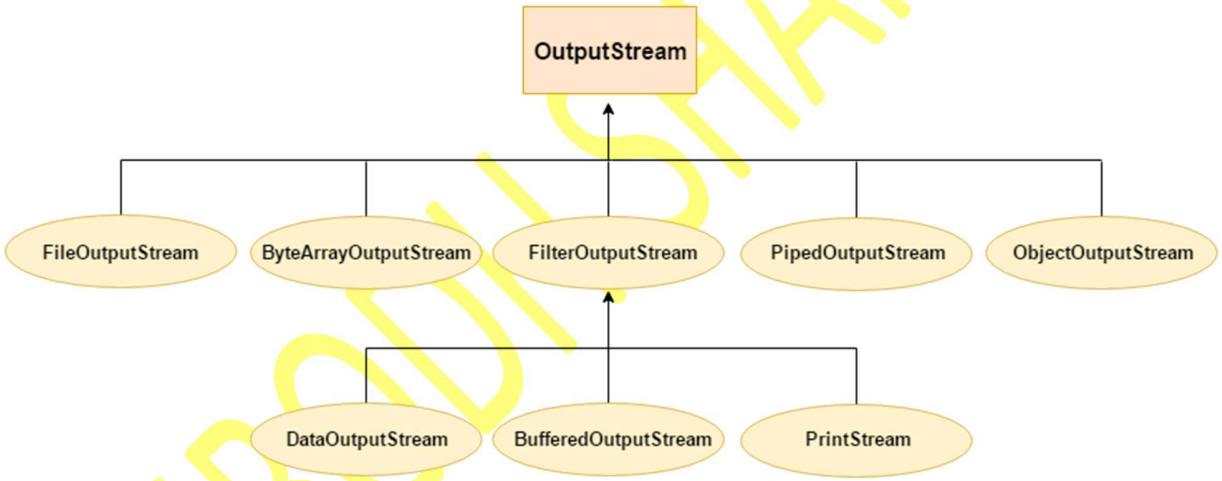
OutputStream :

OutputStream class is an abstract class. It is the super class of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Few important methods are :

Method	Description
1) public abstract void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.

OutputStream Hierarchy



Concrete Subclasses of OutputStream :

BufferedOutputStream	Buffered output stream
ByteArrayOutputStream	That writes to a byte array
DataOutputStream	Contains methods for writing the java standard data types
FileOutputStream	Writes to a file
FilterOutputStream	Implements OutputStream
ObjectOutputStream	
PipedOutputStream	Output pipe
PrintStream	Stream that contains print() and println()

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

the declaration for java.io.FileInputStream :

public class **FileInputStream** extends **InputStream**

The most common constructors are show here :

FileInputStream(String filepath)
FileInputStream(File fileObj)

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("t.txt");
          int i=fin.read();
          System.out.print((char)i);

          fin.close();
         }catch(Exception e){System.out.println(e);}
        }          }
```

Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a file.
If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

The most common constructor used are :

FileOutputStream (String filePath)
FileOutputStream (File fileObj)
FileOutputStream (String filePath, Boolean appnd)
FileOutputStream (File fileObj, Boolean appnd)

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

```
import java.io.*;
class I3
{
  public static void main(String s[]) throws IOException
  {
    FileOutputStream fo = new FileOutputStream("Ofile.txt");
     fo.write(84);
  String str="his is a string";
  byte b[]=str.getBytes();
   for(int i=0;i<b.length;i++)
      fo.write(b[i]);
    fo.write(10);
    fo.write(b);
    fo.close();
  }
}
```

File :

Although most of the classes defined by java.io. operates on stream but the File class does not. It deals directly with files and file system. That is, the File class does not specify how information is retrieved from or stored in files; it describes the properties of a file itself. A File object is used to obtain or manipulate the information associated with a disk file, such as permissions, time, date and directory path etc.

The following constructors can be used to create File objects :

File(String directoryPath)
File(String directoryPath, String filename)
File(File dirObj, String filename)
File(URI uriObj)

Here the example :

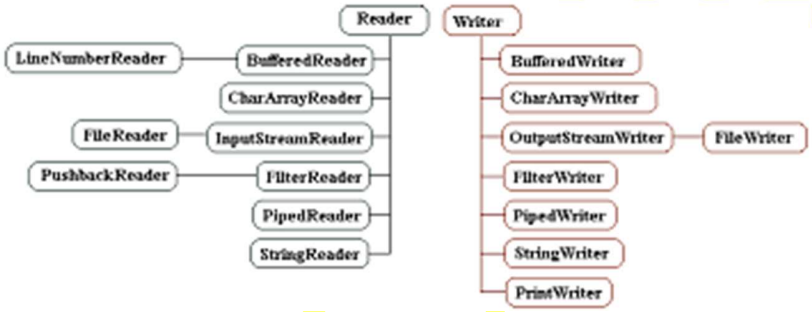
```
import java.io.*;
class I4
{
        static void p(String str)
                        {  System.out.println(str);
                        }
 public static void main(String s[])
 {
   File fl = new File(s[0]);
p("FileName :"+fl.getName());
p("Path     :"+fl.getPath());
p("Abs Path :"+fl.getAbsolutePath());
p("Parent   :"+fl.getParent());
p("exist    :"+(fl.exists() ? "file exist" : " file doesnot exist"));
p("canWrite :"+(fl.canWrite() ? "yes" : " no "));
p("canRead  :"+(fl.canRead() ? "yes" : " no "));
p("isDirectory  :"+(fl.isDirectory() ? "yes A folder" : "not a folder"));
p("isFile       :"+(fl.isFile() ? "yes a file" : " not a file "));
p("lastModified :"+fl.lastModified());
p("Length     :"+fl.length());
 }
}
```

Character Streams – provide a convenient means for handling input/output of characters. They use Unicode and, therefore, can internationalized.

In Java, characters are stored using Unicode conventions. Character stream automatically allows us to read/write data character by character. For example FileReader and FileWriter are character streams used to read from source and write to destination.

Reader :
Reader is an abstract class that defines streaming character input. All of the methods in this class will throw an IOException on error conditions. .

Writer :
Writer is an abstract class that defines streaming character output. All of the methods in this class return a void value and throw IOException in the case of errors.



Java Reader

Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency, additional functionality, or both.

Some of the implementation class are BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader, FileReader

Modifier and Type	Method	Description
abstract void	close()	It closes the stream and releases any system resources associated with it.
void	mark(int readAheadLimit)	It marks the present position in the stream.
boolean	markSupported()	It tells whether this stream supports the mark() operation.
int	read()	It reads a single character.

int	read(char[] cbuf)	It reads characters into an array.
abstract int	read(char[] cbuf, int off, int len)	It reads characters into a portion of an array.
int	read(CharBuffer target)	It attempts to read characters into the specified character buffer.
boolean	ready()	It tells whether this stream is ready to be read.
void	reset()	It resets the stream.
long	skip(long n)	It skips characters.

FileReader :

The FileReader class creates a Reader that we can use to read the contents of a file. Its two most commonly used constructors are :

FileReader(String filePath)
FileReader(File fileObj)

```
// Java Program illustrating that we can read a file in
// a human readable format using FileReader
import java.io.*; // Accessing FileReader, FileWriter, IOException
public class GfG
{
	public static void main(String[] args) throws IOException
	{
		FileReader sourceStream = null;
		try
		{
			sourceStream = new FileReader("test.txt");

			// Reading sourcefile and writing content to
			// target file character by character.
			int temp;
			while ((temp = sourceStream.read()) != -1)
				System.out.println((char)temp);
		}
		finally
		{	// Closing stream as no longer in use
			if (sourceStream != null)
				sourceStream.close();
		}
	}
}
```

Java Writer

It is an abstract class for writing to character streams. The methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.

Constructor

Modifier	Constructor	Description
protected	Writer()	It creates a new character-stream writer whose critical sections will synchronize on the writer itself.
protected	Writer(Object lock)	It creates a new character-stream writer whose critical sections will synchronize on the given object.

Methods

Modifier and Type	Method	Description
Writer	append(char c)	It appends the specified character to this writer.
Writer	append(CharSequence csq)	It appends the specified character sequence to this writer
Writer	append(CharSequence csq, int start, int end)	It appends a subsequence of the specified character sequence to this writer.
abstract void	close()	It closes the stream, flushing it first.
abstract void	flush()	It flushes the stream.
void	write(char[] cbuf)	It writes an array of characters.
abstract void	write(char[] cbuf, int off, int len)	It writes a portion of an array of characters.
void	write(int c)	It writes a single character.
void	write(String str)	It writes a string.
void	write(String str, int off, int len)	It writes a portion of a string.

FileWriter :

The FileWriter class creates a Writer that we can use to write the contents on a file. Its most commonly used constructors are :

FileWriter(String filePath)
FileWriter(String filePath, boolean append)
FileWriter(File fileObj)
FileWriter(File fileObj, boolean append)

They can throw IOException,

```
import java.io.*;
public class FileRead {

   public static void main(String args[])throws IOException {
      File file = new File("t.txt");

      // creates the file
      file.createNewFile();

      // creates a FileWriter Object
      FileWriter writer = new FileWriter(file);

      // Writes the content to the file
      writer.write("This\n is\n an\n example\n");
      writer.flush();
      writer.close();

      // Creates a FileReader Object
      FileReader fr = new FileReader(file);
      char [] a = new char[50];
      fr.read(a);   // reads the content to the array

      for(char c : a)
         System.out.print(c);   // prints the characters one by one
      fr.close();
   }
```

Java BufferedReader Class

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 inherits Reader class.

It improves the I/O performance because :

1. This buffer allows java to do I/O operations on more than a byte at a time.
2. Because the buffer is available, skipping, marking, and resetting of the stream becomes possible.
3. It also support moving backward in the stream of the available buffer.

Java BufferedReader class declaration

The declaration for Java.io.BufferedReader class:

```
public class BufferedReader extends Reader
```

Java BufferedReader class constructors

Constructor	Description
BufferedReader(Reader rd)	It is used to create a buffered character input stream that uses the default size for an input buffer.
BufferedReader(Reader rd, int size)	It is used to create a buffered character input stream that uses the specified size for an input buffer.

Java BufferedReader class methods

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.
long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

Java BufferedReader Example

In this example, we are reading the data from the text file **t.txt** using Java BufferedReader class.

```
import java.io.*;
public class IOBuffRe1
{
 public static void main(String args[])throws Exception
 {     FileReader fr=new FileReader("t.txt");
       BufferedReader br=new BufferedReader(fr);
      int i;
        while((i=br.read())!=-1)
          {   System.out.print((char)i);
          }
        br.close();
        fr.close();
 } }
```

Reading data from console by InputStreamReader and BufferedReader

In this example, we are connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

```
import java.io.*;
public class IOBuffRe2
{
public static void main(String args[])throws Exception
  {  InputStreamReader r=new InputStreamReader(System.in);
     BufferedReader br=new BufferedReader(r);
     System.out.println("Enter your name");
         String name=br.readLine();
       System.out.println("Welcome "+name);
}   }
```

Another example of reading data from console until user writes stop

In this example, we are reading and printing the data until the user prints stop.

```
import java.io.*;
public class IOBuffRe3{
public static void main(String args[])throws Exception
{
   InputStreamReader r=new InputStreamReader(System.in);
  BufferedReader br=new BufferedReader(r);
 String name="";
  while(!name.equals("stop"))
    {         System.out.println("Enter data: ");
              name=br.readLine();
              System.out.println("data is: "+name);
    }
    br.close();
    r.close();
    }    }
```

Java BufferedWriter Class

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.

Class declaration

The declaration for Java.io.BufferedWriter class:

```
public class BufferedWriter extends Writer
```

Class constructors

Constructor	Description
BufferedWriter(Writer wrt)	It is used to create a buffered character output stream that uses the default size for an output buffer.
BufferedWriter(Writer wrt, int size)	It is used to create a buffered character output stream that uses the specified size for an output buffer.

Class methods

Method	Description
void newLine()	It is used to add a new line by writing a line separator.
void write(int c)	It is used to write a single character.
void write(char[] cbuf, int off, int len)	It is used to write a portion of an array of characters.
void write(String s, int off, int len)	It is used to write a portion of a string.
void flush()	It is used to flushes the input stream.
void close()	It is used to closes the input stream

Example of Java BufferedWriter

Let's see the simple example of writing the data to a text file **testout.txt** using Java BufferedWriter.

```
import java.io.*;
public class IOBuffWri01
{    public static void main(String[] args) throws Exception {
          FileWriter fw = new FileWriter("t.txt");
          BufferedWriter buffer = new BufferedWriter(fw);
  String s="this is a string";
  buffer.write('a');       buffer.write(65);
  buffer.write(s);          buffer.write("I write a line into a file.");
  buffer.close();
  System.out.println("Successfully write");
} }
```

Java - RandomAccessFile

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Constructor

Constructor	Description
RandomAccessFile(File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Method

Modifier and Type	Method	Method
void	close()	It closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel()	It returns the unique FileChannel object associated with this file.
int	readInt()	It reads a signed 32-bit integer from this file.
String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.
int	read()	It reads a byte of data from this file.
long	length()	It returns the length of this file.
Void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileExample {
   static final String FILEPATH ="myFile.TXT";
   public static void main(String[] args)
  {
      try {
         System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
         writeToFile(FILEPATH, "I love my country and my people", 31);
          } catch (IOException e) {
            e.printStackTrace();
            }
   }

   private static byte[] readFromFile(String filePath, int position, int size)
         throws IOException
   {
      RandomAccessFile file = new RandomAccessFile(filePath, "r");
      file.seek(position);
      byte[] bytes = new byte[size];
      file.read(bytes);
      file.close();
      return bytes;
   }

   private static void writeToFile(String filePath, String data, int position)
         throws IOException
  {
      RandomAccessFile file = new RandomAccessFile(filePath, "rw");
      file.seek(position);
      file.write(data.getBytes());
      file.close();
   }
}
```

Java ObjectInputStream class declaration

Java ObjectInputStream class

The objectinputstream class is mainly used to deserialize the primitive data and objects which are written by using ObjectOutputStream. ObjectInputStream can also be used to pass the objects between hosts by using a SocketStream. The objects which implement Serializable or Externalizable interface can only be read using ObjectInputStream. Consider the example in the last of this topic to read an object from the file which was previously written by using ObjectOutputStream.

Java ObjectInputStream class declaration :

public class ObjectInputStream **extends** InputStream **implements** ObjectInput, ObjectStreamConstants

1	public int available() throws IOException	The available() method returns the number of bytes that can be read without blocking.
2	public void close() throws IOException	The close() method closes the input stream. It must be called to release any resource associated with the stream.
3	public void defaultReadObject() throws IOException, ClassNotFoundException	The default ReadObject() method reads the non-static and non-transient fields of the current class from this stream. i.e one cannot use defaultReadObject() to read the static fields. This may only be called from the readObject method of the class being desterilized. You cannot use defaultReadObject() directly.
4	Public int read() throws IOException	The read() method of java.io.ObjectInputStream reads a byte of data. This method will block if no input is available. i.e., some data should be present to read in inputstream otherwise method will block. The actual number of bytes read or -1 is returned when the end of the stream is reached.
5	public int read(byte[] buf, int off, int len) throws IOException	The read() method read into an array of bytes. There should be some bytes present to be read otherwise the method will block. This method takes 3 parameters, a byte array buff into which the bytes are stored, starting offset off to start reading, Length of bytes len.
6	public boolean readBoolean() throws IOException	This method reads a Boolean from the stream. It returns true if the byte is non-zero otherwise false
7	public byte readByte()	The readByte() method reads an 8-bit byte.

	throws IOException	
8	public char readChar() throws IOException	The readChar() method reads a 16-bit char.
9	Public double readDouble() throws IOException	
10	public float readFloat() throws IOException	The readFloat() method reads a 32-bit float.
11	Public int readInt() throws IOException	The readInt() method reads a 32-bit int.
12	public long readLong() throws IOException	The readLong() method reads a 64-bit float.
13	Public hort readShort() throws IOException	The readShort() method read 16 bit short.
14	public String readLine() throws IOException	The readLine() method reads in a line that has been terminated by a \n, \r, \r\n or EOF.

Java ObjectInputStream class

The objectinputstream class is mainly used to deserialize the primitive data and objects which are written by using ObjectOutputStream. ObjectInputStream can also be used to pass the objects between hosts by using a SocketStream. The objects which implement Serializable or Externalizable interface can only be read using ObjectInputStream. Consider the following line of code to read an object from the file which was previously written by using ObjectOutputStream.

Java ObjectInputStream class declaration

public class ObjectInputStream extends InputStream

implements ObjectInput, ObjectStreamConstants

List of ObjectInputStream Methods

NO	Method	Description
1	public int available() throws IOException	The available() method returns the number of bytes that can be read without blocking.
2	public void close() throws IOException	The close() method closes the input stream. It must be called to release any resource associated with the stream.
3	public void defaultReadObject() throws IOException, ClassNotFoundException	The default ReadObject() method reads the non-static and non-transient fields of the current class from this stream. i.e one cannot use defaultReadObject() to read the static fields. This may only be called from the readObject method of the class being desterilized. You cannot use defaultReadObject() directly.
4	public int read() throws IOException	The read() method of java.io.ObjectInputStream reads a byte of data. This method will block if no input is available. i.e., some data should be present to read in inputstream otherwise method will block. The actual number of bytes read or -1 is returned when the end of the stream is reached.
5	public int read(byte[] buf, int off, int len) throws IOException	The read() method read into an array of bytes. There should be some bytes present to be read otherwise the method will block. This method takes 3 parameters, a byte array buff into which the bytes are stored, starting offset off to start reading, Length of bytes len.

6	public boolean readBoolean() throws IOException	This method reads a Boolean from the stream. It returns true if the byte is non-zero otherwise false
7	public byte readByte() throws IOException	The readByte() method reads an 8-bit byte.
8	public char readChar() throws IOException	The readChar() method reads a 16-bit char.
9	Public double readDouble() throws IOException	
10	public float readFloat() throws IOException	The readFloat() method reads a 32-bit float.
11	Public int readInt() throws IOException	The readInt() method reads a 32-bit int.
12	public long readLong() throws IOException	The readLong() method reads a 64-bit float.
13	Public short readShort() throws IOException	The readShort() method read 16 bit short.
14	public String readLine() throws IOException	The readLine() method reads in a line that has been terminated by a \n, \r, \r\n or EOF.
15	public void readFully(byte[] buf) throws IOException public void readFully(byte[] buf, int off, int len) throws IOException	The readFully() method of ObjectInputStream class Reads bytes until all the bytes in objectinputstream are read. This method is overloaded as It takes 3 parameters, the buffer into which data is stored, from where to start reading, and how many bytes to read.
16	public ObjectInputStream.GetField readFields() throws IOException, ClassNotFoundException	The readField() method of ObjectInputStream class reads the persistent fields from the stream and makes them available by name.

17	public final Object readObject() throws IOException, ClassNotFoundException	The readObject() method of ObjectInputStream class is used to read an object from objectinputstresm.

Here Serializable is an interface. This is a marker interface.

"A marker interface is an interface that **doesn't have any methods or constants inside it**. It provides **run-time type information about objects**, so the compiler and JVM have **additional information about the object**." A marker interface is also called a tagging interface.

The **Serializable** interface is present in **java.io** package. It is a marker interface. A Marker Interface does not have any methods and fields. Thus classes implementing it do not have to implement any methods. Classes implement it if they want their instances to be Serialized or Deserialized. Serialization is a mechanism of converting the state of an object into a byte stream. Serialization is done using ObjectOutputStream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object. Deserialization is done using ObjectInputStream. Thus it can be used to make an eligible for saving its state into a file.

Write Object into File :

```
import java.io.*;
class Student implements Serializable
{
  String name; String sid;
  Student(String n, String id)
   {   name=n;     sid = id;
   }
}

class ObjWr
{
public static void main(String s[])
  {
   try{
       Student obj = new Student("Sumit","100");
       FileOutputStream fis = new FileOutputStream("tt.txt");
         ObjectOutputStream oos= new ObjectOutputStream(fis);
         oos.writeObject(obj);

     }
   catch(Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

Read Object from file :

```
import java.io.*;
class Student implements Serializable
{
  String name;  String id;
   Student(String n,String i)
    { name=n; id =i;
    }
   void show()
   {    System.out.println(name+":"+id);   }
}

class ObjrT
{
public static void main(String s[])
{
try{
ObjectInputStream ois= new ObjectInputStream(new FileInputStream("tt.txt"));
Student obj=null;
obj=(Student)ois.readObject();
obj.show();
   }
catch(Exception e)
 {
   e.printStackTrace();
   }
}
}
```