Java I/O Tutorial

Java I/O (Input and Output) is used to process the input and produce the output.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 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

Let's see the code to print **output and an error** message to the console.

```
System.out.println("simple message");
System.err.println("error message");
```

Let's see the code to get **input** from console.

```
int i=System.in.read();//returns ASCII code of 1st character
System.out.println((char)i);//will print the character
```

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

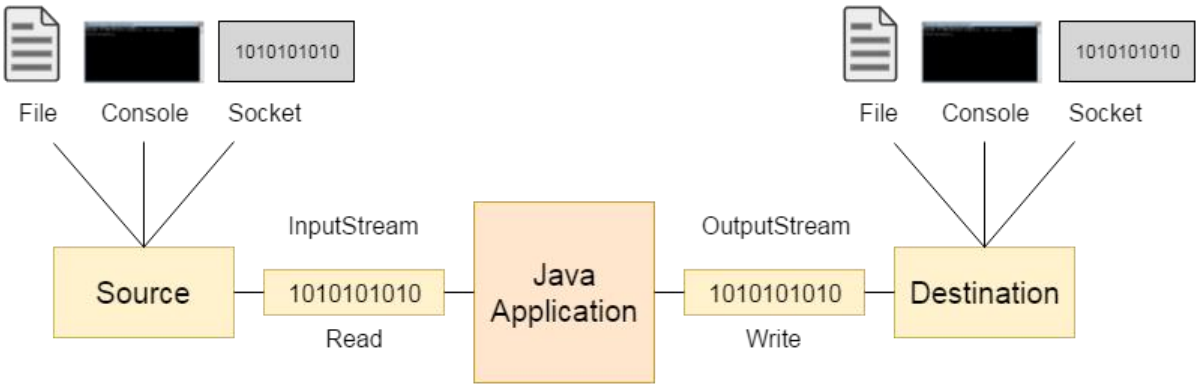
OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream by the figure given below.



OutputStream class

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

Useful methods of OutputStream

Method	Description
1) public 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.

InputStream class

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

Useful methods of InputStream

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.

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.

FileOutputStream class declaration

Let's see the declaration for Java.io.FileOutputStream class:

FileOutputStream class methods

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.

Java FileOutputStream Example 1: write byte

```
        import java.io.FileOutputStream;
        public class FileOutputStreamExample {
           public static void main(String args[]){
                try{
                  FileOutputStream fout=new FileOutputStream("D:\\testout.txt")
;
                  fout.write(65);
                  fout.close();
                  System.out.println("success...");
                 }catch(Exception e){System.out.println(e);}
```

Java FileOutputStream example 2: write string

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
        try{
          FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
          String s="Welcome to javaTpoint.";
          byte b[]=s.getBytes();//converting string into byte array
          fout.write(b);
          fout.close();
          System.out.println("success...");
         }catch(Exception e){System.out.println(e);}
```

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.

Java FileInputStream class methods

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.

Java FileInputStream example 1: read single character

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");

          int i=fin.read();
          System.out.print((char)i);

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

Java FileInputStream example 2: read all characters

```
        try{
                FileInputStream fin=new FileInputStream("D:\\testout.txt"
);
                int i=0;
                while((i=fin.read())!=-1){
                 System.out.print((char)i);
                }
                fin.close();
               }catch(Exception e){System.out.println(e);}
```

Java Writer

It is an abstract class for writing to character streams.

```
        try {
                Writer w = new FileWriter("output.txt");
                String content = "I love my country";
                w.write(content);
                w.close();
                System.out.println("Done");
            } catch (IOException e) {
                e.printStackTrace();
            }
```

Java Reader

Java Reader is an abstract class for reading character streams.

```
try {
        Reader reader = new FileReader("file.txt");
        int data = reader.read();
        while (data != -1) {
            System.out.print((char) data);
            data = reader.read();
        }
        reader.close();
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
```

Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.
Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.

```
try{
        FileWriter fw=new FileWriter("D:\\testout.txt");
        fw.write("Welcome to javaTpoint.");
        fw.close();
       }catch(Exception e){System.out.println(e);}
       System.out.println("Success...");
 }
```

Java FileReader Class

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

It is character-oriented class which is used for file handling in java.

```
FileReader fr=new FileReader("D:\\testout.txt");
    int i;
    while((i=fr.read())!=-1)
    System.out.print((char)i);
    fr.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.

```
FileWriter writer = new FileWriter("D:\\testout.txt");
    BufferedWriter buffer = new BufferedWriter(writer);
    buffer.write("Welcome to javaTpoint.");
    buffer.close();
    System.out.println("Success");
```

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 makes the performance fast. It inherits Reader class.

```
FileReader fr=new FileReader("D:\\testout.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.

```
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 File Class

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

```
try {
        File file = new File("javaFile123.txt");
        if (file.createNewFile()) {
            System.out.println("New File is created!");
        } else {
            System.out.println("File already exists.");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Java Scanner

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.

The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

```
Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        System.out.println("Name is: " + name);
        in.close();
```

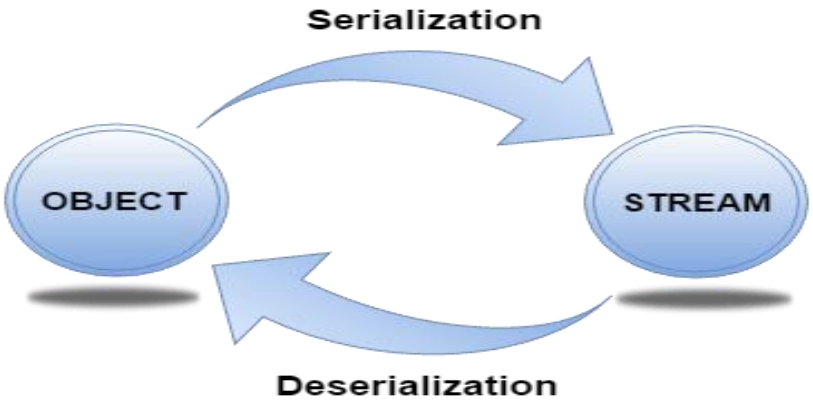
Serialization and Deserialization in Java

Serialization in Java is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object in a platform and deserialize in different platform.

Advantages of Java Serialization

It is mainly used to travel object's state on the network (which is known as marshaling).



java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The Cloneable and Remote are also marker interfaces.

It must be implemented by the class whose object you want to persist.

The String class and all the wrapper classes implement the java.io.Serializable interface by default.

```
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
```

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

```
try{
//Creating the object
Student s1 =new Student(211,"ravi");
//Creating stream and writing the object
FileOutputStream fout=new FileOutputStream("f.txt");
ObjectOutputStream out=new ObjectOutputStream(fout);
out.writeObject(s1);
out.flush();
//closing the stream
out.close();
System.out.println("success");
}catch(Exception e){System.out.println(e);}
}
```

Example of Java Deserialization

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. Let's see an example where we are reading the data from a deserialized object.

```
        try{
         //Creating stream to read the object
         ObjectInputStream in=new ObjectInputStream(new FileInputStream("f
.txt"));
         Student s=(Student)in.readObject();
         //printing the data of the serialized object
         System.out.println(s.id+" "+s.name);
         //closing the stream
         in.close();
         }catch(Exception e){System.out.println(e);}
```

Java Transient Keyword

Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.

Let's take an example, I have declared a class as Student, it has three data members id, name and age. If you serialize the object, all the values will be serialized but I don't want to serialize one value, e.g. age then we can declare the age data member as transient.

```
public class Student implements Serializable{
 int id;
 String name;
 transient int age;//Now it will not be serialized
 public Student(int id, String name,int age) {
  this.id = id;
  this.name = name;
  this.age=age;
 }
```

Searialize your class.

```
        try{
         //Creating stream to read the object
         ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt
"));
         Student s=(Student)in.readObject();
         //printing the data of the serialized object
         System.out.println(s.id+" "+s.name);
         //closing the stream
         in.close();
         }catch(Exception e){System.out.println(e);}
```