IO Package

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, three streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

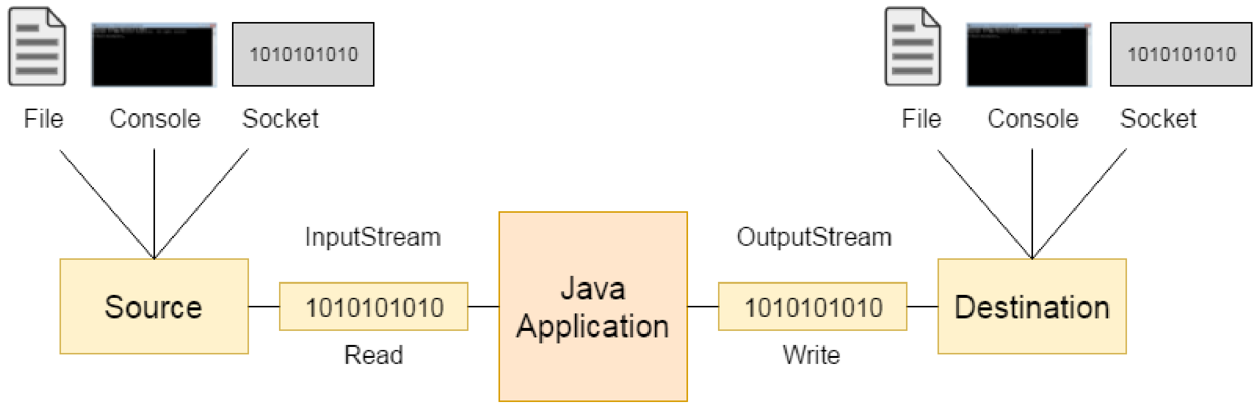
2) System.in: standard input stream

3) System.err: standard error stream

InputStream

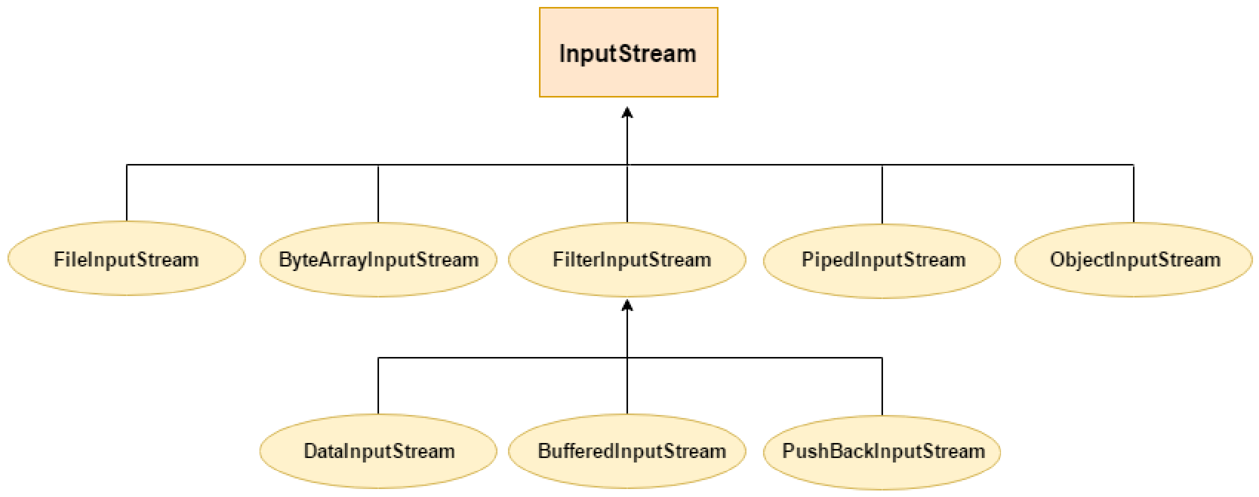
Java applications use an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

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



InputStream class

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

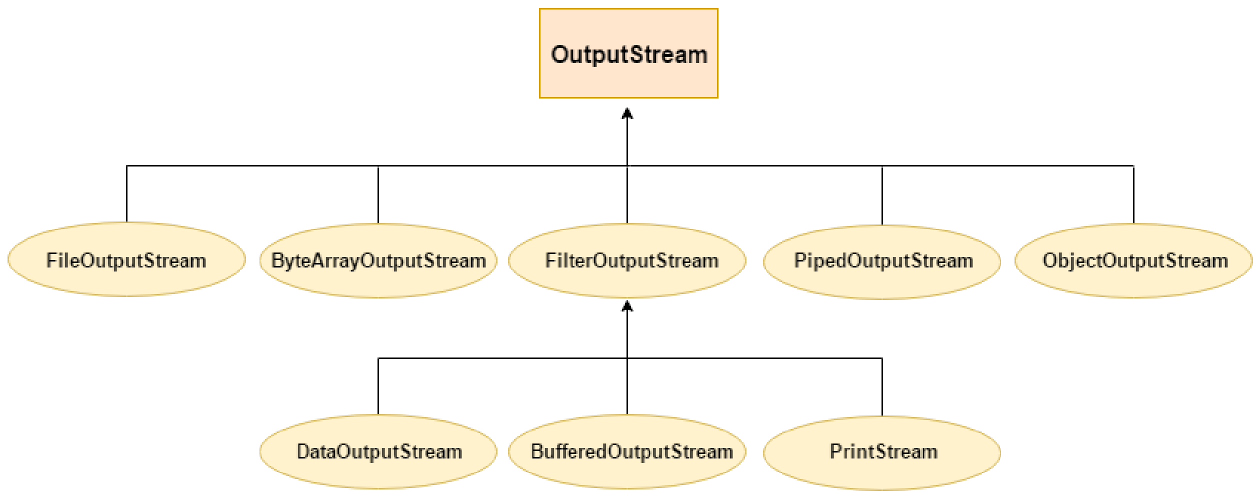


OutputStream

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

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.



FileOutputStream

Java FileOutputStream is an output stream used for writing data to a file.

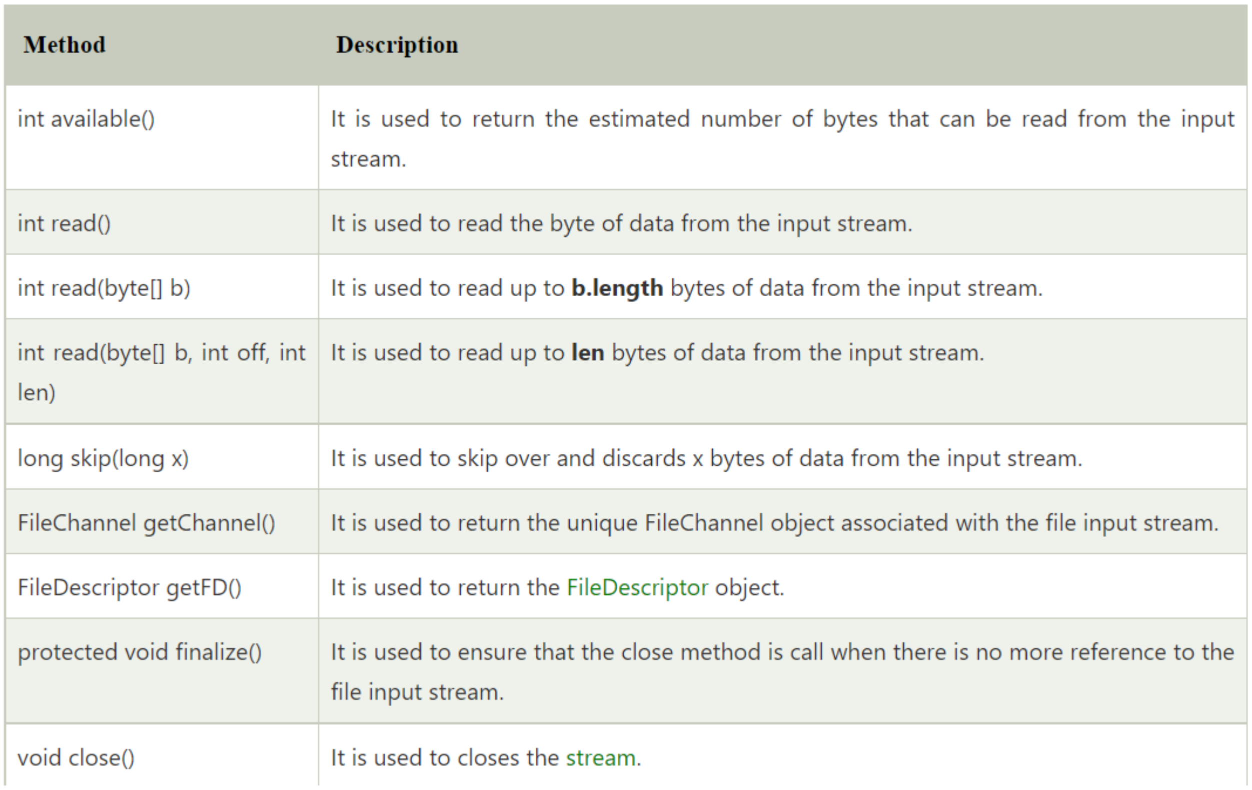
If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

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.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
           try{
             FileOutputStream fout=new FileOutputStream("testout.txt");
             String s="Java Full Stack course";
             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);
           }
      }
}
```

FileInputStream

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.



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 FileInputStreamExample {
     public static void main(String args[]){
          try{
            FileInputStream fin=new FileInputStream("testout.txt");
            int i=0;
            while((i=fin.read())!=-1){
             System.out.print((char)i);
            }
            fin.close();
          }catch(Exception e){System.out.println(e);}
         }
        }
```

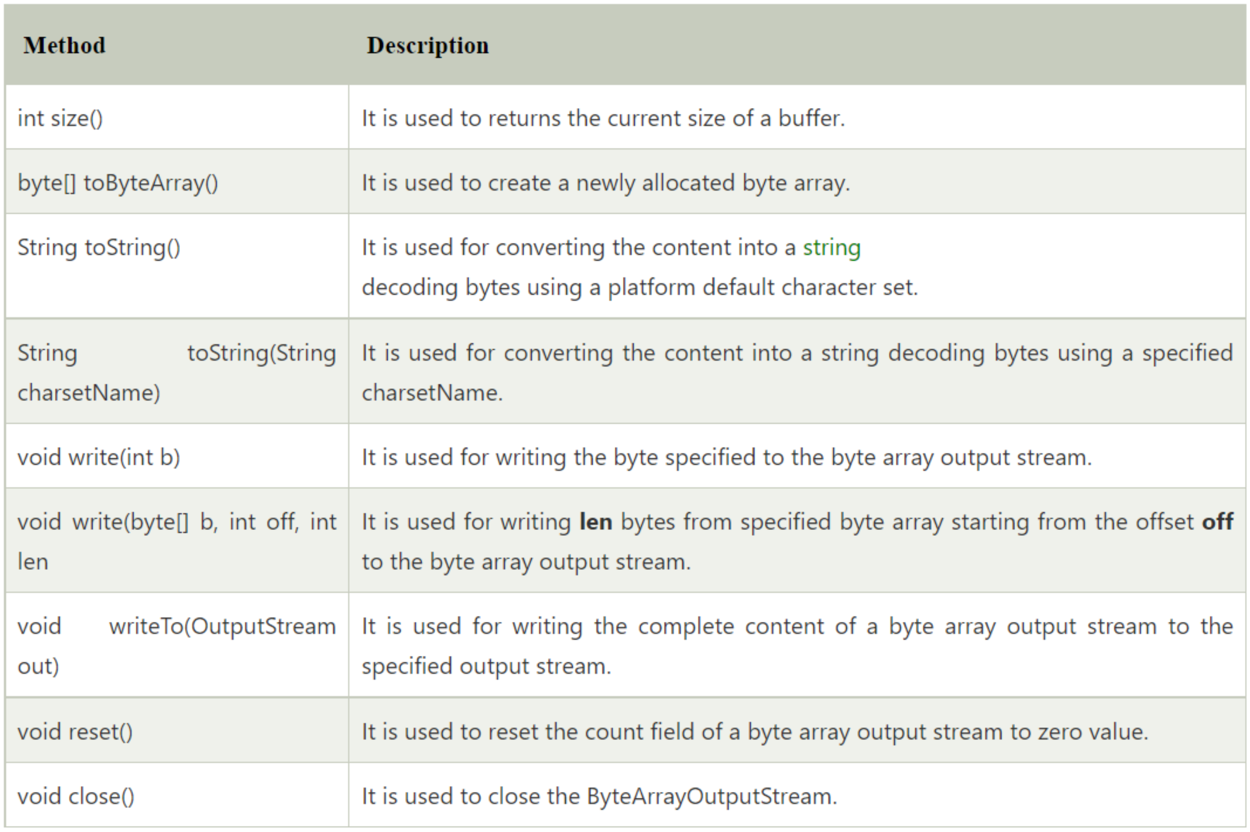
ByteArrayOutputStream

Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte array

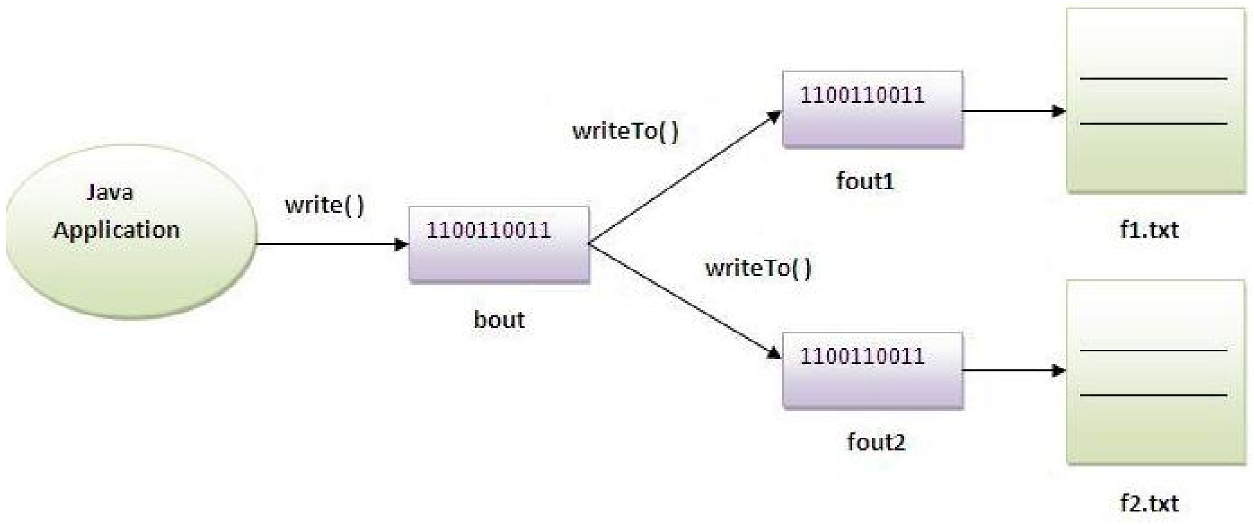
which can be written to multiple streams later.

The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.

The buffer of ByteArrayOutputStream automatically grows according to data.



Method	Description
int size()	It is used to returns the current size of a buffer.
byte[] toByteArray()	It is used to create a newly allocated byte array.
String toString()	It is used for converting the content into a string decoding bytes using a platform default character set.
String toString(String charsetName)	It is used for converting the content into a string decoding bytes using a specified charsetName.
void write(int b)	It is used for writing the byte specified to the byte array output stream.
void write(byte[] b, int off, int len	It is used for writing **len** bytes from specified byte array starting from the offset **off** to the byte array output stream.
void writeTo(OutputStream out)	It is used for writing the complete content of a byte array output stream to the specified output stream.
void reset()	It is used to reset the count field of a byte array output stream to zero value.
void close()	It is used to close the ByteArrayOutputStream.



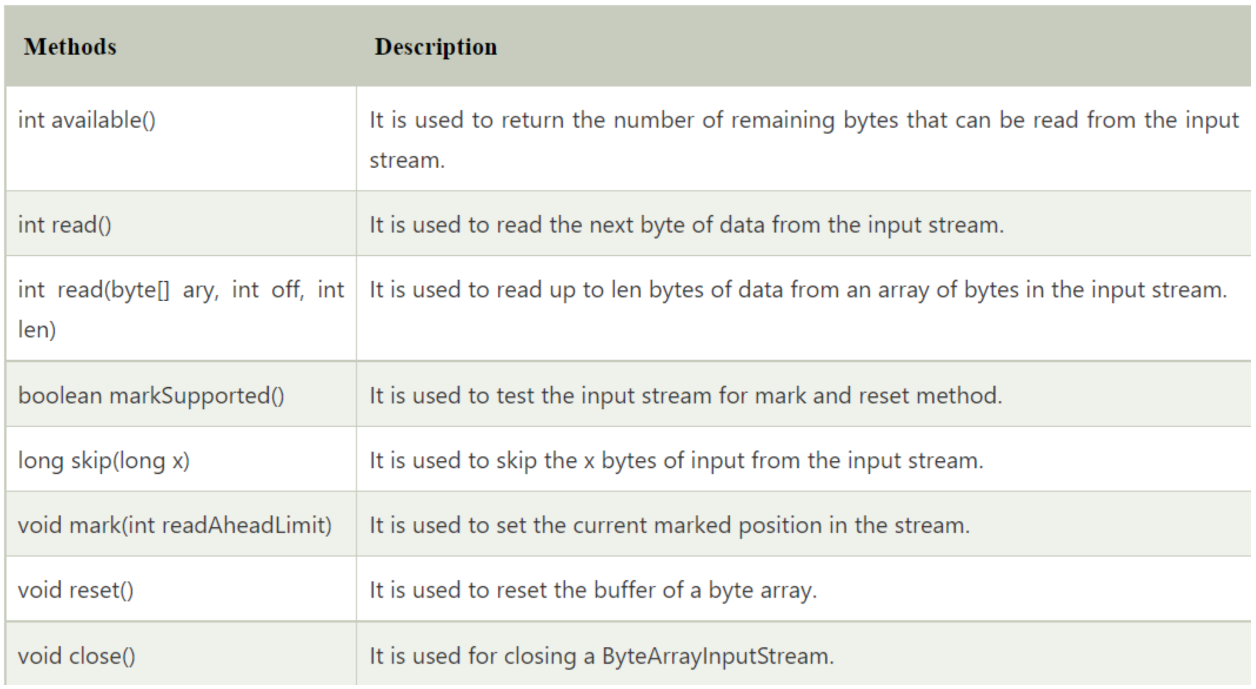
```
import java.io.*;

public class ByteArrayOutputStreamExample {
	public static void main(String args[]) throws Exception {
		FileOutputStream fout1 = new FileOutputStream("f1.txt");
		FileOutputStream fout2 = new FileOutputStream("f2.txt");

		String s="This text will be written in two files.";

		ByteArrayOutputStream bout = new ByteArrayOutputStream();

		byte[] b=s.getBytes();
		bout.write(b);
		bout.writeTo(fout1);
		bout.writeTo(fout2);

		bout.flush();
		bout.close();
		System.out.println("Success...");
	}
}
```

ByteArrayInputStream

The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte array as input stream.

Java ByteArrayInputStream class contains an internal buffer which is used to **read byte array** as stream. In this stream, the data is read from a byte array.

The buffer of ByteArrayInputStream automatically grows according to data.



Methods	Description
int available()	It is used to return the number of remaining bytes that can be read from the input stream.
int read()	It is used to read the next byte of data from the input stream.
int read(byte[] ary, int off, int len)	It is used to read up to len bytes of data from an array of bytes in the input stream.
boolean markSupported()	It is used to test the input stream for mark and reset method.
long skip(long x)	It is used to skip the x bytes of input from the input stream.
void mark(int readAheadLimit)	It is used to set the current marked position in the stream.
void reset()	It is used to reset the buffer of a byte array.
void close()	It is used for closing a ByteArrayInputStream.

```
import java.io.*;

public class ByteArrayInputStreamExample {
        public static void main(String[] args) throws IOException {
                byte[] buf = { 35, 36, 37, 38 };

                ByteArrayInputStream byt = new ByteArrayInputStream(buf);
                int k = 0;
                while ((k = byt.read()) != -1) {

                        char ch = (char) k;
                        System.out.println("ASCII value of Character is:" + k + "; Special
character is: " + ch);
                }
        }
}
```

Serialization

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.

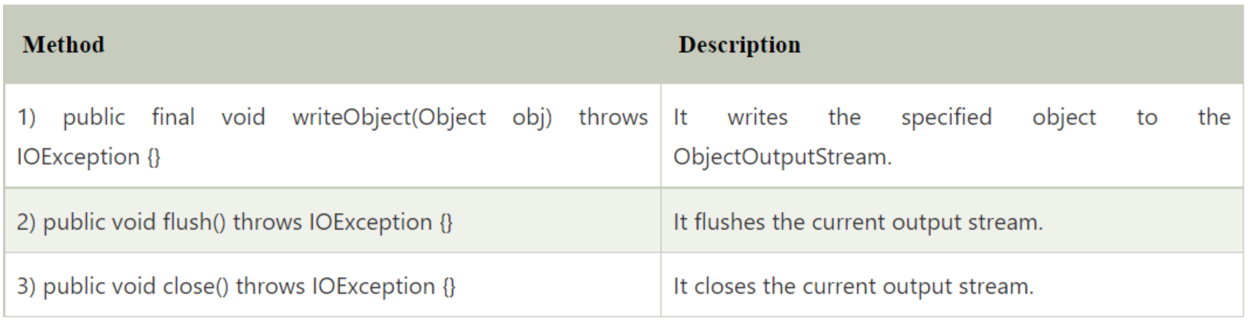
The reverse operation of serialization is called ***deserialization*** where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

For serializing the object, we call the **writeObject()** method of *ObjectOutputStream* class, and for deserialization we call the **readObject()** method of *ObjectInputStream* class.

We must have to implement the ***Serializable*** interface for serializing the object.

ObjectOutputStream class

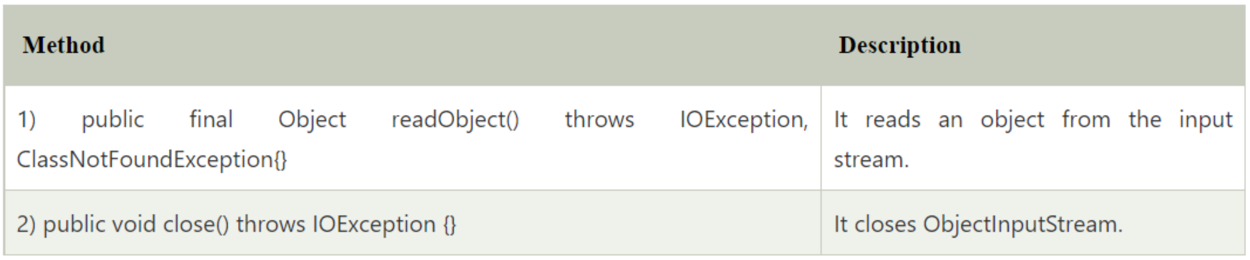
The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.



Method	Description
1) public final void writeObject(Object obj) throws IOException {}	It writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	It flushes the current output stream.
3) public void close() throws IOException {}	It closes the current output stream.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.



Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	It reads an object from the input stream.
2) public void close() throws IOException {}	It closes ObjectInputStream.

```
class Game implements Serializable{
	int score;
	int completedLevel;
}

public class ObjectOutputStreamExample {

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

		Game obj = new Game();
		obj.score = 3400;
		obj.completedLevel = 3;

		File f= new File("GameLogDetails.txt");
		FileOutputStream fo = new FileOutputStream(f);
		ObjectOutputStream oos = new ObjectOutputStream(fo);
		oos.writeObject(obj);


		FileInputStream fi = new FileInputStream(f);
		ObjectInputStream ois = new ObjectInputStream(fi);
		Game obj1 = (Game) ois.readObject();
		System.out.println(obj1.completedLevel+"\t "+obj1.score);
	}
}
```

transient keyword

During the serialization, when we do not want an object to be serialized we can use a transient keyword.

Why use the transient keyword?

The transient keyword can be used with the data members of a class in order to avoid their serialization. For example, if a program accepts a user's login details and password. But we don't want to store the original password in the file. Here, we can use the transient keyword and when the JVM reads the transient keyword it ignores the original value of the object and instead stores the default value of the object.

Syntax

```
	private transient <member variable>;
```

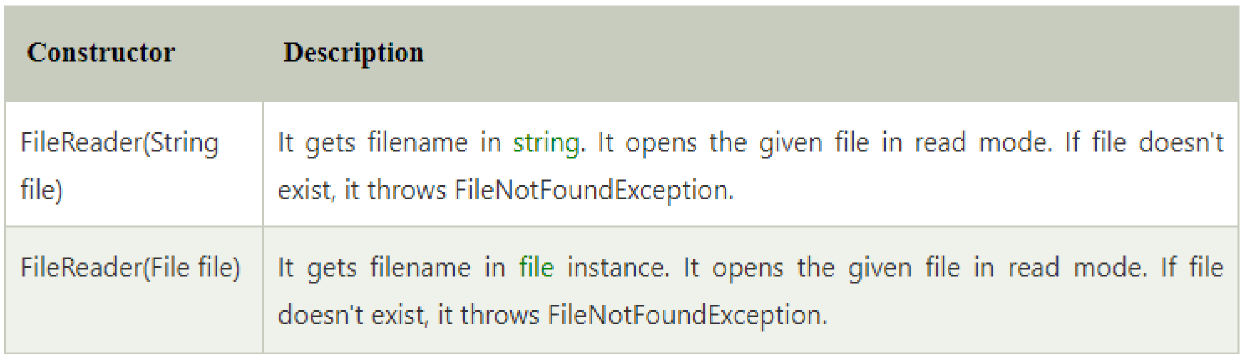
}

Character Streams

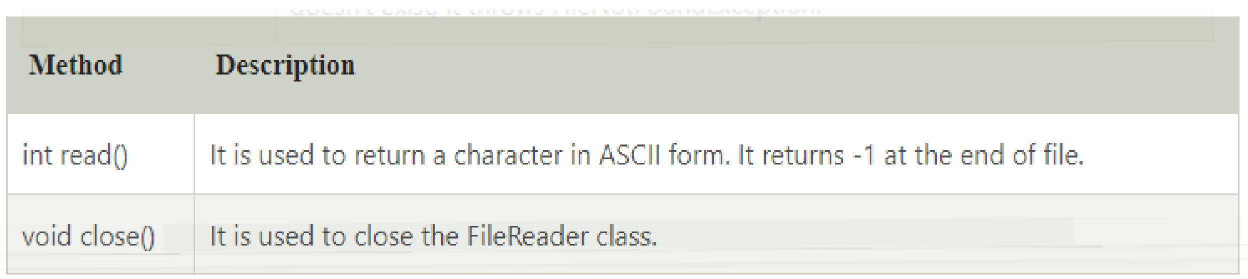
FileReader

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.



Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

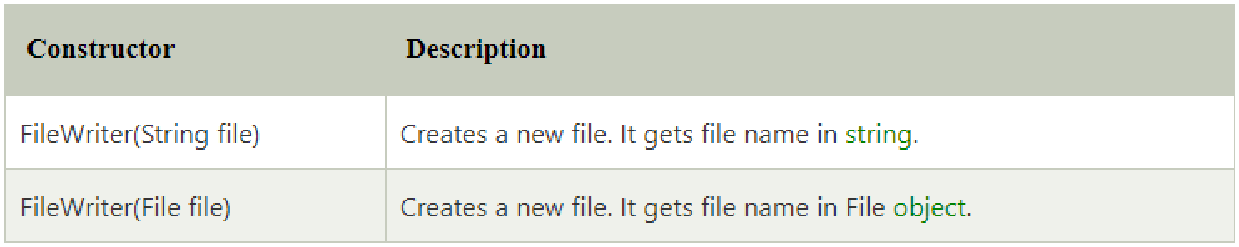


Method	Description
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

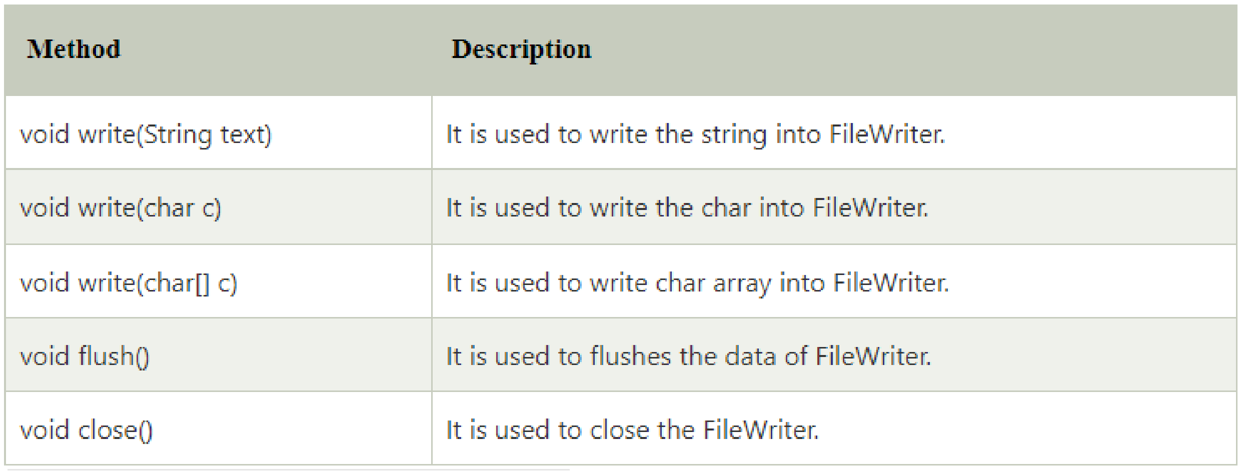
```
public class FileReaderExample {

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

        FileReader fr=new FileReader("test.txt");

        int i;

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

        System.out.print((char)i);

        fr.close();

   }

}
```

FileWriter

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.



Constructor	Description
FileWriter(String file)	Creates a new file. It gets file name in string.
FileWriter(File file)	Creates a new file. It gets file name in File object.



Method	Description
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

```
import java.io.FileWriter;

public class FileWriterExample {

    public static void main(String args[]){

         try{

           FileWriter fw=new FileWriter("testt.txt");

           fw.write("Welcome All.");

           fw.close();

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

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

     }
}
```