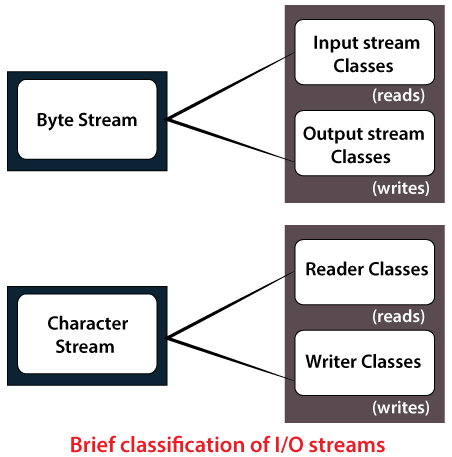
**File Handling in Java**

In Java, a **File** is an abstract data type. A named location used to store related information is known as a **File**. There are several **File Operations** like **creating a new File, getting information about File, writing into a File, reading from a File** and **deleting a File**.

Before understanding the File operations, it is required that we should have knowledge of **Stream** and **File methods**. If you have knowledge about both of them, you can skip it.

**Stream**

A series of data is referred to as **a stream**. In [Java](https://www.javatpoint.com/java-tutorial), **Stream** is classified into two types, i.e., **Byte Stream** and **Character Stream**.



**Byte Stream**

**Byte Stream** is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream.**

import java.io.\*;

public class CopyFile {

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

**FileInputStream in = null;**

**FileOutputStream out = null;**

try {

**in = new FileInputStream("input.txt");**

out = new FileOutputStream("output.txt");

int c;

**while (c!= -1) {**

**in.read(); //input.txt**

**out.write(c); //output.txt**

}

}**finally {**

**if (in != null) {**

**in.close();**

**}**

**if (out != null) {**

**out.close();**

**}**

**}**

**}**

**}**

**Character Stream**

**Character Stream** is mainly involved with character data. A file handling process with a character stream is a process in which an input is provided and executed with the character data. Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**.

import java.io.\*;

public class CopyFile {

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

FileReader in = null;

FileWriter out = null;

try {

in = new FileReader("input.txt");

out = new FileWriter("output.txt");

int c;

**while (c != -1) {**

**in.read();**

**out.write(c);**

**}**

**}finally {**

**if (in != null) {**

**in.close();**

**}**

**if (out != null) {**

**out.close();**

**}**

**}**

**}**

**}**





**Example 1:**

import java.io.\*;

import java.util.\*;

public class DataOutputStreamExample {

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

OutputStream os = new FileOutputStream("**D:\\testout.txt**");

DataOutputStream dos = new DataOutputStream(os);

int itemNo; String itemName, ch; double unitPrice;

Scanner in = new Scanner(System.in);

do

{ System.out.println("Enter the item number, name and unit price:");

itemNo = in.nextInt();

itemName = in.next();

unitPrice = in.nextDouble();

**dos.writeInt(itemNo);**

**dos.writeUTF(itemName); // Unicode Translation Format**

**dos.writeDouble(unitPrice);**

System.out.println("Continue(y/n)?");

ch = in.next();

}while(ch.equals("yes"));

dos.close();

}

}

**Example 2:**

import java.io.\*;

public class DataInputStreamExample {

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

**InputStream is = new FileInputStream("D:\\testout.txt");**

**DataInputStream dis = new DataInputStream(is);**

int itemNo; String itemName, ch; double unitPrice;

while(dis.available() > 0 ){

itemNo = dis.readInt();

itemName = dis.readUTF();

unitPrice = dis.readDouble();

System.out.println(itemNo + "\t" + itemName + "\t" + unitPrice);

}

dis.close();

}

}

**Example 3:**

import java.io.File;



import java.io.FileNotFoundException;

import java.util.Scanner;

class ReadFromFile {

public static void main(String[] args) {

try {

// Create f1 object of the file to read data

File f1 = new File("D:FileOperationExample.txt");

Scanner dataReader = new Scanner(f1);

while (dataReader.hasNextLine()) {

String fileData = dataReader.nextLine();

System.out.println(fileData);

}

dataReader.close();

} catch (FileNotFoundException exception) {

System.out.println("Unexcpected error occurred!");

exception.printStackTrace();

}

}

}

**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 in a platform and deserialize in different platform.

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

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

**Advantages of Java Serialization**

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





**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.

Let's see the example given below:

Filename: Person.java

import java.io.Serializable;

public class Person implements Serializable {

private static final long serialVersionUID = 1L;

private String name;

private int age;

private String gender;

Person() {

};

Person(String name, int age, String gender) {

this.name = name;

this.age = age;

this.gender = gender;

}

@Override

public String toString() {

return "Name:" + name + "\nAge: " + age + "\nGender: " + gender;

}

}

Filename: WriterReader.java

import java.io.\*;

public class WriterReader {

public static void main(String[] args) {

Person p1 = new Person("John", 30, "Male");

Person p2 = new Person("Rachel", 25, "Female");

try {

FileOutputStream f = new FileOutputStream(new File("myObjects.txt"));

ObjectOutputStream o = new ObjectOutputStream(f);

*// Write objects to file*

o.writeObject(p1);

o.writeObject(p2);

o.close();

f.close();

FileInputStream fi = new FileInputStream(new File("myObjects.txt"));

ObjectInputStream oi = new ObjectInputStream(fi);

*// Read objects*

Person pr1 = (Person) oi.readObject();

Person pr2 = (Person) oi.readObject();

System.out.println(“ ” +pr1);

System.out.println(pr2.toString());

oi.close();

fi.close();

} catch (FileNotFoundException e) {

System.out.println("File not found");

} catch (IOException e) {

System.out.println("Error initializing stream");

} catch (ClassNotFoundException e) {

*// TODO Auto-generated catch block*

e.printStackTrace();

}

}

}

Java Serialization with Inheritance (IS-A Relationship)

If a class implements serializable then all its sub classes will also be serializable. Let's see the example given below:

import java.io.Serializable;

class Person implements Serializable{

int id;

String name;

Person(int id, String name) {

this.id = id;

this.name = name;

}

}

class Student extends Person{

String course;

int fee;

public Student(int id, String name, String course, int fee) {

super(id,name);

this.course=course;

this.fee=fee;

}

}