**Java I/O:**

It is used to process the input and produce the output. Java uses the concept of a stream to make I/O operation fast. We can perform file handling in Java by Java I/O API.

Stream – stream is a sequence of data. In Java stream is composed of bytes

Three streams created for us automatically

Sytem.out – standard output stream

System.in – standard input stream

System.err – standard error stream

Example of all above three

**package** com.lokesh;

**import** java.util.Scanner;

**public** **class** HelloWorld2 {

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

System.***err***.println("hello");

System.***out***.println("hello");

Scanner sc =**new** Scanner(System.***in***);

**int** a=sc.nextInt();

System.***out***.println(a);

}

}

Output stream – uses an output to write data to destination, it may be file, array, peripheral device or socket

Input stream – uses an input to read data from source, it may be a file, an array, peripheral device or socket

Below is an example on how to create a text file

**package** com.lokesh;

**import** java.io.File;

**import** java.util.Scanner;

**public** **class** HelloWorld2 {

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

File f1 =**new** File("F:\\testfile.txt");

**boolean** f2 =f1.createNewFile();

**if**(f2==**true**) {

System.***out***.println("File has been created");

}**else** {

System.***out***.println("file already present");

}

}

}

If I run the same above again it doesn’t create file since file already exists and same will be printed in console “file already present”

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

Example:

**package** com.lokesh;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.util.Scanner;

**public** **class** HelloWorld2 {

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

FileOutputStream f1 =**new** FileOutputStream("F:\\testfile.txt");

String s1 ="I am ramu";

//converting string to byte array

**byte** b1[]=s1.getBytes();

f1.write(b1);

f1.close();

}

}

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

Testfile has data “I am ramu”

Example: Below example only gets first letter from above sentence

**package** com.lokesh;

**import** java.io.File;

**import** java.io.FileInputStream;

**public** **class** HelloWorld2 {

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

FileInputStream f1 =**new** FileInputStream("F:\\testfile.txt");

**int** i=f1.read();

**char** a=(**char**)i;

System.***out***.println(a);

f1.close();

}

}

Example: Below example gets total sentence

**package** com.lokesh;

**import** java.io.File;

**import** java.io.FileInputStream;

**public** **class** HelloWorld2 {

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

FileInputStream f1 =**new** FileInputStream("F:\\testfile.txt");

**int** i=0;

**while**((i=f1.read())!=-1) {

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

}

f1.close();

}

}

BufferedOutputStream - is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into stream. So it makes performance fast.

flush() method flushes the buffered output stream. Flushes the data from one stream to another.

Example:

**package** com.lokesh;

**import** java.io.BufferedOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** HelloWorld2 {

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

FileOutputStream f1 =**new** FileOutputStream("F:\\testfile.txt");

BufferedOutputStream b1=**new** BufferedOutputStream(f1);

String s1="I am ramu";

**byte** b[]=s1.getBytes();

b1.write(b);

b1.flush();

b1.close();

f1.close();

System.***out***.println("sucess");

}

}

BufferedInputStream - is used to read information from stream. It internally uses buffer mechanism to make performance fast.

Example:

**package** com.lokesh;

**import** java.io.BufferedInputStream;

**import** java.io.BufferedOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** HelloWorld2 {

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

FileInputStream f1 =**new** FileInputStream("F:\\testfile.txt");

BufferedInputStream b1=**new** BufferedInputStream(f1);

**int** i;

**while**((i=b1.read())!=-1) {

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

}

b1.close();

f1.close();

}

}

FilterOutputStream - it has different subclasses BufferedOutputStream and DataOutputStream to provide additional functionality. So these two are less used individually

Example:

**package** com.lokesh;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.FilterOutputStream;

**public** **class** HelloWorld2 {

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

FileOutputStream f1 =**new** FileOutputStream("F:\\testfile.txt");

FilterOutputStream fil = **new** FilterOutputStream(f1);

String s1="I am ramu";

**byte** b[]=s1.getBytes();

fil.write(b);

fil.flush();

fil.close();

f1.close();

}

}

FilterInputStream - it has different sub classes as BufferedInputStream and DataOutputStream to provide additional functionality. So these two are less used individually

Example:

**package** com.lokesh;

**import** java.io.BufferedInputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FilterInputStream;

**public** **class** HelloWorld2 {

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

FileInputStream f1 =**new** FileInputStream("F:\\testfile.txt");

FilterInputStream fil = **new** BufferedInputStream(f1);

**int** i=0;

**while**((i=fil.read())!=-1) {

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

}

fil.close();

f1.close();

}

}

Writer - for writing to character streams. Most subclasses will override some f methods defined here to provide higher efficiency, functionality or both

Example:

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

Writer f1 =**new** FileWriter("F:\\testfile.txt");

String s1="I am ramu";

f1.write(s1);

f1.flush();

f1.close();

System.***out***.println("success");

}

}

Reader - for reading character streams. Some of implementation classes are Bufferedreader, CharArrayReader, filterReader, InputStreamReader, PipedReader, StringReader

Example:

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

Reader f1 =**new** FileReader("F:\\testfile.txt");

**int** r1=f1.read();

**while**(r1!=-1) {

System.***out***.print((**char**)r1);

r1=f1.read();

}

f1.close();

}

}

FileWriter - used to write character-oriented data to file. Unlike FileOutputStream you don’t need to convert String into byte array because it provides method to write string directly.

Example:

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

FileWriter f1 =**new** FileWriter("F:\\testfile.txt");

f1.write("I am ramu");

f1.close();

System.***out***.println("success");

}

}

FileReader - used to read data from file. It returns data in byte format like FileInputStream

Example:

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

FileReader f1 =**new** FileReader("F:\\testfile.txt");

**int** i;

**while**((i=f1.read())!=-1) {

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

}

f1.close();

}

}

BufferedWriter – used to provide buffering for Writer instances. It makes performance fast.It inherits Writer class

Example:

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

FileWriter f1 =**new** FileWriter("F:\\testfile.txt");

BufferedWriter b1 =**new** BufferedWriter(f1);

b1.write("I am ramu");

b1.flush();

b1.close();

System.***out***.println("success");

}

}

BufferedReader – used to read text from a character based input stream. It can be used to read data line by line by readLine() method. Performance fast

Example: Below is an example reading the text data in normal way

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

FileReader f1 =**new** FileReader("F:\\testfile.txt");

BufferedReader b1 =**new** BufferedReader(f1);

**int** i;

**while**((i=b1.read())!=-1) {

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

}

}

}

Example: Below is an example reading the entire line using readLine() method

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

FileReader f1 =**new** FileReader("F:\\testfile.txt");

BufferedReader b1 =**new** BufferedReader(f1);

String s1 =b1.readLine();

System.***out***.println(s1);

}

}

PrintWriter – is the implementation of Writer class.

Example:

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

//data to write on console using PrintWriter

PrintWriter p1 =**new** PrintWriter(System.***out***);

p1.write("this should come on console");

p1.flush();

p1.close();

//data to write in file using PrintWrite

PrintWriter p2 =**new** PrintWriter("F:\\testfile.txt");

p2.write("I am ramu");

p1.flush();

p2.close();

}

}

Serialization – is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, JPA…etc

The reverse operation of serialization is called deserialization where byte-stream is converted into a object.

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

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

Example: Below is an example of Serialization

**package** com.lokesh;

**import** java.io.\*;

**public** **class** Employee **implements** Serializable {

**int** id;

String name;

Employee(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

}

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

Employee e1=**new** Employee(1, "ramu");

FileOutputStream f1 =**new** FileOutputStream("F:\\testfile.txt");

ObjectOutputStream ob1=**new** ObjectOutputStream(f1);

ob1.writeObject(e1);

ob1.flush();

ob1.close();

System.***out***.println("success");

}

}

De-serialization – is the process of reconstructing the object from serialized state. It is reverse operation of serialization

Example: Below is an example of De-seralization

**package** com.lokesh;

**import** java.io.\*;

**public** **class** Employee **implements** Serializable {

**int** id;

String name;

Employee(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

}

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

ObjectInputStream ob1 =**new** ObjectInputStream(**new** FileInputStream("F:\\testfile.txt"));

Employee e1 =(Employee) ob1.readObject();

System.***out***.println(e1.id+" "+e1.name);

ob1.close();

}

}

Example: Below is an example of serialization and de-serialization with inheritance (IS-A relationship)

**package** com.lokesh;

**import** java.io.\*;

**public** **class** Address **implements** Serializable {

String city;

String state;

String country;

**public** Address(String city, String state, String country) {

**this**.city = city;

**this**.state = state;

**this**.country = country;

}

}

**package** com.lokesh;

**import** java.io.\*;

**public** **class** Employee **extends** Address {

**int** id;

String name;

**public** Employee(**int** id, String name, String city, String state, String country) {

**super**(city,state,country);

**this**.id = id;

**this**.name = name;

}

}

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

Employee e1= **new** Employee(1, "ramu", "pune", "mh", "India");

FileOutputStream f1 =**new** FileOutputStream("F:\\testfile.txt");

ObjectOutputStream ob1 =**new** ObjectOutputStream(f1);

ob1.writeObject(e1);

ob1.flush();

ob1.close();

System.***out***.println("success");

//de-serialization

ObjectInputStream ob2 =**new** ObjectInputStream(**new** FileInputStream("F:\\\\testfile.txt"));

Employee e2=(Employee)ob2.readObject();

System.***out***.println(e2.id+" "+e2.name+" "+e2.city+" "+e2.state+" "+e2.country);

ob2.close();

}

}

Example: Below is an example of serialization and De-serialization using aggregation (HAS-A relationship)

**package** com.lokesh;

**import** java.io.\*;

**public** **class** Address **implements** Serializable {

String city;

String state;

String country;

**public** Address(String city, String state, String country) {

**this**.city = city;

**this**.state = state;

**this**.country = country;

}

}

**package** com.lokesh;

**import** java.io.\*;

**public** **class** Employee {

**int** id;

String name;

Address addr;

**public** Employee(**int** id, String name,Address addr) {

**this**.id = id;

**this**.name = name;

**this**.addr = addr;

}

}

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

Address ad1 =**new** Address("pune", "mh", "india");

Employee e1= **new** Employee(1, "ramu",ad1);

FileOutputStream f1 =**new** FileOutputStream("F:\\testfile.txt");

ObjectOutputStream ob1 =**new** ObjectOutputStream(f1);

ob1.writeObject(e1);

ob1.flush();

ob1.close();

System.***out***.println("success");

//de-serialization

ObjectInputStream ob2 =**new** ObjectInputStream(**new** FileInputStream("F:\\\\testfile.txt"));

Address ad2 =(Address)ob2.readObject();

Employee e2=(Employee)ob2.readObject();

System.***out***.println(e2.id+" "+e2.name+" "+ad2.city+" "+ad2.state+" "+ad2.country);

ob2.close();

}

}

Transient keyword – if we don’t want an object to be serialized we can use transient keyword. The transient keyword can be used with data members of a class to avoid their serialization

For example if a program accepts user’s login details and password but we don’t want to store original password in the file then here we can use transient keyword. It ignores original value of object and instead stores default value of object

Example: In below example since we gave age field as transient keyword, this gets avoided (doesn’t get stored) during serialization and during de-serialization it produces default value (i.e. 0)

**package** com.lokesh;

**import** java.io.\*;

**public** **class** Employee **implements** Serializable {

**int** id;

String name;

**transient** **int** age;

**public** Employee(**int** id, String name,**int** age) {

**this**.id = id;

**this**.name = name;

**this**.age = age;

}

}

**package** com.lokesh;

**import** java.io.\*;

**public** **class** HelloWorld2 {

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

Employee e1= **new** Employee(1, "ramu",25);

FileOutputStream f1 =**new** FileOutputStream("F:\\testfile.txt");

ObjectOutputStream ob1 =**new** ObjectOutputStream(f1);

ob1.writeObject(e1);

ob1.flush();

ob1.close();

System.***out***.println("success");

//de-serialization

ObjectInputStream ob2 =**new** ObjectInputStream(**new** FileInputStream("F:\\\\testfile.txt"));;

Employee e2=(Employee)ob2.readObject();

System.***out***.println(e2.id+" "+e2.name+" "+e2.age);

ob2.close();

}

}