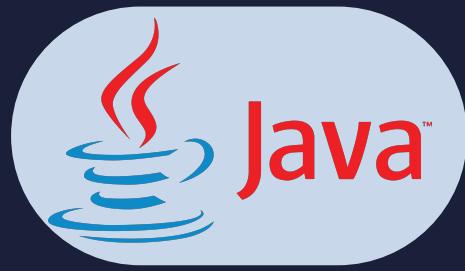


Lesson:



IO Operation(File Handling)



List of Concepts Involved:

- Input Stream
- Output Stream
- File Operation in Java
- Serialization
- Deserialization

File Handling in 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.

Input Stream:

- The InputStream class of the java.io package is an abstract superclass that represents an input stream of bytes.
- InputStream is an abstract class, it is not useful by itself. However, its subclasses can be used to read data.

Output Stream:

- The OutputStream class of the java.io package is an abstract superclass that represents an output stream of bytes.
- OutputStream is an abstract class, it is not useful by itself. However, its subclasses can be used to write data.

File:

```
File f=new File("abc.txt");
```

- This line 1st checks whether the abc.txt file is already available (or) not if it is already available then "f" simply refers to that file.
- If it is not already available then it won't create any physical file just creates a java File object representing the name of the file.

Example:

```
import java.io.*;
class FileDemo{
    public static void main(String[] args)throws IOException{

        File f=new File("cricket.txt");
        System.out.println(f.exists());//false

        f.createNewFile();
        System.out.println(f.exists());//true
    }
}
```

1st run

false
true

2nd run

true
true

=> A java File object can represent a directory also.

Example:

```
import java.io.File;
import java.io.IOException;

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

        File f=new File("cricket123");
        System.out.println(f.exists()); //false
        f.mkdir(); //Creates a new directory
        System.out.println(f.exists()); //true
    }
}
```

1st run

false

true

2nd run

true

true

Note: In UNIX everything is a file, java "file IO" is based on UNIX operating system. Hence in java also we can represent both files and directories by File object only.

File class constructors

1. File f=new File(String name);

=> Creates a java File object that represents the name of the file or directory in the current working directory.
eg#1. File f=new File("abc.txt");

2. File f=new File(String subdirname, String name);

=> Creates a File object that represents the name of the file or directory present in the specified subdirectory.

```
eg#1. File f1=new File("abc");
f1.mkdir();
File f2=new File("abc","demo.txt");
```

3. File f=new File(File subdir, String name);

```
eg#1.File f1=new File("abc");
f1.mkdir();
File f2=new File(f1,"demo.txt");
```

Important methods of file class:

1. boolean exists();

Returns true if the physical file or directory is available.

2. boolean createNewFile();

This method 1st checks whether the physical file is already available or not if it is already available then this method simply returns false without creating any physical file. If this file is not already available then it will create a new file and returns true

3. boolean mkdir();

This method 1st checks whether the directory is already available or not. If it is already available then this method simply returns false without creating any directory. If this directory is not already available then it will create a new directory and returns true

4. boolean isFile();

Returns true if the File object represents a physical file.

5. boolean isDirectory();

Returns true if the File object represents a directory.

6. String[] list();

It returns the names of all files and subdirectories present in the specified directory.

7. long length();

Returns the no of characters present in the file.

8. boolean delete();

To delete a file or directory

FileWriter:

By using the FileWriter object we can write character data to the file.

Constructors:

```
FileWriter fw=new FileWriter(String name);
```

```
FileWriter fw=new FileWriter(File f);
```

The above 2 constructors are meant for overriding the data to the file.

Instead of overriding if we want append operation then we should go for the following 2 constructors.

```
FileWriter fw=new FileWriter(String name,boolean append);
```

```
FileWriter fw=new FileWriter(File f,boolean append);
```

If the specified physical file is not already available then these constructors will create that file.

Methods:

1. int read();

It attempts to read the next character from the file and return its Unicode value. If the next character is not available then we will get -1.

2. int i=fr.read();

3. System.out.println((char)i);

As this method returns unicodevalue , while printing we have to perform type casting.

4. int read(char[] ch);

It attempts to read enough characters from the file into char[] array and returns the number of characters copied from the file into char[] array.

```

5. File f=new File("abc.txt");
6. Char[] ch=new Char[(int)f.length()];
7. void close();

```

Usage of FileWriter and FileReader is not recommended because of following reason

1. While writing data by FileWriter compulsory we should insert line separator(\n) manually which is a bigger headache to the programmer.
2. While reading data by FileReader we have to read character by character instead of line by line which is not convenient to the programmer.
3. To overcome these limitations we should go for BufferedWriter and BufferedReader concepts

BufferedWriter:

By using the BufferedWriter object we can write character data to the file.

Constructor:

```

BufferedWriter bw=new BufferedWriter(writer w);
BufferedWriter bw=new BufferedWriter(writer w,int buffersize);

```

Note: BufferedWriter never communicates directly with the file it should communicate via some writer object.

Which of the following declarations are valid?

1. BufferedWriter bw =new BufferedWriter("cricket.txt"); (invalid)
2. BufferedWriter bw =new BufferedWriter (new File("cricket.txt")); (invalid)
3. BufferedWriter bw =new BufferedWriter (new FileWriter("cricket.txt")); (valid)
4. BufferedWriter bw =new BufferedWriter(new BufferedWriter(new FileWriter("crickter.txt")));

Methods:

1. write(int ch);
2. write(char[] ch);
3. write(String s);
4. flush();
5. close();
6. newline();

Inserting a new line character to the file.

Note:

When compared with FileWriter, which of the following capability(facility) is available as a method in BufferedWriter.

1. Writing data to the file.
2. Closing the writer.
3. Flush the writer.
4. Inserting newline character.

Ans. 4

BufferedReader:

This is the most enhanced(better) Reader to read character data from the file.

Constructors:

```
BufferedReader br=new BufferedReader(Reader r);  
BufferedReader br=new BufferedReader(Reader r,int buffersize);
```

Note

- => BufferedReader can not communicate directly with the File it should communicate via some Reader object.
- => The main advantage of BufferedReader over FileReader is that we can read data line by line instead of character by character.

Methods:

1. int read();
 2. int read(char[] ch);
 3. String readLine();
- It attempts to read the next line and return it , from the File. if the next line is not available then this method returns null.
4. void close();

PrintWriter:

- => This is the most enhanced Writer to write text data to the file.
- => By using FileWriter and BufferedWriter we can write only character data to the File but by using PrintWriter we can write any type of data to the File.

Constructors:

```
PrintWriter pw=new PrintWriter(String name);  
PrintWriter pw=new PrintWriter(File f);  
PrintWriter pw=new PrintWriter(Writer w);
```

PrintWriter can communicate either directly to the File or via some Writer object also.

Methods:

1. write(int ch);
2. write (char[] ch);
3. write(String s);
4. flush();
5. close();
6. print(char ch);
7. print (int i);
8. print (double d);
9. print (boolean b);
- 10.print (String s);
- 11.println(char ch);
- 12.println (int i);
- 13.println(double d);
- 14.println(boolean b);
- 15.println(String s);

Serialization: (1.1 v)

=> The process of saving (or) writing the state of an object to a file is called serialization but strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.

=> By using FileOutputStream and ObjectOutputStream classes we can achieve a serialization process.

|=> writeObject(Object obj)

Ex: big balloon.

De-Serialization:

=> The process of reading the state of an object from a file is called DeSerialization but strictly speaking it is the process of converting an object from file supported form (or) network supported form to java supported form.

=> By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

|=> readObject()

Transient keyword:

1. transient is the modifier applicable only for variables, but not for classes and methods.

2. While performing serialization, if we don't want to save the value of a particular variable to meet the security constant of such a type of variable, then we should declare that variable with the "transient" keyword.

3. At the time of serialization JVM ignores the original value of the transient variable and saves the default value to the file.

4. That is transient means "not to serialization".

static Vs transient :

1. static variable is not part of the object state hence they won't participate in serialization because of this declaring a static variable as transient there is no use.

Transient Vs Final:

1. final variables will be participated into serialization directly by their values.

Hence declaring a final variable as transient there is no use.

//the compiler assign the value to final variable

eg: final int x= 10;

int y = 20;

System.out.println(x); // compiler will replace this as System.out.println(20) becoz x is final.

System.out.println(y);

Externalization : (1.1 v)

1. In default serialization everything is taken care of by JVM and the programmer doesn't have any control.

2. In serialization the total object will always be saved and it is not possible to save part of the object, which creates performance problems at certain points.

3. To overcome these problems we should go for externalization where everything is taken care of by programmers and JVM doesn't have any control.

4. The main advantage of externalization over serialization is that we can save either the total object or part of the object based on our requirement.

5. To provide Externalizable ability for any object, the corresponding class should implement an externalizable interface.

6. Externalizable interface is child interface of serializable interface.

Serialization: (1.1 v)

=> The process of saving (or) writing the state of an object to a file is called serialization but strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.

=> By using FileOutputStream and ObjectOutputStream classes we can achieve a serialization process.

|=> writeObject(Object obj)

Ex: big balloon.

De-Serialization:

=> The process of reading the state of an object from a file is called DeSerialization but strictly speaking it is the process of converting an object from file supported form (or) network supported form to java supported form.

=> By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

|=> readObject()

Transient keyword:

1. transient is the modifier applicable only for variables, but not for classes and methods.

2. While performing serialization, if we don't want to save the value of a particular variable to meet the security constant of such a type of variable, then we should declare that variable with the "transient" keyword.

3. At the time of serialization JVM ignores the original value of the transient variable and saves the default value to the file.

4. That is transient means "not to serialization".

static Vs transient :

1. static variable is not part of the object state hence they won't participate in serialization because of this declaring a static variable as transient there is no use.

Transient Vs Final:

1. final variables will be participated into serialization directly by their values.

Hence declaring a final variable as transient there is no use.

//the compiler assign the value to final variable

eg: final int x= 10;

int y = 20;

System.out.println(x); // compiler will replace this as System.out.println(20) becoz x is final.

System.out.println(y);

Externalization : (1.1 v)

1. In default serialization everything is taken care of by JVM and the programmer doesn't have any control.

2. In serialization the total object will always be saved and it is not possible to save part of the object, which creates performance problems at certain points.

3. To overcome these problems we should go for externalization where everything is taken care of by programmers and JVM doesn't have any control.

4. The main advantage of externalization over serialization is that we can save either the total object or part of the object based on our requirement.

5. To provide Externalizable ability for any object, the corresponding class should implement an externalizable interface.

6. Externalizable interface is child interface of serializable interface.

Externalizable interface defines 2 methods :

1. writeExternal(ObjectOutput out) throws IOException
2. readExternal(ObjectInput in) throws IOException, ClassNotFoundException

public void writeExternal(ObjectOutput out) throws IOException

This method will be executed automatically at the time of Serialization with in this method , we have to write code to save required variables to the file .

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException

This method will be executed automatically at the time of deserialization. Within this method , we have to write code to save the required variable from the file and assign it to the current object.

At the time of deserialization JVM will create a separate new object by executing public no-arg constructor on that object JVM will call readExternal() method.

Every Externalizable class should compulsorily contain a public no-arg constructor otherwise we will get RuntimeException saying "InvalidClassException".

Difference b/w Serialization and Externalization

Serialization

1. It is meant for default Serialization
2. Here every thing takes care by JVM and programmer doesn't have any control doesn't have any control.
3. Here the total object will always be saved and it is not possible to save part of the object.
4. Serialization is the best choice if we want to save a total object to the file.
5. relatively performance is low.
6. Serializable interface doesn't contain any method
7. It is a marker interface.
8. Serializable class not required to contain public no-arg constructor.
9. transient keyword play role in serialization

Externalization

1. It is meant for Customized Serialization
2. Here everything is taken care of by the programmer and JVM does not have any control.
3. Here based on our requirement we can save either total object or part of the object.
4. Externalization is the best choice if we want to save part of the object.
5. relatively performance is high
6. Externalizable interface contains 2 methods :
 1. writeExternal()
 2. readExternal()
7. It is not a marker interface.
8. Externalizable class should compulsory contains public no-arg constructor otherwise we will get RuntimeException saying "InvalidClassException"
9. transient keyword don't play any role in Externalization.

serialVersionUID

- => To perform Serialization & Deserialization internally JVM will use a unique identifier,which is nothing but serialVersionUID .
- => At the time of serialization JVM will save serialVersionUID with the object.
- => At the time of Deserialization JVM will compare serialVersionUID and if it is matched then only object will be Deserialized otherwise we will get a RuntimeException saying "InvalidClassException".

The process in depending on default serialVersionUID are :

1. After Serializing object if we change the .class file then we can't perform deserialization because of mismatch in serialVersionUID of local class and serialized object in this case at the time of Deserialization we will get RuntimeException saying in "InvalidClassException".
2. Both sender and receiver should use the same version of JVM if there is any incompatibility in JVM versions then receive unable to deserializable because of different serialVersionUID , in this case receiver will get RuntimeException saying "InvalidClassException".
3. To generate serialVersionUID internally JVM will use complexAlgorithm which may create performance problems.

We can solve the above problems by configuring our own serialVersionUID .

eg#.

```
import java.io.Serializable;
public class Dog implements Serializable {
    private static final long serialVersionUID=1L;
    int i=10;
    int j=20;
}

import java.io.*;
public class Sender {
    public static void main(String[] args) throws IOException {
        Dog d=new Dog();
        FileOutputStream fos=new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(d);
    }
}

import java.io.*;
public class ReceiverApp {
    public static void main(String[] args) throws IOException, ClassNotFoundException{
        FileInputStream fis=new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Dog d2=(Dog) ois.readObject();
        System.out.println(d2.i+"====>"+d2.j);
    }
}
```

D:\TestApp>javac Dog.java

D:\TestApp>java Sender

D:\TestApp>javac Dog.java

D:\TestApp>java ReceiverApp

10====>20

=> In the above program after serialization even though we perform any change to Dog.class file we can deserialize the object.

=> We can configure our own serialVersionUID both sender and receiver not required to maintain the same JVM versions.

Note : some IDE's generate explicit serialVersionUID