File Handling

2 Jan File Handling IO Operation

03 January 2023 11:27

What is persistency?

It is a mechanism of storing the data permanently on to the file.

In java persistency can be achieved through an API's available inside package called "java.io".

Input and Output Operations

==========

Agenda:

1. File
2. FileWriter
3. FileReader
4. BufferedWriter
5. BufferedReader

File:

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

This line 1st checks whether abc.txt file is already available (or) not if it is already available then "f" simply refers that file.

If it is not already available then it won't create any physical file just creates a java File object represents name of the file.

Example:

```
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("IPLTeams");
        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 name of the file or directory in 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 name of the file or directory present in specified sub directory.

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");

Requirement

==========

=> Write code to create a file named with demo.txt in current working directory.

cwd

|=> abc.txt

Program:

```
import java.io.*;
class FileDemo{
    public static void main(String[] args)throws IOException{
        File f=new File("demo.txt");
        f.createNewFile();
    }
}
```

Requirement

=> Write code to create a directory named with IPLTeam in current working directory and create a file named with abc.txt in that directory.

cwd

|=> IPLTeam

|=> abc.txt

Program:

```
import java.io.*;
```

```java
class FileDemo{
  public static void main(String[] args)throws IOException{
    File f1=new File("IPLTeam");
    f1.mkdir();
    File f2=new File("IPLTeam","abc.txt");
    f2.createNewFile();
  }
}
```

Requirement: Write code to create a file named with rcb.txt present in c:\IPLTeam folder.

C

|=> IplTeam

|-> rcb.txt

Program:

```java
import java.io.*;
class FileDemo{
    public static void main(String[] args)throws IOException{
        File f=new File("c:\\IPLTeam","rcb.txt");
        f.createNewFile();
    }
}
```

Assuming C:\\IPLTeam should be already available otherwise it would result in "FileNotFoundException".

Important methods of file class:

1. boolean exists();

   Returns true if the physical file or directory 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

Requirement:

Requirement: Write a program to display the names of all files and directories present in D:\\Java Job Guarantee Batch

Requirement: Write a program to display only file names.

Requirement: Write a program to display only directory names.

```
import java.io.*;
class Test
{
    public static void main(String[] args)throws Exception
    {
        int dirCount     = 0;
        int jpgFileCount = 0;
        int txtFileCount = 0;
        int zipFileCount = 0;
        String location = "D:\\Java Job Guarantee Batch";
        File f= new File(location);
        String[] names = f.list();
        for(String name : names){
            // D:\\Java Job Guarantee Batch
            // all files we are iterating
            File f1 = new File(f,name); - obj is created only, not file.
            if (f1.isDirectory())
                dirCount++;
            if(f1.isFile()){
                if (name.endsWith(".png"))
                    jpgFileCount++;
                if(name.endsWith(".txt"))
                    txtFileCount++;
                if(name.endsWith(".zip"))
                    zipFileCount++;

            }
            System.out.println(name);
        }
        System.out.println("Total no of JPGfiles  is :: "+jpgFileCount);
        System.out.println("Total no of txtfiles  is :: "+txtFileCount);
        System.out.println("Total no of Zipfiles  is :: "+zipFileCount);
        System.out.println("Total no of Directory is :: "+dirCount);
    }
    //JVM shutdown now
```

}

FileWriter:

By using 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 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. write(int ch);

   To write a single character to the file.

2. write(char[] ch);

   To write an array of characters to the file.

3. write(String s);

   To write a String to the file.

4. flush();

   To give the guarantee the total data include last character also written to the file.

5. close();

   To close the stream.

eg#1.

```
import java.io.FileWriter;
import java.io.IOException;
public class TestApp {
    public static void main(String[] args)throws IOException {
            FileWriter fw=new FileWriter("abc.txt");
            fw.write(73);
            fw.write("neuron\nTechnology\nPrivate\nLimited");
            fw.write("\n");
            char ch[] ={'a','b','c'};
            fw.write(ch);
            fw.flush();
            fw.close();
    }
}
```

A new file will be created automatically

```
abc.txt
=======
Ineuron
Technology
Private
Limited
abc
```

Note:

=> The main problem with FileWriter is we have to insert line separator manually,which is difficult to the programmer. ('\n')

=> And even line separator varing from system to system.

FileReader:

=> By using FileReader object we can read character data from the file.

Constructors:

FileReader fr=new FileReader(String name);

FileReader fr=new FileReader (File f);

Methods

=======

1. int read();

   It attempts to read 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.

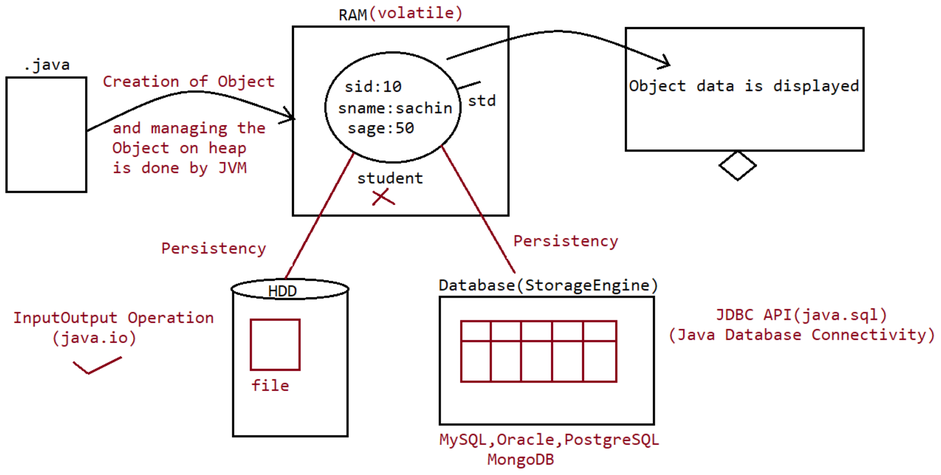
4. int read(char[] ch);

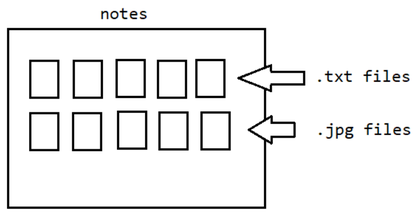
   It attempts to read enough characters from the file into char[] array and returns the no of characters copied from the file into char[] array.

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

eg#1.

```
import java.io.FileReader;
import java.io.IOException;
public class TestApp {
    public static void main(String[]
args)throws IOException {
            FileReader fr=new
FileReader("abc.txt");
            int i=fr.read();
            while(i!=-1){
                System.out.println((char
)i);
                i=fr.read();
            }
}
```





File(C)
   On an existing file/directory we just performed the operations like
          a. isFile() b. isDirectory()
          c. length() d. exists()
          e. mkdir()   f. createNewFile()
          g. list()

If we want to perform Reader operation on a File, then we need to use "FileReader".
If we want to perfrom Write operation on a File, then we need to use "FileWriter".

3rd Jan FileHandling

06 January 2023 16:03

eg#2. Reading an array of characters

abc.txt

=======

1000 characters are available

Scenario 1:

```
FileReader fr=new FileReader("abc.txt");
char[] ch=new char[10];
int noOfCharactersCopied=fr.read(ch);
```

Scenario2:

```
FileReader fr=new FileReader("abc.txt");
char[] ch=new char[10000];
int noOfCharactersCopied=fr.read(ch);
```

```
public class TestApp {
    public static void main(String[] args)throws IOException {
            File f=new File("abc.txt");

            FileReader fr=new FileReader(f);
            char ch[] = new char[(int)f.length()];

            fr.read(ch);
            String data=new String(ch);
            System.out.println(data);
            fr.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.

Assume we need to search for a 10 digit mobile no present in a file called "mobile.txt"

=>Since we can read only character just to search one mobile no 10 searching and to search 10,000 mobile no we need to read 10r times,

so performance is very low.

3. To overcome these limitations we should go for BufferedWriter and BufferedReader concepts.

BufferedWriter:

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

It can't communicate with the file directly, it can communicate only with writer Object.

Constructor

BufferedWriter bw=new BufferedWriter(Writer w);

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

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")));//valid it indicates 2 levels of buffering.
```

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 method in BufferedWriter.

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

Ans. 4

eg#1.

```
class Test
```

```
{
    public static void main(String[] args)throws Exception
    {
        FileWriter fw     = new FileWriter("abc.txt",true);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(105);
        bw.write("Neuron");
        bw.newLine();
        char[] ch ={'P','W','S','K','I','L','L','S'};
        bw.write(ch);
        bw.newLine();
        bw.write("unicorn");
        bw.flush();//to make sure the operation is succesfull on file
        bw.close();//internally fw.close() call will happen

    }
    //JVM shutdown now
}
```

abc.txt

```
105 Neuron
PW Skills
Unicorn
```

Note

```
1.bw.close()// recomended to use
2.fw.close()// not recomended to use
3.bw.close()// not recomended to use fw.close()
```

=> When ever we are closing BufferedWriter automatically underlying writer will be closed and we are not close explicitly.

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 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 next line and return it , from the File. if the
next line is not available then                                    //this method returns
null.
4. void close();
```

eg#1.Read the data from the file called "abc.txt"4

```
public class TestApp {
    public static void main(String[] args)throws IOException {
            FileReader fr=new FileReader("abc.txt");
            BufferedReader br=new BufferedReader(fr);
            String line= br.readLine();
            while(line!=null){
                System.out.println(line);
                line=br.readLine();
            }
            br.close();
    }
}
```

abc.txt

Hi yash
You r awesome

Hi yash
you are awesome

Note:

```
1.br.close()  // recomended to use
2.fw.close() // not recomended to use
3.br.close() // not recomended to use both
   fw.close()
```

=> Whenever we are closing BufferedReader automatically underlying FileReader will be closed it is not required to close explicitly.

=> Even this rule is applicable for BufferedWriter also.

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);
```

What is the difference between write(100) and print(100)?

=> In the case of write(100) the corresponding character "d" will be added to the File but

=> In the case of print(100) "100" value will be added directly to the File.

Note 1:

1. The most enhanced Reader to read character data from the File is BufferedReader.
2. The most enhanced Writer to write character data to the File is PrintWriter.

Note 2:

1.In general we can use Readers and Writers to handle character data. Where as we can use InputStreams and OutputStreams to handle binary data(like images, audio files, video files etc).

2. We can use OutputStream to write binary data to the File and we can use InputStream to read binary data from the File

Character Data => Reader and Writer

Binary Data => InputStream and OutputStream

Requirement => file1.txt ,file2.txt copy all the contents to file3.txt

```
import java.io.*;
class Test
{
    public static void main(String[] args)throws Exception
    {
        PrintWriter pw = new PrintWriter("file3.txt");
        //reading from first file and writing to file3.txt
        BufferedReader br= new BufferedReader(new FileReader("file1.txt"));
        String line = br.readLine();
        while (line!=null)
        {
            pw.println(line);
            line=br.readLine();
```

```
        }
        //reading from second file and writing to file3.txt
        br =new BufferedReader(new FileReader("file2.txt"));
        line = br.readLine();
        while (line!=null)
        {
            pw.println(line);
            line=br.readLine();
        }
        //To write all the data to file3.txt
        pw.flush();

        br.close();
        pw.close();
        System.out.println("Open file3.txt to see the result");
    }
    //JVM shutdown now
}
```

Requirement => file1.txt file2.txt copy one line from file1.txt and from file2.txt to file3.txt.

```
import java.io.*;
class Test
{
    public static void main(String[] args)throws Exception
    {
        PrintWriter pw = new PrintWriter("file3.txt");
        //reading from first file and writing to file3.txt
        BufferedReader br1= new BufferedReader(new FileReader("file1.txt"));
        String line1 = br1.readLine();
        BufferedReader br2 =new BufferedReader(new FileReader("file2.txt"));
        String line2 = br2.readLine();
        while (line1!=null || line2!=null)
        {
            if (line1!=null)
            {
                pw.println(line1);
                line1=br1.readLine();
            }

            if(line2!=null)
            {
                pw.println(line2);
                line2=br2.readLine();
            }

        }
        //To write all the data to file3.txt
        pw.flush();

        br1.close();
        br2.close();
        pw.close();
        System.out.println("Open file3.txt to see the result");
    }
    //JVM shutdown now
}
```

Requirement => Write a program to remove duplicates from the file

```
class Test
{
    public static void main(String[] args)throws Exception
    {
        BufferedReader br = new BufferedReader(new FileReader("input.txt"));
        PrintWriter    pw = new PrintWriter("output.txt");
        String target = br.readLine();
        while (target!=null)
        {
            boolean isAvailable =false;
            BufferedReader br1 =new BufferedReader(new FileReader("output.txt"));
            String line = br1.readLine();
            //control comes out of while looop in smooth fashion without break
            while (line!=null)
            {
                //if matched control should come out with break
                if (line.equals(target))
                {
                        isAvailable = true;
                        break;
                }
                line=br1.readLine();
            }

            if (isAvailable==false){
                pw.println(target);
                pw.flush();
            }
            target = br.readLine();
        }
        br.close();
        pw.close();
    }
    //JVM shutdown now
}
```

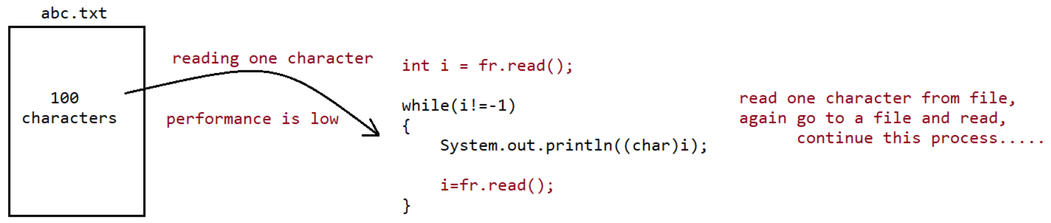
Requirement => Write a program to perform extraction of mobile no only if there is no duplicates

```
class Test
{
    public static void main(String[] args)throws Exception
    {
        BufferedReader br = new BufferedReader(new FileReader("input.txt"));
        PrintWriter    pw = new PrintWriter("output.txt");
        String target = br.readLine();
        while (target!=null)
        {
            boolean isAvailable =false;
            BufferedReader br1 =new BufferedReader(new FileReader("delete.txt"));
            String line = br1.readLine();
            //control comes out of while looop in smooth fashion without break
            while (line!=null)
            {
```

```
                //if matched control should come out with break
                if (line.equals(target))
                {
                        isAvailable = true;
                        break;
                }
                line=br1.readLine();
            }

            if (isAvailable==false){
                pw.println(target);
                pw.flush();
            }
            target = br.readLine();
        }
        br.close();
        pw.close();
    }
    //JVM shutdown now
}
```

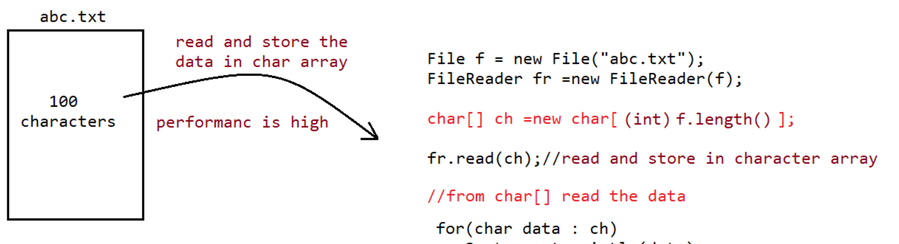
Write a code to read the data from the file and identify which data is of larger in length(assuming the data is String)

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

        BufferedReader br = new BufferedReader(new FileReader("data.txt"));
        String data = br.readLine();

        int maxLength = 0;
        String result = "";
        while (data!=null)
        {
            int resultLength=data.length();
            if (maxLength<resultLength)
            {
                maxLength = resultLength;
                result=data;
            }
            data= br.readLine();
        }
        System.out.println("The maxLength string data in a file is :: "+result);
        System.out.println("The maxLength string in a file is :: "+maxLength);
    }
    //JVM shutdown now
}
```




```
int i = fr.read();

while(i!=-1)
{
    System.out.println((char)i);

    i=fr.read();
}
```

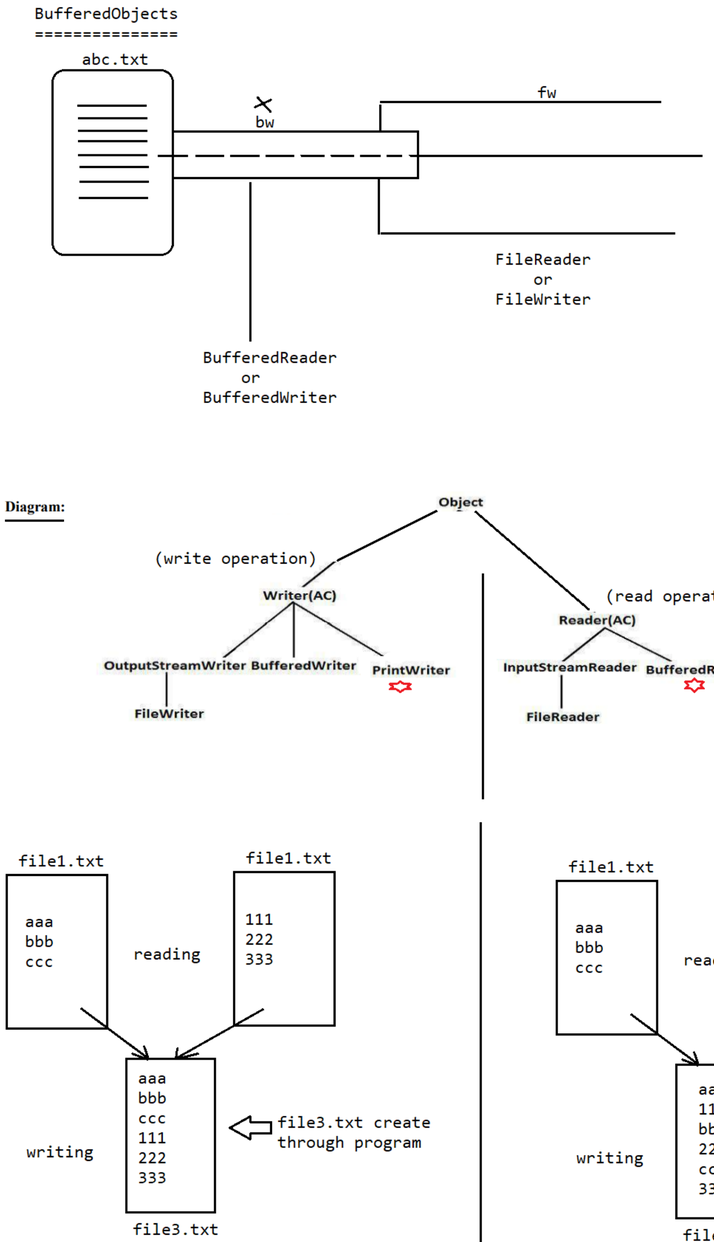
read one character from file,
again go to a file and read,
continue this process.....




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

char[] ch =new char[ (int) f.length() ];

fr.read(ch);//read and store in character array

//from char[] read the data

 for(char data : ch)
    System.out.println(data);
```

read one character from a file,
store in array,continue the reading.....

now use the file data from char[]



FileWriter
BufferedWriter
⇒ write(int)
write(ch[])
only one character at a time
String data writing is also possible
write(String)

⇩ boolean,long,float,double

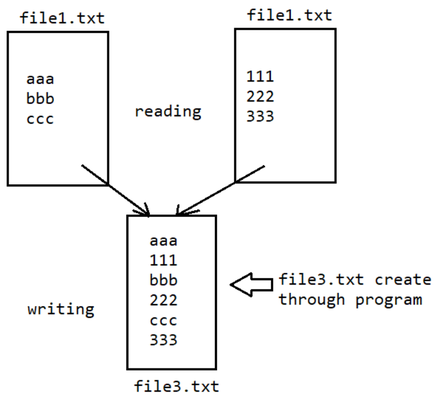
To OverCome the limitation of writing any type of data we need to use "PrintWriter"

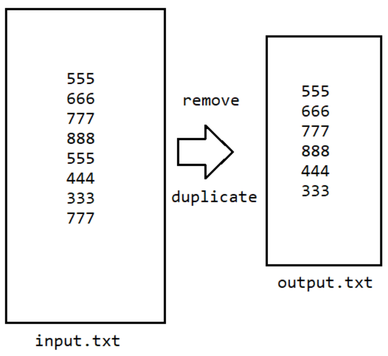
To OverCome the limitation of \n, we use newLine(),but remembering a method name is challenging.

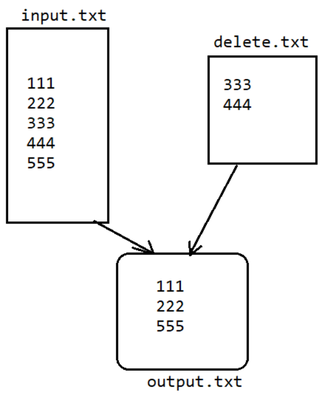
To resolve this problem we need to use println()

FileReader
BufferedReader
⇒ readLine()

Diagram:







Serialization

&

De-Serialization

4th Jan Serialization , De-Serialization

06 January 2023 16:04

Agenda :

1. Serialization
2. Deserialization
3. transient keyword
4. static Vs transient
5. transient Vs final
6. Object graph in serialization.
7. customized serialization.
8. Serialization with respect inheritance.
9. Externalization
10. Difference between Serialization + Externalization
11. SerialVersionUID

Serialization

=> The process of saving (or) writing 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 serialization process.

|=> writeObject(Object obj)

De-Serialization

=> The process of reading 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()

```
import java.io.*;
/*
 public java.io.ObjectOutputStream(java.io.OutputStream) throws java.io.IOException;
 public java.io.FileOutputStream(java.lang.String) throws java.io.FileNotFoundException;
 public final void writeObject(java.lang.Object) throws java.io.IOException;
 public java.io.ObjectInputStream(java.io.InputStream) throws java.io.IOException;
 public java.io.FileInputStream(java.lang.String) throws java.io.FileNotFoundException;
 public final java.lang.Object readObject() throws java.io.IOException,
java.lang.ClassNotFoundException;
```

```
*/
class Dog implements Serializable
{
    static{
        System.out.println("static block gets executed...");
    }
    Dog(){
        System.out.println("Object is created...");
    }
    int i = 10;
    int j = 20;
}
class Test
{
    public static void main(String[] args)throws Exception
    {
            Dog d = new Dog();
            System.out.println("Serialization started.....");
            String fileName = "abc.ser";
            FileOutputStream fos   = new FileOutputStream(fileName);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(d);
            System.out.println("Serialized Object refernce is ::"+d);
            System.out.println("Serialization ended.....");
            //To pause the execution till we press some key from keyboard
            System.in.read();
            System.out.println("De-Serialization started.....");
            FileInputStream fis   = new FileInputStream("abc.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            Object obj=ois.readObject();
            Dog d1 = (Dog)obj;
            System.out.println("De-Serialized Object refernce is ::"+d1);
            System.out.println("De-Serialization ended.....");

    }
    //JVM shutdown now
}
```

i=10
j=20

eg#2.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.Serializable;
class Dog implements Serializable{
     int i=10;
     int j=20;
}
class Cat implements Serializable{
    int i=100;
    int j=200;
}
public class TestApp {
    public static void main(String[] args)throws IOException,ClassNotFoundException {

        Dog d1=new Dog();
```

```
        Cat c1=new Cat();
        System.out.println("serialization started");
        FileOutputStream fos= new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(d1);
        oos.writeObject(c1);
        System.out.println("Serialization ended");

        System.out.println("Deserialization started");
        FileInputStream fis=new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Dog d2=(Dog) ois.readObject();
        Cat c2=(Cat) ois.readObject();
        System.out.println("Deserialization ended");
        System.out.println("Dog object data");
        System.out.println(d2.i+"\t" +d2.j);
        System.out.println("Cat object data");
        System.out.println(c2.i+"\t" +c2.j);

    }
}
Output
serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      20
Cat object data
100     200
```

Note:

1. We can perform Serialization only for Serilizable objects.
2. An object is said to be Serilizable if and only if the corresponding class implements Serializable interface.
3. Serializable interface present in java.io package and does not contain any abstract methods. It is marker interface.

   The required ability will be provided automatically by JVM.
4. We can add any no. Of objects to the file and we can read all those objects from the file but in which order we wrote

   objects in the same order only the objects will come back. That is order is important.
5. If we are trying to serialize a non-serializable object then we will get RuntimeException saying "NotSerializableException".

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

security constant such

type of variable ,then we should declare that variable with "transient" keyword.

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

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

eg#1.

```
class Dog implements Serializable{
    int i=10;
   transient int j=20;
}
public class TestApp {
    public static void main(String[] args)throws IOException,ClassNotFoundException {

        Dog d1=new Dog();
        System.out.println("serialization started");
        FileOutputStream fos= new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(d1);
        System.out.println("Serialization ended");

        System.out.println("Deserialization started");
        FileInputStream fis=new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Dog d2=(Dog) ois.readObject();
        System.out.println("Deserialization ended");
        System.out.println("Dog object data");
        System.out.println(d2.i+"\t" +d2.j);

    }
}
```

```
Output
serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      0
```

static Vs transient :

1. static variable is not part of object state hence they won't participate in serialization because of this declaring a static variable as

transient there is no use.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.Serializable;
```

```
class Dog implements Serializable{
     static transient int i=10;
     int j=20;
}
public class TestApp {
    public static void main(String[] args)throws IOException,ClassNotFoundException {

        Dog d1=new Dog();

        System.out.println("serialization started");
        FileOutputStream fos= new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(d1);

        System.out.println("Serialization ended");

        System.out.println("Deserialization started");
        FileInputStream fis=new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Dog d2=(Dog) ois.readObject();

        System.out.println("Deserialization ended");
        System.out.println("Dog object data");
        System.out.println(d2.i+"\t" +d2.j);
    }
}
Output
serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      20
```

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);
```

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.Serializable;
class Dog implements Serializable{
     int i=10;
```

```
    transient final int j=20;
}

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

        Dog d1=new Dog();

        System.out.println("serialization started");
        FileOutputStream fos= new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(d1);

        System.out.println("Serialization ended");

        System.out.println("Deserialization started");
        FileInputStream fis=new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Dog d2=(Dog) ois.readObject();

        System.out.println("Deserialization ended");
        System.out.println("Dog object data");
        System.out.println(d2.i+"\t" +d2.j);

    }
}
Output
Serialization started
Serialization ended
Deserialization started
Deserialization ended
Dog object data
10      20
```

Declaration output

==========

```
1.
int i=10;
int j=20;
output
  10 -------- 20
2.
transient int i=10;
int j=20;
output
   0-------20
3.
transient int i=10;
transient static int j=20;
output
    0---------20
4.
transient final int i=10;
transient int j=20;
output
     10-------0
```

5.

```
transient final int i=10;
transient static int j=20;
output
    10--------20
```

Note:

We can serialize any no of objects to the file but in which order we serialized in the same order only we have to deserialize,

if we change the order then it would result in "ClassCastException".

Example :

```
Dog d1=new Dog( );
Cat c1=new Cat( );
Rat r1=new Rat( );
FileOutputStreamfos=new FileOutputStream("abc.ser");
ObjectOutputStreamoos=new ObjectOutputStream(fos);
oos.writeObject(d1);
oos.writeObject(c1);
oos.writeObject(r1);
FileInputStreamfis=new FileInputStream("abc.ser");
ObjectInputStreamois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
Cat c2=(Cat)ois.readObject();
Rat r2=(Rat)ois.readObject();
=> If we don't know the order of Serialization then we need to use the following code
    FileInputStream fis =new FileInputStream("abc.ser");
    ObjectInputStream ois=new ObjectInputStream(fis);

    Object obj=ois.readObject();
    if(obj instanceof Dog){
        Dog d=(Dog)obj;
        //perform operation related to Dog
    }
    if(obj instanceof Cat){
        Cat C=(Cat)obj;
        //perform operation related to Cat
    }
    if(obj instanceof Rat){
           Rat r=(Rat)obj;
         //perform operation related to Rat
}
```

Object graph in serialization:

1. Whenever we are serializing an object the set of all objects which are reachable from that object will be serialized

   automatically. This group of objects is nothing but object graph in serialization.
2. In object graph every object should be Serializable otherwise we will get runtime exception saying

"NotSerializableException".

eg#1.

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;

class Dog implements Serializable{
    Cat c=new Cat();
}
class Cat implements Serializable{
    Rat r=new Rat();
}
class Rat implements Serializable{
     int i=10;
}
public class Test {
    public static void main(String[] args)throws IOException,ClassNotFoundException{

        Dog d= new Dog();
        System.out.println("Serialization Started");

        FileOutputStream fos= new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);

        oos.writeObject(d);

        System.out.println("Serialization ended");

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

        System.out.println("DeSerialization Started");

        FileInputStream fis= new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);

        Dog d1=(Dog)ois.readObject();

        System.out.println(d1.c.r.i);
        System.out.println("DeSerialization ended");

   }
}
Output
======
Serialization Started
Serialization ended
**********************************
DeSerialization Started
10
DeSerialization ended
```

CustomizedSerialization

==========

During default Serialization there may be a chance of lose of information due to transient keyword.

example: remember mango and money inside it.

eg#1.

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;
class Account implements Serializable{
    String name="sachin";
    transient String password="tendulkar";
}
public class Test {
    public static void main(String[] args)throws IOException,ClassNotFoundException{

        Account acc=new Account();
        System.out.println(acc.name +"=====> "+ acc.password);
        System.out.println("Serialization Started");
        FileOutputStream fos= new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(acc);
        System.out.println("Serialization ended");
        System.out.println("**********************************");
        System.out.println("DeSerialization Started");
        FileInputStream fis= new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        acc=(Account)ois.readObject();
        System.out.println(acc.name +"=====> "+ acc.password);
        System.out.println("DeSerialization ended");
   }
}
```

=> In the above example before serialization Account object can provide proper username and password.

But after Deserialization Account object can provide only username but not password. This is due to declaring password as transient.

Hence doing default serialization there may be a chance of loss of information due to transient keyword.

=> We can recover this loss of information by using customized serialization.

We can implements customized serialization by using the following two methods.

1. private void writeObject(ObjectOutputStream os) throws Exception.

=> This method will be executed automatically by jvm at the time of serialization.

=> Hence at the time of serialization if we want to perform any extra work we have to define that in this method only. (prepare encrypted password and write encrypted password seperate to the file)

2. private void readObject(ObjectInputStream is) throws Exception.

=> This method will be executed automatically by JVM at the time of Deserialization.

Hence at the time of Deserialization if we want to perform any extra activity we have to define that in this method only.

(read encrypted password , perform decryption and assign decrypted password to the current object password variable)

eg#1.

```
class Account implements Serializable{
    String name="sachin";
    transient String password="tendulkar";
    private void writeObject(ObjectOutputStream oos)throws Exception{
        oos.defaultWriteObject();//performing default Serialization
        String epwd="123"+password;//performing encryption
        oos.writeObject(epwd);//write the encrypted data to file(abc.ser)

    }
    private void readObject(ObjectInputStream ois)throws Exception{
        ois.defaultReadObject();//performing default Serialization
        String epwd=(String)ois.readObject();//performing decryption
        password=epwd.substring(3);//writing the extra data to Object
    }
}
public class Test {
    public static void main(String[] args)throws IOException,ClassNotFoundException{

        Account acc=new Account();
        System.out.println(acc.name +"=====> "+ acc.password);
        System.out.println("Serialization Started");
        FileOutputStream fos= new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(acc);

        System.out.println("Serialization ended");
        System.out.println("**********************************");

        System.out.println("DeSerialization Started");

        FileInputStream fis= new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        acc=(Account)ois.readObject();

        System.out.println(acc.name +"=====> "+ acc.password);
        System.out.println("DeSerialization ended");
```

```
    }
}
```

=> At the time of Account object serialization JVM will check is there any writeObject() method in Account class or not.

=> If it is not available then JVM is responsible to perform serialization(default serialization).

=> If Account class contains writeObject() method then JVM feels very happy and executes that Account class writeObject() method.

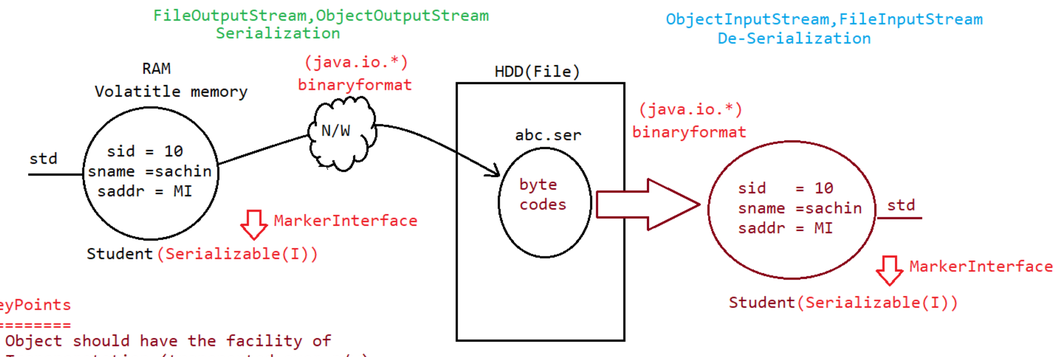
=> The same rule is applicable for readObject() method also.

```
import java.io.*;
/*
 public java.io.ObjectOutputStream(java.io.OutputStream) throws java.io.IOException;
 public java.io.FileOutputStream(java.lang.String) throws java.io.FileNotFoundException;
 public final void writeObject(java.lang.Object) throws java.io.IOException;
 public java.io.ObjectInputStream(java.io.InputStream) throws java.io.IOException;
 public java.io.FileInputStream(java.lang.String) throws java.io.FileNotFoundException;
 public final java.lang.Object readObject() throws java.io.IOException,
java.lang.ClassNotFoundException;
*/
class Account implements Serializable
{
    String userName  = "sachin";
    transient String password = "tendulkar";//loss of information
    transient int pin=4444;//loss of information

       //Write a logic of Serialization
     private void writeObject(ObjectOutputStream oos) throws Exception{
        System.out.println("writeObject method is called....");

        // perform default serialization
        oos.defaultWriteObject();
        // perform encryption on password
        String encypwd  = "123" + password;// 123tendulkar
        int encypin     = 1111  + pin;// 5555
        // write the encrytped data as object to serialized file
        oos.writeObject(encypwd);
        oos.writeInt(encypin);

     }
      //Write a logic of Deserialization
     private void readObject(ObjectInputStream ois) throws Exception{
        System.out.println("readObject method is called....");

        //perform default deserialization
        ois.defaultReadObject();
        //read encrypted data from serialized file
        String encypwd = (String)ois.readObject();
        int encypin    = ois.readInt();
        // perform decryption and attach it to instance variable
        password = encypwd.substring(3);// tendulkar
```

```
        pin      = encypin - 1111;// 4444
     }
}

class Test
{
    public static void main(String[] args)throws Exception
    {
            Account account =new Account();

            System.out.println("Serialization started.....");
            String fileName = "abc.ser";
            FileOutputStream fos   = new FileOutputStream(fileName);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(account);
            System.out.println("Serialization ended.....");
            //To pause the execution till we press some key from keyboard
            System.in.read();
            System.out.println("De-Serialization started.....");
            FileInputStream fis   = new FileInputStream("abc.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            Account acc=(Account)ois.readObject();

            System.out.println("Username is :: "+acc.userName);
            System.out.println("Password is :: "+acc.password);
            System.out.println("Pin      is :: "+acc.pin);
            System.out.println("De-Serialization ended.....");

    }
    //JVM shutdown now
}
```

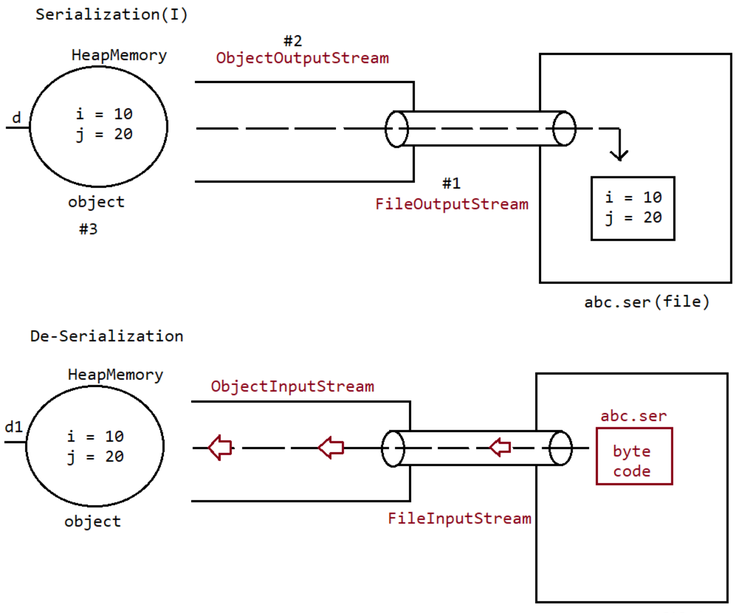
Reader ====> To read character type of data.(BufferedReader)
Writer ====> To write character type of data.(PrintWriter)

Binary Type of data
==========
InputStream
OutputStream



KeyPoints
=========
1. Object should have the facility of
   Transporatation.(transported over n/w).
2. Object should be supported to store inside
   file system.(Using Streams)

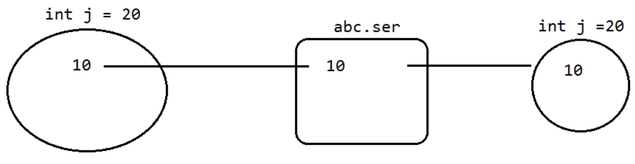
Serialization(I)



De-Serialization

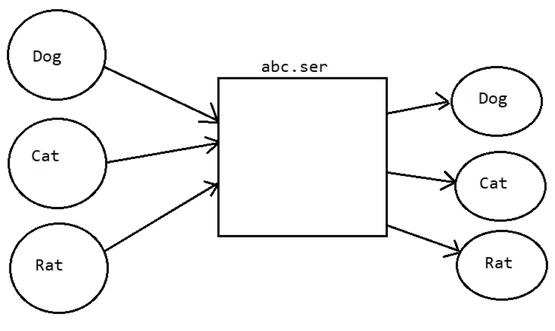
Transient
=========

```
class Dog implements Serializable
{
     final  transient int i = 10;
     static transient int j = 20;
}
```



transient -> variable don't participate in serialization

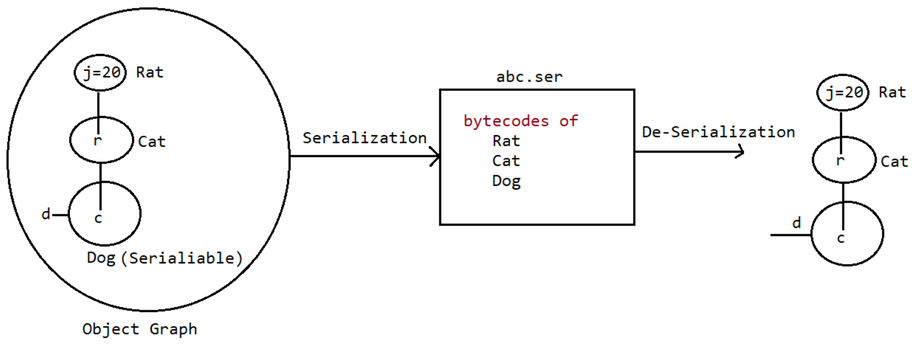
final and transient -> final means variable won't come into pitcure it is directly the values
static and transient -> variable which is static is not a part of Object, so it is not a part
                         of Serialization.



If we don't know the order of Objects we use the following approach

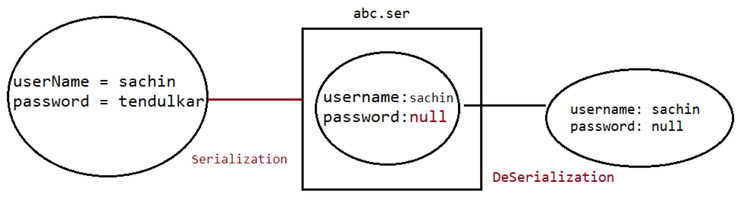
```
Object o  = oos.readObject();

if ( o instanceof Dog)
    // perform Dog related operation

if (o instanceof Cat)
   // perform Cat related operation

if (o instanceof Rat)
   // perform Rat related operation
```

Object Graph
============

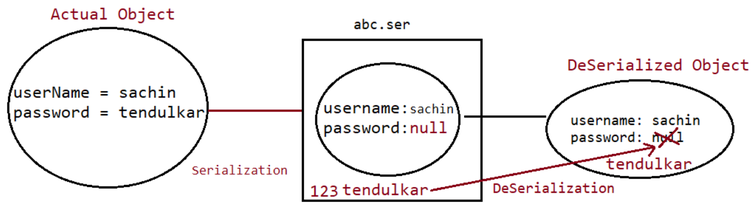


Serialization
=============

```
class Account implements Serializable
{
     String userName  = "sachin";
     transient String password = "tendulkar";
}
```



CustomSerialization
===================



Serialization
=============
1. Default Serialization should happen (password =null,username=sachin)
2. write the encrypted password data as shown below
        encypd = "123"+password;
3. Now write the encypted password also to the serialized Object

De-Serialization
================
1. Default De-Serialization should happen (password =null,username=sachin)
2. Read encrypted password and decrypt the encrypted password
3. Attach it to password variable with decrypted value.

5th jan

06 January 2023 16:05

Serialization w.r.t Inheritance

==========

Case 1:

If parent class implements Serializable then automatically every child class by default implements Serializable.

That is Serializable nature is inheriting from parent to child.

Hence even though child class doesn't implements Serializable , we can serialize child class object if parent class implements serializable interface.

```
class Animal implements Serializable{
    int i=10;
}
class Dog extends Animal{
    int j=20;
}
public class Test {
    public static void main(String[] args)throws IOException,ClassNotFoundException{

    Dog d=new Dog();
    System.out.println("Serialization started");
    FileOutputStream fos=new FileOutputStream("abc.ser");
    ObjectOutputStream oos=new ObjectOutputStream(fos);
    oos.writeObject(d);
    System.out.println("Serialization ended");

    System.out.println("*****************************");
    System.out.println("DeSerialization started");
    FileInputStream fis=new FileInputStream("abc.ser");
    ObjectInputStream ois=new ObjectInputStream(fis);
    Dog d1=(Dog)ois.readObject();
    System.out.println(d1.i+"====> "+d1.j);
    System.out.println("DeSerialization ended");

   }
}
```

```
Output
Serialization started
Serialization ended
*****************************
DeSerialization started
10====> 20
DeSerialization ended
```

Even though Dog class does not implements Serializable interface explicitly but we can Serialize Dog object because its parent class

Animal already implements Serializable interface.

Note :Object class doesn't implement Serializable interface.

Case 2:

1. Even though parent class does not implementsSerializable we can serialize child object if child class implements Serializable
   interface.

2. At the time of serialization JVM ignores the values of instance variables which are coming from non Serializable parent then instead of original value JVM saves default values for those variables to the file.

3. At the time of Deserialization JVM checks whether any parent class is non Serializable or not.
   If any parent class is nonSerializable JVM creates a separate object for every non Serializabl parent and
   shares its instance variables to the current object.

4. To create an object for non-serializable parent JVM always calls no arg constructor (default constructor) of that non Serializable parent hence every non Serializable parent should compulsory contain
   no arg constructor otherwise we will get runtime exception "InvalidClassException".

eg#1.

```
class Animal {
    int i=10;
    Animal(){
        System.out.println("No arg Animal constructor");
    }
}
class Dog extends Animal implements Serializable{
    int j=20;
    Dog(){
        System.out.println("No arg Dog constructor");
    }
}
public class Test {
    public static void main(String[] args)throws IOException,ClassNotFoundException{

    Dog d=new Dog();
    d.i=888;
    d.j=999;
```

```
    System.out.println("Serialization started");
    FileOutputStream fos=new FileOutputStream("abc.ser");
    ObjectOutputStream oos=new ObjectOutputStream(fos);
    oos.writeObject(d);
    System.out.println("Serialization ended");

    System.out.println("*****************************");
    System.out.println("DeSerialization started");
    FileInputStream fis=new FileInputStream("abc.ser");
    ObjectInputStream ois=new ObjectInputStream(fis);
    Dog d1=(Dog)ois.readObject();
    System.out.println(d1.i+"====> "+d1.j);
    System.out.println("DeSerialization ended");
   }
}
Output
No arg Animal constructor
No arg Dog constructor
Serialization started
Serialization ended
*****************************
DeSerialization started
No arg Animal constructor
10====> 999
DeSerialization ended
```

Agenda :

1. Externalization
2. Difference between Serialization & Externalization
3. SerialVersionUID

Externalization : (1.1 v)

1. In default serialization every thing takes care by JVM and programmer doesn't have any control.
2. In serialization total object will be saved always and it is not possible to save part of the object, which creates performance problems at certain point.
3. To overcome these problems we should go for externalization where every thing takes care by programmer and JVM doesn't have any control.
4. The main advantage of externalization over serialization is we can save either total object or part of the object based on our requirement.
5. To provide Externalizable ability for any object compulsory the corresponding class should implements 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 with in this method , we have to write code to save read required variable from file and assign to the current object.

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

Every Externalizable class should compusory contains public no-arg constructor otherwise we will get RuntimeExcepion saying "InvaidClassException".

eg#1.

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;
import java.io.Externalizable;
import java.io.ObjectOutput;
import java.io.ObjectInput;

class ExternalizableDemo implements Externalizable{
    String i;
    int j;
    int k;
    ExternalizableDemo(String i,int j,int k){
        this.i=i;
        this.j=j;
        this.k=k;
    }
    public ExternalizableDemo(){
        System.out.println("Zero arg constructor");
    }
```

```
    //Performing Serialization as per our requirement
    public void writeExternal(ObjectOutput out) throws IOException{
        System.out.println("call back method used while Serialization");
        out.writeObject(i);
        out.writeInt(j);
    }
    //Performing DeSerialization as per our requirement
    public void readExternal(ObjectInput in) throws IOException,ClassNotFoundException{
        System.out.println("call back method used while DeSerialization");
        i=(String)in.readObject();
        j=in.readInt();
    }
}
public class Test {
    public static void main(String[] args)throws IOException,ClassNotFoundException{

    ExternalizableDemo d=new ExternalizableDemo("nitin",100,200);
    System.out.println("Serialization started");
    FileOutputStream fos=new FileOutputStream("abc.ser");
    ObjectOutputStream oos=new ObjectOutputStream(fos);
    oos.writeObject(d);
    System.out.println("Serialization ended");

    System.out.println("*****************************");
    System.out.println("DeSerialization started");
    FileInputStream   fis=new FileInputStream("abc.ser");
    ObjectInputStream ois=new ObjectInputStream(fis);
    d=(ExternalizableDemo)ois.readObject();
    System.out.println(d.i+"======>"+d.j+"======>"+d.k);
    System.out.println("DeSerialization ended");
   }
}
Output
Serialization started
call back method used while Serialization
Serialization ended
*****************************
DeSerialization started
Zero arg constructor
call back method used while DeSerialization
nitin======>100======>0
DeSerialization ended
```

1. If the class implements Externalizable interface then only part of the object will be saved in the case output is
2. public no-arg constructor
3. nitin---- 10 ----- 0
4. If the class implements Serializable interface then the output is nitin --- 10 --- 20
5. In externalization transient keyword won't play any role , hence transient keyword not required.

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 total object will be saved always and it is not possible to save part of the object.
4. Serialization is the best choice if we want to save total object to the file.
5. relatively performence is low.
6. Serializable interface doesn't contain any method
7. It is a marker interface.
8. Serializable class not required to contains public no-arg constructor.
9. transient keyword play role in serialization

Externalization

==========

1. It is meant for Customized Serialization
2. Here every thing takes care by 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 performence 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 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 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 any incompatability 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 performence problems.

We can solve above problems by configuring our own serialVersionUID .

eg#1.

```
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);
```

```
    }
}
```

Output

```
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 if we perform any change to Dog.class file we can desenialize 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.

Usage of StringTokenzier

==========

=> It is a part of java.util package

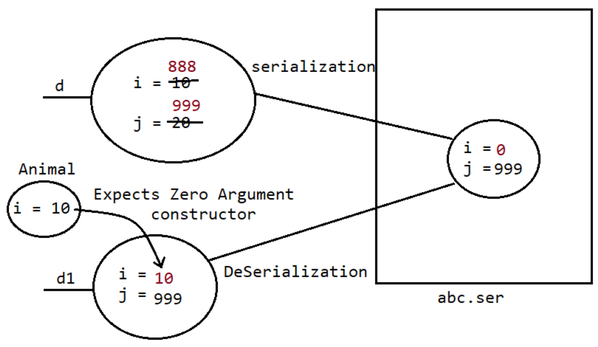
=> It is used to split the entire string into mulitple tokens based on the delimiter we supply

```
eg: String  data= "sachin ramesh tendulkar";
StringTokenzier stk = new StringTokenizer(data);
StringTokenzier stk = new StringTokenizer(data," ");
```

```
=> public boolean hasMoreTokens()
=>  public String nextToken()
```

eg#1.

```
import java.util.*;
class TestApp {
    public static void main(String[] args) {
        StringTokenizer stk = new StringTokenizer("sachin$ramesh$tendulkar","$");
        System.out.println(stk);
        int tokenCount = stk.countTokens();
        System.out.println(tokenCount);
        while (stk.hasMoreTokens())
        {
            String data = stk.nextToken();
            System.out.println(data);
        }
    }
}
```



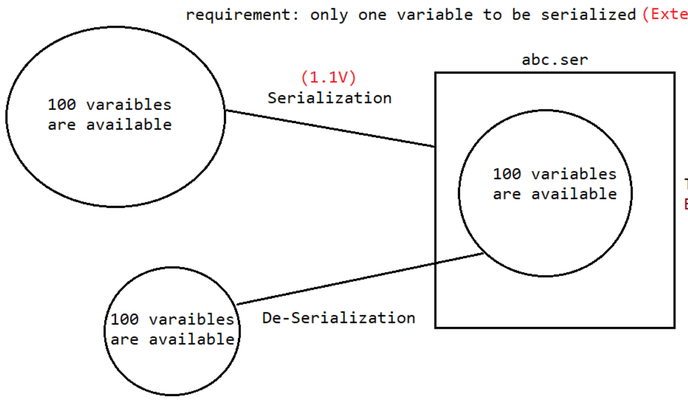
1. Even though parent class is not implementing serializable still we can serialize child object, if the child class is implementing "Serializable interface".

2. JVM while performing Serialization will check for instance variable coming from NonSerializable parent,for these variables jvm will give default value.

3.At the time of DeSerialization,JVM will check whether any Parent class is NonSerizliable or not.
   if any parent class is nonSerializalbe then jvm will create a seperate object for every nonSerialiable parent and shares its instance variable to the current Object.

4. To create an Object for NonSerializable parent, jvm will make a call to zero argument consturctor, if zero argument constructor is not avaialble then it would throw an exception called "InvalidClassException"

requirement: only one variable to be serialized (Externalization 1.1v)

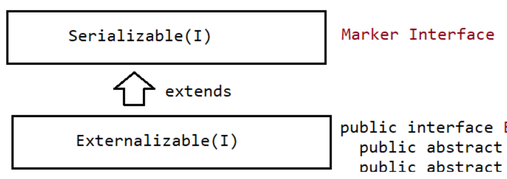


Performance is low

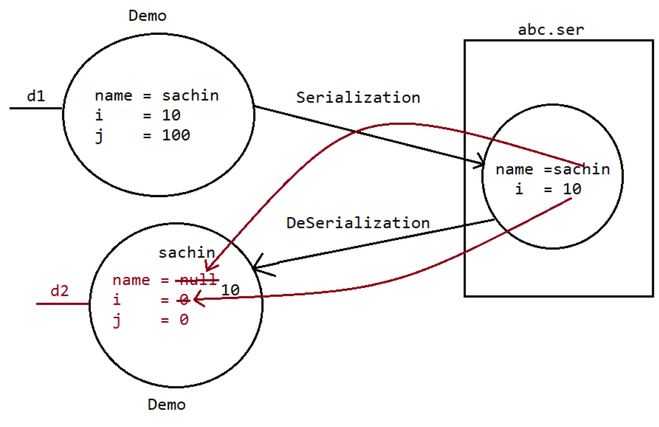
To avoid this we need to go for Externalization
Benefit
  a. full object we can save.
  b. part of object also can be saved based on our requirement.

Code should be taken care by programmer, henceforth Externalization is not appreciated by developers and it i not that much popular in java.

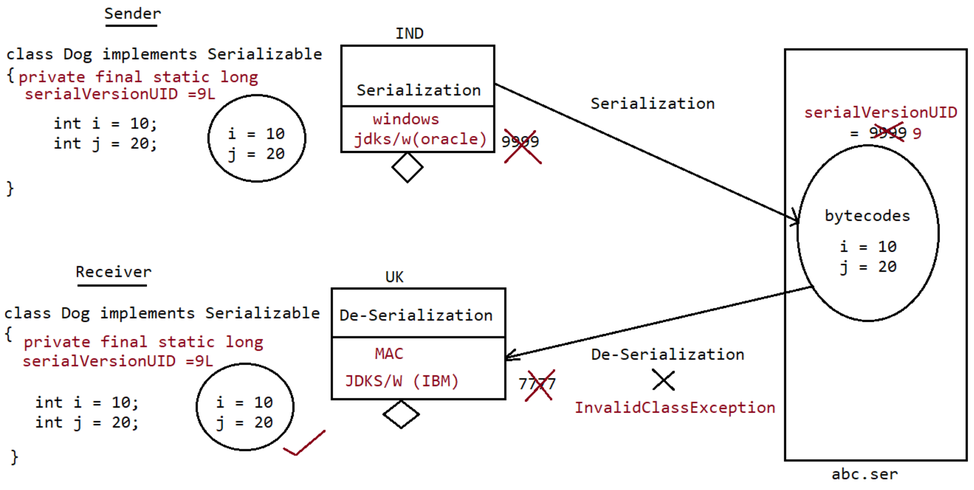
Note




```
public interface Externalizable extends Serializable {
  public abstract void writeExternal(ObjectOutput) throws IOException; //serialization
  public abstract void readExternal (ObjectInput) throws IOException,ClassNotFoundException;
```



1. Object gets created using zero argument consturctor if not available it would result in "InvalidClassException"



1. JVM while performing serialization will use "SerialVersionUID" with Object.

2. JVM while performing DeSerialization it wil serialVersionUID, if it matches only then De-Serialization will happen.