**Java.lang Package**

The following are some of important classes present in java.lang package.

1. Object class

2. String class

3. StringBuffer class

4. StringBuilder class (1.5 v)

5. Wrapper Classes

6. Autoboxing and Autounboxing(1.5 v)

**Note :**

· For writing any java program the most commonly required classes and interfaces are encapsulated in the separate package which is nothing but java.lang package.

· It is not required to import java.lang package in our program because it is available by default to every java program.

**The following is the list of all methods present in java.lang Object class :**

1. public String toString();

2. public native int hashCode();

3. public boolean equals(Object o);

4. protected native Object clone()throws CloneNotSupportedException;

5. public final Class getClass();

6. protected void finalize()throws Throwable;

7. public final void wait() throws InterruptedException;

8. public final native void wait()throws InterruptedException;

9. public final void wait(long ms,int ns)throws InterruptedException;

10. public final native void notify();

11. public final native void notifyAll();

**toString() Method:**

1. We can use this method to get string representation of an object.

2. Whenever we are try to print any object reference internally toString() method will be executed.

3.If our class doesn't contain toString() method then Object class toString() method will be executed.

class Student

{

String name;

int rollno;

Student(String name, int rollno)

{

this.name=name;

this.rollno=rollno;

}

public static void main(String args[]){

Student s1=new Student("saicharan",101);

Student s2=new Student("ashok",102);

System.out.println(s1);

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

System.out.println(s2);

}

}

Output:

Student@3e25a5

Student@3e25a5

Student@19821f

**hashCode() Method:**

1.For every object jvm will generate a unique number which is nothing but hashCode.

2. Jvm will using hashCode while saving objects into hashing related data structures like HashSet, HashMap, and Hashtable etc.

| **class Student**  **{**  **public int hashCode()**  **{**  **return 100;**  **}**  **}**  **It is *improper way* of overriding**  **hashCode() method because for every**  **object we are generating same**  **hashcode.** | **class Student**  **{**  **int rollno;**  **public int hashCode()**  **{**  **return rollno;**  **}**  **}**  **It is *proper way* of overriding hashcode()**  **method because for every object we are**  **generating a different hashcode.** |
| --- | --- |

**Equals() Method:**

1.We can use this method to check equivalence of two objects.

2. If our class doesn't contain .equals() method then object class .equals() method will be executed which is always meant for reference comparison[address comparison]. i.e., if two references pointing to the same object then only .equals() method returns true .

**class Student**

**{**

**String name;**

**int rollno;**

**Student(String name,int rollno)**

**{**

**this.name=name;**

**this.rollno=rollno;**

**}**

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

**Student s1=new Student("vijayabhaskar",101);**

**Student s2=new Student("bhaskar",102);**

**Student s3=new Student("vijayabhaskar",101);**

**Student s4=s1;**

**System.out.println(s1.equals(s2));**

**System.out.println(s1.equals(s3));**

**System.out.println(s1.equals(s4));**

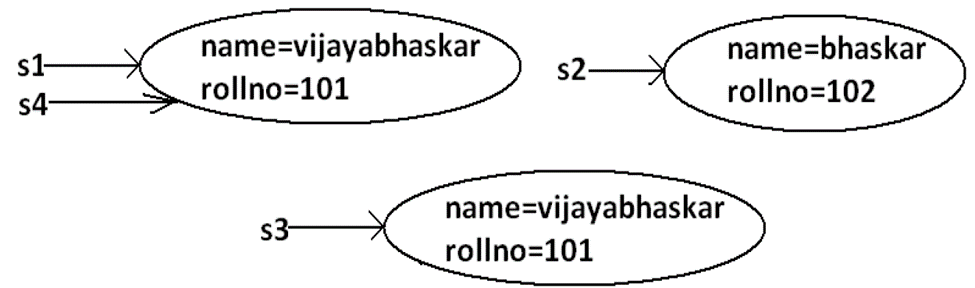
**}}**

**Output:**

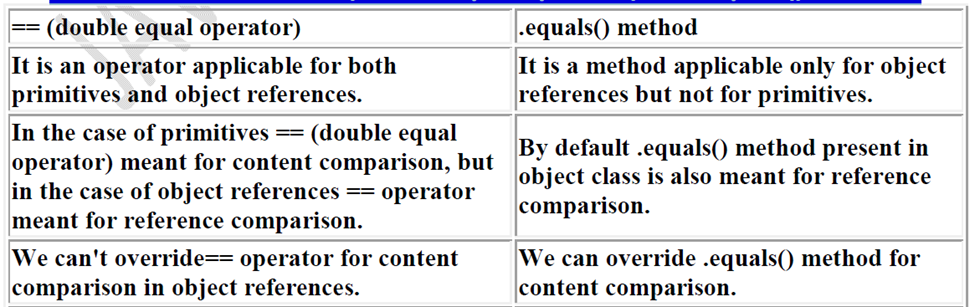
**False**

**False**

**True**



**Difference between == operator and .equals() method:**



**Clone() Method:**

1. The process of creating exactly duplicate object is called cloning.

2. The main objective of cloning is to maintain backup purposes.(i.e., if something goes wrong we can recover the situation by using backup copy.)

3. We can perform cloning by using clone() method of Object class.

**protected native object clone() throws CloneNotSupportedException;**

**Example:**

**class Test implements Cloneable**

**{**

**int i=10;**

**int j=20;**

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

**CloneNotSupportedException**

**{**

**Test t1=new Test();**

**Test t2=(Test)t1.clone();**

**t2.i=888;**

**t2.j=999;**

**System.out.println(t1.i+"---------------"+t1.j);**

**System.out.println(t2.i+"---------------"+t2.j);**

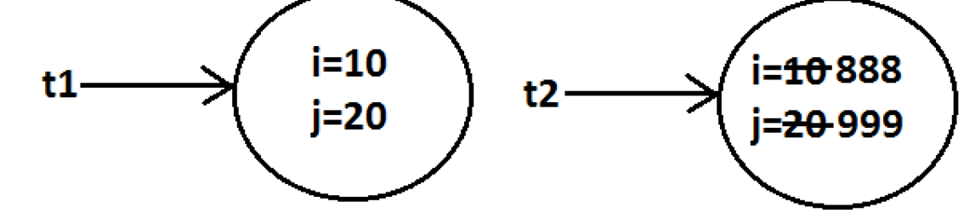
**}**

**}**

**Output:**

**10---------------20**

**888---------------999**

****

**getClass() Method:**

This method returns runtime class definition of an object.

**Example :**

**class Test implements Cloneable {**

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

**CloneNotSupportedException {**

**Object o=new String("ashok");**

**System.out.println("Runtime object type of o is**

**:"+o.getClass().getName());**

**}**

**}**

**Output:**

**Runtime object type of o is: java.lang. String**

**Ex : To print Connecton interface implemented vendor specific class name .**

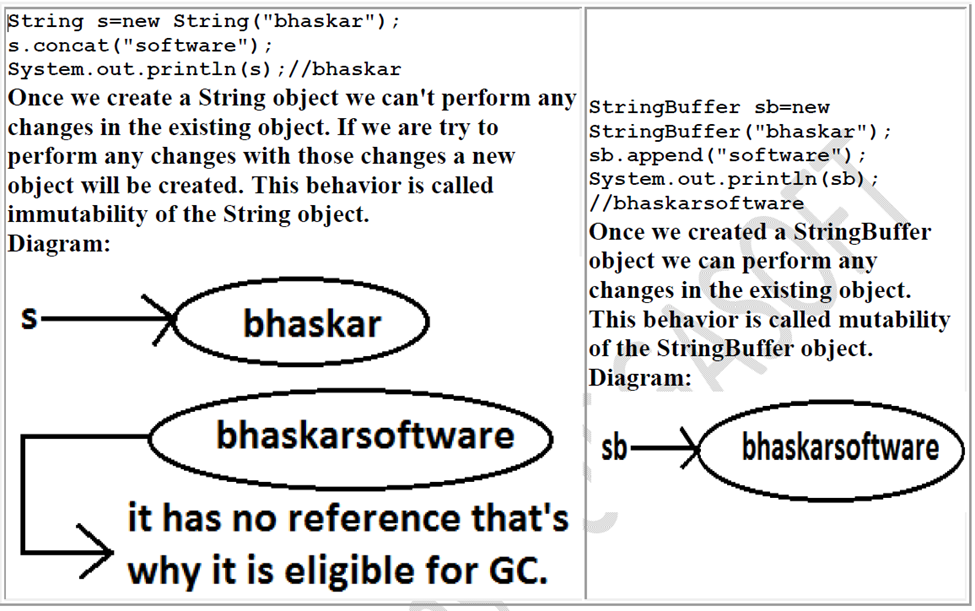
**System.out.println(con.getClass( ).getName( ) );**

**Finalize() Method:**

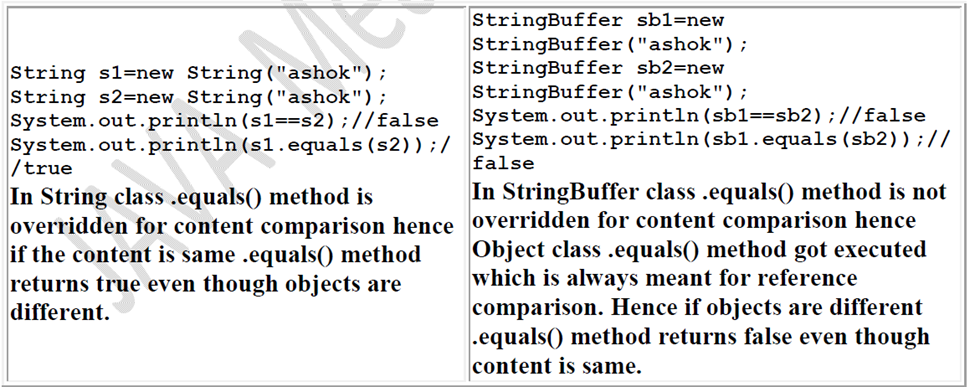
Just before destroying an object GC calls finalize( ) method to perform CleanUp activities .

**Java.lang.String class:**

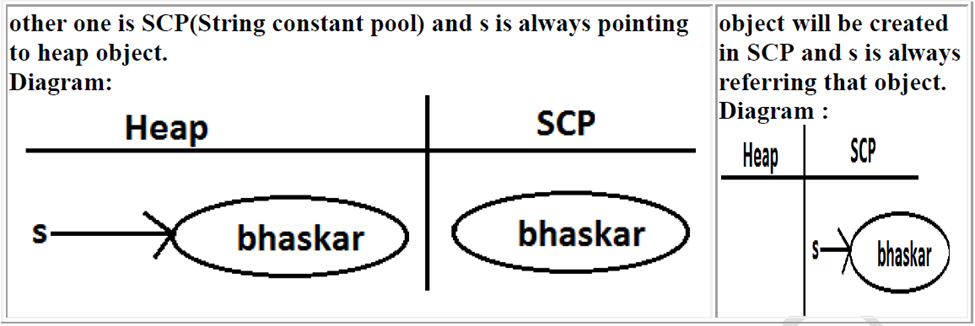
**Case 1:**

****

**Case 2:**

****

**Case 3:**

****

**Note :**

1. Object creation in SCP is always optional 1st JVM will check is any object already created with required content or not. If it is already available then it will reuse existing object instead of creating new object. If it is not already there then only a new object will be created. Hence there is no chance of existing 2 objects with same content on SCP that is duplicate objects are not allowed in SCP.

2. Garbage collector can't access SCP area hence even though object doesn't have any reference still that object is not eligible for GC if it is present in SCP.

3. All SCP objects will be destroyed at the time of JVM shutdown automatically.

**Example 1:**

**String s1=new String("bhaskar");**

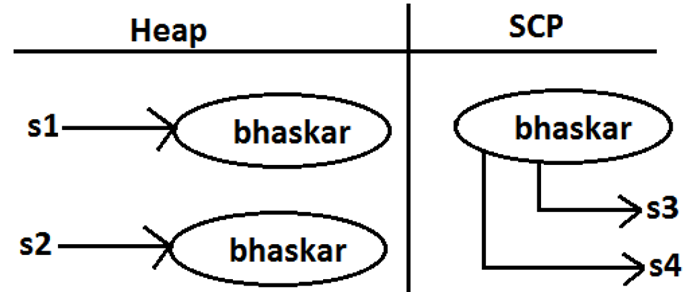
**String s2=new String("bhaskar");**

**String s3="bhaskar";**

**String s4="bhaskar";**

**Note :**

When ever we are using new operator compulsory a new object will be created on the Heap . There may be a chance of existing two objects with same content on the heap but there is no chance of existing two objects with same content on SCP . i.e., duplicate objects possible in the heap but not in SCP .



**String Constant Pool:**

1. In our program if any String object is required to use repeatedly then it is not recommended to create multiple object with same content it reduces performance of the system and effects memory utilization.

2. We can create only one copy and we can reuse the same object for every requirement. This approach improves performance and memory utilization we can achieve this by using "scp".

3. In SCP several references pointing to same object the main disadvantage in this approach is by using one reference if we are performing any change the remaining references will be impacted. To overcome this problem sun people

implemented immutability concept for String objects.

4. According to this once we creates a String object we can't perform any changes in the existing object if we are trying to perform any changes with those changes a new String object will be created hence immutability is the main disadvantage of scp.

**String Buffer:**

1.If the content will change frequently then never recommended to go for String object because for every change a new object will be created internally.

2. To handle this type of requirement we should go for StringBuffer concept.

3. The main advantage of StringBuffer over String is, all required changes will be performed in the existing object only instead of creating new object.(won't create new object)

Object

**Note :**

Every method present in StringBuffer is synchronized hence at a time only one thread is allowed to operate on StringBuffer object , it increases waiting time of the threads and creates performance problems , to overcome this problem we should go for StringBuilder.

**String Builder:**

1. Every method present in StringBuffer is declared as synchronized hence at a time only one thread is allowed to operate on the StringBuffer object due to this, waiting time of the threads will be increased and effects performance of the system.

2. To overcome this problem sun people introduced StringBuilder concept in 1.5v.

**String vs StringBuffer vs StringBuilder**

1. If the content is fixed and won't change frequently then we should go for String.

2. If the content will change frequently but Thread safety is required then we should go for StringBuffer.

3. If the content will change frequently and Thread safety is not required then we should go for StringBuilder.

**Wrapper Classes:**

The main objectives of wrapper classes are:

1. To wrap primitives into object form so that we can handle primitives also just like objects.

2. To define several utility functions which are required for the primitives.

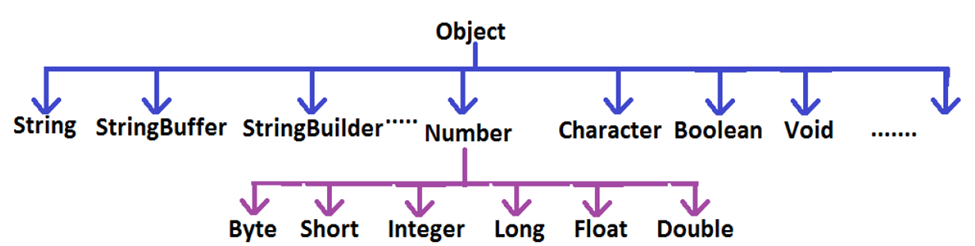
All most all wrapper classes define the following 2 constructors one can take

corresponding primitive as argument and the other can take String as argument.

**Example:**

**1) Integer i=new Integer(10);**

**2) Integer i=new Integer("10");**



**Note:**

1. String, StringBuffer, StringBuilder and all wrapper classes are final classes.

2. The wrapper classes which are not child class of Number are Boolean and Character.

3. The wrapper classes which are not direct class of Object are Byte, Short, Integer, Long, Float, Double.

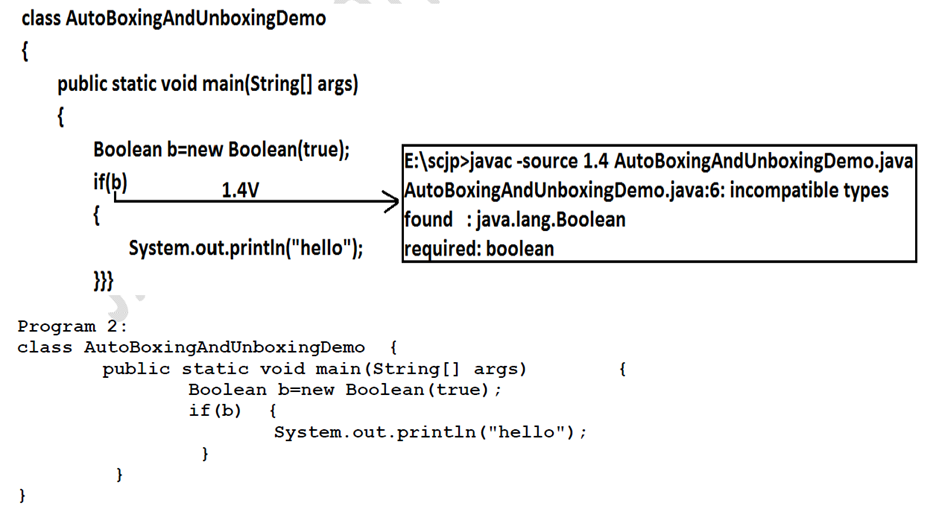
4. Sometimes we can consider Void is also as wrapper class.

5. In addition to String objects , all wrapper class objects also immutable in java.

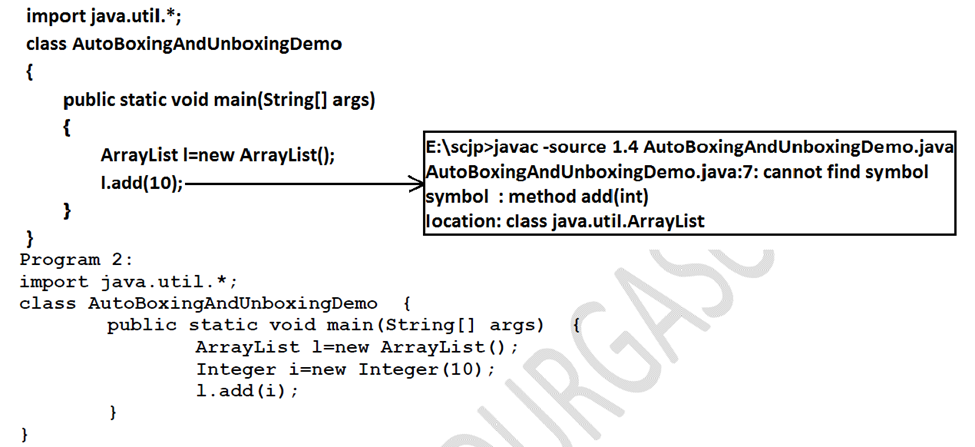
**Autoboxing and Auto-unboxing(1.5v):**

Until 1.4 version we can't provide wrapper object in the place of primitive and primitive in the place of wrapper object all the required conversions should be performed explicitly by the programmer.

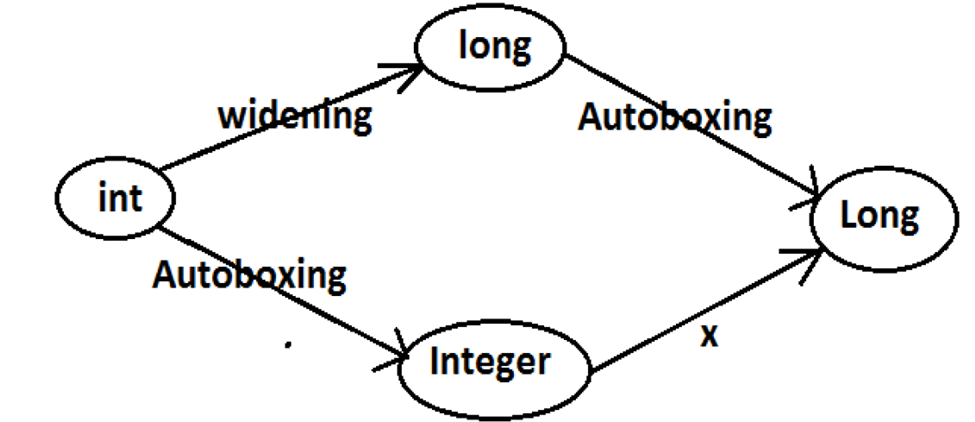
**Example 1:**

****

**Example 2:**

****

**Widening vs Autoboxing:**



**Java.io package**

**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:**

**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**

**}**

**}**

**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.**

**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.**

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

**FileWriter:**

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

**Constructors:**

**FileWriterfw=new FileWriter(String name);**

**FileWriterfw=new FileWriter(File f);**

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

**FileWriterfw=new FileWriter(String name,boolean append);**

**FileWriterfw=new FileWriter(File f,boolean append);**

**Methods:**

**1. write(intch);**

**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.**

**Example:**

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

**classFileWriterDemo**

**{**

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

**{**

**FileWriterfw=new FileWriter("cricket.txt",true);**

**fw.write(99);//adding a single character**

**fw.write("haran\nsoftware solutions");**

**fw.write("\n");**

**char[] ch={'a','b','c'};**

**fw.write(ch);**

**fw.write("\n");**

**fw.flush();**

**fw.close();**

**}**

**}**

**Output:**

**charan**

**software solutions**

**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.**

**File Reader:**

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

**Constructors:**

**FileReaderfr=new FileReader(String name);**

**FileReaderfr=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();**

**BufferedWriter:**

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

**Note: BufferedWriter never communicates directly with the file it should communicates**

**via some writer object.**

**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.(Answer)**

**BufferedReader:**

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

**Note: BufferedReadercan 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.**

**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);**

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

**Methods:**

**1. write(intch);**

**2. write (char[] ch);**

**3. write(String s);**

**4. flush();**

**5. close();**

**Example:**

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

**classPrintWriterDemo {**

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

**{**

**FileWriterfw=new FileWriter("cricket.txt");**

**PrintWriter out=new PrintWriter(fw);**

**out.write(100);**

**out.println(100);**

**out.println(true);**

**out.println('c');**

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

**out.flush();**

**out.close();**

**}**

**}**

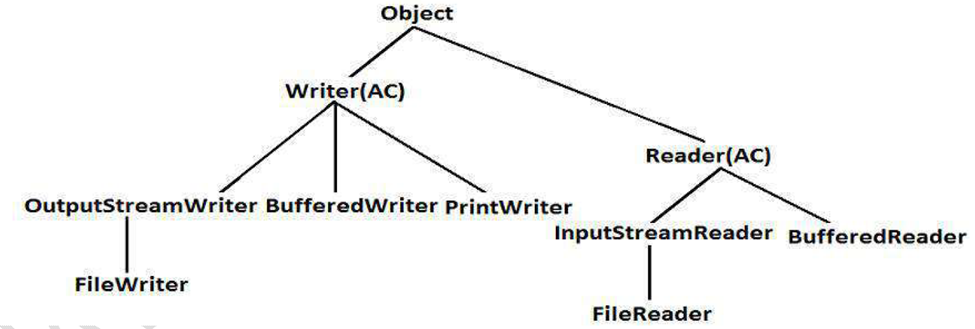
**Output:**

**d100**

**true**

**c**

**SaiCharan**

****

**Serialization**

**Serialization:**

1. The process of saving (or) writing state of an object to a file is called serialization

2. But strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.

3. By using FileOutputStream and ObjectOutputStream classes we can achieve serialization process.

**De-Serialization:**

1. The process of reading state of an object from a file is called DeSerialization

2. But strictly speaking it is the process of converting an object from file supported form (or) network supported form to java supported form.

3. By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.

**Example 1:**

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

**class Dog implements Serializable**

**{**

**int i=10;**

**int j=20;**

**}**

**classSerializableDemo**

**{**

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

**Dog d1=new Dog();**

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

**FileOutputStreamfos=new FileOutputStream("abc.ser");**

**ObjectOutputStreamoos=new ObjectOutputStream(fos);**

**oos.writeObject(d1);**

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

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

**FileInputStreamfis=new FileInputStream("abc.ser");**

**ObjectInputStreamois=new ObjectInputStream(fis);**

**Dog d2=(Dog)ois.readObject();**

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

**System.out.println(d2.i+"................"+d2.j);**

**}**

**}**

**Output:**

**Serialization started**

**Serialization ended**

**Deserialization started**

**Deserialization ended**

**10................20**

Note:

1. We can perform Serialization only for Serializable objects.

2. An object is said to be Serializable if and only if the corresponding class implements Serializable interface.

3. Serializable interface present in java.io package and does not contain any 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".