```
public static void methodOne (ArrayList l){
        l.add(10);
        l.add("dhoni");
        l.add(true);
}
```

Conclusions :

Generics concept is applicable only at compile time
at runtime there is not such type of concept.
At the time of compilation, as the last step
generics concept is removed, hence for jvm generics
syntax won't be available.
Hence the following declarations are equal

```
ArrayList l=new ArrayList<String>();
ArrayList l=new ArrayList<Integer>();
ArrayList l=new ArrayList<Double>();
```

All are equal at runtime, becoz compiler will
remove these generics syntax.

```
ArrayList l=new ArrayList();
```

Example :

```
import java.util.*;
class Test {
    public static void main (String[] args){
        Array list l=new Array List<String>();
```

```
    l.add(10);
    l.add(10.5);
    l.add(true);
  System.out.println(l); // [10, 10.5, true]
}
```

Example 2:

```
import java.util.*;
class Test {
    public void methodOne (ArrayList<String> l) {
    public void methodone (ArrayList<Integer> l) {
}
```

CE: duplicate methods found.

Behind the scenes by the Compiler:

1. Compiler will scan the code
2. check the argument type
3. If generics found in the argument type remove the generics syntax
4. Compiler will again check the syntax.

Example 3:

The following 2 declarations are equal.

```
ArrayList<String> l1 = new ArrayList();
ArrayList<String> l2 = new ArrayList<String>();
```

for these ArrayList Objects we can add only String type of objects.

```
l1.add("A"); // Valid
l1.add(10); // Invalid
```

Comparable Vs Comparator.

public Treeset();

1 => when we use the above constructor jvm will internally use comparable interface method to sort the objects based on default natural sorting order.

what is Comparable interface?

it is a functional interface present in java.lang pa

This interface is internally used by Treeset object during sorting process of the Object.

```
@FunctionalInterface.
public interface java.lang.comparable<T> {
    public abstract int comparable(T);
}
```

eg#1

```
import java.util.*;
class Test
{
    public static void main (String[] args)
        // Sorting of objects will happen based on
default natural sorting order Treeset ts = new
                                Treeset();
```

```
ts.add ("A");
ts.add ("z");
ts.add ("C");
ts.add ("B");
ts.add (null); // Null pointer Exception.
ts.add (10); //Class cast Exception.
System.out.println (ts); // [A, B, C, Z]
}
}
```

Note:
If we are keeping the data inside Treeset object, then the data should be:

a. Homogenous ⟹ because it uses compareTo() to sort the Object.

b. The Object should compulsory implements an interface called "Comparable".

if we fail to do so, it would result in "ClassCast Exception".

Eg #2

```
import java.util.*;
class Test {
    public static void main (String[] args)
    {
        //sorting of objects will happen based on default
natural sorting order
        Treeset ts = new Treeset();
```

```
ts.add(new StringBuffer("A"));
ts.add(new StringBuffer("Z"));
ts.add(new StringBuffer("L"));
ts.add(new StringBuffer("B"));
System.out.println(ts); // ClassCastException
}
}
```

Note:- All wrapper classes and String class has implemented "Comparable" interface. StringBuffer class has not implemented Comparable interface, so the above program would result in "ClassCastException".

21/12/22 Wednesday:-

Note: If we are Depending on Default Natural Sorting order Compulsory objects should be Homogeneous and Comparable.
Otherwise we will get RE: ClassCastException.
An object is said to be comparable if and only if corresponding class implements Comparable Interface
All wrapper classes, String class Already Implements Comparable Interface. But StringBuffer class doesn't Implement Comparable Interface.

@functionalInterface

```
public interface Comparable<T>{
    public abstract int compareTo(T);
}
```

obj1.compareTo(obj2)

- → returns -ve value if obj1 has to come before obj2
- → returns +ve value, If obj1 has to come after obj2
- → returns 0 value, If both obj1 and obj2 are equal

```
import java.util.*;
Class Test {
    public static void main (String[] args){
        Treeset ts = new Treeset();
        ts.add ("A");
        ts.add ("Z");
        ts.add ("L");
        ts.add ("K");
        ts.add ("B");
        System.out.println (ts);
    }
}
```

<u>Comparable (I);</u>

Comparable Interface present in Java.lang package and it contains only one method CompareTo().

obj1.compareTo(obj2)

Returns

Reutun -ve if and only if obj1 has to come Before obj

Returns +ve if and only if obj1 has to come After obj

Return 0 if and only if obj1 and obj2 are Equal.

eg#1

```
System.out.println("A".compareTo("z"));//-ve v
System.out.println("z".compareTo("k"));//+value
System.out.println("z".compareTo("z"));//zero
System.out.println("z".compareTo(null));//NPE
```

whenever we are Depending on Default Natural sorting order and if we are trying to Insert Elements then Internally JVM will Call CompareTo() to identify sorting order.

```
Treeset t=new Treeset();
t.add("k");
t.add("z");  "z".compareTo("k");
t.add("A");  "A".compareTo("k");
t.add("A");  "A".compareTo("A");
System.out.println(t);// [A, k, z] => sorting is ascend
```

Note for string default natural sorting order is "Ascending order".
for Number default natural sorting order is "Ascending order".

Comparator (I)

Note: If we are NOT satisfied with Default Natural sorting order OR if Default Natural sorting order is Not Already Available then we can Define our own sorting by using Comparator object.

```
public interface java.util.Comparator<T> {
    public abstract int compare (T, T);
    public abstract boolean equals (Java.lang.Object);
}
```

Comparator (I):

This Interface present in java.util package.

methods: It contains 2 methods compare() and equals()

public int compare (object obj1, object obj2);

Returns -ve if and only if obj1 has to come Before obj2
& +ve if and only if obj1 has to come After obj2
Return 0 if and only if obj1 and obj2 are Equal.

public boolean equals (Object o);

Whenever we are implementing Comparator Int compulsory we should provide Implementation for compare ().

Implementing equals() is optional because it is Already Available to our class from object class through Inheritance.

```
import java.util.*;
Class TreesetDemo {
public static void main (String [] args ){
    Treeset t=new Treeset (new mycomparator ())
                                        //line-1
    t.add (10);
    t.add (0);
    t.add (15);
    t.add (5);
    t.add (20))
    t.add (20))
System.out.println (t);// [20, 15, 10, 5, 0]
}}
Class Mycomparator implements Comparator {
    public int Compare (Object obj1, Object obj2
        Integer i1 = (Integer) obj1;
        Integer i2 = (Integer) obj2;
```

```
    if (i1 < i2)
        return +1;
    else if (i1 > i2)
        return -1;
    else
        return 0;
}
```

At line 1 if we are Not passing Comparator object as an Argument then Internally JVM will call compareTo(), which is meant for Default Natural Sorting order (Ascending order).

In this case output is [0, 5, 10, 15, 20].

At line 1 if we are passing Comparator object then JVM will call compare() Instead of compareTo(). which is meant for Customised sorting (can be Ascending / Descending order).

In this case the output is [20, 15, 10, 5, 0]

<u>Various possible Implementations of compare():</u>

```
public int compare (Object obj1, Object obj2) {
    Integer I1 = (Integer) obj1;
    Integer I2 = (Integer) obj2;
```

return I1.compareTo(I2);

return -I1.compare(I2);

return I2.compareTo(I1);

return -I2.compareTo(I1);

return +1;

return -1;

return 0;

}

<u>Output</u>

1. Ascending order
2. Descending order
3. Descending order
4. Ascending order
5. Insertion order
6. reverse of insertion order
7. only first element will be inserted.

(*) Write a program to Insert String Objects into the Treeset where the Sorting order is of Reverse of Alphabetical order:

) import java.util.;

Class TreeSetDemo {

public static void main (String[] args) {

Treeset t = new Treeset (new MyComparator());

```
        t.add ("sachin");
        t.add ("pointing");
        t.add ("Sangakara");
        t.add ("fleming");
        t.add ("kara");
        System.out.println (t);
    }
}
class  my comparator implements Comparator {
    public int compare(object obj1, objectobj2){
        String s1 = obj1.toString();
        String s2 = (string)obj2;
    return s2.compareTo(s1);
    //return -s1.compareTo(s2);
    }
}
```

Write a program to Insert String Buffer Objects into the Treeset where sorting order is Alphabetical order:

```
import java.util.*;
  class TreeSetDemo {
    public static void main(String[]args){
  Treeset t=new Treeset(new MyComparator1());
    t.add (new StringBuffer("A"));
    t.add (new StringBuffer ("z"));
    t.add (new StringBuffer ("K"));
```

```
t.add(new StringBuffer("C"));
System.out.println(t);
}
}

class MyComparator1 implements Comparator {
    public int compare(Object obj1, Object obj2){
        String s1 = obj1.toString();
        String s2 = obj2.toString();
        return s1.compareTo(s2); //[A, K, L, Z]
}}
```

Write a program to Insert String and StringBuffer objects into the Treeset where sorting order is Increasing length order.

If 2 objects having same length the consider their Alphabetical order.

Eg: A, ABC, AA, XX, ABCE, A

<u>output</u> A, AA, XX, ABC, ABCE

```
import java.util.*;
class TreeSetDemo {
public static void main(String[] args){
    Treeset t = new Treeset(new MyComparator());
    t.add("A");
    t.add(new StringBuffer("ABC"));
```

```
        t.add (new StringBuffer("AA"));
        t.add ("XX");
        t.add ("ABCE");
        t.add ("A");
        System.out.println (t);
    }
}
class myComparator implements Comparator {
    public int compare (Object obj1, Object obj2) {
        String s1 = obj1.toString();
        String s2 = obj2.toString();
        int i1 = s1.length();
        int i2 = s2.length();
        if (i1 < i2) return -1;
        else if (i1 > i2) return 1;
        else return s1.compareTo(s2);
    }
}
```

<u>Note</u> :

if we are use Treeset(), then the condition is

a. Object should be homogenous.

b. object should be comparable (class should implement comparable(I)).

If we use Treeset (Comparator c) then what is the condition?

a. object need not be homogenous

b. object need not implement comparable.

when to go for comparable interface and when to go comparator interface?

Ans predefined Comparable classes like String, Wrapper Class ⇒ Defaultt natural sorting is already availble

if we are not happy with natural sorting order, we want customization then we need to go for "Comparator (I)".

for predefined Non-comparable class like StringBuffer ⇒ comparator (I) is used for both natural sorting order and customized sorting order.

for user defined class like Employee, Student ⇒ Developer if he comes up with own logic of sorting, then he should implement Comparable (I) and give it as a ready made logic.

~~Ddi~~

```
class Employee implements Comparable
{
    int id;
    String name;
    int age;
    public int compareTo(Object obj){
        //sorting is done based on "id"
        ;;//
}
}
```

If the developer who is using Employee class if he is not interested with sorting based on "id" given by the api, then he can use "comparator" when we go for Comparable and when go for comparator:

Comparable vs Comparator:

→ for predefined Comparable classes (Like String) Default Natural sorting order is Already Available if we are Not satisfied with that we can Define our own sorting by Comparator Object

→ for predefine Non-Comparable Classes (like StringBuffer) Default Natural sorting order is Not Already Available.

If we want to Define our own sorting we can use Comparator Object.

→ for our own Classes (like Employee) the person who is writing Employee class he is Responsible to Define Default Natural. Sorting order by implementing Comparable Interface.

→ the person who is using our own Class if he is Not satisfied with Default Natural sorting order he can define his own Sorting by using Comparator Object.

If he Satisfied with DNSO then he can use Directly our Class.

Write a program to Insert Employee Objects into the Treeset where DNSO is based on Ascending order of Employee id & customized order is Based on Alphabetical order of Names

DNSO → Default Natural Sorting order

```
Impost java.util.*;
Class Employee implements Comparable {
    String name;
    int eid;
    Employee (String name, int eid) {
        this.name = name;
        this.eid = eid;
    }
public String toString () {
    return name + " ---- " + eid; }
public int compareTo (object obj) {
    int eid1 = this.eid;
    Employee e = (Employee) obj;
  int eid2 = e.eid;
  if (eid1 < eid2) return -1;
  else if (eid1 > eid2) return 1;
  else return 0;
}
```

```
Class Test {
    public static void main (String[] args ) {
  employee e1 = new Employee ("sachin", 10);
  employee e2 = new Employee ("pointing", 14);
  employee e3 = new Employee ("lara", 9);
  employee e4 = new Employee ("flin off", 17);
  Employee e5 = new Employee ("anwar", 23);
  Treeset t = new Treeset ();
    t.add (e1);
    t.add (e2);
    t.add (e3);
    t.add (e4);
    t.add (e5);
    System.out.println (t);

  Treeset t1 = new Treeset (new MyComparator ());
    t1.add (e1);
    t1.add (e2);
    t1.add (e3);
    t1.add (e4);
    t1.add (e5);
    System.out.println (t1); } }
```

```
class myComparator implements Comparator {
    public int compare(Object obj1, Object obj2){
        Employee e1 = (Employee) obj1
        Employee e2 = (Employee) obj2
        String s1 = e1.name;
        String s2 = e2.name;
        return s1.compareTo(s2);
    }
}
```

Comparision of Comparable and Comparator.

1) <u>Comparable(I)</u>

Present in java.lang package

It is meant for Default Natural Sorting Order

Defines only one method compareTo()

All wrapper classes and String class implements Comparable Interface.

<u>Comparator(I)</u>

Present in java.util package.

It is meant for Customized Sorting Order.

Defines 2 methods compare() and equals()

the only implemented classes of Comparator are collator and RuleBasedCollator.

22/12/22

functional interfaces

→ If an interface contains only one abstract method then such interfaces are called as "functional interface".

```
public interface java.util.function.predicate<T> {
    public abstract boolean test (T);

  //default methods
  public java.util.function.predicate<T> and (java.util.function.predicate<? super T>);
  public java.util.function.predicate<T> negate();
  public java.util.function.predicate<T> or(java.util.function.predicate<? super T>);
  //static method
  public static <T> java.util.function.predicate<T> isEqual (java.lang.Object);
}
```

→ Usage of predicate

```
Class my predicate implements predicate<Integer>
{
  @oreride
    public boolean test (Integer i) {
      if (i>10)
        return true;
    } else return false
  }
```

Instead of writing a seprate class, we can write lambda expression as shown below

```
class mypredicate implements predicate<string>
{
  @overide
    public boolean test (string name){
        if (name.length()>=3)
          return true
        else
          return false
    }
}

class Test {
    public static void main (string[] args){
      predicate<string> p= name -> name.length()>=
      System.out.println(p.test("pwc"));//true
      System.out.println(p.test("cs"));//false
    }
}
```

default methods available as utility methods for developer

```
public default predicate<T> and (predicate p
public default predicate<T> negate();
public default predicate<T> or (predicate p);
```

```
import java.util.function.*;
public class Test {
    public static void main (String [] args) {
        int [] arr = {0, 5, 10, 15, 20, 25, 30};
predicate <Integer> p1 = i -> i>10;
System.out.println ("Elements greater than 10 are
:: ");
m1 (p1, arr);
predicate <Integer> p2 = i -> i%2 == 0;
System.out.println ("Elements which are even n
:: ");
m1 (p2, arr);
System.out.println ("Elements which are greater 10
Should be even no");
m1 (p1.or (p2), arr);
System.out.println ("the elements which are not
even are :: ");
m1 (p2.negate(), arr);
}
public static void m1 (predicate <Integer> p,
int [] x) {
```

```
for (int ele : x)
        if (~~int ele~~ p.test(ele)) //ele -> ele>10
        System.out.println(ele);
    }
}
```

<u>function (I)</u>

T → Input Type

R → return Type

```
public interface java.util.function.Function<T,R>
{
    // 1 abstract method.
        public abstract R apply(T);
    //default methods

public <V> java.util.function.Function<V, R>
Compose (java.util.function.Function<? super V,?
extends T>);
    public <V> java.util.function.Function<T, V>
and then (java.util.function.Function<? super R,
? extends V>);
//static method.
    public static <T> java.util.function.Function<T, T> identi
()
}
```

writing a code using Implementation class

```
class myfunction implements function<String,Integer>
{
 @Override
   public Integer apply (String name) {
        return name.length();
   }
}

public class Test {
    public static void main (String[] args) {
    function<String,Integer> f= new myfunction()
    int output = f.apply ("Sachin");
    System.out.println (output);
    }
}
```

Note :- when to go for predicate and when to go for function?

predicate → To implement some conditional checks we should go for predicate

function → To perform some operation and to return some result we should go for function.

Method Reference (::) and Constructor reference (::)

:: ⇒ scope resolution operator.

Syntax for method reference

1. Static method
   ClassName :: methodName

2. instance method
   object :: methodName

eg#1.

```
public class Test {
    public static void m1() {
        for (int i=1; i<=10; i++)
        {
            System.out.println("child thread");
        }
    }
    public static void main (String[] args) throws Excep
    //using method reference binded the method call
    run () of interface Runnable
        Runnable r= Test :: m1;
        Thread t=new Thread (r);
        t.start();
        for (int i=1; i<=10; i++)
        {
            System.out.println ("main thread");
            Thread.sleep (1000);
        }
}
```

23/12/22

```
class Sample {
    private String s;
    Sample (String s) {
        this.s=s;
        System.out.println ("Constructor executed...."+s);
    }
}
@functionalInterface.
interface Interf
{ public Sample get (String s);
}
public class Test {
    public static void main (String[] args) {
        Interf i=s -> new Sample (s);
        i.get ("from lambda expression ...");
        System.out.println();
        // ~~constructor.out.println()~~
        //constructor reference.
        Interf i1 = Sample::new;
        i1.get ("from Constructor reference ....");
}
}
```

Example of Method reference.

```
@FunctionalInterface
interface Interf
{
  public void m1 (int i);
}
public class Test {
    // logic coded by other developer
    public void m2 (int i) {
        System.out.println (i*i);
        System.out.println ("logic coming from
method reference ...");
}
public static void main (String[] args){
        Interf i = x -> System.out.println (x);
        i.m1 (10);
        System.out.println();

// method reference (binding the body of m2() to
abstract method m1)
        Interf i1 = new Test ()::m2;
        i1.m1 (20);
  }
}
```

Eg: To demonstrate the usage of forEach () to print the elements of ArrayList

```
import java.util.*;
import java.util.function.*;
// public void forEach (java.util.function.Consumer <? super E>)
// public abstract void accept (T t)

class MyConsumer implements Consumer <String>
{
  @Override
    public void accept (String name){
        System.out.println ("accept method got called...");
        System.out.println (name);
    }
}

public class Test {
    public static void main (String[] args){
        ArrayList <String> names = new ArrayList <String>();
        names.add ("sachin");
        names.add ("dhoni");
        names.add ("kohli");
        names.add ("dravid");
    // Traditional approach
    Consumer <String> consumer = new MyConsumer();
        names.forEach(consumer);
        System.out.println ();
```

// lambda Expression

names.forEach (name -> System.out.println(name

System.out.println ();

//method reference

names.forEach (System.out::println);

}

}.

Stream API

Stream ----> Channel through which there is a free flow movement of data.

Streams

To process objects of the collection, in 1.8 version, streams concept introduced.

Ans what is the differences between java.util.streams and java.io streams?

java.util streams meant for processing objects from the collection. Ie, it represents a stream of objects from the collection. but java.io streams meant for processing binary and character data with respect to file. i.e it respresents stream of binary data or character data from the file hence java.io streams and java.util streams both are different.

what is the difference b/w Collection and stream?

-> If we want to represent a group of individual objects as a single entity then we should go for collection.

-> If we want to process a group of objects from the collection then we should go for streams.

-> we can create a stream object to the collection by using stream() method of collection interface. stream() method is a default method added to the collection in 1.8 version.

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main(String[] args) {
        ArrayList<Integer> al=new ArrayList<Integer>();
        al.add(0);
        al.add(5);
        al.add(10);
        al.add(15);
        al.add(20);
        al.add(25);
    System.out.println(al); // [0,5,10,15,20,25]
    // 从 jdk.1.7y
    ArrayList<Integer> evenList=new ArrayList<Integer>();
```

```
for (Integer i1 : al)
    if (i1 % 2 == 0)
        evenList.add(i1)
System.out.println(evenList); // [0, 10, 20]

// from JDK 1.8V we use streams
  //1. configuration ==> al.streams
  //2. processing ==> filter(i -> i % 2 == 0).collect
                                    (collectors
                                    .toList())
List<Integer> streamList = al.stream().filter(i -> i %2 ==
                    collect(collectors.toList());
                System.out.println(streamList);
            streamList.forEach(System.out::println);
}
}
```

eg#2 :

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void (String[] args ){
    ArrayList<Integer> al = new ArrayList<Integer>
        al.add(0);
        al.add(5);
        al.add(10);
        al.add(15);
        al.add(20);
        al.add(25);
```

```
System.out.println(al);

// till JDK 1.7V
ArrayList<Integer> doubleList = new ArrayList<
Integer>();
for (Integer i : al)
        doubleList.add(i*2);

System.out.println(doubleList);
// from JDK 1.8V
// map -> for every object, if a new object has to
be created then go for map.

    List<Integer> streamList = al.stream().map(obj-
> obj*2).collect(Collectors.toList());
        System.out.println(streamList);
        streamList.forEach(i -> System.out.println(i));
        System.out.println();
        streamList.forEach(System.out::println);
}
}
```

=> Stream is an interface present in java.util.stream.
Once we got the stream, by using that we can process objects of that collection.
we can process the objects in the following 2 phases
1. Configuration
2. processing

1) <u>Configuration</u>: we can configure either by using filter mechanism or by using map mechanism

<u>filtering</u>: we can configure a filter to filter ~~mechanism or by using map mechanism~~ elements from the collection based on some boolean condition by using.

filter() method of stream interface.

public stream filter (predicate <T> t)

here (predicate <T> t) can be a boolean value function / lambda expression

<u>Ex</u>:- stream s = c.stream();

stream s1 = s.filter(i -> i % 2 == 0);

Here to filter elements of collection based on some Boolean condition we should go for filter() meth

<u>mapping</u>: if we want to create a sepreate new object for every object present in the collection based on our requirement then we should go for map() method of stream interface.

public stream map (function f);

It can be lambda expression also

ex:- Stream s = c.stream();

stream s1 = s.map(i -> i+10);

once we performed configuration we can process objects by using several methods

2) processing

processing by collect() method

processing by count() method

processing by sorted() method

processing by min() and max() methods

for Each() method

toArray() method

stream.of() method.

eg#1 :-

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main (String[] args){
ArrayList<String> names = new ArrayList<String>();
names.add("sachin");
names.add("saurav");
names.add("dhoni");
names.add("dravind");
names.add("kohli");
names.add("raina");
```

```
System.out.println (names);

List < String > result = names.stream().filter (name
-> name.length ()>5).collect (Collectors.toList());
System.out.println (result.size());

long count = names.stream().filter (name -> name
length ()>5).count();
System.out.println ("the no of objects whose strin
length >5 is :: " + (ount);
}
}
import java.util.*;
import java.util.stream.*;
//comparable (predefined API for natural sorting
order) -> Compare To (object obj)
//comparable (for userdefined class for customized
sorting order) ->
compare (obj1, obj2)
public class Test {
public static void main (String [] args){
ArrayList < Integer > al = new ArrayList < Integer
al.add (10);
al.add (0);
al.add (15);
al.add (5);
al.add (20);
```

```
System.out.println("Before Sorting :: "+al);
  //using stream api
List<Integer> resultList = al.stream().sorted().
collect(Collectors.toList());
    System.out.println("After Sorting :: "+resultList);

  List<Integer> customizedResult = al.stream().
sorted((i1, i2) -> i2.compareTo(i1)).collect(Collectors.
                                    toList());
    System.out.println("After Sorting :: "+customized
                                    Result);

  }
}
import java.util.*;
import java.util.stream.*;
//comparable (predefined API for natural sorting order)
-> compareTo(object obj)
//comparable (for user defined class for customized
sorting order) -> compare(obj1, obj2)
  public class Test {
    public static void main(String[] args) {
      ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(10);
        al.add(0);
        al.add(15);
        al.add(5);
```

```
al.add(20);
System.out.println("Array List is ::"+al);
Object[] objArr = al.stream().toArray();
    for (object obj: objArr)
        System.out.println(obj)
System.out.println();
Integer[] objArr1 = al.stream().toArray(Integer
                                        [] :: new)
    for (Integer obj1: objArr1)
        System.out.println(obj1);
}}
```

Q8)

```
import java.util.*;
import java.util.stream.*;
public class Test{
    public static void main (String[] args){
        //stream API ==> Collections (group of
objects)
    Stream s=stream.of(9,99,999,9999,99999);
    s.forEach(System.out::println);
    System.out.println();
    Double[] d={10.0, 10.1, 10.2, 10.3, 10.4};
    Stream s1 = Stream.of(d);
}} s1.forEach(system.out::println);
```

collect()

This method collects the elements from the stream and adding to the specified to the collection indicated (specified) by argument.

eg#1

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main (String[] args) {
        ArrayList<String> names = new ArrayList<String>();
        names.add ("sachin");
        names.add ("saurav");
        names.add ("dhoni");
        names.add ("yuvi");
    System.out.println (names); // [sachin, saurav, dhoni, yuvi
//predicate (I)
// public abstract boolean test (T);
    List<String> result = names.stream().filter(name
    name.length()>5).collect (Collectors.toList());
    System.out.println (result);
//function (I) <T, R>
    //public abstract R apply(T);
    List<String> mapResult = names.stream().ma
                                (name -> name.
                                toUpperCase()).collect(Collectors.
                                toList());
    System.out.println (mapResult);
}}
```

Count

This method returns number of elements present in the stream pu

```
public long count()
```

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main (String[] args){
        ArrayList<String> names = new ArrayList<String>();
        names.add("sachin");
        names.add("saurav");
        names.add("dhoni");
        names.add("yuvi");
    System.out.println(names); // [sachin, saurav, dhoni, yuvi]
    long count = names.stream().filter(name -> name.length()>5).count();
        System.out.println(count);
    }
}
```

(III) processing by sorted() method

If we sort the elements present inside stream then we should go for sorted() method.

the sorting can either default natural sorting order or customized sorting order specified by comparator

sorted() - default natural sorting order

sorted(comparator c) - Customized sorting order.

```
import java.util.*;
import java.util.stream.*;

public class Test {
    public static void main (String[] args){
    ArrayList <Integer> al = new ArrayList<Integer>
                                                ();
        al.add (10);
        al.add (20);
        al.add (0);
        al.add (5);
        al.add (25);
        al.add (15);
        System.out.println(al);
    List <Integer> result = al.stream().sorted().
    Collect (collectors.toList());
        System.out.println (result);
    List <Integer> customized Result = al.stream().
    Sorted ((i1, i2) -> - i1.compareTo (i2)).collect
        (collectors.toList());
    System.out.println (customized Result);
}

}
```

IV processing by min () and max () methods.

min (comparator c) returns minimum value according to specified comparator.

max (comparator c) returns maximum value according to specified comparator.

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main (String[] args){
        ArrayList<Integer> al = new ArrayList<Integer>
        al.add (10);
        al.add (20);
        al.add (0);
        al.add (5);
        al.add (25);
        al.add (15);
        System.out.println (al);
        Integer minValue = al.stream().min((i1,i2
i1. compareTo (i2)).get();
        System.out.println (minValue);
```

```
Integer maxValue = al.stream().max((i1,i2)->
i1.compareTo(i2)).get();
        System.out.println(maxValue);
    }
}
```

V. forEach() method

This method will not return anything.
This method will take lambda Expression as argument and apply that lambda expression for each element present in the stream

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main (string[] args) {
        ArrayList<string> names = new ArrayList<string>();
        names.add("AAA");
        names.add("BBB");
        names.add("CCC");
        names.add("DDD");
    names.stream().forEach(name -> System.out.println(name));
    names.stream().forEach(System.out::println);
```

VI. toArray() method

we can use toArray() method to copy elements present in the stream into specified array

```
impot java.util.*;
impot java.util.stream.*;

public class Test {
    public static void main (String [] args) {
        ArrayList<Integer> al = new ArrayList<
Integer>();
        al.add (0);
        al.add (10);
        al.add (5);
        al.add (20);
        al.add (15);
        System.out.println (al);

Integer [] array = al.stream().toArray(Integer[
:: new);
    for (Integer element : array)
        System.out.println (element);
    }
}
```

VII. stream.of() method.

we can also apply a stream for group of values and for arrays

Ex: Stream s = stream.of(99,999,9999,99999);
s.forEach(System.out::println);

Double[] d = {10.0, 10.1, 10.2, 10.3};
Stream s1 = stream.of(d);
s1.forEach(System.out::println);

File Handling

02/0/23

What is persistency?

It is a mechanism of saving 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. File writer
3. File Reader
4. Buffered writer
5. Buffered Reader..

file

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

This line 1st checks wheather 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 = f~~
        f.create NewFile ();
        System.out.println (f.exists());//true
    }
}
```

1st run	2nd run
false	true
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 ("IPL Teams");
    System.out.println (f.exists ());  //false

    f.mkdir ();  // creates a new directory
    System.out.println (f.exists ());  // true
    }
}
```

1st run	2nd run
false	true
true	true

Note:- In Unix everything is a file, java "File" is based on UNIX operating system. hence in java also we can represent bothe files and directions 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 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");

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 Ipl team in current working directory and create a file named with abc.txt in that directory

```
cwd
    |=> Ipl team
        |=> abc.txt
```

Program

```
import java.io.*;
Class fileDemo{
public static void main (string[] args) throws
                                    IOException {
  file f1 = new file("Ipl team");
  f1. mdir();
file f2 = new file ("IPL Team", "abc.txt");
```

```
        f2.createNewFile();
    }
}
```

<u>Requirement:</u> write code to create a file name with rcb.txt present in C:\IplTeam folder.

```
C
|=> IplTeam
    |-> rcb.txt
```

<u>program:</u>

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

<u>Assuming</u> C:\\IplTeam should be already available otherwise it would result in "File Not Found Exception.

<u>Important methods of file class:</u>

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 avaible then this method simply returns false without creating any physical file.
If this file is not already available then it will create anew file and returns true

③ boolean mkdir();
This method 1st checks wheather 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 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 & 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);
  if (f1.isDirectory())
      dirCount ++;
  if (f1.isFile()) {
    if (name.endsWith (".png"))
        jpg fileCount ++;
    if (name endsWith(".txt"))
        txt file Count ++;
    if (name.endsWith.(".zip"))
        zip file Count ++;
  }
System.out.println (name);
}
```

↓

```
System.out.println("total no of JPG files is::"+
                                    jpgfilecount);
System.out.println("total no of txt files is::"+txtfil
                                    count);
System.out.println("total no of zip files is::"+zipfile
                                    count);
System.out.println("total no of Directory is::"+dircount
}
{ //JVM shutdown now
}
}
```

File Writer:

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 da
to the file.

Instead of overiding 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 file writer (file f, boolean append)

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

<u>methods</u>:

1. write (int ch);
   To write a single character to the file

2. write (char[] ch);
   To write an array of characters to the fil

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.

<u>eg#1</u>

```
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 \n Technology \n private
                                    \n limit
        fw.write ("\n");
        Char ch[] = {'a', 'b', 'c'};
        fw.write (ch);
        fw.flush ();
        fw.flush ();
    }
}
```

A new file will be created automatically

abc.txt

ineuron
Technology
private
limited
abc

Mentors for this handwritten notes :

Hyder Abbas (linkedin.com/in/hyder-abbas-081820150/)
Nitin M (linkedin.com/in/nitin-m-110169136/)
Navin Reddy (linkedin.com/in/navinreddy20/) (Telusko)
wrttien by pallada vijaykumar
@ineuron.ai

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

file Reader fr = new file Reader (String name);

file Reader 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

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 to 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 IOEx
        FileReader fr=new FileReader ("abc.txt");
        int i=fr.read();
        while (i!=-1){
            system.out.println ((Char) i);
            i=fr.read();
        }
    }
```

3/1/23 Part 2

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 noOfCharacters Copied = fr.read(ch);
```

Scenario 2:

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

```
import java.io.FileReader;
import java.io.IOException;
import java.io.File;
public class TestApp {
    public static void main (String[] args) throw
                                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 recommened 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 10 times So performance is very low.

③ To overcome these limitations we should

so for BufferedWriter and Buffered Reader concepts.

Buffered writer

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);

BufferedWritter bw = new BufferedWriter (writer w int buffer size);

which of the following declarations are valid?

1. BufferedWriter bw = new BufferWriter ("cricket.txt"); //invalid
2. BufferedWriter bw = new BufferWriter (newfile("cricket
3. BufferedWriter bw = new BufferWriter (newfilewriter ("cricket.txt"));
4. BufferedWriter bw = new BufferWriter (new FileWriter ("cricket.txt"))); //Valid

   //Valid BufferedWriter (new

   // 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 avaiable 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

```
import java.io.*;
/*
public void newline() throws java.io.IOE
exception.
{
```

```
FileWriter fw = new FileWriter("abc.txt", true)
bw.write (105);
bw.write ("INeuron");
bw.newLine ();
char[] ch= {'p', 'N', 's', 'k', 'I', 'c', 'l', 's'};
bw.write (ch);
bw.newline ();
bw.write ('unicorn');
bw.flush(); //to make sure the operation is
successful on file
bw.close (); //internally fw.close() call will
happen
}
//JVM shutdown now
}
```

Note.

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

→ when ever we are closing BufferedWriter automatically underlying writer will be closed

we are not close explicity

Buffered Reader:

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

Constructors

BufferedReader br = new BufferedReader (Reader)

BufferedReader br = new BufferedReader (Reader, int buffersize)

Note

→ Buffered Reader can not communite directly with file it should communicate via some Reader Object.

→ the main advantage of Buffered Reader over file Reader 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"

/*

*/ public java.lang.string readline() throws java.io.IOException.

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

Note

1. br.close() // recommended to use
2. fw.close() // not recommended to use

3. br.close() // not recommended to use both
fw.close()

=) Whenever we are closing BufferedReader automatically underlying fileReader will be closed it is not required to close explicity

-) 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 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);

eg#1 import java.io.*;

/* public void print (xxxx type)
* public void print (xxxx type);

```
class Test
{
public static void main (String[] args) throws Exception.
    FileWriter fw = new FileWriter ("abc.txt");
    PrintWriter out = new PrintWriter (fw);
```

```
Out.write(100); //100 Unicode value will be
written to a file.
Out.write('\n');
Out.println(100); // 100 only will be written to
the file
 out.println(true);
 out.println('c');
 out.println("Denis Ritchie");
  Out.flush();
  out.close();
 }
 // JVM shut down now
}
```

What is the difference b/w 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. the most enhanced Reader to Read character data from the file is Buffered Reader.
2. The most enhanced Writer to write character.

data to the file is print writer.

Note 2

① In general we can use Readers and writers to handle character data whereas we can use Input streams and output streams to handle binary data (like images, audio files, video files etc).

② we can use output stream to write binary data to the file and we can use Input stream to read binary data from the file.

Character Data ⇒ Reader and writer

Binary Data ⇒ Input stream and output stream.

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

```
import java.io.*;
Class Test {
{ public static void main (String[] args) throwes Excep
  {
  print writer pw = new print writer("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 BufferReader(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.
}
```

<u>Requirement</u> => 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 BufferReader (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 file 3.txt
pw.flush();
br1.close();
br2.close();
pw.close();
System.out.println("open file 3.txt to see the result");
}
//JVM shutdown now
}
```

<u>Requirement</u> => Write a program to remove duplicate from the file.

```
import java.io.*;
Class Test
{
  public static void main (String[] args) throws
                                Exception.
  {
    BufferedReader br = new BufferReader (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 File
                                Reader ("output.txt"));
String line = br1.readline();
//control comes out of while loop 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 (isAvalable == 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 noduplicate

```
impost java.io.*;

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 BufferedReder (new
fileReader ("delete.txt"));
            String line = br1.readline();
        //control comes out of while loop in smooth
fashin 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)

```
import java.io.*;
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 maximum string in a
file is :: "+maxlength);
}
//JVM shutdown now
}
```

—o—

4/1/23 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 b/w Serialization & Externalization
11. SerialVersionUID

Serialization

→ The process of Saving (or) Writing State of an object to a file is called Serialation 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.

Supposted form (or) network supposted form to

Java supposted form

→ By using file Input stream and Object Input stream.

Classes we can achieve Deserialization :

|=> ~~r~~ read Object ()

```
impost java.io.*;
/*
  public java.io.objectOutputstream (java.io.OutputStream)
                                        throws
                                        java.io.IOException;

public java.io.fileOutputStream (java.lang.String) throwy
  java.io.fileNotFoundException;
public final void writeObject (java.lang.object) throwy
                                        java.io.IOException;
public java.io.ObjectInputstream (java.io.Inputstream) throwy
  java.io.IOException;
public java.io.fileInputputstream (java.lang.String) throwy
java.io.fileNot-found Excephion;
public final java.lang.object readobject () throwy java.io.
  java.lang.ClassNot-found Excephion;                IOExcephion
*/
```

```
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 fileOutputOutputStream(filename);
        objectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(d);
        System.out.println("serialization Object reference 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 ....");
file Inputstream fis = new fileInputstream ("abc.ser");
Object Inputstream ois = new ObjectInput Stream (fis);
Object obj = ois.readObject();
    Dog d1 = (Dog) obj;
System.out.println("De-serialized object reference is::",
System.out.println("De-serialized ended ....");
}
  // JVM shutdown now
}
```

<u>eg#12</u>

```
import java.io.fileOutstream;
import java.io.IOexception;
import java.io.objectOutput Stream;
import java.io.fileInput Stream;
import java.io.Object Input stream;
import java.io.serialization;

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 serilization only for serializable objects.
2. An object is said to be serilizable if and only if the corresponding class implements Serlable interface.
3. Serializable interface present in java.io package and does not contain any abstract methods. It is marker 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-serializabl object then we will get RuntimeException saying "NotSerializableException".

Transient Key word:

① transient is the modifier applicable only for variables, but not for classess 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" keywo

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

④ That is transient means "not to serialize".

eg#1

```
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 int j=20;
}
public class TestApp {
        public static void main (String[] args) throw
IOException, ClassNotFoundException {
        Dog d1 = new Dog();
System.out.println("Serialization Started");
FileOutputStream fos = new FileOutputStream("abc.se
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 ("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.ObjectInputputStream;
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 ObjectOutput Stream(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
De-serialization 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) becaz 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 Serialable {
        int i=10;
        transient final int j=20;
}
public class TestApp {
        public static void main (String[] args) throw
IOException, ClassNotFoundException {
        Dog d1 = new Dog();
System.out.println ("Serialization Started");
fileOutputStream fos = new fileOutputStream ("abc.se
ObjectOutputStream oos = new ObjectOutputStream (fos)
oos.writeObject (d1);
System.out.println ("serilization 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 ("Dog Object data");
System.out.println (d2.i+"\t"+d2.j);
}
}
```

output

Serialization Started.
Serialization ended
Deserialation Started
Deserialation 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 "Class Cast Exception".

Example :-

```
Dog d1 = new Dog();
Cat c1 = new Cat();
Rat r1 = new Rat();
FileOutputStream fos = new FileOutputStream("abc.ser
ObjectOutputStream oos = new ObjectOutputStream(fos)
oos.writeObject(d1);
oos.writeObject(c1)
oos.writeObject(r1);

FileInputStream fis = new FileInputStream("abc.ser
ObjectInputStream ois = 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 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 "Not Serializable Exception".

Q#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 Serialiable{
        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) thro
IOException, ClassNotFoundException{
        Dog d= new Dog();
System.out.println ("Serialization Started");
FileOutputStream fos = new FileOutputStream("ab.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
*****
Deseseridition started
10
Deserialization ended
```

—o—

Customized Serialization

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.IOException;
```

```
import java.io.ObjectInputStream;
class Account implements serializable {
    String name = "sachin";
    transient string password = "ten dulkar";
}
public class Test {
        public static void main (String[] args) throw
IOException, ClassNotFound Exception {
    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 ("xxxxxxxxxxxxxxxx");
System.out.println ("Deserialization started");
  FileInputStream fis = new FileInputstream ("abc.ser")
Object InputStream ois = new Object Inputstream (fis
  acc = (Account) ois.readObject();
System.out.println (acc.name + "====>" + acc.passpwo
System.out.println ("Deserilization ended);
```

⇒ In the above example before Serialation 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 ~~chang~~ chance of loss of information due to transient keyword.

⇒ we can recover this loss of information by using customized Serialization.
we can implements customized Serialation 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 serialatation.

⇒ 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 Separate to the file)

② private void readObject(ObjectInputStream is) throws Exception.
⇒ This method will be executed automactically 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

```
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 serialisable {
    String name = "sachin";
    transient String password = "tendulkar";
private void writeObject (ObjectOutputStream oos)
                                    throws Exception){
    oos.defaultWriteObject(); // performing default ser
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 sa
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) thro
IOException, ClassNotFoundException {
  Account acc = new Account();
  System.out.println(acc.name + "=====>" + acc.
                                        password);
  System.out.println("Serialization started");
  FileOutputStream fos = new FileOutputStream("abc.s
  ObjectOutputStream oos = new ObjectOutputStream(fos)
  oss.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 Serilization).

⇒ If account class contains writeObject() method then JVM feels very happy and executes that Account class writeObjects() method.

→ the Same rule is applicable for readObject() method also.

```
import java.io.*;
/*
public java.io.objectOutputStream(java.io.outp
                                    Stream)
                                    throws
                                    Java.io.Excep
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.~~lang~~ 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 keyboa
    System.in.read();

System.out.println("De-serilization started ----");
FileInputStream fis = new ~~ObjectInputStream~~("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
}
```

<u>output</u>

```
D:\IOoperations>Java C Test.java
D:\IOoperations>Java Test
Serialization Started ....
writeObject method iscalled ....
Serialization ended ....
```

be-serialization started ....
readObject method is called ....
Username is :: Sachin
password is :: tendulkar
pin is :: 6444
De-serialization ended ....

5/12/23

Serialization w.r.to 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 seriable child class object if parent class implements.
serialible interface.

```
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 Animal implements Serilizable {
    int i=10;
}
class Dog extends Animal {
    int j=20;
}
public class Test {
    public static Void main (String [] args) throws
IOException, ClassNotfound Exception {
    Dog d = new Dog ();
System.out.println ("Serilization 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 ("abcser");
ObjectInputstream ois = new ObjectInputStream (fis);
Dog d1 = (Dog) ois.readObject ();
System.out.println (d1.j+ "====> "+d1.j);
System.out.println ("Deserilazation ended");
}}
```

Output

Serialization started
Serialization ended
.....

Deserialization started
Deserialization ended

- 10 => 20

Even though Dog class does not implements Serializable interface explicity but we can seriable dog object because its parent class Animal already implements Serialazable interface.

Note: Object class doesn't implement serializable interface

Case 2:

(1.) Even though parent class does not implemen serializable we can seriable child object if child class implements serializable interface.

(2) At the time of Seriatizable JVM ignores the values of instance variable which are coming from non-serializable parent then instead of original value JVM saves default value for those variables to the file.

(3) At the time of Deserialazation JVM checks wheather any parent class is non Serializable or not:

if any parent class is non serializable JVM creates a separate object for every non Serializable parent and Shares its instance Variable to the current object.

(4) To create an object for non-serialization parent JVM always ~~cor~~ calls no arg Constructor. (default Constructor) of that non-serialization parent. hence every non serializable parent should compulsory Contain no arg Constructor otherwise we will get runtime exception. "InvalidClass Exception".

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 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 ("Serializable started");

FileOutputStream fos = new FileOutputStream("abc
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.se
ObjectInputstream ois = new Object Input stream.
dog d1 = (Dog) ois.readObject();
System.out.println (d1.i+"===>"+d1.j);
System.out.println ("Desarilization 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 btw Serialization & Externalization.
3. SerialVersionUID

Externalization : (I.I.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 Exterinalizable 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 separate 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 Runtime Exception Saying "InvalidClassException".

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)
IOException, ClassNotFoundException throws {
```

```
ExternalizationDemo d = new Externalization ("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 ("seriliation 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 ("Deserialation ended");
}
}
```

Output

```
Serialization started
Call back method used while Serialation
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) ~~In externalization transient~~

If the class implements Serializable Interface then the output is nitin --- 10 --- 20

(5) In exterilization transient keyword won't play any role, hence transient keyword not requirement..

* 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 performance is low.

(6) Serialization interface doesn't contain any method.

(7) It is a marker interface

(8) Serialisable class not required to contains public no-arg constructor.

(9) transient keyword play role in serialization

Externalization..

→ Serial Version UID

→ To perform Serialization & Deserialization internally JVM will use a unique identifier which is nothing but SerialVersionUID..

⇒ At the time of serialization JVM will save SerialVersion UID with object.

→ At the time of Deserialization JVM will compare SerialVersion UID and if it is matched then only object

Will be

Deserialized otherwise we will get Runtime Exception Saying

"Invalid class Exception".

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 Runtime Exception Saying in "Invalid class Exception

2. Both sender and receiver should use the same Version of JVM if there any incompatability in JVM versions then reciever unable to deserialization because of different SerialVersionUID, in this case reciever will get Runtime Exception Saying "Invalid Class Exception".

(3) To generate SerialVersionUID internally JVM will use complex Algorithm which may create performance problems

we can solve p above problems by configuring our own serialVersionUID

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

Mentors for this handwritten notes :

Hyder Abbas (linkedin.com/in/hyder-abbas-081820150/)
Nitin M (linkedin.com/in/nitin-m-110169136/)
Navin Reddy (linkedin.com/in/navinreddy20/) (Telusko)
wrttien by pallada vijaykumar
@ineuron.ai

```
import java.io.*;
public class Sender {
    public static void main (String[] args) throw
                                    IOExpetion {
    Dog d=new Dog()
    FileOutStream fos = new FileOutputStream("abc.ser");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(d);
    }
}
import java.io.*;
public class RecieveApp {
    public static void main (String[] args) throw
IOException, ClassNotFoundException {
    ~~FileInputStream ois = new ObjectInputStream(fis);~~
```

```
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 RecieverApp

10 ====> 20.

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

=> we can configure our own SerializableVersionUID

both sender and reciever 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 multiple tokens based on the delimiter we supply

eg:- String data = "sachin ramesh tendulkar";

StringTokenzier stk = new StringTokenzier (data);

StringTokenzier stk = new StringTokenzier (data, " ");

⇒ public boolen 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);
    }
}
}
```

End

core java Completed.