FAQ’s

1. **Which JDK should we install?**

It is compatible with any JDK higher than Java 11. If installing new go with either JDK 17 or 18 either works.

2. **Which IDE is being used? Eclipse, IntelliJ, Netbeans, or Visual Studio Code?**

   Depending on your preferences, any of the IDEs listed above will suffice. However, because we will be using Eclipse in our course, please use the Eclipse IDE, which is free. It costs money to use IntelliJ with JEE and the Spring framework. As a result, we chose Eclipse over others.

3. **How long will the course go? Duration**

   The course will take at least 6 months to finish. It may be extended as we want to include additional examples and concepts to make you the best.

4. **How long will we have access to the live class's recorded video?**

   You will have lifetime access to live class recordings.

5. **Can a fresher or non-technical learner enrol in this course? Are there any requirements?**

   This course does not have any prerequisites. Anyone who wants to get into IT can register. We will start from scratch.

6. **Will there be a Spring Boot microservices project?**

Yes, we will create  project with SpringBoot Microservices, making you industry ready.

7. **What if I can't attend live classes? Or miss any class**

You can view recorded videos of live classes. It is strongly advised to watch recorded videos before attending the next live class.

8. **Will attending this course assist me in switching jobs?**

   Yes, if you attend all of your classes with focus and finish all of your projects.

9. **Can I include the projects I work on in class on my resume?**

   Yes, because it will be industry relevant.

10. **Will there be assignments during the course?**

    Yes, you will be given an assignment to help you improve.

11. **Will there be quizzes or assessments following each major topic?**

    Yes, There will be assessments and quizzes to prepare well.

12. **Will there be question-answer sessions and support during the course and throughout our projects?**

    Yes, every Wednesday and Friday at 8 p.m. IST, you will have live doubt clearing sessions twice a week. There is also chat support where you may share your screen.

**Variables in Java:**

**Instance Variables :**

1. Such variables which are directly created within a class outside any block or method.
2. For every object separated copy of instance variables will be created.
3. Memory for these variables would be given on the heap area, so jvm would perform initialisation with default values.
4. Variables whose value changes from Object to object (i.e for every new object new memory map on heap and new values it will hold)

```
public class Student{
    private String sname;
    private int sid;                /// Instance varibales
    private int sage;

        public Student(String sname,int sid,int sage){
                this.sname=sname; this.sid=sid; this.sage=sage;
        }
}
```

Note : If we make instance variable as final, then compulsorily we should perform initialisation explicitly jvm wont provide default values. (bcz final variables act constant - watch the class lecture on final keyword)

Whether we use or not otherwise we get compile time error.

**Local variables :**

1. Variables which are a part of method signature, method body and those variables are called as "Local variables". (for the first time we declare within a method or any block)
2. Variables which are created by the programmer to meet the temporary requirements are called "local variables".
3. Memory for those variables will be given on the stack memory,jvm will not initialise any value for those variables, compulsorily the user should initialise the value.

```
eg:: class Test{
        public static void main(String... args){
                int i;  // local variable
                System.out.println("Hello");
    }
}
```

Note : If the local variables are final then we should initialise before we use.(final keyword topic done during Inheritance concept kindly watch)

The only modifier applicable for local variables is final,if we use any other modifier it would result in a compile time error.

**String in Java:**

String refers to a collection of characters enclosed within double quotes( “ ”).

String refers to an Object in java present in a package called java.lang.String.

```
eg:: String s= "sachin";
    System.out.println(s);//sachin

    String s =new String("sachin");
    System.out.println(s);//sachin
```

Two types of String:
1- Immutable String (String class)
2- Mutable String (StringBuilder, StringBuffer)

- In java String object is by default immutable (String class if you store), meaning once the object is created we cannot change the value of the object, if we try to change then those changes will be reflected on the new object not on the existing object.

Ways to compare String ;

equals(): Compare the content of String Object
== : Compare references of String Objects.
equalsIgnoreCase(): Compares values of String ignoring case sensitivity.
compareTo(); compare value of string lexicographically

- Memory in case of String will be allocated in Heap and within heap there is String Constant pool. If we are creating a String object without a new keyword things get resolved during compile time and Memory created in the String Constant pool.
- Meaning : Direct literals are always placed in SCP,Because of runtime operation if object is required to create compulsorily that object should be placed on the Heap(Using new keyword or using any method like concate() ) ,but not on SCP.

(Watch the class recording for understanding of same)

Note::

- Before allocating memory in String Constant Pool(SCP) 1st jvm will check is any object is already created with required content or not.
- If it is already available then it will reuse the existing object instead of creating the new Object.
- If it is not available only then a new object will be created, so we say in SCP there is no chance of existing 2 objects with the same content.
- In SCP duplicates are not permitted.

- Garbage Collector cannot access SCP Area, Even though Object does not have any reference  still object is not eligible for GC.
- All SCP objects will be destroyed only at the time of JVM ShutDown.

Note :

- In our program if any String object is required to be used repeatedly then it is not recommended to create multiple objects with the same content it reduces performance of the system and affects memory utilisation.
- We can create only one copy and we can reuse the same object for every requirement. This approach improves performance and memory utilisation we can achieve this by using "scp". (Adv of SCP)

String(java.lang)
==========
We classify String into 2 types
a. Immutable
b. Mutable(StringBuffer(1.0V),StringBuilder(1.5v))

Case1:
String s=new String("sachin"); // 2 object (SCP and other one in heapArea)
s.concat("tendulkar");// 1 object(SCP)
System.out.println(s);
Since String object are immutable, if we try to change the object, that change would not happen in the same memory
the changes will happen in the new memory this concept we call it as "Immutable".

VS

StringBuffer sb=new StringBuffer("sachin"); // 2 objects( SCP and Other one in heapArea)
sb.append("tendulkar"); // 1 object(SCP)
System.out.println(sb);
Since StringBuffer is mutable, if we try to change the object, that change would happen in the same memory, this
mechanism is called "mutable".

Case2:
String s1=new String("sachin");// 2 objects(SCP and Other one in heapArea)
String s2=new String("sachin");// 1 Object (heapArea)
System.out.println( s1.equals(s2) ); // equals method is implemented to check the content of String

vs

StringBuffer sb1=new StringBuffer("sachin");
StringBuffer sb2=new StringBuffer("sachin");
System.out.println(sb1.equals(sb2); //equals method is not available in StringBuffer class,it is a part of Object class
so the implementation is coming from Object class which compares the reference(address)
not the data.

Code
====
```
StringBuffer sb1= new StringBuffer("sachin");
StringBuffer sb2= new StringBuffer("sachin");
System.out.println(sb1==sb2);//compares the refernce so => false
System.out.println(sb1.equals(sb2));//false
```

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

String s1=new String("sachin");
String s2=new String("sachin");
System.out.println(s1==s2);//false
System.out.println(s1.equals(s2));//true
```

final keyword( access modifer)
==========

It is an access modifier which can be applied at 3 levels

a. variable(primitive and reference)
b. class level.
c. method level.

If the variable is made as final then the value for those variables can't be changed,if we try to change
it would result in "CompileTimeError".
final variables would be resolved at the compile time only by the compiler.

eg:

```
class Test{
      public static void main(String[] args){
                  final int a=10;
                  int b=20;
                  b++;
                  a++;// a= a+1; change for the variable:CE(compiletime
error)
                  System.out.println(a);
                  System.out.println(b);

      }
}
```

final vs Immutablity
==========

```
class Test{
      public static void main(String[] args){
                  final StringBuffer sb=new StringBuffer("sachin");
                        sb.append("IND");
                  System.out.println(sb);//sachinIND

                        sb=new StringBuffer("tendulkar");//CE
                  System.out.println(sb);
      }
}
```

If the variable is of primitive type and if it is final then the value of the variable should not be changed,if we try to

change it would result in "CompileTimeError".

If the variable if of reference type and if it is of mutable nature then as per its mutable nature the object data can be changed,it would not result in "CompileTimeError",but if we try to reassign the reference variable with a new object address then it would result in "CompileTimeError".

note:
  final variable(valid concept)
  final object(not valid)
  immutable variable(not valid)
  immutable object(valid)

primitive
=========
 int,float,byte,short,long,double,....

reference
=========
 String,StringBuffer,StringBuilder,Object,.......

StringBuffer
============
   It is avaialble in java.lang package.

Methods of StringBuffer
=======================
   capacity() => the default capacity of StringBuffer is 16
                        if the capacity is filled internally jvm will increase the size using the following formulae
                             newCapacity= (currentcapcity + 1) * 2

eg:
```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());//16

sb.append("abcdefghijklmnop");
System.out.println(sb.capacity());//16

sb.append("q");
System.out.println(sb.capacity());// (16+1) * 2

sb.append("rstuvwxyz");
System.out.println(sb.capacity());// 34
```

eg:
 StringBuffer sb = new StringBuffer(19);//here the integer no specifies the capacity of StringBuffer

```
 System.out.println(sb.capacity());

eg:
StringBuffer sb = new StringBuffer("sachin");//here the capacity will be
(length of String + 16)
System.out.println(sb.capacity());
System.out.println(sb.length());

Important methods of StringBuffer
================================
  length() => it counts the no of characters present in the StringBuffer
Object
  append(String)
  append(Boolean) => it appends the given data to the Old StringBuffer
object
  append(float)

eg:
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());//16

sb.append("The value of PIE IS :: ");
sb.append(3.1414);
sb.append(", This is  exactly ::");
sb.append(true);

System.out.println(sb);
System.out.println(sb.capacity());//70
System.out.println(sb.length());//54


  insert(int,String)
  insert(int,int)    => it inserts the String at the specified index
  insert(int,long)

eg:
StringBuffer sb = new StringBuffer("abcdefgh");
System.out.println(sb.capacity());// 8 + 16 => 24

sb.insert(2, "xyz");
System.out.println(sb);// abxyzcdefgh

sb.insert(11, 9);
System.out.println(sb);      // abxyzcdefgh9


    delete(int,int)
    deleteCharAt(int)

eg:
 StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
 System.out.println(sb.capacity());// 21+16 = 37
 System.out.println(sb.length());//21
```

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

  sb.delete(6,12);// start to end-1
  System.out.println(sb);//sachintendulkar

  sb.deleteCharAt(6);
  System.out.println(sb);//sachinendulkar

  sb.deleteCharAt(21);//SIOBE(StringIndexOutOfBoundsException)
```

reverse() => It is used to reverse the StringBuffer Object

eg:

```
  StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
  System.out.println(sb.capacity());// 21+16 = 37
  System.out.println(sb.length());//21

  sb.reverse();
  System.out.println(sb);
```

setLength(int) => it is possible to reduce the length of the String at the runtime

eg:

```
  StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
       System.out.println(sb);//sachinrameshtendulkar
       System.out.println(sb.length());// 21

       sb.setLength(6);

       System.out.println(sb.length());//6
       System.out.println(sb);//sachin
```

trimToSize() => It will change the capacity to the length of the String

eg:

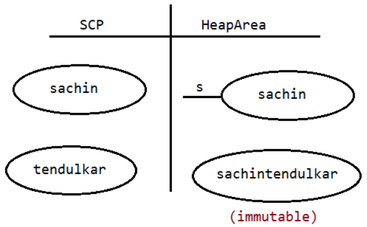
```
  StringBuffer sb = new StringBuffer(1000);
   System.out.println(sb.capacity());// 1000
            sb.append("iNeuron");
   System.out.println(sb.capacity());// 1000
          sb.trimToSize();
   System.out.println(sb.capacity());//7
```

ensureCapacity(int) => it is used to increase the capacity to the specific limit

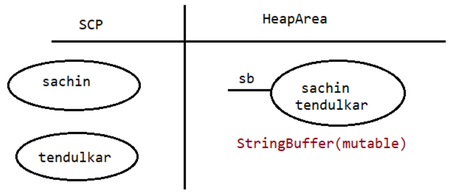
eg:

```
 StringBuffer sb = new StringBuffer();
  System.out.println(sb.capacity());//16

  sb.ensureCapacity(10000);
```

```
  System.out.println(sb.capacity());//10000


Difference b/w StringBuilder and StringBuffer
=============================================
    StringBuffer  => 1.0 V(1996)
                              All the methods present are synchronized
                              At a time on StringBuffer object only one
thread can operate
                              Since only one thread operate it is
"ThreadSafety".
                              Performance is low.


    StringBuilder => 1.5 V
                              All the methods present are not
synchronized
                     At a time on StringBuilder object many threads can
operate
                              So it is not "ThreadSafety".

                              Performance is high.
```

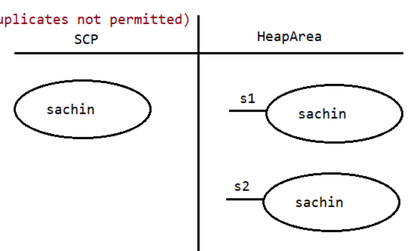



```
String s=new String("sachin"); // 2 object
                                  (SCP and other one in heapArea)

   s.concat("tendulkar");// 1 object(SCP)

   System.out.println(s); //sachin
```

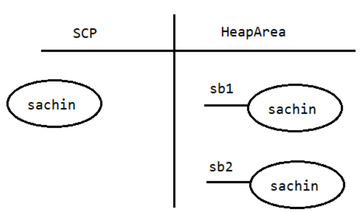



```
StringBuffer sb=new StringBuffer("sachin"); // 2 objects
                                              ( SCP and Other one in heapArea)
sb.append("tendulkar"); // 1 object(SCP)
System.out.println(sb);
```

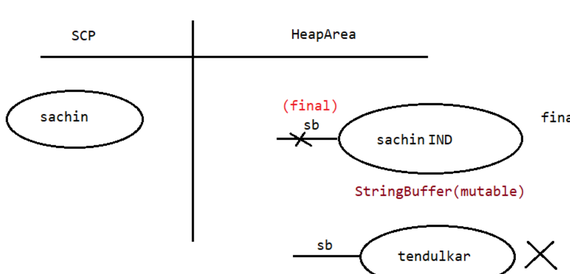
(duplicates not permitted)




```
String s1=new String("sachin");// 2 objects(SCP and Other one in heapArea)
String s2=new String("sachin");// 1 Object (heapArea)
System.out.println( s1.equals(s2) );  //true
```




```
StringBuffer sb1=new StringBuffer("sachin");
StringBuffer sb2=new StringBuffer("sachin");
System.out.println(sb1.equals(sb2);
```




```
final StringBuffer sb=new StringBuffer("sachin");
        sb.append("IND");
    System.out.println(sb);
```

```
sb=new StringBuffer("tendulkar");
System.out.println(sb);
```

Constructor :

- Whenever we are creating an object some piece of the code will be executed automatically to perform initialization of an object this piece of code is nothing but a constructor.
- Main objective of the constructor is nothing but initialisation of Object.
- Constructor can be referred to as a specialised setter whose name is the same that of the class name and it doesn't have any return type.
- Constructor is invoked during object creation.

Rules for writing a constructor
==========
1. Name of the constructor and name of the class must be same.
2. Return type concept not applicable for constructor, even if we provide it won't result in compile time error, if we do so then java language will treat this as "normal method".

```
eg:: class Test{
        void Test(){
            System.out.println("Hello");// It is not a constructor,it is a method.
        }
    }
```

3. It is not a good practise to take the method name same as that of the classname.
4. The modifiers applicable for constructors are private,public,protected,default.
5. The other modifiers if we use, it would result in compile time error.

```
eg:: class Test{
        static Test(){
      }
      }
   output::CE
```

Default constructor
==========
1. For every java class constructor concept is applicable.
2. If we don't write any constructor, then compiler will generate default constructor.
3. If we write atleast one constructor then compiler won't generate any default constructor, so
   we say every java class will have compiler generated default constructor or programmer written
   constructor but not both simultaneously.

Prototype of default constructor
==========
1. It is always no argument constructor.

2. The access modifier of the default constructor is same as class modifier.
   [applicable for public and default]
3. Default constructor contains one line, super(), It is a call to super class constructor.
   eg:: constructor.png

super() vs this()
==========
1. The first line inside the constructor can be super()/ this().
2. If we are not writing anything then compiler will generate super();

case1: We have to take super()/this() only in the first line of constructor, if we are writing any where else it would result in compile time error.
eg#1::

```
class Test{
        Test(){
                System.out.println("Constructor");//CE
                super();
        }
}
```

eg#2::we can either use super()/this() but not simulatenously

```
class Test{
        Test(){
            super();
            this();//CE
        }
}
```

eg#3:: we can use super()/this() only inside the constructor otherwise it would result in
compile time error.

```
class Test{
        void methodOne(){
                super();
                this();
        }
}
```

Note::
super() => It should be first line in the constructor.
It should be used only in constructor.
It will take the control to parent class constructor.

this() => It should be first line in the constructor.
It should be used only in constructor.

It will make the call of current class constructor.

Difference b/w super(),this() and super,this?

super(),this()

1. These are constructor calls
2. These are used to invoke super class and current class constructor directly
3. We should use only inside the constructor that to first line otherwise we get compile time error.

super, this

1. These are reserved words meant while working with object creation
2. These are used to refer to parent class and child class instance members.
3. We can use anywhere(instance area),except static area otherwise we get compile time error.

eg:: class Test{

```
        public static void main(String... args){
                System.out.println(super.hashCode());//CE
        }
}
```

Constructor Overloading

==========

- A class can contain more than one constructor and all these constructors have same name they differ only in the type of argument, hence these constructors are considered as "Overloaded constructor".

eg::

```
class Test {
        Test(double d) {
                System.out.println("double argument constructor");
        }

        Test(int i) {
                this(10.5);
                System.out.println("int argument constructor");
        }

        Test() {
                this(10);
                System.out.println("no argument constructor");
        }
}
public class MainApp {
        public static void main(String[] args) throws Exception {
                Test t1= new Test();//double int no argument constructor
                Test t2= new Test(10);// double int argument constructor
```

```
                Test t3= new Test(10.5);//dobule argument constructor
        }
}
```

3 pillars of oops
==========
1. Encapsulation => It speaks about security
2. Inheritance  => It speaks about reusability
3. Polymorphism  => It speaks about flexibility

Datahiding
==========
  Our internal data should not go to the outside world directly that is outside person should not
  access our interal data directly.
  By using private modifiers we can implement "datahiding".

eg:: class Account{
        private double balance;
   }
Advantage of Datahiding is security.
Recommended modifier for data members is private.

Encapsulation
==========
- Binding of data and corresponding methods into a single unit is called "Encapsulation".
- It also refers to providing controlled access to the most important component (data) of a class using the private keyword, setter and getter method.

  Syntax for setter method
    a. compulsory the method name should start with set.
    b. it should be public.
    c. return type should be void.
    d. compulsorily it should have some argument.

  Syntax for getter method
    a. compulsory the method name should start with get.
    b. it should be public.
    c. return type should not be void.
    d. compulsorily it should not have any argument.

  JavaBean
  ========
    It is a simple java class with private properties and public getter and setter methods.

Inheritance :

It is one of the pillars of Object Orientation.
It always speaks about reusability.
In java inheritance is achieved through the "extends" keyword.

Inheritance is the process of one class acquiring properties(Fields) and behaviour(methods) of another class.

It is also referred to as a writing project as a hierarchy of classes rather than a single class.

Important points to be noted:

- Private members of a class do not participate in Inheritance.
- Constructor will not get inherited but it will get executed because of super() call present in child class Constructor.
- Single Inheritance is allowed ( One class extending another class)
- Multilevel Inheritance is Allowed
- Multiple Inheritance is not permitted. It will lead to ambiguity to compiler also to the Diamond shape problem.
- Cyclic Inheritance is not permitted.
- Hierarchical Inheritance is allowed. (One class can have multiple child classes).

  Parent class== Existing class== Base class== Super class

  Child class== Derived class== Subclass

  This relationship also known as is - A relationship.

  Inherited Methods vs Overridden methods vs Specialized method

  Inherited Method : Such method which is inherited from parent class to child class and used as it is in child class without any modification.

  Overridden method : Such method which is inherited from parent class to child class and modified in child class to meet child class requirement.

  Specialized method : Such method which is present only in child class but not in parent class.

  Upcasting vs Downcasting:

  Upcasting : Creating Parent type reference for child type Object.
  It will achieve loose coupling best suited for polymorphism.

Downcasting : Temporarily converting parent type reference to child type so that specialised method of child class can be accessed.

Note : This note will help to answer questions in an interview and also will act like a summary only if you attend live class or watch live class recording. Kindly watch live class recording and treat it as a summary.

Relationship in java
==========

As a part of application developent, we have to use few entities(class) as per our application requirement.

By promoting relationship we can achieve
a. Optimisation over our code(less lines of code)
b. Code Reusability
c. Execution time
d. sharability.

How many types of relationship exists in Java?
we have 2 types of relationship
a. IS-A(achieved through extends keyword)
b. HAS-A

What is the difference b/w IS-A relationship and HAS-A relationship?
IS-A relationship
=> It is able to define inheritance b/w 2 entity classes
=> It also promotes code reusability in java application.

HAS-A relationship
=> It will define associations b/w 2 entities in java application.
=> Through association b/w entites it will imporve communication b/w 2 entites and data navigation b/w 2 entites.

Associations in Java
==========

In Java we have 4 types of association
a. 1 to 1 association
b. 1 to many association
c. Many to One association
d. Many to Many association.

To achieve association b/w 2 entities we have to declare either single reference or array of reference variable of an entity in another entity class.

eg:

```
class Address{
 Integer pinNo;
 Integer doorNo;
 String state;
 String city;
 String country;
}
class Student{
 String name;
 Integer age;
 Integer sid;

 //HAS-A variable
 Address address;//1 to 1 Association

 //HAS-A variable
```

```
 Address[] address;//1 to Many Association
}
```

Key points of Associations
==========

DependencyInjection
a. The process of Injecting the dependent object into target object is called "Dependancy Injection".

Target Object => The object which is been given to the developers for usage is called "Target Object".

Dependent Object => The object which is an helper object for the target object is called "Dependant Object".

```
eg#1
class Address{}//Dependant Object
class Student{}//Target Object
```

How to perform Dependancy Injection in Java?
It can be done in 2 ways
a. Constructor Injection.
a. We can inject primitive value to object.
b. We can inject one more object into another object.

b. Setter Injection.
a. We can inject primitive value to object.
b. We can inject one more object into another object.

One-One Association
==========

It is a relationship b/w 2 entities, where one entity of one class is mapped to one more entity of another class.

eg:
Employee(target object)
|
|HAS-A
|
Account(dependent object)

refer: ONE_ONE_EXample.png

refer: One-One-Association(Constructor Injection)
One-One-Association(Setter Injection)

When to use Constructor Injection and when to use Setter Injection?
Ans. It totally depends on the project requirement

Constructor => If the dependant object is ready at the time of target object creation then perform "Constructor injection".

Setter => If the dependant object is not ready at the time of target object creation then perform "Setter injection".

One to Many Assocation
==========

It is a relationship b/w 2 entities, where one entity of one class is mapped to multiple entities of another class.

eg:

```
Department(1) //Target Object
|
|HAS-A
 |
Employee(Many)//Dependent Object
```

refer: ONE_ONE_EXample.png

refer: One-Many-Association(Constructor Injection)

Previous session : HAS-A relationship
(1:1,1:M)
==========
Today session : HAS-A relationship(M:1,M:M)
Association
a. Composition
b. Aggregation

Many to one Association
==========
It refers to relationship b/w entites, where mulitple instance of one entity should be mapped to one instance of another entity.

eg: Multiple student have joined with single branch

```
public class Branch{
 private String bid;
 private String bloc;
}

public class Student {
 private Integer sid;
 private String sname;
 private Integer age;

 //HAS-A
 private Branch branch;
}
```

Many to Many Association
==========
It refers to relationship b/w entites, where mulitple instance of one entity should be mapped to Multiple instance of another entity.

eg: Many students have taken Many courses

```
public class Course{//Many
 private Integer cid;
 private String cname;
 private Integer cost;

}

public class Student{//Many
 private Integer sid;
 private String sname;

 //HAS-A
 private Course[] course;
}
```

refer: Many-Many-Association(Construtor Injection)
Many-One-Association(Construtor Injection)

refer: Many-To-Many.png

Many-To-One.png

Association
a. Composition
b. Aggregation

What is the difference b/w Composition and Aggregation?
Aggregation => In case of Aggregation,even if the contained object does not exist still the Container object can be reused.

Compostion => In case of Composition,if the contained object does not exist then container Object can't be reused.

eg: Relationship b/w library and student => Aggregation
Relationship b/w library and books => Composition

Relationship b/w Department and Professor => Aggregation
Relationship b/w University and Department => Composition

Relationhsip b/w Human and Heart => Composition
Relationship b/w Mobile and charger => Aggregation

refer:Composition vs Aggregation.png

Package => It is an encapuslation mechanism to group all the related classes and interface in single module

While using inbuilt classes also, internally sunmicrosystem has used the pacakges to keep the classes.

Objectives of packages
==========
1. It resolves the name conflicts
2. It imporoves modularity(maintainence) of the application
3. It provides security
4. It helps the programmer to debug the application in a easy way be referring to the pacakges.

packages naming convention followed by sunmicrosystem
==========
StringBuilder
StringBuffer
String
NullPointerException
ArithmeticException
System
Scanner

javac Sample.java => it would just compile the .class file
javac -d . Sample.java
  -d => indicates the destination folder where the package should be created.
  . => indicates the location where the package should be created.
      it represent the curren location.

eg:
```
package in.ineuron;

public class Sample
{
 public static void main(String[] args)
 {
  System.out.println("Hey i am working with packages");
 }
}
```

upon using the command javac -d . Sample.java
 =>java compiler has created the package and stored the .class file inside the package.

Standard Structure of java program
  1. package statement(only one should be there)
  2. import statement(can be any number)
  3. declartion of class/interface/enum (any number, but only one should be marked as public)

 refer: packageConcepts.png

Question :
If I have 2 try blocks and there corresponding catch blocks, and there is an exception catched
in the first try block, will the second try block be executed?

```
try{
            //risky code
}catch(Exception e){
            //handling code
}

try{
      //risky code
}catch(Exception e){
      //handling code
}
```

working of throws is not clear
   throws => to work with checked exception
   throw  => to work with unchecked exception

to handle the checked exceptions we use throws keyword????
    throws => this keyword is used in realtime coding

Sir how is the exception handeled when we use ducking?
  => excpetion is ducked then the exception object reaches to DefaultExceptionhandler
     DefaultException handler uses printStackTrace() to handle it.

if we are use ducking for compile time error, does it cause any problem for run time
exception handles? I mean does that key word Duck the User Exception during run time?
    exception => occurs in runtime and it will be ducked.

int c = 100/0;   is it a runtime or compile time error as value for constants are assigned at compile time ???
  it is runtime error as the execution happens at the runtime.

can we use different try blocks and one catch block

```
    try{

     }//compile time error
    try{

    }
    catch(Exception e){

    }
```

sir can u pls repeat compile time and runtime exception briefly
   CompileTime => no exception(just compiler will see what code would create the problem at
                                                    runtime)
   RuntimeExeption => problem occured at the runtime due to some unwanted disturbance.

which is recommended try catch or throw throws

```
    try{}catch(XXXX e) => it is used for uncheckedExcpetion
    throws -> checked Exception
    throw  -> handle the exception and to throw the exception manually to the
caller.


          Student s[] = new Student[3];
             Scanner sc = new Scanner(System.in);
           for(int i=0; i<s.length; i++) {
                 int age= sc.nextInt();
                 float salary =  sc.nextDouble(),
                 String name = sc.nextLine()
                 s[i] = new Student(age,salary,name);
           }

           for(int i=0; i<s.length; i++) {
             System.out.println(s[i].getName() + " " + s[i].getCpi() + " " +
s[i].getId());
           }

      code works fine


ducking => user is not intereseted in handling the exception,so jvm automatically
sends
           the exception object to the caller.

but here we have just ducking it by throws keyword but still exception not occured
    Thread.sleep(5000);//here exception wont occur still why compiler warned ?
        Compiler will support our code to make sure at the run time problem should
not come for jvm exeution.

Sir Could you please explain throws keyword.. how we are handling exception there?

//Vijet wrote the code
class Demo{
            public void m1()throws Exception {
                Thread.sleep(5000);
       }
 }
 public class Sample{
            public static void main(String[] args) throws Exception{

                  try{
                        new Demo().m1();
                  }catch(Exception e){
                        //handling logic
                  }
                        or

                  new Demo().m1();
            }
   }

JVM -> deafaultExceptionhandler ->handle it by using printStackTrace().


interface Calculator
{
```

```
      public float add(int a, int b);
}

 class launch4Calculator {
      public static void main(String[] args) {
            Scanner scan = new Scanner(System.in);
            int x = scan.nextInt();
            int y = scan.nextInt();

            Calculator cal = (a,b) -> a+b;
            System.out.println(cal.add(x, y));
            cal.add(x,y);
      }

}
why a and b creates error?


I didn't got exact diff between aggregation and compostion can you please explain?
   Composition => HAS-A realtionship
                              (container and contained object)

   Aggregation => HAS-A relationsihp
                              (container and contained object)


difference between error and exception
   Error => Problem occured in a program at runtime,but not able to handle the
problem through
               language specification.
                              eg: int[] a= new int[99999999];
                                    output: OutOfMemoryError.
                  here the problem is with the RAMSpace,we can't write a handling
code in catch block
                  to increase the RAM Space so it is called "Error".

   Exception => problem occured in program at runtime and programmatically we can
handle the problem through
                        language specification(Exception handling).
                                    syntax: try{} catch(XXXX e){}

//JDK code(sunmicorsystem team code)
public final class String{
                  public String subString(int x,int y)throws
StringIndexOutBoundsExeption{
                              //logic
                  }
}

public class Demo{
  main() throws SIOBE{
      new String("dhoni").substring(0,3);//dho
      new String("sachin").substring(-1,-100);//SIOBE
            ;;;;
            ;;;;
            ;;;;
            ;;;;
            ;;;;
  }
```

}

sir how to read exception message like exception(e){
system.out.print(e) or

System.out.println(e.getString());
e.printStackTrace();

System.out.println(e.getMessage());
}

```
try{
   int res=10/0;//risky code
 }catch(ArithmeticExeption ae){
      //handling exception
      int res=10/2;
 }
```

```
try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;
```


If no Exception occurs.

Statements - 1, 2, 3, 4, 5, 6, 8, 9, 11 & 12 will be executed.
Resulting in Normal Termination.


If an Exception occurs at statement-2 and the corresponding catch block is matched.

Statements - 1, 10, 11 & 12 will be executed.
Resulting in Normal Termination.


If an Exception occurs at statement-2 and the corresponding catch block is not matched.

Statements - 1 & 11 will be executed.
Resulting in Abnormal Termination.

```
try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;
```


If an Exception occurs at statement-5 and the corresponding inner catch block is matched.

Statements - 1, 2, 3, 4, 7, 8, 9, 11 & 12 will be executed.
Resulting in Normal Termination.


If an Exception occurs at statement-5 and the corresponding inner catch block is not matched, but outer catch block is matched.

Statements - 1, 2, 3, 4, 8, 10, 11 & 12 will be executed.
Resulting in Normal Termination.


If an Exception occurs at statement-5 and both inner and outer catch blocks are not matched.

Statements - 1, 2, 3, 4, 8 & 11 will be executed.
Resulting in Abnormal Termination.

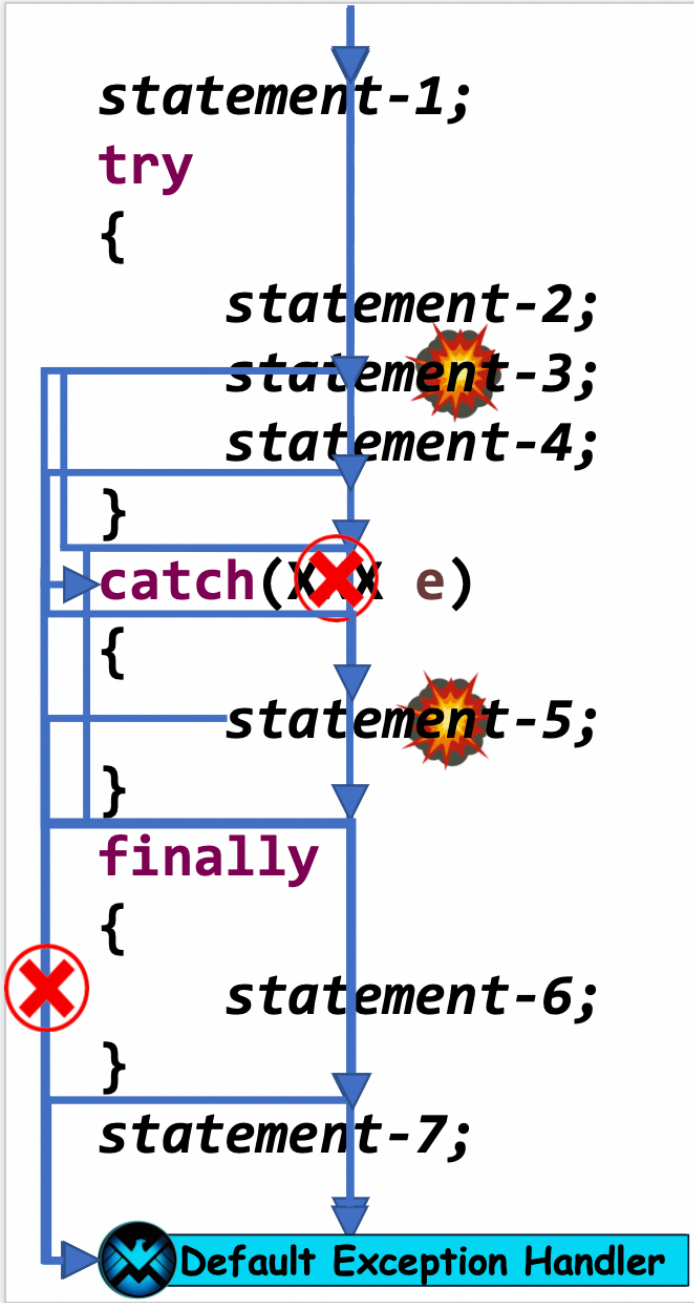
```
try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;
```


If an Exception occurs at Statement-8 and the corresponding catch block is matched.

Statements - 1, 2, 3, X, X, X, X (4, 5, 6, 7 optional), 10, 11, 12 will be Executed with Normal termination.


If an Exception occurs at Statement-8 and the corresponding catch block is not matched.

Statements - 1, 2, 3, X, X, X, X ( 4, 5, 6, 7 optional), 11 will be Executed with Abnormal termination.




CASE - 1

If there is no Exception raised.

statement-1, 2, 3, 4, 6 & 7 will be executed.
Resulting in Normal Termination.

CASE - 2

If an Exception is raised at statement-3 and the corresponding catch block is matched.

statement-1, 2, 5, 6 & 7 will be executed.
Resulting in Normal Termination.

CASE - 3

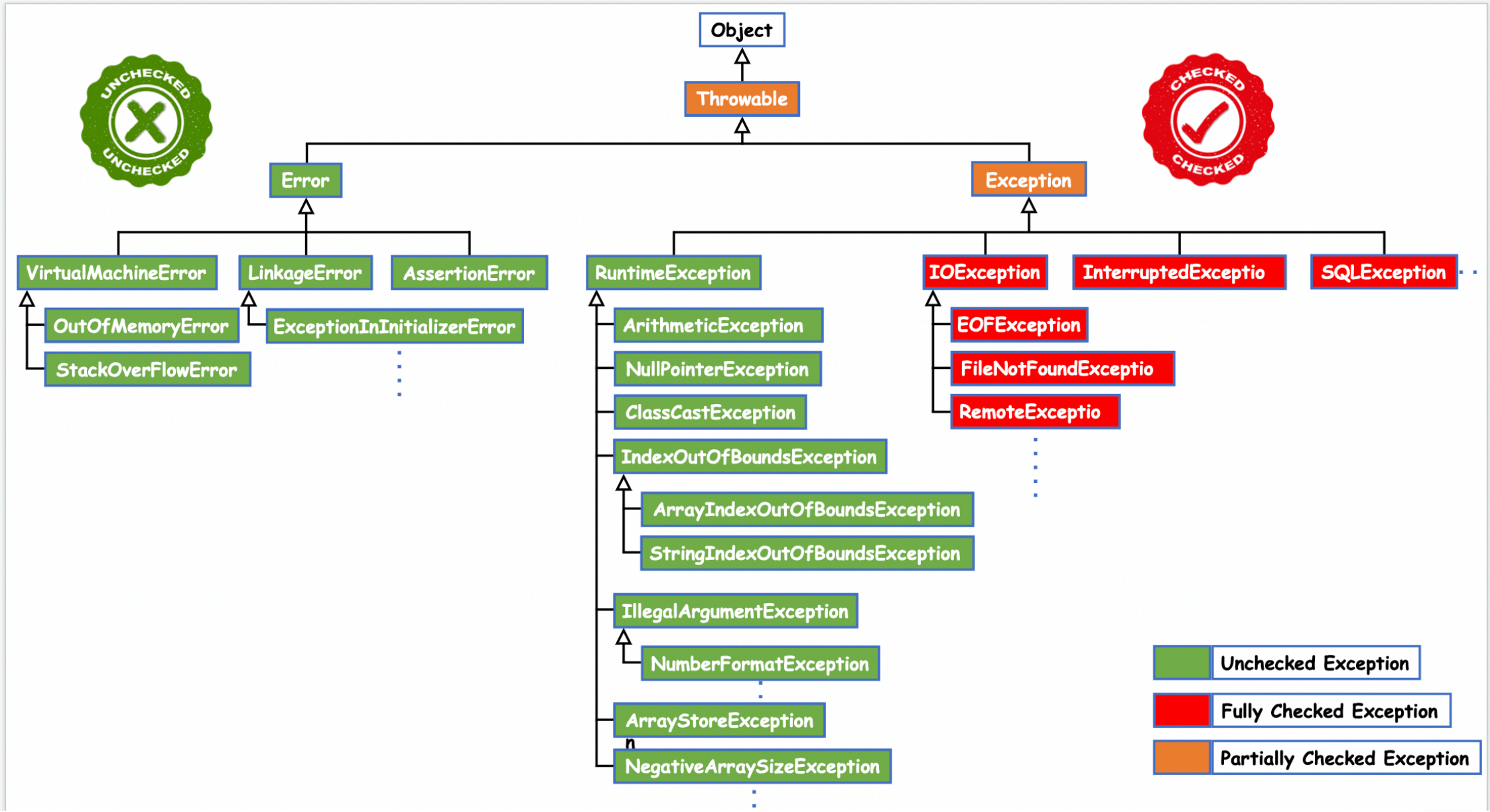
If an Exception is raised at statement-3 and the corresponding catch block is not matched.

statement-1, 2 & 6 will be executed.
Resulting in Abnormal Termination.

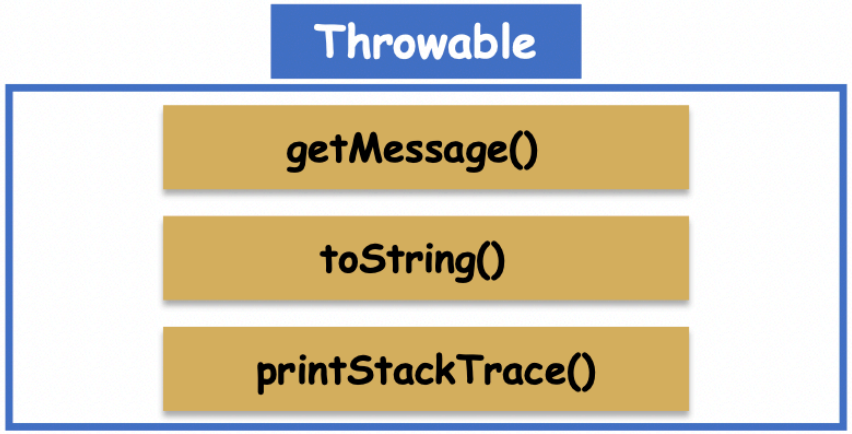
CASE - 4

If an Exception is raised at statement-5.

Resulting in Abnormal Termination.
But before that finally block will be executed.


Object
Throwable
Error
Exception
VirtualMachineError
LinkageError
AssertionError
OutOfMemoryError
StackOverFlowError
ExceptionInInitializerError
RuntimeException
IOException
InterruptedExceptio
SQLException
ArithmeticException
NullPointerException
ClassCastException
IndexOutOfBoundsException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException
IllegalArgumentException
NumberFormatException
ArrayStoreException
NegativeArraySizeException
EOFException
FileNotFoundExceptio
RemoteExceptio
Unchecked Exception
Fully Checked Exception
Partially Checked Exception

# Methods to Print Exception Information




| | |
|---|---|
| getMessage() | Prints the description of the exception |

Example: / by zero

| | |
|---|---|
| toString() | Prints the name and the description of the Exception |

Example: ArithmeticException: / by zero

| | |
|---|---|
| printStackTrace() | Prints the name and the description of the Exception along with the stack trace. |

Example: ArithmeticException: / by zero
at Demo.alpha()

what about the exception object creation if the constructor gives an exception, we know that the
stacktrace methods are in Throwable class,.....
so how will we get the exception object from constructor?
   Constructor => if exception is generated it will given to main() by JVM


what happens if we are keeping an empty catch block
      It would result in smoothful termination of the code.
    try{
            int a=10/0;
            System.out.println(a);
   }catch(Exception e){
      //handling code
   }

so sir if there is an error and whether or not the catch will match exception or not all the code
outside the finally block will not get executed?
      Ans. yes finally block will be executed even in this case,but not the code outside finally
            block and program would result in "Abnormal Termination".
            if exception occurs and if it is handled, then finally followed by other statments also
            would be executed and program would result in "Smoothful termination".
try{
      //risky code
      Excpetion genereated is ArrayIndexOutOfBoundsException
}catch(ArithmeticException e){
      //handling code
}finally{
      //resource releasing code
}
      //stmt-1
      //stmt-2

sir u said in slides that multiple try blocks cannot present but in the last code multiple try sare there i did not understood the last code  snipet sir
      try with catch and finally -> valid
      try and finally -> valid
      try and catch -> valid
      try{

       }catch(XXXX e){

      }finally{

      }

valid
------
try{
      try{

      }catch(XXXX e){

      }
}finally{

```
}
```

if the inner try catch block has thrown exception, inner catch does not matched, and it
matches outer catch block, before going control to outer catch, it will execute both finally block
and followed statement which are present in the inner try catch block?

```
try{
      try{
            //executed and exception occured not handled
      }catch(XXXX e){

      }
      finally{
            //gets executed normally
      }
      //will not be executed
      stmt-1
      stmt-2;
}catch(YYYY e){
      //exception handled
}finally{
      //statment executed
}
```

can you tell us again throw keyword? How the flow go the throw e?
        throws -> it is normally used with checkedException.
                  CheckException -> compiler will scan the code and it will check wheter the
                                                    Exception would occur or not at the runtime.

            eg: IOException,SQLExeption,....

      throw -> It is normally associated with UnCheckedException
                  CheckException -> compiler will scan the code and it will not check wheter the
                                                Exception would occur or not at the runtime.
                                                It is the duty of the programmer to make sure the termination
                                                should happen normally.

sir how it is possible to use return keyword in try block? as per my knowledge return is used to
transfer control back to caller method. plz explain.

```
public class Test{

      public int m1(){
          try{
            return 10;
          }catch(Exception e){
                  //handling logic
         }finally{
                  stmt-1;
         }
```

```
      }
}
```

sir if checked exception are not handled then it will be handled by DEH in runtime yes or no sir

Sir please can you explain throw keyword not cleared

```
   BankManager
         |
      Account
      public String deposit(){
                  //exception occured in the balance part for this customer
                  //exception should be handled
                  throw new problemInDepostingMoneyException();
      }
      public String changeName(){
                  //exception of changing Name
                  //exception occured and it is handled by Account class only.
        }
          |
       Customer
```

sir if checked exception are not handled then it will be handled by DEH in runtime yes or no sir

JDBCApi code(not written by end user)

```
==============================
public class DriverManger{
      public static Connection getConnection(String url,String username,String
password) throws SQLException
}

public class DemoApp{
      public static void main(String[] args) throws SQLException{
            Connection connection =
DriverManger.getConnection("jdbc:mysql://localhost3306/demo","root","root123");
      }
}
```

Exception had not occured => then connection will happen for MySQL database
Exception occured => JDBC Api will throw SQLException to JVM, JVM will check whether main()
has the handling code or not.
if handling code exists inside main then smoothful termination of application
if handling code does nt exists then JVM will delegate that SQLException object
to its own "Handler" called "DefaultExceptionHandler" and program would result
in "Abnormal termination".

sir we know that finally has cleanup code is it possible that finally can also generate exception, then in that case resource is stuck right?
then what to do?

```
      //Resource used in try block
```

```
      Connection con = null;
      try{
            //risky code
            con=DriverManager.getConnection(url,username,password);
      }catch(Exception e){
            //handling logic
      }finally{
            //resource releasing logic
            if(con!=null)
                  con.close();
      }

int method1(){
try{
      sop("");
      return 10;
}
finally{
      SOP(inside finally)
}

}
```

what will be the flow and whether again and again flow will enter into try just for return...

what is logical difference in ducking a exception and rethrowing an exception as they are giving same result

ducking -> u r just using throws keyword(no handling logic)

reThrowing-> we are handling it and also informing the caller that Exception occured(handling logic is available).

sir my another question will be as you told compiler will automatilaay initialize inbuild class object as
arithmatic excetion - new arithmaticex(); how compiler do that ? and what in case of custom exceptions?

Compiler => it will just whether the code is generating any checkedException or not and if yes it will also
check whether the programmer has wrote the handling logic or not.
hanlding logic can be either try{} catch(){} or using throws keyword

```
      try{
            int a=10/0; //JVM will execute this and it throws and Exception called "ArithmeticException".
                              //new ArithmeticException("/ by Zero");
      }catch(Exception e){//Exception e =new ArithmeticException("/ by Zero");

      }

public class A {

    public static void main(String[] args) {

        try{
            System.out.println("In try Block A");/stmt will be executed
            int c=10/0;//new ArithemeticExceptin("/ by zero");
```

```
        }catch (NullPointerException e){
            System.out.println("Null Pointer Exception");
        }finally {
            System.out.println("Finally Block A");//stmt will be executed
            try{
                System.out.println("In try Block B");//stmt will be executed
                int d=10/0;//new ArithemeticExceptin("/ by zero");
            }catch (ArithmeticException e){
                System.out.println("Arithemetic Exception");//stmt will be executed
            }
            System.out.println("outside catch block finally");//stmt will be
executed
        }

        System.out.println("I am outside of try and catch");
}
```

sir exception occured at inner catch ,please explain sir statements would be executed??

```
statement 1;
try
{
      statement 2;
      statement 3;
      statement 4;
      try{
            statement 5;
      }catch(xx e){
            statement 6;
      }
      statement 7;
}
catch(xy e)
{
      statement 8;
}
finally
{
      statement 9;
}
```

1,2,3,4,5,8,9(smoothful termination) if exception is  handled.
1,2,3,4,9(abnormal termination) as excepition is not handled.

```
package Exception;

import java.util.InputMismatchException;
import java.util.Scanner;

public class InnerExc
{
    public static void main(String[] args) {
        try {
            Scanner scan = new Scanner(System.in);
            System.out.println("enter first number for division");
            int a = scan.nextInt();
            System.out.println("enter second  number for division");
            int b = scan.nextInt();
            try {
```

```
                        int c = a / b;
                        System.out.println(c);
            }catch (InputMismatchException ex){
                        ex.printStackTrace();
            }
            finally {
                        System.out.println("inner finally block");
            }
             System.out.println("inner loop");
           }catch (ArithmeticException ex){
               ex.printStackTrace();
           }finally {
               System.out.println("out side finally block");
           }
           System.out.println(" in main method");
    }
}//when i execute this code if i enter wrong input  InputMismatchException has to
caught but its

//Compile Time error
try{
      finally{
            statement 2;
                  try{
                        statement; //error occured
                  }catch(xx e) //not matched
                  {
                        statement1;
                  }

      }
} catch(exception e)//exception matched {
      sop("Hi")
 }

what is the extra use of new ArithmaticException("/ by zero") when jvm already
preparing object, for only custom msg?
    Exception=> They are Objects in java.
                  If problem occurs in our code immediately jvm will generate the
suitable object
                  for the problem.


While ducking compiler will check for exception but rethrow user has to do it what
does it meant by sir?
      ducking -> compiler warns u to write the handling code for the  statements

JDBCApi code(not written by Oracle developers)
==============================
public class DriverManger{
      public static Connection getConnection(String url,String username,String
password) throws SQLException
}

Progammer code
============
public class DemoApp{
      public static void main(String[] args) throws SQLException{
            Connection connection =
```

```
DriverManger.getConnection("jdbc:mysql://localhost3306/demo","root","root123");
      }
}


Code
===
try{
            System.out.println("In try Block A");//stmt will be executed
            int c=10/0;//new ArithmeticException("/ by Zero")
    }catch (NullPointerException e){
            System.out.println("Null Pointer Exception");
    }finally {
            System.out.println("Finally Block A");//stmt will be executed
            try{
                System.out.println("In try Block B");//stmt will be executed
            }catch (ArithmeticException e){
                System.out.println("Arithemetic Exception");
            }
            System.out.println("outside catch block finally");//stmt will be
executed
  }
      System.out.println("I am outside of try and catch");
 }
}

NullPointer Exception
==============
String name=null;
System.out.println(name.length());//NullPointerException

int arr[]=null;
arr[4]=10; //NullPointerException
will get null pointer exception

sir can you give overview of jar vs war file just basic overview?
    JAR -> collection of .class files(jdk s/w which we install is actually .class
files only)
                  It is used when we work with standalone applications.(JSE)

   WAR-> collection of .class files + html files +css files  +javascript files
                  It is used when we work with webapplications/distrubuted
applciation(JEE)


try{
            int res=10/0; // throw new ArithemeticException("/ by zero");
}catch(NullPointerException e){

}

custom exception
============

class StudentRecordNotFoundException extends RunTimeException{
            public StudentRecordNotFoundException(String msg){
                        this.msg=msg;
            }
}
```

```
class TestApp{
      public Student getRecord(String id){
            Student std = get the record from the database
            if(std!=null)
                  throw new StudentRecordNotFoundException("record not found");
            else
                  return std;
  }
}

class Demo{
      public static void main(String...  args){
            try{
                  Student student = new TestApp().getRecord(10);
            }catch(StudentRecordNotFoundException se){
                        se.printStackTrace();
            }catch(Exception e){
                  e.printStackTrace();
            }
       }
}




try{
      stmt-1
      stmt-2
            try{
                  stmt-3//exception occured
            }catch(XXXX e){//exception not handled
                  stmt-4
            }finally{
                  stmt-5
            }
}catch(YYYY e){//exception not handled
            stmt-6
}finally{
            stmt-7
}
```

finally block will be executed and JVM will handled by using DEH.

# Differences between throw & throws

| throw | throws |
| --- | --- |
| throw keyword is used to explicitly throw an exception to the JVM. | throws keyword is used to declare an exception to delegate exception handling responsibility to the caller. |
| throw keyword is followed by an instance of Throwable or a subclass of Throwable. | throws keyword is followed by exception class name. |
| throw keyword is used within the method body. | throws keyword is used with the method signature. |
| Multiple exceptions can't be thrown with a single throw clause. | Multiple exceptions can be declared with a single throws clause. |
| Used in rethrowing an exception. | Used in both rethrowing and ducking an exception. |
| try {<br>}<br>catch (Exception e) {<br>throw e;<br>} | void test() throws Exception<br>{<br><br>} |

CASE - 1

Only try

```
try
{


}
```

CASE - 2

Only catch

```
catch(XXX e)
{


}
```

CASE - 3

Only finally

```
finally
{


}
```

CASE - 4

try - catch

```
try
{

}
catch(XXX e)
{

}
```

CASE - 5

Reverse order

```
catch(XXX e)
{

}
try
{

}
```

CASE - 6

**Multiple try**

```
try
{

}
try
{

}
catch(XXX e)
{

}
```

CASE - 7

**Multiple try**

```
try
{

}
catch(XXX e)
{

}
try
{

}
```

CASE - 8

**Multiple try - catch**

```
try
{

}
catch(XXX e)
{

}
try
{

}
catch(XXX e)
{

}
```

CASE - 9

**Multiple catch**

```
try
{

}
catch(XXX e)
{

}
catch(YYY e)
{

}
```

CASE - 10

**Multiple catch**

```
try
{

}
catch(XXX e)
{

}
catch(XXX e)
{

}
```

CASE - 11

**Multi - catch**

```
try
{

}
catch(XXX | YYY e)
{

}
```

Exception handling
==========
1. try catch and finally
2. throws(best suited for checkedException)
3. throw(best sutied for uncheckedException and customException)

Syntax
=====
```
  try{
      //risky code
  }catch(XXXXX e){
      //handling code
  }finally{
      //resource releasing code
  }
```

In realtime application,we use many resources where all the resource should be closed inside finally block.
            resource => In File operations we use FileReader,FileWriter,BufferedReader,BufferedWriter
                                In JDBC Operations we use Connection,Statement,PreapredStatement,CallableStatement,....

Realtime coding
===========
```
 //declaration of resources
  try{
      //risky code
      use the resource
  }catch(XXXXX e){
      //handling code
  }finally{
      //resource releasing code
  }
```

JDK1.6V for developers
==========
eg:
```
BufferedReader br=null;
      FileReader fr =null;
      try{
            fr  = new FileReader("sample.txt");
            br = new BuffereReader();
      }catch(IOException e){
            e.printStackTrace();
      }finally{
            if(br!=null)
                  br.close();
            if(fr!=null)
                  fr.close();
      }
```

boilerplate -> the code which is repeated in multiple modules of project with no change or with small change.
                  whenever boiler plate code comes into pitcutre, we always try to avoid it by using
                        a. using JDK software higher version(jdk1.0,jdk1.2,..... jdk18)
                        b. using 3rd party API's

In JDK1.7 version they made few enhancement in the Excpetion handling area
==========
1. try with resource
2. try with multicatch block

try with resource
==========
Syntax:  try(R){
                    use resource as per ur application requirement
                    if exception occurs or not occurs and if it is handled or not handled
                    still Resources will be closed once the control comes out of try block
          }catch(XXXX e){

          }

eg: without using try with resource
```
    ==========
      BufferedReader br=null;
    FileReader fr =null;
    try{
          fr  = new FileReader("sample.txt");
          br = new BuffereReader();
    }catch(IOException e){
          e.printStackTrace();
    }finally{
          if(br!=null)
                br.close();
          if(fr!=null)
                fr.close();
    }
```

eg: try with resource
```
       try(BufferedReader br= new BufferedReader(new FileReader("sample.txt"))){
                //use the resource
       }catch(IOException e){
                e.printStatckTrace();
       }
```

Advantage of try with Resource
==========
  1. The main advantage of try with resource is the resources which are a part of try block gets close automatically.
    once the control comes of out try block automatically that resource will be closed.
    while try block is getting executed
          a. exception occured and handled
          b. exception occured and not handled
     In both these cases also jvm will close the resource automatically, if we use resource with "try with resource".

  2. Using try with resource increases readabilty and reduces redundant code in our application.

Conclusions
==========

1. we can declare any no of resources ,but all the resoures should be seperated with ; symbol.

```
try(R1;R2;R3; ........){

}catch(XXXXX e){

}
```

2.From JDK1.7 for Resource Releasing logic Requirement specification they had come with an interface called "AutoCloseable" which is added in "java.lang" package.

```
interface AutoCloseable{
      public abstract void close() throws Exception;
}
public class BufferedReader implements AutoCloseable{
            @Override
            public void close(){
                  //logic of closing.
            }
}
try(BufferedReader br=new BufferedReader(new FileReader("sample.txt"))){
            //logic of using br
}
  catch(IOException e){
            //handling logic
}
```

Note:

```
try(String name =new String("sachin")){
                  //using name object
   }catch(Exception e){}
```

output: Compile Time Error

All java.io classes and java.sql classes has implemented AutoCloseable interface.

3. All resources reference variable are been made as final automatically when they are used, so we can't re-assign the reference of the Resource Variable.

CompileTime Error

==========

```
try(BufferedReader br=new BufferedReader(new FileReader("sachin.txt"))){
            br=new BufferedReader(new FileReader("kohli.txt")));
}catch(IOException e){}
```

4. Before JDK1.6

```
try{

}catch(XXXX e){

}finally{

}
```

After JDK1.7, do we need finaly block ?

Ans. no

finally block becomes dummy if we use "try with Resource".

5. JDK1.6V

```
try{
```

```
      }finally{

      }
      a. if exception does not occur => normal termination/smoothfull termination
still finally executes.
      b. if exception occurs => abnormal termination still finally executes.

     JDK1.7
      try(R){

      }
      it is possible to write only try also from JDK1.7 version ,but that try
should be associated with Resource.


JDK1.5 features
===========
1.Wrapper classes
2.Var-Args

Wrapper class
==========
      1. To wrap primtive into object form so that we can handle primtive also just
like objects
      2. To define several utility function which are requried for primitives.
      3. Wrapper classes are a part of "java.lang" package.

primtive data types
==============
  1. byte,short,int,long
  2. float,double
  3. char
  4. boolean

For every primitive type we have equavilent Wrapper class as shown below
      byte -> Byte
      short -> Short
      int    -> Integer
      long -> Long
      float -> Float
      double-> Double
      ***char  -> Character(1 constructor)
      ***boolean -> Boolean(2 constructor(String is important))

With Respect to wrapper class how is toString() implemented?

class Object{
      public String toString(){
            // returns the reference(address/hashCodeValue) of the object
      }
}
public final class Integer extends Object{

      @Override
      public String toString(){
            //returns the data present in the Object
      }
}
```

Almost every Wrapper class contains 2 constructors which takes
a. primtive type as the argument.
b. String type as the argument.

eg#1.

```
Integer i1 = new Integer(10);
System.out.println(i1);//jvm calls i1.toString()
Integer i2 = new Integer("10");
System.out.println(i2);//jvm calls i1.toString()
```

output

```
10
10
```

eg#2
If the String input is not properlyformatted,mean if it is not reprsenting any number then we will get an Exception called
"NumberFormatException"

```
  Integer i2 = new Integer("ten");//NumberFormatException
```

eg#3.
Character class contains only constructor which can take only primitive argument of type char only.

```
Character c1=new Character('a');
System.out.println(c1);

Character c1=new Character("a");//Compile Time Error.
System.out.println(c1);
```

eg#4.

```
Boolean b=new Boolean(true);
System.out.println(b);//true

Boolean b=new Boolean(false);
System.out.println(b);//false

Boolean b=new Boolean(True);//CE
Boolean b=new Boolean(False);//CE
```

Note: If we are passing String argument, then case is not important and content is important.
if the content is case insensitive String of true then it is treated as true and in all other cases it is false.
eg#5

```
Boolean b1=new Boolean("false");
System.out.println(b1);//false

Boolean b2=new Boolean("False");
System.out.println(b2);//false
```

eg#6

```
Boolean b1=new Boolean("true");
System.out.println(b1);//true

Boolean b2=new Boolean("True");
System.out.println(b2);//true
```

```
eg#7.
Boolean b1=new Boolean("yes");
System.out.println(b1);//false

Boolean b2=new Boolean("no");
System.out.println(b2);//false

Boolean b1=new Boolean("tRuE");
System.out.println(b1);//true

Boolean b2=new Boolean("TrUe");
System.out.println(b2);//true

Object class methods
===============
public class java.lang.Object {
  public java.lang.Object();
  public final native java.lang.Class<?> getClass();
  public native int hashCode();
  public boolean equals(java.lang.Object);
  protected native java.lang.Object clone() throws
java.lang.CloneNotSupportedException;
  public java.lang.String toString();
  public final native void notify();
  public final native void notifyAll();
  public final native void wait(long) throws java.lang.InterruptedException;
  public final void wait(long, int) throws java.lang.InterruptedException;
  public final void wait() throws java.lang.InterruptedException;
  protected void finalize() throws java.lang.Throwable;
  static {};
}

String toString()
      JVM will always call toString() when we try to print any reference variable.
      reference varaible can be
            a. inbuilt class
            b. user defined class

eg#1.

class Object{
      public String toString(){
            // returns the reference(address/hashCodeValue) of the object
      }
}
public final class String extends Object{

      @Override
      public String toString(){
            //returns the data present in the Object
      }
}

String name= new String("sachin");
System.out.println(name);// jvm internally calls name.toString()

eg#2.

class Object{
```

```
	public String toString(){
		// returns the reference(address/hashCodeValue) of the object
	}
}
public class Student extends Object{
	String name;

	Student(String name){
		this.name =name;
	}

	public String toString(){
		// returns the reference(address/hashCodeValue) of the object
	}
}

Student student = new Student("sachin");
   System.out.println(student);//JVM calls student.toString()

output: hashCode value of Student object


eg#3.
class Object{
	public String toString(){
		// returns the reference(address/hashCodeValue) of the object
	}
}
public class Student extends Object{
	String name;

	Student(String name){
		this.name =name;
	}

	@Override
	public String toString(){
		return this.name;
	}
}

Student student = new Student("sachin");
   System.out.println(student);//JVM calls student.toString()

output: sachin
```

If we use our own class inside try with resource then will it consider as resource
 ans. no

how we can edit the predefined class as implement autocloseable
   predefined not possible as source code would not be shared.

Can we still close the resouce that does not implement autoclosable in finally
block?
    yes possible, iff the class contains close()

On which criteria they have made only java.io and java.sql as auto closable ?
   Normally resources are present inside java.io and java.sql

sir is it required us to remember version from where these added concept came
    only jdk1.5 and Jdk1.8 version feature are used so remember them.

almost all of these constructors are deprecated in latest version of java ?
    use jdk8, later we speak about valueOf()

```
Boolean b1 = new Boolean("1");
Boolean b2 = new Boolean("0");
```

what will be the output of this code
false
false

I have one small doubt in Integer class Why they have implemented constructor which
will accept
String type of arguement ..why we need to pass int values in double quotes ?
I want to understand the use case of wrapper class, please

ans. In real time data movement always happens in String.

```
            String age = request.getParameter("age");
            Integer data = new Integer(age);
```

Sir can you give 2-3 examples of application/proj build using the entire Java
stack...so that we
could get a idea..what type of application we will be able to develop after this
course?
    CoreJava-> standalone application
    JEE        -> WebApplication's
   Springboot and hibernate -> Enterprise application

you are handling the unchecked expections with try and catch and using throw
keyword to send to JVM, but in the
method signature you are using throws which is checked expections. How is this
possible?
    throws => checked Exceptions
    throw   -> uncheckedException and CustomException

I have one doubt regarding for each loop int the following code

```
Scanner scan = new Scanner(System.in);
int[] arr = new int[5];

//It can't be used to inserting the values to the array
for(int a : arr)
{
        a = scan.nextInt();
}
```

```
//foreach is used only for reading the data from collection/array.
for(int a : arr)
{
        System.out.print(a + " ");
}
```
input: 1 2 3 4 5
output: 0 0 0 0 0
I don't understand why it is not updating values, can you explain.

```
class InvalidUserException extends Exception
{
      InvalidUserException(String msg)
      {
            super(msg);
      }
}
```
  sir this calls => Exception class Parent constructor

Sir when we have wrapper class concept in Java so why Java is still called not fully object
 oriented?We can easily convert the primitive types to object type?
   Because java still supports primitive data types, so we can't say it is fully object oriented.

Sir i heard so many times from u that creating object is costly event for the jvm......
Wt does that mean by costly event/(just curiosity to know sir)

```
class Object{
      static{
            ....
      }
      Object(){

      }
}
class Parent  extend Object{
      static{
            System.out.println("parent class loading");
      }
      Parent(){
            super()
      }
}
class Child extends Parent{
      static{
            System.out.println("child class loding");
      }
      Chid(){
            super();
      }
}
new Child();
```

JDK8
====

```
try(br1=new BufferedReader(new FileReader("sample.txt"));br2=new BufferedReader(new
FileReader("sample.txt"));){

}
```

JDK9
====

```
br1=new BufferedReader(new FileReader("sample.txt"));
br2=new BufferedReader(new FileReader("sample.txt"));
try(br1;br2){

}
```

Wrapper class
==========
  The need of wrapper class is to wrap primtives into objects,so that we can handle primitives also
  just like Objects.

Constructor summary
===============
byte -> Byte(byte,String)
short-> Short(short,String)
int   -> Integer(int,String)
long -> Long(long,String)
float -> Float(float,String,Double)
double->Double(double,String)
---------------------------------------
char-> Character(char)
boolean->Boolean(boolean,String)

Note: In all wrapper class toString() is overriden to return the content

Object class method
===============
```
public class Object{
      public String toString(){
            //return the reference of the Object
      }
      public boolean equals(Object o){
            //compares the reference
      }
}

public final class String{

      @Override
     public String toString(){
            //returns the content of the Object
      }

      @Override
      public boolean equals(Object o){
            //compares the content of the String
      }
}
```

Note: equals() method is also overriden in all Wrapper class to compare the content.

```
public final class Integer{

      @Override
     public String toString(){
            //returns the content of the Object
      }

      @Override
      public boolean equals(Object o){
            //compares the content.
      }
}
```

```
eg#1.
Integer i1=new Integer(10);
Integer i2=new Integer(10);
System.out.println(i1); //i1.toString() -> 10
System.out.println(i1.equals(i2));//true


Setter methods and Getter Methods
==========================
public void setXXXX(XXXX data){

}
public XXXX getXXXX(){

}

Usage of Wrapper class
--------------------------
 utility method(helper methods/static methods)
1.valueOf()
2.XXXXValue()
3.parseXXX()
4.toString()

1.valueOf()
            signature: public static wrapper valueOf(primtive data)
                              public static wrapper valueOf(String data)
       It is used to create wrapper object for the given primtive or String type of
data.
       It is alternative to constructor,but good practise is to use valueOf() only.

eg#1.
//constructor usage of Wrapper class to create Wrapper Object
Integer i1= new Integer(10);
Integer i2= new Integer("10");

//usage of utility methods to create Wrapper Objet
Integer i3= Integer.valueOf(10);
Integer i4= Integer.valueOf("10");

      System.out.println(i1);
      System.out.println(i2);

      System.out.println();

      System.out.println(i3);
      System.out.println(i4);

output
10
10

10
10

Note: valueOf() is also a part of Character class.

eg#2.
Integer i1=Integer.valueOf(10);
```

```
Double d1= Double.valueOf(10.5);
Boolean b1=Boolean.valueOf("Nitin");
Character c1=Character.valueOf('a');

System.out.println(i1);//10
System.out.println(d1);//10.5
System.out.println(b1);//false
System.out.println(c1);//a
```

2.xxxxValue()

We can use xxxxValue() to convert wrapper to prmitive type.

Every Number type wrapper class(Byte,Short,Integer,Long,Float,Double) contains the following 6 xxxxValue() method to convert the wrapper object to primitive type.

Number
- Byte
- Short
- Integer
- Long
- Float
- Double

- Character
- Boolean

eg#1.

```
Integer i=new Integer(130);
System.out.println(i.byteValue());//-126
System.out.println(i.shortValue());//130
System.out.println(i.intValue());//130
System.out.println(i.longValue());//130
System.out.println(i.floatValue());//130.0
System.out.println(i.doubleValue());//130.0
```

Note: 130 is not in the range of byte so jvm will perform operation in the following manner

```
range = -128 to 127
result =  -128,-127,-126
                last value will be stored.
```

eg#2.

```
Character c1=new Character('c');
char c2= c1.charValue();
System.out.println(c2);//c

Boolean b1=new Boolean("nitin");
Boolean b2=b1.booleanValue();
System.out.println(b2);//false
```

3.parseXXX()

Every wrapper class except Character class Contains parseXXXX() to convert String to Corresponding primtive type.

```
signature: public static xxxx parseXXX(String data)
```

eg#1.

```
int i1= Integer.parseInt("10");
```

```
            System.out.println(i1);//10

            boolean b1=Boolean.parseBoolean("TrUE");
            System.out.println(b1);//true

            short s1=Short.parseShort("Ten");
            System.out.println(s1);//NumberFormatException

4. toString()
            We can use toString() to convert wrapper object/primitve data to
String.
            signature:   public static String toString(XXXX data)
                               public static String toString(xxxx data)

eg#1.
            Integer i=Integer.valueOf("10");
            System.out.println(i);//10(in String format)
            System.out.println(i.toString());//10(in String format)

            System.out.println();


            String s1=Integer.toString(10);
            String s2= Boolean.toString(true);
            String s3= Character.toString('a');

            System.out.println(s1);//10(in string format)
            System.out.println(s2);//true(in string format)
            System.out.println(s3);//a(in string format)

                  refer :diagram for conversion chart


AutoBoxing and AutoUnBoxing(JDK1.5V)
================================
     valueOf()      -> To convert String/primitive to Wrapper Object
     xxxxValue() -> To convert Wrapper to primitive type.

Integer i = 10;
            |
            |compiler will  make the following change
            |
Integer i = Integer.valueOf(10);

Automatic conversion of primitive type to wrapper type done by the compiler is
called "AutoBoxing".


Integer i1= new Integer(10);
int  i2 = i1;
      |
      |Compiler will do the following change
      |
int i2= i1.intValue();
Automatic conversion of  wrapper type to primtive type done by the compiler is
called "AutoUnBoxing".

Autoboxing and UnBoxing
===================
```

eg#1.

```
class TestApp
{
      static Integer I=10;//AutoBoxing(valueOf())
      public static void main(String[] args)
      {
            int i=I;//AutoUnBoxing(intValue())
            System.out.println(i);//10
      }
}
```

eg#2.

```
class TestApp
{
      static Integer I=0;//AutoBoxing(valueOf())
      public static void main(String[] args)
      {
            int i=I;//AutoUnBoxing(intValue())
            System.out.println(i);//0
      }
}
```

eg#3.

```
class TestApp
{
      static Integer I=null;//AutoBoxing(valueOf())
      public static void main(String[] args)
      {
            int i=I;//AutoUnBoxing(intValue())//NullPointer Exception
            System.out.println(i);//
      }
}
```

a. null
b. 0
c. CE
d. NumberFormatException
e. NullPointerException
f. None of the above

Note:
  Immutable Object -> String,all wrapper classes
(if we try to make a change, then with the change new object will be created)

```
            Integer x=10;
            Integer y=x;
            x++;
            System.out.println(x);//11
            System.out.println(y);//10
            System.out.println(x==y);//false
```

Snippets
======

```
Integer x=new Integer(10);//new object
Integer y=new Integer(10);//new object
System.out.println(x==y);//false
```

```
Integer x=new Integer(10);
Integer y=10;
System.out.println(x==y);//false

Integer x=new Integer(10);
Integer y=x;
System.out.println(x==y);//true

Integer x=10;
Integer y=10;
System.out.println(x==y);//true

Integer x=100;
Integer y=100;
System.out.println(x==y);//true

Integer x=1000;
Integer y=1000;
System.out.println(x==y);//false

Note:
   byte,short,int,long,float,double the buffer concpet which internally jvm
maintains  is "byte range only".
   character  -> 0 to 127
   Boolean   -> always(true or false)
```

for INTEGER buffer is -127to 127
For other data types also same Range sir?
      For all Wrapper classes buffer range is of byte only.
      Character -> 0 to 127
        Boolean   -> true, false

sir, diff. b/w buffer and array r they same?
      Array -> objects which we should create
      buffer-> programmer won't create rather jvm will create for performance.

sir u said that wrapper classes are immutable but how value of x is changed in code snippet that u explained
     Integer x= 10;(immutable)
     Integer y =x;(immutable)
         x++;//x = x+1(since it is immutable change won't happen in the same object
                          rather change will happen and new object will be created)

for boolean type, in a java class when we write boolean/Boolean, both are considered same,
which is not the same with other datatypes, why?
      That is the convention Oracle team /SunMicroSystem team had followed.

one interview qst i got..
wap to get only integer number if any other number or char it should throw error...
in short only integer is allowed
            logical question through if else.

in string you have SCP like we have for Wrapper classes also
    String => SCP
   Wrapper class -> Buffer of range byte

sir can u  explain about capacity  method in StringBuffer class
    StringBuffer sb=new StringBuffer();
      System.out.println(sb.capacity());//16(it indicates the no of characters which can be stored)

  StringBuffer sb=new StringBuffer("sachin");
      System.out.println(sb.capacity());// capacity = 16 + length of String
                                                                  16 + 6
                                                                = 22

StringBuffer sb=new StringBuffer(10);
    System.out.println(sb.capacity());//capacity = 10


If we use value of method explicitly buffer won't be created right?
    Integer i1 =new Integer(10);
    Integer i2 =new Integer(10);
      System.out.println(i1==i2);//false
                  vs
   Integer i1=10;//Autoboxing(valueOf())
   Integer i2=10;//Autoboxing(valueOf())
      System.out.println(i1==i2);//true

Sir buffer which jvm is creating and the StringBuffer class which we used in Mutable concept are same or different?
          Buffer -> Wrapper class and StringBuffer Class buffer are different.

what are different ways to create objects?
    a. new
    b. instanceOf()
    c. Using Factory methods like valueOf()
    d. using clone()

Integer i1=10; will this create any object?
   First time it will use the object already available in Buffer.

sir for string we have same value it will refer same object but in wrapper class if we have same value using new keyword is different?

```
    String s1=new String("sachin");
    String s2=new String("sachin");
                  vs
    Integer i1=new Integer(10);
    Integer i2=new Integer(10);
```

Question
sir interface methods implemented in class those methods by default public or default in class..
and interface methods implemented in abstract class those methods are by default public or
default in abstract class becoz in abstract class all the methods are public by default

```
abstract class Bank{'
      // it is default
      abstract void depositAmount();
      abstract long withDrawAmount();
      abstract String giveCheque();

      void giveNotification(){
            //implementation
      }
}
            vs

interface Bank{
      //abstract and public
      void depositAmount();
      long withDrawAmount();
      String giveCheque();
}
```

Will the valueOf() create an object in buffer ?
      it will use the Object, but can't create an object inside buffer

```
Double c=2.0;//not in range of byte
Double cc=2.0;//not in range of byte
System.out.println(c==cc); It's giving false
```

Sir,In Eclipse can we run the program in  .class file  without .java file?
      yes possible using war/jar file approach using maven tools..

Integer i1=10 by default it is valueof(10) to convert primitive to wrapper class or by default it is
      Integer i1=new Integer(10)?
    Integer i1=10;

```
         compiler will make
    Integer i1=Integer.valueOf(10);
```

is there any way to increase the size of buffer like we have ensureCapacity in strings
   Not possible

String is immutable how can you explain with Memomery also?

```
    String s=new String("sachin");
      s.concat("tendulkar");
    System.out.println(s);// sachin
```

when auto unboxing  comes into picture?

```
      Integer i1=new Integer(10);
      int i2=i1;
                  or
public void m1(int i){//AutounBoxing
      System.out.println(i);
}
  Integer i=10;//autoboxing
      m1(i);
```

Question
string is an type of object and wrapper is also used to wrap string or primitive to an object,
then both things are quite similar,can u please explain me

```
String firstInput = "10";//String
String secondInput = "20";//String
String result = Integer.value(firstInput)+ Integer.valueOf(secondInput);
System.out.println(result);//30
```

wrapper buffer is part of stack or heap?
   heap.

How to print address of object?

```
 Integer i = new Integer(10);
     System.out.println(i);//i.toString() prints data
```

```
String s1= new String("sachin");
String s2=s1.concat("tendulkar");
System.out.println(s1);//sachin
System.out.println(s2);//sachintendulkar
```

Question

```
String str1="10";
String str2="20";
String res1 = Integer.valueOf(str1) + Integer.valueOf(str2);//CE
```

Yesterday you come with 2 interface class with same signature implementing class had override function with super class could you explain it once again again sir?. because i missed it some where

```
interface Right{
      default void m1(){
            System.out.println("hiee");
      }
}
interfae Left{
```

```
	default void m1(){
		System.out.println("hello");
	}
}
class TestImpl implements Left,Right{

	@Override
	public void m1(){
		System.out.println("byee");
		Left.super.m1();//hello
		Right.super.m1();//hiee
	}
	public static void main(String[] args){
		Left l =null;
		l.m1();//CE

		TestImpl t = new TestImpl();
		t.m1();//byee

		Left l =new TestImpl();
		l.m1();//byee
	}
}

interface Right{
	default void m1(){
		System.out.println("hiee");
	}
}
interfae Left{
	default void m1(){
		System.out.println("hello");
	}
}
class TestImpl implements Left,Right{}//Compile Time Error


can u explain casting between wrappers..?
     Casting to happen we need to have parent child relationship, but wrapper
classes are siblings for Number class.

interface Left{
	public void m1();
}
interface Right{
	public void m1();
}
public class TestImpl implments Left,Right{
	@Override
	public void m1(){

	}
}

public final class Integer extends Number
Integer is extending Number class then how they are siblings
       Number
		|-> Byte
		|-> Short
```

```
|-> Integer
```

```
WrapDemo.java:10: warning: [removal] Byte(byte) in Byte has been deprecated and
marked for removal
      Byte g1 = new Byte(grade); // wrapping

JDK9 constructor usage is deprecated use "valueOf"

can we write our own method in runnable interface or only run() is allowed in
runnable interface?
   inbuilt class/interface can't be modified, we need to just take the benift.

if any excepiton occur in one thead is there any effect on other thread in program?
      MulitThreading -> Best suited only when tasks are indepedent of each other
                                It is choose to improve the application performance
by using CPU time effectively.
      main() --------> t1---------> t2
     Thread => seperate stack for execution
     3 Threads -> 3 seperate Stacks

if main method is present in program,that means automatically thread is created for
main thread,or
we need to manually create a thread for main method?
sir can you please explain why hder sir said that main thread is not named after
main i did
not understood that conecpt??
default thread creation happens first ? or bringing the method to the runtime stack
area first ? could you clarify pls
            main() is loaded from method Area to stack and thread will start the
execution.
            JVM will create one thread with that thread it starts the execution.
            When the execution starts the stack should be given for the Thread.
            Inside stack the body of main method is available so the stack name is
"main".

sir in single run() example little confusion came, all the threads created and
linked....
but there is if-else condition ...so once one thread get executed then only others
will get chance ?
  Ans.  totally depends on ThreadScheduler Algorithm(part of JVM)

At what time thread go to dead state : for example t1.join(); -> for this when will
t1 will be in dead state?
      once the complete execution of t1 is done only then Thread will enter into
dead state.

Life cycle of thread is same as "Life cycle of Human Being".
     new/born ----------------> Ready/runnable -----run()-----> running
state--------finsished with execution---------> dead state

Q>
is multithreding with sigle run method used in real time or in industry level
projects?
MultiThreading -> Many tasks
        1 Thread --> 1 Task

can duck the exception in run method when thread.sleep method is used in run
method...?
     interface Runnable{
            public void run();
     }
```

Q>at first point of entry jvm will look forward for public static void method and later invoke
main thread without main method no java application will start to work is it correct?
yes,it results in Exception.

Q> Can write our own custom Scheduler ?
We can write but for this sequence we don't have any "SRS"(interface)

in thread class constructor what all the lines will be there?
Thread class constructor will have a call to Object class constructor.

Q>
sir at the starting when hyder sir print thread name,priority and method name,it was printing
main,5 and main,how 5 cames as a priority for main thread?
JVM will give defautl priorirty for the main thread.
Default priority is "5".

ContextSwitching => Switching the control from one thread to another thread by ThreadScheduler is called "ContextSwitching".
q1. why is it considered that threads are faster in contect switching also?
ContextSwitching in mulithreading is done by JVM(program) compared to ContextSwitching done by OS
since os is not involved we say ContextSwitching is faster at threads level.
q2. what does, Threads use shared memory area mean?
In mulitThreading application jvm will maintain the stack region for sepearte threads and these regions
data can be interchanged b/w threads, so we say Threads works in Shared memory.

if run method was called inside start method then what is the difference between calling direct run method and start method?
Thread is a class present in "java.lang.Thread"
Thread class start method
1. Register Thread with ThreadScheduler
2. It performs all low level memory activities(usage of shared memory)
3. It makes a call to run()

```
Thread t1=new Thread();
    t1.run();
```

sir please explain, is it possible to start thread two times, & can we directly call the
run() instead start(), multiple time this question asked to me while i am giving interview.

```
    Thread t1=new Thread();
      t1.start();
      t1.start();//IllegalThreadStateException
```

Q>

```
Integer i=12;//AutoBoxing using valueOf()
System.out.println(i);//i.toString() will be called so "12" in String format.
System.out.println(i+7);//19(Integer Object) will be printed.
```

Q>

```
Integer i=new Integer("20");
```

```
Integer i2=new Integer("21");
System.out.println(i2);//calls i2.tostring() to print the data
System.out.println(i+i2);// System.out.println(20+21);//41 will be printed.
```

If 3 threads are in runnable state and if thread scheduler has given control to say

thread 1 and if it's not going to sleep or wait or blocked state in between will it

complete it's execution first or in between will thread 2 or 3 will begin it's execution

Thread-1 -> If it is not entered into sleeping state/waiting state then it will complete the execution

Then the T.S will give control to other thread(T2,T3)depends of Algorithm.

can we start a thread again, Sir?

Ans.No

why does for the same application, let's say just incrementing variable by one each time by multiple threads,

we get different outputs each time? why is it so

Ans. Becoz of Concurrency.

How ThreadScheduler decide which thread to help chance ? And is ThreadScheduler behave like queue ?

It is not in the hands of the programmer,totaly depends on the vendor algorithm.

byte bb=12;//compiler will treat as "int", but the datatype u supplied is byte where the value is with in the range.
Byte b=Byte.valueOf(bb);
short s=153;//compiler will treat as "int", but the datatype u supplied is short where the value is with in the range of short
Short S=Short.valueOf(s);
Sir,Above Examples working fine but In below example, why we can't to pass value directly in valueOf() method

```
Byte b1=Byte.valueOf(11);//compile time
Short S1=Short.valueOf(44);//compile time
```

Q>
Sir, We have main thread to execute. In the main method consider we have two threads. To enter the two

threads in running state we need to have a delay in the currently running thread?

```
main(){
            Thread t1=new Thread();
            t1.start();//create a sepearte statck for 't1' and inform Scheduler to schedule the Thread.

            Thread t2=new Thread();
            t2.start();
}
```

need one example for converting primitive to stringtype

```
String s= Integer.toString(10);
System.out.println(s);
```

Q1.****if run method was called inside start method then what is the difference

between calling direct run method and start method?******

Q2.if main thread has default priority as 5 then wt will have 1,2,3,4 and why 5 will execute first instead of 1,2,3,4 and
wt are those whose has priority as 1,2,3,4?

```
public static void main(String[] args) {
            Demo d = new Demo();
            d.start();
            System.out.println(d.getName());
            System.out.println(d.getPriority());
               System.out.println(d.getState());  //TIMED_WAITING what it means?
      }
```

Actually singly thread is being executed in runningstate then how we are calling it as multiple task at a time
overall application -> multiThreading
JVM perspective  -> it is single threading only

if we have created a t1 thread inside main and t1 got exception then entire remaining execution part of main thread also stops?

```
class Demo extends Thread{
      public void run() {
                  int c= 10/0;
      }
}
  class Test{
            public static void main(String[] args){

                        Thread t =new Thread(new Demo());
                        t.start();

                        System.out.println("hello");//prints becoz seperate
stack(Thread scheduler control)

            }
  }
```

output: depends on T.S(if main thread is given a chance then hello) otherwise "Abnormal termination".

is there zombie thread?
   Zombie -> A process which is getting executed without any parent.
   Threads -> Jvm will create main thread----> using main thread -> our threads
                  MainThread(P) -------------> UserDefinedThread(Child)

we are not making the main to daemon thread then y it is showing the exception
      DaemonThread-> A thread which runs in the background and supports other
threads
                                    are called "Daemon thread".
                                Main thread is not a Deamon thread it is normal
thread.
                               Excpetion will come only if
                                    a. u start a thread and then try to make it as
"Daemon".

can we invoke garbage collector implicitly sir ?
      yes ,it is possible through finalize().(Garbagecollector topic)


Q> Can we create our own immutable class?
            Immutable -> Once object created with a data,if we try to make a change
then
                                 that change will not happen on the same object
rather new Object
                                 will be created.
                                             eg: String,Wrapper class.

eg:
  class CreateImmutable{
            //instance variable
            private int i;

            //constructor
            CreateImmutable(int i){
                  this.i=i;
            }

            //instance method
            public   CreateImmutable   modify(int i){
                        if(this.i ==i)
                              return this;//return current object
                        else
                              return new CreateImmuatable(i);//return by creating
new object
            }

 }
public class Demo{
      public static void main(String[] args){
                  CreateImmutable c1= new CreateImmutable(10);
                  CreateImmutable c2= c1.modify(10);
                  CreateImmutable c3= c1.modify(100);
                  System.out.println(c1==c2);//true
                  System.out.println(c1==c3);//false
      }
}

is it possible to set priority for daemon threads? if yes then is this legal or
possible

: t1.setPriority(10) where t1 is daemon thread?
Daemon thread priority if u give then also no impact as they run behind the main thread

synchronized
When multiple threads tries to act on single resource simultaneously there would be a problem
of "Data Inconsistency". This problem can be avoided through "synchronized keyword".
synchronized can be applied at 2 levels
a. method level
a. instance method level-> JVM will put lock of the Thread at Object level
b. static method level -> JVM will put lock of the Thread at class level.
b. block level

```
class Demo{
      public synchronized void m1(){

      }
      public synchronized void m3(){

      }

      public static synchronzied void m2(){

      }
      public static synchronzied void m4(){

      }
      public  void m5(){

      }

}
```

Threads
t1-> m1()[Object level lock is required]
t2-> m3()[t2 will be waiting till object level lock is released by t1]
t3-> m2()[Class level lock is required]
t4-> m4()[t4 will be waiting till Class level lock is released by t3]
t5-> m5()[No lock is required]

Hi sir what happens if we create two threads and assign them with same priority. Will conflict occur?
No, becoz TS uses differetnt algorithm to assign the cpu time.

how a string is used as a resource in synchronized block
syntax: synchronized(object){

}

why main thread has default priority of 5 as max priority allowed is 10 for higher priority
JVM thread priority
1 5 10
MIN AVG MAX

Thread.sleep(10) how it invoke sleep for current thread?
Thread.currentThread().sleep(1000);

```
pubic class Thread{
            public static void sleep(int millisecond){
                        //logic of sleep
            }
}
```

If join() is not best case, what’s its purpose please?
task are dependent on each other and they are such that without completeing one
task other task should not be continued, in these scenarious we use "join()".
eg:

```
wedding card                distrubution of              book wedding
  printed                         wedding card                  hall
      t1                                        t2
t3
                                        |
|
                                        |
|
                                    t1.join()
t2.join()
```

if we give high priority for thread1 and lower priorities for thread2 and 3 will thread1 go to
running state first or still it depends on thread scheduler's internal logic?
thread1 -> 10
thread2 -> 1
thread3 -> 1
since the thread1 is having high priority it will enter into running state.
b/w thread2 and thread3 Threadscheduler decides whom to give the cpu time.

how can we say stringbuffer is synchronized,reason?
StringBuffer -> Jdk1.0
All the methods present in StringBuffer are synchronized so we say the StringBuffer
resource is "synchronized".
Any resource if it is synchornzied it means the resource is "Thread safe".

how to check how mamy threads are runing currently
Thread.isAlive() -> returns boolean value through which u can check
vch thred is alive.

q. sir, as you said in try(R) R should be the classes which implement AutoClosable(I),
so are there any resource classes which do not implement it but are required in our programming?
In that case we should write finally block, right??

try(R) ----->An object which implements java.io.AutoCloseable
multiThreading-> resource it is any Object
if u r using try(R) compulsorily it should implement AutoCloseable otherwise use
try{} catch(){}finaly{}

can you pls explain the relationship (if any) between Java Thread, OS Thread and

number of processors on the host?
Java -> Architectural neutral
it would not worry about the underyling os and its architecture becoz of JVM.
OS -> os concepts and its alogrithms
no of processors -> Microprocessor architecture.

sir is it the rule that in case of single resources for multiple thread object we use synchroed keyword
if required? or we can use synchrozed for different resources?
synchronzied -> one resource used by multiple threads.
to avoid data inconsistency we use "synchronized".

Can we set main Thread priority less than user-defined threads? -> please answer
main() -> 5
t1 -> change(t1.setPriority).
If we change the priority order of execution will be different and we can't predict results.

MulitThreading
   1. Different ways of creating a thread
                a. Extending Thread class
                b. Implements Runnable interface
   2. Setting a name and getting name from the Thread
                a. public void setName(String name)
                b. public String getName()

Lifecycle of a Thread
          new/born -----start()--------> ready/runnable ---ts allocates cpu time---------> running----run() complets---------> deadstate

   3. Methods to prevent a Thread from execution
                a. join() ->  To make another thread to wait till it finsihes the execution.
                b. sleep() -> To stop/pause the execution of a thread for sometime
   4. Synchronization
                => this concept is applicable at method level and block level.
                => if we apply synchronization at block level or at method level then only one thread
                        is allowed to execute the block or a method.
                => Advantage -> it resolves the problem of "Data Incosistencey/race condition".
                => DisAdvantage->It increase the waiting time for other threads so it affects the peformance of the
                                                system.

Note: In java we have 2 levels of lock
                a. class level lock  => A thread which needs to execute static synchronized block/method needs
                                                        class level lock.
                                                        This lock is very unique at the class level.
                b. object level lock => A thread which needs to execute synchronized block/method needs
                                                         object level lock.
                                                         This lock is very unique at the Object level.


InterThread Communication
-------------------------------------
      Two threads should interact with each other, how?

eg: Producer Consumer Problem

ProducerEnd
      => Producer duty is to produce the data and once the data is produced update the variable
           called "DataProvider" to true
      => This action should be done by "Producer Thread"

```
for (int i = 1; i <= 10; i++) {
                try {
                      sb.append(i + ": ");
                      Thread.sleep(100);
                      System.out.println("appending");
                } catch (InterruptedException e) {
```

```
                        e.printStackTrace();
                }

}
dataProvider = true;


Consumer End
      => Consumer Thread should consume the data produced by the Producer
      => Consumer  Thread should check the dataprovider status,if it is true
consume the data
           otherwise sleep for some time and again check for the dataprovider
status

while(producer.dataProvider == false) {
                try {
                        Thread.sleep(10);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
}

//consume the data produced by the producer
System.out.println(producer.sb);

In the above code when the interaction happens b/w 2 threads, always the consumer
thread is ready for
consumption,but the consumer thread will get the data only when the dataProvider
value is set to true.
This increases the waiting time of a thread and makes the cpu time idle,through
which communication b/w
2 threads wont be efficient.
      To reduce the effieciency problem we use the methods given by Object class
a.wait()
b.notify()
c.notifyAll()

Note:
 wait(),notify(),notifyAll() methods are present in Object class not in Thread
class.
 If any thread has to call wait(),notify(),notifyAll() then that thread should be
the owner of the thread.
      We say the thread as owner iff the thread has the lock of the object.
 If the thread calls notify(),notifyAll() and wait() and the thread is not a owner
then it would result in
      RE: "IllegalMonitorStateException".

wait() -> Whichever thread is excpecting the updated result from the object that
thread should call wait method.
               Whenever wait() is called automatically that thread will release the
lock of the Object to the other
               thread to use that lock.

notify()     -> Which ever thread wants to update the Object, that thread should
call notify() to the Other thread(one).
notifyAll() ->Which ever thread wants to update the Object, that thread should call
notifyAll() to the Other
                     waiting threads(many)
```

Note: only wait(),notify(),notifyAll() have the mechanism to release the automatically, where as sleep(),join() these methods
can't release the lock.

FAQ:
Why wait(),notify(),notifyAll() methods are part of Object class, why not Thread class?
ans. These methods will be used by the thread on different types of Objects like StringBuffer,Student,Customer,Account,.....
For every object the parent class is Object,so these methods comes from object class.
=>methods like join,yield,sleep will be applied only on Threads, so only these methods
are part of Thread class not Object class.

What is the difference b/w notify() and notifyAll()?
notify() -> It will give notification only to one thread which needs the lock of that object
notifyAll() -> It will give notification to mulitple threads which needs the lock of that object.

Example of wait() and notify()
---

```
class Demo extends Thread
{
	//data is updated
	int total = 0;

	public void run(){

		//producer thread
		synchronized(this){
			System.out.println("Child thread starts the calculation");//step-2

			//sum of first 100 numbers
			for (int i =1;i<=100 ; i++)
			{
				total = total + i;
			}
			System.out.println("Child thread is giving the notifcation call");//step-3
			this.notify();

		}
	}
}
class Test
{
	public static void main(String[] args)  throws Exception
	{
		Demo d = new Demo();
		d.start();

		//consumer thread
		synchronized(d){
			System.out.println("Main Thread is calling  wait() method....");//step-1
			d.wait();
```

```
                System.out.println("Main Thread got the notification
call");//step-4
                System.out.println(d.total);//5050
            }
      }
}
```

Output

```
Main Thread is calling  wait() method....
Child thread starts the calculation
Child thread is giving the notifcation call
Main Thread got the notification call
5050
```

Note:

Demo class had total variable

Main thread[5]

- => needs Demo class total variable with proper value(5050)
- => lock is applied on Demo object and call wait()
- => wait() releases the lock of Demo object and main thread enters into sleeping state.

User Defined Thread[5]

- -> should update total variable in Demo class and it should send the notification
- -> notify() is used so the code should be in synchronize region
- -> it needs the lock of Demo object, now the calculation is started to update.
- -> it will give the notification

Producer consumer problem

---

refer : InterThreadCommunicationApp,
InterThreadCommunication.png

Writing the code in lambda Expression style

---

refer: LambdaExpressionApp

new Runnable() is for creating object which implements the runnable interface but no class name right?
  Ans. Yes (Ananomyos implementation)

notify have inbuilt logic to inform to the exact thread which release the lock?
   notifyAll does not have this?
notify() -> afte the updation is done it would just notify to the waiting thread
                  eg:
                              obj1.wait()(1 thread)
                                    |
                                    | ==========> obj1.notify()
                                    |
                               waiting state

notifyAll() -> after the updating is done it will notify all the waiting threads

                          obj1.wait() =====> 30 threads
                         obj2.wait() =====> 60  threads

                         obj1.notifyAll()---> notification will be sent to 30 threads


Q>
```
psvm(Striing[] args){
A a = new A();
B b = new B();
 Thread t1 = new Thread(a);
 Thread t2 = new Thread(b);
      t1.start();
      t2.start();
}
```

Sir here will the main method immediately execute t1.start() and t2.start()?
Will the thread scheduler allocate cpu for t1 thread, will make t1 in running and main thread in  runnable
(in the line t1.start())?
Could you please explain that sir?

3 threads
      a. main thread
      b. t1 thread
      c. t2 thread

Q>
when we have sb.notify() how will will waiting thread know from which point of code to start?
      sb.notify() and sb.wait() both are under synchronized and they share the common communication channel.

            sb.wait() -> release the lock and enter into waiting state
                  |
            sb.notify() -> use the lock update the object and send the notification to sb.wait()


can you please tell the difference between join and yield?
    join() ->  stop ur execution and wait for another thread to execute later join
   yield() -> pause ur execution for sometime and check whether there are any

threads of same priority available or
not, if not continue ur execution, if available give the other thread a chance.

so how we are assuming consumer will take the lock , it is like lock will not be given to producer and given to
consumer first?
Producer ->produce the item and sent the notification to the consumer[notify()]
Consumer -> consume the item if avaialble otherwise wait as long as the producer produces the item
and gets the notification.[wait]

consumer.start();//no enter into waiting state by releasing the lock
producer.start()// get the lock and update the data and finally relase the lock.

Q>
main thread -> trigger logic[t2.start(),t1.start()]
producer thread -> notify()[produces the item and gives the notification for waiting thread expecting the lock]
consumer thread -> wait()[lock relased and waiting for notification]

Q>
```
class ThreadB extends Thread{
            int total =0;
            public void run(){
                        for(;;;)
                              update total;
            }
}
class Test{
      p.s.v.m(String... args){
            ThreadB b=new ThreadB();
            b.start();

            System.out.println(b.total);
      }
}
```
main thread[5] ->  System.out.println(b.total) //0
user thread[5]->
```
      int total =0;
      for(;;)
            update total
```

Q>
```
main(String.. args){
      Thread t1=new Thred();
      Thread t2 =new Thread();

            t1.start();
            t2.start();

}
```
3 threads
main thread(currently executing)
t1
t2[ got  a chance]

```
Q>
     t1 -----> ;;;;;;;;;;;;;Thread.sleep(100);;;;;;;;;;
     t2 -----> t1.join()[waiting state]
     t3------> ;;;;;;;;;;;;;;;;Thread.sleep(4000);;;;;;;

Q>
class Thread{
      public void start(){
            //register the thread with ThreadScheduler
            // perofm low level memeory management
            //makes a call to run()
      }
}
class Test extends Thread{
            public void start(){
                  super().start();
            }
}

SharedResource usage in effiecient way without datainconsistency(synchronized
region)

t1[5]---> b.wait()
t2[5]--->;;;;;;; b.notify()
```

Variable                -> hold data(only one data)
ArrayVaraible      ->10,20,30,40

Advnatage of Array
----------------------
   1. Large volume of data can be holded by single varaible
   2. To access the data we use "index".

DisAdvantage of Array
----------------------------
   1. It expects continous memory
   2. Once Array is fixed, u can't increase or decrease the size based on our requirement.
   3. It can hold only homogenous data.
   4. Array is not implemented based on any standard datastructure,so  ready made methods are not avaiable
       to perform some operation like sorting,searching,etc....It increase the burden on developers to write
        their own logic to perform these common operations.
                  int a[5]= { 10,20,0,5,2,1};
                        Arrays.sort(a);

To overcome all the limitations of an array we use "Collections"
  Advantage of Collections
      1. Collections are growable in nature,u can increase or decrease the size of an Collection.
       2.  It can hold both homogenous and heterogenous Objects.
      3. All Collection(I)implementation classes are built by following some standard DataStructure.
      4. The data stored inside collection would always be in Object type only
                  primtive ---autoboxing----------> object

Collection(I)
       |
  List(I)

List(I)
1. If we want insertion order to be preserved and duplicates to be allowed
2. Index plays a vitol role here.
3. internally datastructure followed is "Array".

      eg: ArrayList,LinkedList, Vector,Stack


Set(I)
 1. Insertion order won't be preserved and duplicates should not be allowed.
 2. index won't play any vitoal role
 3. internally datastrucure follwed is "Hash Table".

      eg: HashSet,LinkedhashSet

SortedSet(I)
 1. Inside Set, if the elements/Object has to be Sorted then we need to opt for SortedSet.
            eg: TreeSet

Note: For Collection interface the parent class is "Iterable"

Iterable -> The entity which can be used in iteration.

eg: loops

When to use List, Set?
List -> This implementaion classes should be used for the following conditions
a. insertion order and duplicates are allowed
eg: ArrayList, LinkedList

ArrayList -> We add the elements, we need to remove
The array size should be shrinked at the runtime(more time -> don't use)
If the frequent operation is read operation then we use ArrayList.
ArrayList implements an interface called "RandomAccess".
Note:ArrayList and Vector are the only 2 classes which implements "RandomAccess".

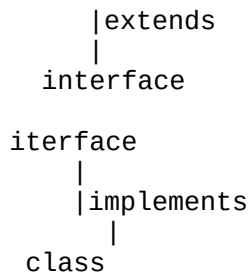
LinkedList-> We add the elements, we need to remove
linkedlist -> data will be stored in scattered manner not in continous mode.
deletion is easy,but reading is tough.
if the frequent operation is deletion then we use LinkedList

Marker Interface
An interface which does not contain any abstract methods that interface is called "Marker Interface".
If an class implements Marker interface, then that class "object" will get additional funcitionality at the runtime by the
JVM.
eg: RandomAccess(searching is fast in the Collection object), Serializable(makes the object transportable),.....

Set
------
Set
|
SortedSet(I)
| implements
TreeSet
=> Best suited to keep the elements in sorted order
=> Sorted order can be "Ascending/Descending" order.
=> 2 important interfaces
a . Comparable
b. Comparator

```
int a[5] = {10,20,30,40,50};
System.out.println(a[3]);// directly 40 printed by accessin at the memory level

ArrayList al =new ArrayList();
al.add(10);
al.add(20);
al.add(30);
System.out.println(al.get(2));//performance is low compared with Array.
```

Relation b/w 2 interface
iterface
|


|extends
|
interface
iterface
|
|implements
|
class

```
package: java.util.*;
                        |-> refers to all classes/interfaces/enum present in the
current package(util package)

Inside any of Collection What kind of data is been stored?
      A. primitive(no)
                  Even if the programmer gives primitive data, internally from
JDK1.5Version JVM will use
                  wrapper class concepts to convert primitive type of data to
object and it would be stored.
                              eg:    al.add(8);
                                                |------
>al.add( Integer.valueOf(8));
      B. Object(yes)

Collection(I) ----> it is the root interface for all type of collection
List(I)
      a. ArrayList
      b. LinkedList
                  To utilise the scattered/dispursed memory in efficient way we use
LinkedList.
                  a. SinglyLinkedList
                  b. DoublyLinkedList
                        Note: All the Node creation and address maintainence is
totally managed by JVM since
                                    java does not support the concept of pointers for
programmer, this only is the
                                  reson to say java is "abstract high level
language".

      c. Vector  -> All the methods of Vector is synchronized(Thread safe, slow in
execution)
                        this class also implements Random Access interface so it is
best suited for "Retreival operation",
                        but it is not suited for insertion and deletion at the
middle.
                              methods
                                    a. addElement(Object o)
                                    b. removeElement(Object o)
                                    c. removeAllElements()
                                    d. Object elelementAt(int idnex)
                                    e. Object firstElement()
                                    f.  Object lastElement()
            a. Stack
                  Constructors
                              a. Stack s =new Stack();
                        methods
                              a.  Object push(Object o)
                              b.  Object pop() -> removes and returns the top
element of  the stack
                              c.  boolean empty()
                              d. Object peek() -> It will give the top element of
the stack without removal
                              e. search(Object) -> it returns the offset if the
element is present,otherwise it returns -1.


There are 3 cursor in java
   -> Inside collection data would stored as Objects.
```

=> After storing the data as Objects, it is comon requirement to take the Object one by one from Collection
=> To do this we have cursors in java
a. Enumeration(I) -> It is applicable for legacy classes only.

```
public interface Enumeration{
  public abstract boolean hasMoreElements();//will check in the collection whether elements are there are not
  public abstract E nextElement(); // this method will get the current cursor data and makes the cursor to point to next collection object
}
```

eg#1.

```
import java.util.*;

class Test
{
      public static void main(String[] args)
      {
            Vector v = new Vector();
            for (int i=1;i<=10 ; i++)
            {
                  v.addElement(i);
            }
            System.out.println(v);//internally v.toString() is called.

            Enumeration e = v.elements();// to get the cursor
            System.out.println("Reading elements one by one from collection");
            while (e.hasMoreElements())
            {
                     Integer data=(Integer)e.nextElement();
                     System.out.println(data);
                     if (data%2==0){
                           System.out.println(data + ":  is an even number");
                     }
            }


      }
}
```

Limitation

1. It is applicable only on legacy classes
2. using this cursor we can perform only read operation and we can't perform remove operation
3. To resovle this problem only we use "iteator".

b. Iterator(I) ----> Universal Cursor( applied on any type of Collection Object)

```
public interface Iterator {
  public abstract boolean hasNext();//check whether the collection has next element or not
  public abstract E next(); //retreive the element and takes the cursor to the next element
  public void remove(); //to remove the object from collection.
  public void forEachRemaining(java.util.function.Consumer<? super E>);//Stream api's
}
import java.util.*;
```

```
class Test
{
      public static void main(String[] args)
      {
            ArrayList al = new ArrayList();
            for (int i=1;i<=10 ; i++)
            {
                  al.add(i);
            }
            System.out.println(al);//internally al.toString() is called.

            Iterator itr = al.iterator();
            System.out.println("Reading elements one by one from collection");
            while (itr.hasNext())
            {
                     Integer data=(Integer)itr.next();
                     System.out.println(data);
                     if (data%2==0)
                           System.out.println(data +" : is an even number");
                     else
                           itr.remove();


            }
            System.out.println(al);


      }
}
```

Limitation

1. Using this cursor we can move only in forward direction, not in backward direction so we say the cursor is "UniDirectional cursor".
2. Using this cursor we can perform only remove operation, operations like adding the object,replacing the object is not possible.
3. To overcome this limitation we need to use ListIterator.

c. ListIterator(I)

```
public interface java.util.ListIterator<E> extends java.util.Iterator<E> {

//for forward traversing
  public abstract boolean hasNext();
  public abstract E next();
  public abstract int nextIndex();

// for backward traversing
  public abstract boolean hasPrevious();
  public abstract E previous();
  public abstract int previousIndex();

//for operations like add,remove and update
  public abstract void set(E);
  public abstract void add(E);
  public abstract void remove();
}
```

Limitation

a. Eventhough it is a powerful cursor it can be applied only on List(I) implementation object,but not on all Collections.

Revise and get back for doubts
---
Set(I)
SortedSet(I)
NavigableSet(I)
Queue(I)

Concurrent Collections(java.util.concurrent.*)
failfast -> while one thread is trying to perform iteration on collection Object and if another thread is trying to do some structural modification to the same collection object, then immedieatly iterator would fail, by resulting in an exception called ConcurrentModificationException,such type of iterators are called as "FailFastIterator".

eg:

```
import java.util.*;

class Test
{
	public static void main(String[] args)
	{
		ArrayList al = new ArrayList();
		al.add("A");
		al.add("B");
		al.add("C");

		Iterator itr = al.iterator(); // fail fast iterator
		while (itr.hasNext())
		{
			String data = (String) itr.next();
			System.out.println(data);

		//Assume one more thread is doing up modification on ArrayList
			al.add("D");//Trying to change the structure of an ArrayList
		}
	}
}
```

If we don't want the exception to occur even during mulithreading events then prefers using "Concurrent Collections" which supports concurrent modifcations.

fail safe: while one thread is trying to perform iteration on collection Object and if another thread is trying to do some structural modification to the same collection object, then also iteration won't fail becoz the iterator is "fail safe iterator".
here exception wont occur becoz every update operation will be performed on seperate cloned copy.

eg#1.

```
import java.util.concurrent.*;
import java.util.*;

class Test
{
	public static void main(String[] args)
	{
		CopyOnWriteArrayList al = new CopyOnWriteArrayList();
```

```
            al.add("A");
            al.add("B");
            al.add("C");

            Iterator itr = al.iterator(); //fail safe iterator
            while (itr.hasNext())
            {
                  String data = (String) itr.next();
                  System.out.println(data);

            //Assume one more thread is doing up modification on ArrayList
                  al.add("D");//Trying to change the structure of an ArrayList
            }
            System.out.println(al);
      }
}
```

It will be discussed in tommo session

---

Map(I)
NavigableMap(I)
SortedMap(I)

Today topics of discussion
---
1. Summary of Collection
2. Map and its internal working
3. Map and its implementation classes
4. Comparable and Comparator Interface

Map
===
=> It is not a child interface of Collection.
=> If we want to represent group of Objects as key-value pair then we need to go for Map.
=> Both keys and values are Objects only
=> Duplicate keys are not allowed but values are allowed.
=> Key-value pair is called as "Entry".

refer: MapHierarchy.png

Map methods
---
1. It contains 12 methods which is common for all the implementation Map Objects
a. Object put(Object key,Object value) // To add key,value pair
b. void putAll(Map m) // To add another map
c. Object get(Object key) // To get the value based on key
d. Object remove(Object key) //To remove an entry based on key
e. boolean containsKey(Object key) //Check whether it contains key or not
f. boolean containsValue(Object value)//Check whether it contains value or not
g. boolean isEmpty() //To check wheter the Map is empty or not
h. int size() //To get the size of a Map
i. void clear() //To remove all Entry from a map

views of a Map
j.Set keySet() //Convert the key's of Map into Set for reading purpose
k.Collection values() //Convert the values of Map into Collection for reading purpose
l.Set entrySet() // Convert whole Entry of Map into Set for reading purpose.

Entry(I)
---
1. Each key-value pair is called Entry.
2. Without existence of Map,there can't be existence of Entry Object.
3. Interface entry is defined inside Map interface.

```
interface Map{
      interface Entry{
            Object getKey();  //To get the key using Map.Entry Object
            Object getValue();//To get the value using Map.Entry Object
            Object setValue(Object newValue);//To update the value Using Map.Entry Object
      }
}
```

HashMap
---
Underlying DataStructure: Hashtable
insertion order : not preserved
duplicate keys : not allowed

```
 duplicate values        : allowed
 Heterogenous objects    : allowed
 null insertion          : for keys allowed only once,but for values can be any no.
 implementation interface: Serializable,Cloneable.

Difference b/w HashMap(c) and Hashtable(c)
==========================================
 HashMap  => All the methods are not synchronized.
 Hashtable => All the methods are synchronzied.

 HashMap   => At a time multiple threads can operate on a Object, so it is not
ThreadSafe.
 Hashtable  => At a time only one Thread can operate on a Object, so it is
ThreadSafe.

 HashMap   => Pefromance is high.
 Hashtable  => Performance is low.

 HashMap   => null is allowed for both keys and values.
 Hashtable  => null is not allowed for both keys and values,it would result in
NullPointerException.

 HashMap   => Introduced in 1.2v
 Hashtable => Introduced in 1.0v

Constructors
============
 1. HashMap hm=new HashMap()
            //default capacity => 16, loadfactor => 0.75(upon increase of data by
75% automatically

       size of HashMap will be doubled)

 2. Hashmap hm=new HashMap(int capacity);

 3. HashMap hm=new HashMap(int capacity,float fillration);

 4. HashMap hm=new HashMap(Map m);

eg#1.
eg#1.
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            HashMap hm = new HashMap();
            hm.put(10,"sachin");
            hm.put(7,"dhoni");
            hm.put(18,"kohli");
            hm.put(45,"rohith");
            System.out.println(hm);//hm.toString() will be called

            Set s = hm.keySet();//To get the keys from Map
            System.out.println(s);
            System.out.println(s.getClass().getName());

            System.out.println();
```

```
            Collection c = hm.values();//To get the values from Map
            System.out.println(c);
            System.out.println(c.getClass().getName());

            System.out.println();

            Set mapData = hm.entrySet();//To get the K,V from Map as Set
            System.out.println(mapData);
            System.out.println(mapData.getClass().getName());

            System.out.println();
            Iterator itr = mapData.iterator();
            while(itr.hasNext()){
                  //Object is return type of next(), i am converting to Map.Entry
Object to call methods of Entry interface
                  // The data of Map.Entry is stored as object
                  Map.Entry data =(Map.Entry)itr.next();
                  System.out.println(data.getClass().getName());
                  System.out.println(data.getKey() + ": " + data.getValue());
                  if (data.getKey().equals(10))
                  {
                        data.setValue("SRT");
                  }
            }
            System.out.println();
            System.out.println(hm);
      }
}
```

```
LinkedHashMap
=============
 => It is the child class of HashMap.
 => It is same as HashMap,but with the following difference

       HashMap             => underlying datastructure is hashtable.
       LinkedHashMap => underlying datastructure is LinkedList + hashtable.

       HashMap             => insertion order not preserved.
       LinkedHashMap => insertion order preserved.

       HashMap             => introduced in 1.2v
       LinkedHashMap => introduced in 1.4v


eg#1.
import java.util.*;

class Test
{
      public static void main(String[] args)
      {
            LinkedHashMap hm = new LinkedHashMap();
            hm.put(10,"sachin");
            hm.put(7,"dhoni");
            hm.put(18,"kohli");
            hm.put(45,"rohith");
            System.out.println(hm);//hm.toString() will be called

            System.out.println();
```

```
            Set s = hm.keySet();//To get the keys from Map
            System.out.println(s);
            System.out.println(s.getClass().getName());

            System.out.println();

            Collection c = hm.values();//To get the values from Map
            System.out.println(c);
            System.out.println(c.getClass().getName());

            System.out.println();

            Set mapData = hm.entrySet();//To get the K,V from Map as Set
            System.out.println(mapData);
            System.out.println(mapData.getClass().getName());

            System.out.println();
            Iterator itr = mapData.iterator();
            while(itr.hasNext()){
                  Map.Entry data =(Map.Entry)itr.next();
                  System.out.println(data.getKey() + ": " + data.getValue());
                  if (data.getKey().equals(10))
                  {
                        data.setValue("SRT");
                  }
            }
            System.out.println();
            System.out.println(hm);
      }
}



import java.util.*;

class Test
{
      public static void main(String[] args)
      {
            HashMap h = new HashMap();

            //Creating a key
            Integer i1= new Integer(10);
            Integer i2= new Integer(10);

            //Adding the data to HashMap
            h.put(i1,"sachin");
            h.put(i2,"Messi");

            System.out.println(h);//{10=Messi}

      }
}
```

IdentityHashMap
==========

It is same as HashMap, with the following differences

a. In case of HashMap,jvm will use equals() to check whether the keys are

duplicated or not.
      equals() => meant for ContentComparison.
b. In case of IdentityHashMap,jvm wil use == operator to identify whether the keys are duplicated.
    or not.
                        refer: HashMap vs IdentityHashMap .png

Note:
Garbage collector actions
---
```
import java.util.*;
class Test
{
      public static void main(String[] args) throws Exception
      {
            Employee e = new Employee();
                  ;;;;;;;;
                  ;;;;;;;;
                  ;;;;;;;;

            e = null;//Garbage object
            System.gc();//Informing JVM to active GC thread to clean garbage object
            Thread.sleep(5000);
      }
}
class Employee
{
      @Override
      public void finalize(){
            System.out.println("Cleaning the object");
      }
}
```

WeakHashMap
==========
 It is exactly same as HashMap, with the following differences.
1. HashMap will always dominate Garbage Collector, that is if the Object is a part of HashMap
   and if the Object is Garbage Object, still Garbage Collector won't remove that Object from
   heap since it is a part of HashMap. HashMap dominates GarbageCollector.
2. Garbage Collector will dominate WeakHashMap, that is if the Object is part of WeakHashMap and
   if that Object is Garbage Object, then immediately Garbage Collector will remove that Object
   from heap even though it is a part of WeakHashMap, so we say Garbage Collector dominates
   "WeakHashMap".

eg#1.
```
import java.util.*;

class Test
{
      public static void main(String[] args) throws Exception
      {
            HashMap hm = new HashMap();
            Temp t= new Temp();
            hm.put(t,"shri");
```

```
            System.out.println(hm);//{temp=shri}

            t= null;//Making eligible for Garbage Collection
            System.gc();//Triggering garbage collector thread to clean 't'
            Thread.sleep(5000);

            System.out.println(hm);//{temp=shri}

      }
}

class Temp
{
      @Override
      public String toString(){
                  return "temp";
      }

      @Override
      public void finalize(){
            System.out.println("cleaning temp object");
      }
}
eg#2.
import java.util.*;

class Test
{
      public static void main(String[] args) throws Exception
      {
            WeakHashMap hm = new WeakHashMap();
            Temp t= new Temp();
            hm.put(t,"shri");
            System.out.println(hm);//{temp=shri}

            t= null;//Making eligible for Garbage Collection
            System.gc();//Triggering garbage collector thread to clean 't'
            Thread.sleep(5000);

            System.out.println(hm);//{}

      }
}
class Temp
{
      @Override
      public String toString(){
                  return "temp";
      }

      @Override
      public void finalize(){
            System.out.println("cleaning temp object");
      }
}
```

Hashtable:
=> The Underlying Data Structure for Hashtable is Hashtable Only.
=> Duplicate Keys are Not Allowed. But Values can be Duplicated.

=> Insertion Order is Not Preserved and it is Based on Hashcode of the Keys.
=> Heterogeneous Objects are Allowed for Both Keys and Values.
=> null Insertion is Not Possible for Both Key and Values. Otherwise we will get Runtime
Exception Saying NullPointerException.
=> It implements Serializable and Cloneable,but not RandomAccess.
=> Every Method Present in Hashtable is Synchronized and Hence Hashtable Object is Thread
Safe.

Constructors:
1) Hashtable h = new Hashtable();
Creates an Empty Hashtable Object with Default Initial Capacity 11 and
Default Fill Ratio 0.75.
2) Hashtable h = new Hashtable(intinitialcapacity);
3) Hashtable h = new Hashtable(intinitialcapacity, float fillRatio);
4) Hashtable h = new Hashtable(Map m);

eg#1.

```
import java.util.*;

class Test
{
	public static void main(String[] args)
	{
		Hashtable hm = new Hashtable();//Default capacity is 11
		hm.put(new Temp(5),"A");
		hm.put(new Temp(2),"B");
		hm.put(new Temp(6),"C");
		hm.put(new Temp(15),"D");
		hm.put(new Temp(23),"E");
		hm.put(new Temp(16),"f");

		System.out.println(hm);
	}
}
class Temp
{
	int i;
	Temp(int i){
		this.i=i;
	}
	public int hashCode(){
		return i;
	}

	public String toString(){
		return i+" " ;
	}
}
```

Note;

```
public class Object{
	public native int hashCode();//Code is not from java language it will binded during runtime

	@Override
	public String toString(){
			return getClass().getName()+ "@" +
```

```
Integer.toHexString(hashCode());
	}
}

eg#1.
class Test{
	@Override
	public int hashcode(){
		return 10;
	}
}
Test t1= new Test();//Test@A
Test t2= new Test();//Test@A

class Student{
	int rollNo;
	Student(int rollNo){
		this.rollNo = rollNo;
	}
	@Override
	public int hashCode(){
		return rollNo;
	}
}
Student std1= new Student(10);//Student@A
Student std2= new Student(100);//Student@64

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.
3. If the objects are stored according to hashCode searching will become very
efficient
   (The most powerful search algorithm is hashing which will work based on
hashCode).
4. If we didn't override hashCode() method then Object class hashCode() method will
be executed   which generates hashCode based on address of the object but it
doesn't mean hashCode represents    address of the object.
5. Based on our programming requirement we can override hashCode() method to
generate our own   hashcode.
6. Overriding hashCode() method is said to be proper if and only if for every
object we have to generate a unique number as hashcode for every object.

 public native int hashCode()
	=> It generates the hashCode based on the address of the Object.

 public String toString(){
	return getClass().getName() + "@" + Integer.toHexString(hashCode());
 }
 here getClass().getName() =>
classname@hexa_decimal_String_representation_of_hashCode

Example1:
class Student {
	public int hashCode() {
		return 100;
	}
}
```

It is improper way of overriding hashCode() method because for every object we are generating same
hashcode.

Example2:

```
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.

toString() method vs hashCode() method

==========

eg#1.

```
class Test{
      int i;
      Test(int i){
            this.i=i;
      }
      public static void main(String[] args){
            Test t1=new Test(10);
            Test t2=new Test(100);
            System.out.println(t1);//Test@....
            System.out.println(t2);//Test@....
      }
}
```

Object==>toString() called.
Object==>hashCode() called.
In this caseObject class toString( )method got executed which is internally calls Object  class hashCode( ) method.

eg#2.

```
class Test{
      int i;
      Test(int i){
            this.i=i;
      }
      public int hashCode(){
            return i;
      }
      public static void main(String[] args){
            Test t1=new Test(10);
            Test t2=new Test(100);
            System.out.println(t1);//Test@A
            System.out.println(t2);//Test@64
      }
}
```

Object==>toString() called.
 Test  ==>hashCode() called.
In this case Object class toString( ) method got executed which is internally calls Test
class hashCode( ) method.

eg#3.

```
class Test{
      int i;
      Test(int i){
            this.i=i;
      }
      public int hashCode(){
            return i;
      }
      public String toString(){
            return i+"";
      }
      public static void main(String[] args){
            Test t1=new Test(10);
            Test t2=new Test(100);
            System.out.println(t1);//10
            System.out.println(t2);//100
      }
  }
```

Output:
10
100
In this case Test class toString() method got executed and hashCode() wont be executed.

Note :
1. if we are giving opportunity to Object class toString() method it internally calls hashCode()  method. But if we are overriding toString() method it may not call hashCode() method.
2. We can use toString() method while printing object references and we can use hashCode() method  while saving objects into HashSet or Hashtable or HashMap

Properties:
=> It is the Child Class of Hashtable.
=> In Our Program if anything which Changes Frequently (Like Database User Name, Password, Database URLs Etc)
     Never Recommended to Hard Code in Java Program.
=> Because for Every Change in Source File we have to Recompile, Rebuild and Redeploying
     Application and Sometimes Server Restart Also Required, which Creates Business Impact to the Client.
=> To Overcome this Problem we have to Configure Such Type of Propertiesin Properties File.
=> The Main Advantage in this Approach is if a there is a Change in Properties File, to Reflect that Change Just Redeployment is Enough,
      which won't Create any Business Impact.
=> We can Use Properties Object to Hold Properties which are coming from Properties File.

Constructor:
      Properties p = new Properties();

1) public String getProperty(String pname);
      To Get the Value associated with specified Property.
2) public String setProperty(String pname, String pvalue);
      To Set a New Property.
3) public Enumeration propertyNames();It Returns All Property Names.
4) public void load(InputStream is);

```
      To Load Properties from Properties File into Java Properties Object.
5) public void store(OutputStreamos, String comment);
      To Store Properties from Java Properties Object into Properties File


eg#1
import java.util.*;
import java.io.*;

class Test
{
      public static void main(String[] args)throws Exception
      {
            Properties  p = new Properties();//properties object is created

            //Creating  a FileInputStream to read the data from a file called
"database.properties"
            FileInputStream fis = new FileInputStream("database.properties");

            //Data loaded into properties object throug fis
            p.load(fis);

            System.out.println(p);
            System.out.println();

            System.out.println("URL IS      :: "+p.getProperty("url"));
            System.out.println("USERNAME IS :: "+p.getProperty("username"));
            System.out.println("PASSWORD IS :: "+p.getProperty("password"));

            p.setProperty("iNeuron","NavinReddy");//Setting a new property

            FileOutputStream fos = new FileOutputStream("database.properties");
            p.store(fos,"MAP operation got concluded");//Added like a comment

      }
}
abc.properties
------------------
#MAP operation got concluded
#Sun Oct 16 12:47:16 IST 2022
password=root123
url=jdbc\:mysql\:///abc
iNeuron=NavinReddy
username=root
```

Topics of discussion for upcoming sessions
a. Generics
b. Collection vs Collections
c. Disucssion on few utility methods
d. Comparable vs Comparator(I)
e. Stream API's
f. Date and Time API
g. enum
h. annotations
i. inner classes
j. fileoperations and io streams

I am still not understanding how you are using 2 methods in a single object.
eg: mapData.getClass().getName(); data.getKey().equals(10)

eg:
```
String name ="sachin";
String result = name.toUpperCase();
int count=result.length();
System.out.println(count);

          vs
String name ="sachin";
System.out.println(name.toUpperCase().length());
```

Q>Is e.printStack() and sysout(e) same?
Exception object
name of the exception
cause of the exception
line where the exception occured

e.printStackTrace() -> prints the name,cause and line where exception occured
System.out.println(e) similar to System.out.prinltn(e.toString()) -> name and cause of exception

Q> Why do we have to always declare System.out.println() inside a method?
Any statement should always be inside a method because of oop's style coding.

```
class NameOfTheClass{
            //properties(datatypes)

            //behaviours(methods)
                System.out.println("");
}
```

Q> Can we implement multithreading with abstract class and interface?
Ans. yes using interface only multithreading is implemented
eg: Runnable(I)

Q>

```
import java.util.*;
import java.io.*;
public class Properties {

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

```
            Properties p1=new Properties();
            FileInputStream fin=new FileInputStream("C:\\Users\\VLR\\Desktop\\
db.properties");

            p1.load(fin);//------->method load is undefined for type property
      }
}
```

Properties object is not created for java.util.Properties, rather it  is created for userdefined class called "Properties".
solution : change the class name to PropertiesApp then run  it will work.

Q>

```
package Arraylist;
import java.util.*;
public class IteratorArraylist
{
      public static void main(String[] args)
      {
            ArrayList a=new ArrayList();
            a.add(100);
            a.add(10);
            a.add(80);
            a.add(50);

            Iterator i=a.iterator();
            System.out.println("Acccesing data in Forward direction using
Iterator");
            while(i.hasNext())
            {
                  System.out.println(i.next());
            }

            ListIterator l=a.listIterator(a.size());
            //cursor is in last position
            System.out.println("Accessing data in Backward direction using
ListIterator");
            while(l.hasPrevious())
            {
                  System.out.println(l.previous());
            }
            System.out.println("Accessing data in Forward direction using
ListIterator");
            while(l.hasNext())
            {
                  System.out.println(l.next());
            }
      }
}
```

Sir, Here why it is necessary to pass size in ListIterator when we want to access data from last but in LinkedLlist it is not required?
  Even in LinkedList, we need to send size() return type otherwise the logic won't work.

```
package LinkedList;
import java.util.*;
public class IteratorLinkedList {
```

```
	public static void main(String[] args) {
		LinkedList l1=new LinkedList();
		l1.add(10);
		l1.add(20);
		l1.add(30);
		l1.add(40);

		System.out.println("---------------");
		Iterator i=l1.iterator();
		while(i.hasNext())
		{
			System.out.println(i.next());
		}
		System.out.println("---------------");

		ListIterator l=l1.listIterator();
		//here cursor is in first positions
		while(l.hasNext())
		{
			System.out.println(l.next());
		}

		//cursor already in last location
		System.out.println("---------------");
		while(l.hasPrevious())
		{
			System.out.println(l.previous());
		}
		System.out.println("---------------");
		Iterator d=l1.descendingIterator();
		while(d.hasNext())
		{
			System.out.println(d.next());
		}

	}

}
```

Sir can you please explain higher, ceiling, lower, floor method Sir

higher() -> This method returns the highest value

ceiling(100) -> This method returns the highest value including the supplied one

lower() -> This method returns the lowest value

floor(100) -> This method returns the lowest value including the supplied one

A ={100,101,102,103,104,105}

higher(102)-> 103

ceiling(102)-> 102

lower(102) -> 101

floor(102) -> 102

Q>

StringBuffer sb=new StringBuffer("ten");

```
String s=new String("ten");

System.out.println(s.equals(sb)); //false

//if(this == Object) //String == StringBuffer (false)
      retur true;
   else if ( this instanceof String)   // String instanceof StringBuffer(false)
      return true;
   return false;
```

Q>
Sir, multithreading pls,

```
public class Launch {
      public static void main(String[] args) {
            ThreadA obj = new ThreadA();
            Thread t = new Thread(obj);
            t.start();
// 2threads(user defined and main thread)
            synchronized (obj.total) {
                  obj.wait();
                  System.out.println(obj.total);
            }
      }
}
class ThreadA implements Runnable{
      public Integer total=0;
      @Override
      public void run() {
            synchronized (total) {
                  for(int i=0; i<5; i++) {
                        total = total + i;
                  }
                  total.notify();
            }
      }
}
```

1. why total should be made 0 or else it gives NullPointerException
2.Producer synchronize block why total only doesn't work and also with notify it doesn't work
3. consumer gives exception for obj.total and wait also together? Please correct the above code and explain...

Generics(JDK1.5V)
==========
Agenda:
1. Introduction
2. Type-Safety
3. Type-Casting
4. Generic Classes
5. Bounded Types
6. Generic methods and wild card character(?)
7. Conclusions
8. Collection vs Collections

Deff : The main objective of Generics is to provide Type-Safety and to resolve Type-Casting problems.

Case 1: Type-Safety
Arrays are always type safe that is we can give the guarantee for the type of elements present inside array.
For example if our programming requirement is to hold String type of objects it is recommended to use String array.
In the case of string array we can add only string type of objects by mistake if we are trying to add any other type we will get
compile time error.

eg:
```
 String name[] =new String[500];
   name[0] = "Navin Reddy";
   name[1] = "Haider";
   name[2] =  new Integer(100); //CE: incompatbile types found: java.lang.Integer
required: java.lang.String
```

That is we can always provide guarantee for the type of elements present inside array and hence arrays are safe to use with respect to type
that is arrays are type safe.

But collections are not type safe that is we can't provide any guarantee for the type of elements present inside collection.

For example if our programming requirement is to hold only string type of objects it is never recommended to go for ArrayList.
By mistake if we are trying to add any other type we won't get any compile time error but the program may fail at runtime.

eg:
```
   ArrayList al =new ArrayList();
      al.add("NavinReddy");
      al.add("Haider");
      al.add(new Integer(10));
            ;;;
            ;;;
            ;;;
      String name1 = (String)al.get(0);
      String name2 = (String)al.get(1);
      String name3 = (String)al.get(2);//Exception in thread "main" ::
java.lang.ClassCastException
                                                        java.lang.Integer cannot
be cast to java.lang.String
```

Hence we can't provide guarantee for the type of elements present inside

collections that is collections are not safe to use with respect to
type.

Case 2: Type-Casting
In the case of array at the time of retrieval it is not required to perform any
type casting.

```
eg::
String name[] =new String[500];
   name[0] = "Navin Reddy";
   name[1] = "Haider";
      ;;;;
      ;;;;
  String data =name[0];//here type casting is not required.
```

But in the case of collection at the time of retrieval compulsory we should perform
type casting otherwise we will get compile time error.

```
eg::
ArrayList al =new ArrayList();
      al.add("NavinReddy");
      al.add("Haider");
String name1= al.get(0);//CE: incompatible types : found : java.lang.Object
                                                                  required:
java.lang.String

String name1=(String) al.get(0);//At the time of retrieval type casting is
madantory
```

That is in collections type casting is bigger headache.
To overcome the above problems of collections(type-safety, type casting)sun people
introduced generics concept in 1.5v
hence the main objectives of generics are:

1. To provide type safety to the collections.
2. To resolve type casting problems.

To hold only string type of objects we can create a generic version of ArrayList as
follows.

```
ArrayList<String> al =new ArrayList<String>();
      al.add("NavinReddy");
      al.add(10);//CE: can't find symbol
                              incompatible type: required java.lang.String   found:
int
```

For this ArrayList we can add only string type of objects by mistake if we are
trying to add any other type we will get compile time error
that is through generics we are getting type safety.
At the time of retrieval it is not required to perform any type casting we can
assign elements directly to string type variables.

```
eg:
ArrayList<String> al =new ArrayList<String>();
      al.add("NavinReddy");
            ;;;;
            ;;;;
      String name =al.get(0);//type casting is not required as it is an TypeSafe
```

That is through generic syntax we can resolve type casting problems.

Conclusions
==========

1. Polymorphism concept is applicable only for the base type but not for parameter type
    [usage of parent reference to hold child object is called polymorphism].

```
       |-> basetype
eg: ArrayList<String> al =new ArrayList<String>();
                     |=> parameter type
       List<String> al =new ArrayList<String>();
       Collection<String> al =new ArrayList<String>();
       Collection<Object> al =new ArrayList<String>();//CE: incompatible types
```

2.
Collections concept applicable only for objects , Hence for the parameter type we can use any class or interface name but not primitive
 value(type).Otherwise we will get compile time error.

```
            eg: ArrayList<int> al =new ArrayList<int>();//CE: unexcpected type

      found:primitive
                                                                    required:
reference
```

Generic classes:
Until 1.4v a non-generic version of ArrayList class is declared as follows.

Example:

```
class ArrayList{
      add(Object o);
      Object get(int index);
}
```

add() method can take object as the argument and hence we can add any type of object to the ArrayList.
Due to this we are not getting type safety.
The return type of get() method is object hence at the time of retrieval compulsory we should perform type casting.

But in 1.5v a generic version of ArrayList class is declared as follows.

```
                      |=> Type parameter
class ArrayList<T>{
      add(T t);
      T get(int index)
}
```

Based on our requirement T will be replaced with our provided type.
For Example to hold only string type of objects we can create ArrayList object as follows.

```
      Example:
            ArrayList<String> l=new ArrayList<String>();
```

For this requirement compiler considered ArrayList class is
Example:

```
class ArrayList<String>{
            add(String s);
            String get(int index);
}
```

add() method can take only string type as argument hence we can add only string type of objects to the List.
By mistake if we are trying to add any other type we will get compile time error.

eg#1.

```
ArrayList<String> al =new ArrayList<String>();
      al.add("NavinReddy");
      al.add(10);//CE: can't find symbol
                              symbol: method add(int)
                              location : class
java.util.ArrayList<java.lang.String>
                                                  al.add(10)
```

eg#2.

```
ArrayList<String> al =new ArrayList<String>();
      al.add("NavinReddy");
String name = al.get(0);//type casting is not requried
```

Hence through generics we are getting type safety.
At the time of retrieval it is not required to perform any type casting we can assign its values directly to string variables.

In Generics we are associating a type-parameter to the class, such type of parameterised classes are nothing but Generic classes.
Generic class : class with type-parameter.
Based on our requirement we can create our own generic classes also.
Example:

```
class Account<T>
{}
Account<Gold> g1=new Account<Gold>();
Account<Silver> g2=new Account<Silver>();
```

Example:

```
class Gen<T>{
      T obj;
      Gen(T obj){
            this.obj=obj;
      }
      public void show(){
                  System.out.println("The type of object
is :"+obj.getClass().getName());
      }
      public T getObject(){
                  return obj;
      }
}
class GenericsDemo{
      public static void main(String[] args){
                  Gen<Integer> g1=new Gen<Integer>(10);
                  g1.show();
                  System.out.println(g1.getObject());

                  Gen<String> g2=new Gen<String>("iNeuron");
                  g2.show();
                  System.out.println(g2.getObject());

                  Gen<Double> g3=new Gen<Double>(10.5);
```

```
                g3.show();
                System.out.println(g3.getObject());
      }
}
Output:
The type of object is: java.lang.Integer
10

The type of object is: java.lang. String
iNeuron

The type of object is: java.lang. Double
10.5


Bounded types
--------------------
We can bound the type parameter for a particular range by using extends keyword
such types are called bounded types.
Example 1:
class Test<T>
{}
Test <Integer> t1=new Test< Integer>();//valid
Test <String> t2=new Test < String>();//valid

Here as the type parameter we can pass any type and there are no restrictions hence
it is unbounded type.

Example 2:
class Test<T extends X>
{}
If x is a class then as the type parameter we can pass either x or its child
classes.
If x is an interface then as the type parameter we can pass either x or its
implementation classes.

    eg#1.
      class Test <T extends Number>{}
      class Demo{
            public static void main(String[] args){
                  Test<Integer> t1 = new Test<Integer>();
                  Test<String>   t2 = new Test<String>(); //CE
            }
      }

      eg#2.
      class Test <T extends Runnable>{}
      class Demo{
            public static void main(String[] args){
                  Test<Thread> t1 = new Test<Thread>();
                  Test<String>   t2 = new Test<String>(); //CE
            }
      }

Keypoints about bounded types
----------------------------------------
 => We can't define bounded types by using implements and super keyword
 => But implements keyword purpose we can replace with extends keyword.
            eg: class Test<T implements Runnable>{}//invalid
```

```
                class Test<T super String>{}//invalid

=> As the type parameter we can use any valid java identifier but it convention to
use T always.
            eg: class Test<T>{}
                   class Test<iNeuron>{}

=> We can pass any no of type parameters need not be one.
            eg: class HashMap<K,V>{}
            HashMap<Integer,String> h=new HashMap<Integer,String>();

Which of the following are valid?
class Test <T extends Number&Runnable> {}//valid(first class and then interface)
class Test<T extends Number&Runnable&Comparable> {} //valid(first class and then
multiple interfaces)
class Test<T extends Number&String> {} //invalid(both are classes becoz multiple
inheritance through class is not supported)
class Test<T extends Runnable&Comparable> {}//valid (both are interfaces)
class Test<T extends Runnable&Number> {}//invalid(first class then interface)


Generic methods with wildcard pattern
=================================
  ? => it is a wild chard symbol to indicate any type i can collect.

methodOne(ArrayList<String> l):
This method is applicable for ArrayList of only String type.
Example:
      l.add("A");
      l.add(null);
      l.add(10);//(invalid)

Within the method we can add only String type of objects and null to the List.
      methodOne(ArrayList<?> l):
We can use this method for ArrayList of any type but within the method we can't add
anything to the List except null.
Example:
l.add(null);//(valid)
l.add("A");//(invalid)
l.add(10);//(invalid)
This method is useful whenever we are performing only read operation.

methodOne(ArrayList<? Extends x> l):
If x is a class then this method is applicable for ArrayList of either x type or
its child classes.
If x is an interface then this method is applicable for ArrayList of either x type
or its implementation classes.
In this case also within the method we can't add anything to the List except null.

methodOne(ArrayList<? super x> l):
If x is a class then this method is applicable for ArrayList of either x type or
its super classes.
If x is an interface then this method is applicable for ArrayList of either x type
or super classes of implementation class of x.
But within the method we can add x type objects and null to the List.

eg:    Runnable
            |
           Thread<==super class====== Object
```

Which of the following declarations are allowed?
1. ArrayList<String> l1=new ArrayList<String>();//valid
2. ArrayList<?> l2=new ArrayList<String>();//valid
3. ArrayList<?> l3=new ArrayList<Integer>();/valid
4. ArrayList<? extends Number> l4=new ArrayList<Integer>();//valid
5. ArrayList<? extends Number> l5=new ArrayList<String>();//invalid(String and Number no relationship)
6. ArrayList<?> l6=new ArrayList<? extends Number>(); //invalid becoz of <? extends Number is right hand side>
7. ArrayList<?> l7=new ArrayList<?>(); //invalid

Declaring type parameter at class level
==========
class Test<T>{
	We can use anywhere this 'T'.
}

Declaring type parameter at method level
==========
We have to declare just before return type.

Which of the following declarations are allowed?
public<T> void methodOne1(T t){} //valid
public<T extends Number> void methodOne2(T t){}//valid
public<T extends Number&Comparable> void methodOne3(T t){}//valid
public<T extends Number&Comparable&Runnable> void methodOne4(T t){}//valid
public<T extends Number&Thread> void methodOne(T t){}//invalid(2 classes extends not possible)
public<T extends Runnable&Number> void methodOne(T t){}//invalid(first interface not possible)
public<T extends Number&Runnable> void methodOne(T t){}//valid

Collection vs Collections
---
Collection(I)  => It is a root interface in Collection hierarchy
Collections(C) => It is a utility class(static methods/helper methods would be available)

```
import java.util.*;
public class Test {
	public static void main(String[] args) {
		ArrayList al =new ArrayList();
		al.add(10);
		al.add(5);
		al.add(0);
		al.add(15);
		System.out.println(al);//[10,5,0,15]

		Collections.sort(al);//sorting is done in Ascending order
		System.out.println(al);
	}
}
```

Usage of Compartor will be discussed to work with "Descending order".

Generics
========
   a. To promote typesafety and to avoid type casting problems


Type parameter at class level
   class Test<T>{

  }

a. <T extends X>
b. <T extends X&Y>
c. <T extends X&Y&Z>

X,Y,Z can beinterface
X,Y  can be interface
X -> it should be class name ,Y,Z is interface name


 method level Type parameter
    1. methodOne( ArrayList<String> t)
    2. methodOne( ArrayList<? extends X> t)
    3. methodOne(ArrayList< ? super X> t)

return type of method Type Parameter
   1. public <T> void m1(T t)
   2. public <T extends X&Y> void m1(T t)
   3. public <T extneds X&Y&Z> void m1(T t)

Generics concept is applicable only at the compiler level, not at JVM
level(runtime)
Even if we have code in generics in .classfile the syntax of generics will be
removed
      eg:  ArrayList<String> al  = new ArrayList<String>();
              ArrayList<String> al = new ArrayList<>();
                                |
                                |
                                |
               ArrayList al =new ArrayList();


class Test{
      public void m1(Arraylist<String> al){  ======> m1(ArrayList al)

      }
      public void m1(ArrayList<Double> al ){  ========> m1(ArrayList al)

      }
}
this mechanism is called "type erasure"


Basically my question is what is the difference between Arrays.asList() & List.of()
methods to create objects in collaboration?
     Arrays.asList() --> gives a copy of Array to read as List(only for reading
purpose)
     List.of() ---> creates a list which are immutable.

```
SpringORM
---------------
public T m1(T t) {}
       and
public <T> void m1(T t) {
      //control the behaviour with T
}

Then how to JVM resolve the type casting problem , if its erase the generics part
at compile time itself?
   class ArrayList<String>{
            add(String data)
            String get(int index)
   }

ArrayList<String> al = new ArrayList<String>();
      al.add("sachin");
      al.add("saurav");
      ;;;;;
      ;;;;;
      String data = al.get(0);

JVM
ArrayList al = new ArrayList();
        al.add("sachin");
      al.add("saurav");
      ;;;;;
      ;;;;;
      String data = al.get(0);

How can we restrict Generics to a subclass of particular class?
           Person
            |
            |
       Student

 Demo<? extends Person> d = new Demo<Student>();

Difference b/w Callable and Runnable

Compiled from "Runnable.java"
public interface java.lang.Runnable {
      public abstract void run();
}

Compiled from "Callable.java"
public interface java.util.concurrent.Callable<V> {
  public abstract V call() throws java.lang.Exception;
}

Q>ArrayList<?> al = new ArrayList<Number>();Is it allowed?

Arraylist<Number> al =new ArrayList<Number>();
       m1(al);

public void m1(ArrayList<?> al){// ArrayList<?> al =new Arraylist<Number> ();
      ;;;;
      ;;;;
      ;;;;
```

```
}
AL<Object> al =new AL<String>; //invalid
```

java.util.Date vs java.sql.Date
==========

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

		//To use Date in general operations like printing the date and time
		java.util.Date utilDate = new java.util.Date();
		System.out.println(utilDate);


		long l = utilDate.getTime();//It is giving the information of utilDate
in milliseconds from 1970
		System.out.println(l+"ms");

		//To use Date in DB operations like insert,update,delete query we use
sqlDate
		java.sql.Date sqlDate = new java.sql.Date(l);
		System.out.println(sqlDate);
	}
}
```

Output

```
Sun Oct 30 10:05:33 IST 2022
1667107558145ms
2022-10-30(yyyy-mm-dd)
```

Difference b/w java.util.Date and java.sql.Date
==========

java.util.Date

- => It is a utility class to handles Date in our java program.
- => It represents both Date and Time

java.sql.Date

- => It is designed class to handle Dates w.r.t DB operations
- => It represents only Date,but not Time.

Note: In sql package

| | |
|---|---|
| Time(C) | represents only => Time value |
| TimeStamp(C) | represents both => Date and Time value |

Date and Time API: (Joda-Time API)

Until Java 1.7 version the classes present in Java.util package to handle Date and Time (like Date, Calendar,
TimeZone etc) are not up to the mark with respect to convenience and performance.

To overcome this problem in the 1.8 version oracle people introduced Joda-Time API. This API developed by joda.org and available in Java in the form of "java.time" package.

```
# program for to display System Date and time.
import Java.time.*;
public class DateTime {
		public static void main(String[] args) {
				LocalDate date = LocalDate.now();
				System.out.println(date);

				LocalTime time=LocalTime.now();
```

```
                        System.out.println(time);
            }
}
Output
======
2022-10-30
11:15:41.698
```

Once we get LocalDate object we can call the following methods on that object to retrieve Day,month and year values separately.

Ex:

```
import java.time.*;
 class Test {
       public static void main(String[] args) {
                  LocalDate date = LocalDate.now();
                  System.out.println(date);
                  int dd =   date.getDayOfMonth();
                  int mm = date.getMonthValue();
                  int yy = date.getYear();
                  System.out.println(dd+"..."+mm+"..."+yy);
                  System.out.printf("\n%d-%d-%d",dd,mm,yy);
      }
}
Output
2022-10-30
30...10...2022
30-10-2022
```

Once we get LocalTime object we can call the following methods on that object.

```
import java.time.*;
class Test {
       public static void main(String[] args) {
                  LocalTime time = LocalTime.now();
                  int h = time.getHour();
                  int m = time.getMinute();
                  int s = time.getSecond();
                  int n = time.getNano();
                  System.out.printf("\n%d:%d:%d:%d",h,m,s,n);
      }
}
Output
9:22:31:795000000
```

Note::

If we want to represent both Date and Time then we should go for LocalDateTime object.

```
LocalDateTimedt = LocalDateTime.now();
System.out.println(dt);
O/p: 2015-11-23T12:57:24.531
```

We can represent a particular Date and Time by using LocalDateTime object as follows.

Ex:

```
 LocalDateTime dt1 = LocalDateTime.of(1995,Month.APRIL,28,12,45);
 sop(dt1);
```

Ex:

```
 LocalDateTime dt1=LocalDateTime.of(1995,04,28,12,45);
```

```
 Sop(dt1);
 Sop(“After six months:”+dt.plusMonths(6));
 Sop(“Before six months:”+dt.minusMonths(6));
```

ZoneId
=====
To Represent Zone:
ZoneId object can be used to represent Zone.
Ex:

```
import Java.time.*;
  class ProgramOne {
             public static void main(String[] args) {
                        ZoneId zone = ZoneId.systemDefault();
                        System.out.println(zone);
             }
}
```

Output
Asia/Calcutta

We can create ZoneId for a particular zone as follows
Ex:

```
 ZoneId la = ZoneId.of("America/Los_Angeles");
 ZonedDateTimezt = ZonedDateTime.now(la);
 System.out.println(zt);
```

Output
2022-10-29T23:19:59.718-07:00[America/Los_Angeles]

Period Object:
Period object can be used to represent quantity of time
Ex:

```
 LocalDate today = LocalDate.now();
 LocalDate birthday = LocalDate.of(1994,01,3);
 Period p = Period.between(birthday,today);
 System.out.printf("age is %d year %d months %d
days",p.getYears(),p.getMonths(),p.getDays());
```

Output
age is 28 year 9 months 27 days

# write a program to check the given year is leap year or not

rule for leap year
==========
=> A year may be a leap year if it is evenly divisible by 4.
=> Years that are divisible by 100 (century years such as 1900 or 2000) cannot be leap years unless they are also
      divisible by 400.

```
import Java.time.*;
 public class Leapyear {
      public static void main(String[] args){
             int n = Integer.parseInt(args[0]);
             Year y = Year.of(n);
              if(y.isLeap())
                  System.out.printf("%d is Leap year",n);
             else
                  System.out.printf("%d is not Leap year",n);
```

```
      }
}


Note:
    Date -> LocalDate(C)
    Time -> LocalTime(C)
    Date & Time -> LocalDateTime(c)

      now() --->Current information
        of()     --> user specific information

      ZoneId -> Setting up the particular zone to fetch the information
       ZondDateTime-> To get the Date and time information of any zone.

       Period ---> To find difference b/w 2 date Objects
      Year    --->  To check whether the supplied year is leapYear or not.
```

contact details:
    haider  :syedhyder@ineuron.ai
    nitin      : nitin@ineuron.ai

Q>
Doubts:
1. When we create a String object with string buffer or builder will a copy be created in string pool too, if so how does it become mutable.
      My understanding of String being immutable is because they are created in SCP area of memory too.

String,WrapperClass-> Immutable
                String s = "sachin"; => SCP area
                String s=new String("kohli"); => SCP, Heap area(s)

StringBuffer,StringBuilder => mutable
                StringBuffers sb =new StringBuffer("dhoni"); ===> SCP, Heap area(sb)
                String s =new String(sb);====> HeapArea(s)

2. If prewritten code that we use are API, why Arrays are not called as API but class
      Every class which we are using we call it as "API" only,but while talking we say as class.


3. static and non static variable
            static variable => a variable which is declared using static keyword are called "static variables".
                                                eg: static int a = 10;
            non static variable => a variable which is not declared using static keyword are called "instance variables".
                                                eg: int a =100;

4. How to create temporory Array and assign existing arrays values?
                int a[] =new int[5];
                int b[] = {10,20,30};
                a=b;

5. public int lastIndexOf(char ch)
Does this method search the character in the string in reverse order(last to first)?
            searching from last index to first index

6. Sir, i have confusion on return statement.....can u explain it

```
class Demo
{
      public int m1(){
                  System.out.println("hello");
                  return 10;
      }

      public void m2(){
                  System.out.println("hiee");
                  return;
      }
}
```

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

			Demo d = new Demo();
			int result=d.m1();
			System.out.println(result);

			d.m2();


	}
}
```

Q> Ananomyous Array

```
public class Test {
	public static void main(String[] args) {
		display(new int[]{10,20,30,40,50});

	}
	static void display(int[] arr){
		for(int ele:arr)
			System.out.println(ele);
	}
}
```

Q>

```
class Demo
{
	Demo(){
		System.out.println("constructor called");
	}
}
public class Test {
		Demo d = new Demo();

		Test(){
			System.out.println("Test class Constructor");
		}
	public static void main(String[] args) {

		Test t =new Test();

	}

}
```

Q>1 plz check at line 1, new object will create on heap ?
Q>2 sir can you explain at what time object will create ,at the of classloading?

Q>

```
String s1 = new String("Test1");
String s2 = new String("Test1");
System.out.println(s1.equals(s2)); //output true

StringBuffer s3 = new StringBuffer("Test1");
StringBuffer s4 = new StringBuffer("Test1");
System.out.println(s3.equals(s4)); //output false
```

Q>

```
String s1 = new String("Nitin");//SCP(Nitin) and in heaparea(s1)
```

```
String s2 = "Nitin";//SCP s2,s3-> Nitin
String s3 = "Nitin";
String s4 = "nitin";//SCP s4->nitin
System.out.println("Value is: "+ s1==s2); //false
System.out.println("Value is: "+ s2==s3); // true
System.out.println("Value is: "+ s3==s4); //false

Q>
String s1 = new String("Nitin");
String s2 = "Nitin";
System.out.println("Value is: "+ s1==s2)//false


Q>
StringBuffer sb=new StringBuffer("sachin");//new object created in heap(sachin)and
SCP(sachin)
sb.append("tendulkar");//sb in heap area now pointing to sb-> sachin tendulkar
Q)will the existing object in SCP also gets modified to sachin tendulkar?
Ans. no

Q> sir in int[] array if we give char value it is showing compile time error Hyder
sir  said
      it will give run time error called array store exception. but it is not only
compiling

      int[] a = {'a','b','c'};

Q>sir, in Inheretence only we create object reference parent name or another also
we can write ?
     please explain?
 Ans. yes
            eg: Parent ref =new Child(); //only in inheritance.


Q> System.out.println();
      Here out points to null then how it can access println() method?
            out => refers to the Object of PrintStream(java.io)

Q> String name = new String("sachin");
            name-> "sachin" (heap area)
            ========================
              sachin(SCP)
                  "memory will be deallocated during jvm shutdown"
```

Agenda
1. Introduction.
2. Normal or Regular inner classes
   - o Accessing inner class code from static area of outer class
   - o Accessing inner class code from instance area of outer class
   - o Accessing inner class code from outside of outer class
   - o The applicable modifiers for outer & inner classes
   - o Nesting of Inner classes
3. Method Local inner classes
4. Anonymous inner classes
   - o Anonymous inner class that extends a class
   - o Anonymous Inner Class that implements an interface
   - o Anonymous Inner Class that define inside method arguments
   - o Difference between general class and anonymous inner classes
   - o Explain the application areas of anonymous inner classes ?
5. Static nested classes
   - o Compression between normal or regular class and static nested class ?
6. Various possible combinations of nested class & interfaces
   - o class inside a class
   - o interface inside a class
   - o interface inside a interface
   - o class inside a interface
7. Conclusions

=> Sometimes we can declare a class inside another class such type of classes are called inner classes.

SUNMS came up with JDK1.0 in 1991 with features like
1. Platform Independent
2. Robust
3. OOPs
4. Secured
5. Simple
   ..
   ..
   ..

But java was not upto the mark in 2 areas
a. Performance was not upto the mark
b. GUI concepts(AWT)

Performance was improved through "JIT compiler".
=> Sun people introduced inner classes in 1.1 version as part of "EventHandling" to resolve GUI bugs.
=> But because of powerful features and benefits of inner classes slowly the programmers starts using in regular coding also.
=> Without existing one type of object if there is no chance of existing another type of object then we should go for inner classes.

Eg1
Without existing University object there is no chance of existing Department object
hence we have to define Department class inside University class.

```
class University{//Outer class
      class Department{//Inner class

      }
}
```

Eg2
 Without existing Bank object there is no chance of existing Account object hence we
 have to define Account class inside Bank class.

```
      class Bank{//Outer class
            class Account{//Inner class

            }
      }
```

Eg3:
  Without existing Car Object, there is no chance of existing Engine Object hence we have to
  define Engine class Inside Car class.

```
      class Car{//Outer class
            class Engine{//Inner class

            }
      }
```

Eg4:
 Without existing Map object there is no chance of existing Entry object hence Entry interface is define inside Map interface.
  Map is a collection of key-value pairs, each key-value pair is called an Entry.

```
      interface Map{//Outer interface
            interface Entry<K,V>{//Inner interface

            }
        }
```

Note:
The relationship between outer class and inner class is not IS-A relationship and it is Has-A relationship.
It promotes Composition/Aggregation.

Based on the purpose and position of declaration all inner classes are divided into 4 types.
They are:
1. Normal or Regular inner classes.
2. Method Local inner classes.
3. Anonymous inner class.
4. static nested classes.

Regular inner class
==========
If we are declaring any named class inside another class directly without static modifier such type of inner classes are called normal or
regular inner classes.
Example:

```
      Outer.java
class Outer{
      class Inner{
      }
}
```

Output:

```
javac Outer.java
   Outer.class
   Outer$Inner.class
```

```
java Outer
      NoSuchMethodError/main method not found
java Outer$Inner
      NoSuchMethodError/main method not found

Example:
class Outer{
      class Inner{
      }
      public static void main(String[] args){
            System.out.println("outer class main method");
      }
}
Output:
 javac Outer
 java Outer
      outer class main method
 java Outer$Inner
      NoSuchMethodError/main method not found

=> Inside inner class we can't declare static members. Hence it is not possible to
declare main() method and we can't invoke
     inner class directly from the command prompt.
Example
class Outer {
      class Inner{
            public static void main(String[] args) {
                  System.out.println("Inner class main method");
              }
      }
}
Output: CE: can't declare static inside Inner class.


Accessing inner class code from static area of outer class:
============================================================
class Outer {
      class Inner{
            public void m1(){
                  System.out.println("inner class instance m1()");
            }
      }
      public static void main(String[] args) {
            Outer o= new Outer();
            Outer.Inner i=o.new Inner();
            i.m1();
      }
}
Output
=====
 javac Outer
 java Outer
   inner class instance m1()

Note:
Scenario1:
 Outer.Inner i= new Outer().new Inner();
    i.m1();
```

Scenario2:
```
   new Outer().new Inner().m1();
```

Accessing inner class code from instance area of outer class
==========

```
class Outer {
      class Inner{
            public void m1(){
                  System.out.println("inner class instance m1()");
            }
      }
      public void m2(){
            Inner i= new Inner();
            i.m1();
      }
      public static void main(String[] args) {
            Outer o= new Outer();
            o.m2();

      }
}
```

```
javac Outer
java Outer
  inner class instance m1()
```

Accessing inner class code from outside of outer class
==========

```
class Outer {
      class Inner{
            public void m1(){
                  System.out.println("inner class instance m1()");
            }
      }
}
class Test{
    public static void main(String[] args) {
            Outer o= new Outer();
            Outer.Inner i=o.new Inner();
            i.m1();

      }
}
```

Output
=====

```
javac Test
java Test
  inner class instance m1()
```

=>From inner class we can access all members of outer class (both static and non-static, private   and non private methods and variables) directly.

example

```
class Outer {
      int x=10;
      static int y=20;
      class Inner{
            public void m1(){
                  System.out.println(x);
```

```
                System.out.println(y);
            }
      }
      public static void main(String[] args) {
            new Outer().new Inner().m1();
      }
}
javac Outer
java Outer
10
20


eg#2.
class Outer {
      int y=10;
      class Inner{
            int y=100;
            public void m1(){
                  int z=1000;
                  System.out.println(x);
                  System.out.println(y);;
                  System.out.println(z);
            }
        }
      public static void main(String[] args) {
            new Outer().new Inner().m1();
      }
}

=> Within the inner class "this" always refers current inner class object. To refer
current outer class object we have to use "outer class name.this".
class Outer {
      int x=10;
      class Inner{
            int x=100;
            public void m1(){
                  int x=1000;
                  System.out.println(x);
                  System.out.println(this.x);
                  System.out.println(Outer.this.x);

            }
        }
      public static void main(String[] args) {
            new Outer().new Inner().m1();
      }
}
Output
======
javac Outer
java Outer
1000
100
10
The applicable modifiers for outer classes are:
1. public
2. default
3. final
```

4. abstract
5. strictfp

But for the inner classes in addition to this the following modifiers also allowed.
 1.private
 2.protected
 3.static

Nesting of Inner classes :
We can declare an inner class inside another inner class

```
class A {
      class B{
            class C{
                  public void m1(){
                        System.out.println("C class method");
                  }
            }
      }
}
class Test{
      public static void main(String[] args) {
            A a = new A();
            A.B b = a.new B();
            A.B.C c = b.new C();
                 c.m1();
        }
}
```

```
javac Test
java Test
C class method
```

Note:
 new A().new B().new C().m1();

Method local inner classes:
=> Sometimes we can declare a class inside a method such type of inner classes are called method  local inner classes.
=> The main objective of method local inner class is to define method specific repeatedly required   functionality.
=> Method Local inner classes are best suitable to meet nested method requirement.
=> We can access method local inner class only within the method where we declared it.
     That is from outside of the method we can't access. As the scope of method local inner classes is very less,
     this type of inner classes are most rarely used type of inner classes.

eg::

```
class Outer {
            public void m1(){
                  class Inner{
                        public void sum(int x,int y){
                              System.out.println("The sum is ::"+(x+y));
                        }
                  }
                  new Inner().sum(10,20);
                        ;;;;
                        ;;;;
                        ;;;;
```

```
                new Inner().sum(100,200);
                      ;;;;
                      ;;;;
                      ;;;;
                      ;;;;
                new Inner().sum(1,2);

      }
      public static void main(String[] args) {
            new Outer().m1();
      }
}
javac Outer
java  Outer
 The sum is :: 30
 The sum is :: 300
 The sum is :: 3
```

=> If we are declaring inner class inside instance method then we can access both static and non  static members of outer class directly.
=> But if we are declaring inner class inside static method then we can access only static  members of outer class directly and
     we can't access instance members directly.

```
eg#1.
class Outer {
      int x=10;
      static int y=20;
      public  void m1(){
                class Inner{
                      public void m2(){
                            System.out.println(x);
                            System.out.println(y);
                      }
                }
                Inner i=new Inner();
                i.m2();
      }
      public static void main(String[] args) {
            new Outer().m1();

      }
}
Output
10
20
```

case2:
if we declare inner class inside static method then it would result in CE.

=> If we declare methodOne() method as static then we will get compile time error saying "non-static variable x cannot be referenced from a static context".
=> From method local inner class we can't access local variables of the method in which we  declared it.
      But if that local variable is declared as final then we won't get any compile time    error.

```
eg#1.
class Outer {
```

```
	public  void m1(){
		final int x=10;
		class Inner{
			public void m2(){
					System.out.println(x);
			}
		}
		Inner i=new Inner();
		i.m2();
	}
	public static void main(String[] args) {
		new Outer().m1();

	}
}
```

CaseStudy
=========

```
class Outer {
	int i=10;
	static int j=20;
	public void m1(){
		int k=30;
		final int l=40;
		class Inner{
			public void m2(){
				//line-1
			}
		}
		Inner i=new Inner();
		i.m2();
	}
	public static void main(String[] args) {
		new Outer().m1();

	}
}
```

At line-1 how many variables we can access?
  i,j,l
if we make m1() as static how many variables we can access?
  j,l
if we make m2() as static how many variables we can access
 compile time error.

The only applicable modifiers for method local inner classes are:
1. final
2. abstract
3. strictfp

Anonymous inner classes:
=> Sometimes we can declare inner class without name such type of inner classes are called  anonymous inner classes.

=> The main objective of anonymous inner classes is "just for instant use".
=> There are 3 types of anonymous inner classes
	1. Anonymous inner class that extends a class.
	2. Anonymous inner class that implements an interface.
	3. Anonymous inner class that defined inside method arguments.

1. Anonymous inner class that extends a class.

```
class PopCorn{
      public void taste(){
            System.out.println("spicy");
       }
}
class Test {
      public static void main(String[] args) {
            PopCorn p=new PopCorn()
            {
                  public void taste(){
                        System.out.println("salty");
                  }
            };
            p.taste();//salty
            PopCorn p1=new PopCorn()
            p1.taste();//spicy
      }
}
```

Analysis:

1. PopCorn p=new PopCorn();
   We are just creating a PopCorn object.

2. PopCorn p=new PopCorn(){
   };
   We are creating child class without name for the PopCorn class and for that child class we are creating an object with Parent PopCorn reference.

```
2. PopCorn p=new PopCorn()
    {
        @Override
        public void taste(){
            System.out.println("salty");
        }
    };
```

1. We are creating child class for PopCorn without name.
2. We are overriding taste() method.
3. We are creating object for that child class with parent reference.

Note:
Inside Anonymous inner classes we can take or declare new methods but outside of anonymous inner classes we can't call these methods directly because we are depending on parent reference.[parent reference can be used to hold child class object but by using that reference we can't call child specific methods]. These methods just for internal purpose only.

eg#1.
Example 1:

```
class PopCorn{
      public void taste(){
            System.out.println("spicy");
      }
}
class Test {
      public static void main(String[] args) {
            PopCorn p=new PopCorn(){
```

```
                  public void taste(){
                        methodOne();//valid call(internal purpose)
                  System.out.println("salty");
                   }
                  public void methodOne(){
                        System.out.println("child specific method");
                     }
               };
      //p.methodOne();//here we can not call(outside inner class)
      p.taste();//salty
      PopCorn p1=new PopCorn();
      p1.taste();//spicy
   }
}
Output:
Child specific method
Salty
Spicy

Example 2:
class Test {
      public static void main(String[] args){
            Thread t=new Thread(){
                  public void run(){
                        for(int i=0;i<10;i++){
                              System.out.println("child thread");
                        }
                  }
            };
            t.start();
            for(int i=0;i<10;i++){
                  System.out.println("main thread");
            }
      }
}

Anonymous Inner Class that implements an interface
==================================================
Example:
class InnerClassesDemo{
      public static void main(String[] args) {
            Runnable r=new Runnable(){ //here we are not creating for Runnable
interface are creating implements class object.

            public void run(){
                  for(int i=0;i<10;i++){
                        System.out.println("Child thread");
                  }
            }
            };
            Thread t=new Thread(r);
            t.start();
            for(int i=0;i<10;i++){
                  System.out.println("Main thread");
            }
      }
}
```

Anonymous Inner Class that define inside method arguments
==========

```
Example:
class Test {
      public static void main(String[] args) {
            new Thread(new Runnable(){
                  public void run(){
                        for(int i=0;i<10;i++){
                              System.out.println("child thread");
                        }
                  }
                  }).start();
            for(int i=0;i<10;i++){
                  System.out.println("main thread")}
            }
      }
}
```

Output:

=> This output belongs to example 2, anonymous inner class that implements an interface example and anonymous inner class that define inside method arguments example.

```
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Main thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
```

lambda Expression
==========

```
new Thread(
                  ()-> {
                              for (int i =1;i<=5;i++ )
                              {
                                    System.out.println("child thread");
                              }
                        }
      ).start();
```

When we use lambda expression,code would be compact and no .class file for lambda expression.

Difference between general class and anonymous inner classes:

General Class
1) A general class can extends only one class at a time.
2) A general class can implement any no. Of interfaces at a time.
3) A general class can extends a class and can implement an interface simultaneously.
4) In normal Java class we can write constructor because we know name of the class.

Anonmyous Inner class
1. Of course anonymous inner class also can extends only one class at a time.
2) But anonymous inner class can implement only one interface at a time.
3) But anonymous inner class can extends a class or can implements an interface but not
both simultaneously.
4) But in anonymous inner class we can't write constructor because anonymous inner class not
having any name of the class.

static nested classes
==========
=> Sometimes we can declare inner classes with static modifier such type of inner classes are called static nested classes.
=> In the case of normal or regular inner classes without existing outer class object there is no chance of existing inner class
object.
i.e., inner class object is always strongly associated with outer class object.
=> But in the case of static nested class without existing outer class object there may be a chance of existing static nested class
object.
i.e., static nested class object is not strongly associated with outer class object.

eg#1.
Example:
```
class Test {
   static class Nested{
      public void methodOne(){
            System.out.println("nested class method");
      }
   }
   public static void main(String[] args){
      //Test.Nested t=new Test.Nested();
      //t.methodOne();

      Nested n=new Nested();
      n.methodOne();
   }
}
```
=> Inside static nested classes we can declare static members including main() method also.
Hence it is possible to invoke static nested class directly from the command prompt.

Example:
```
class Test {
      static class Nested{
            public static void main(String[] args){
                  System.out.println("nested class main method");
            }
```

```
	}
	public static void main(String[] args){
		System.out.println("outer class main method");
	}
}
java Test
outer class main method
java Test$Nested
nested class main method
```

=> From the normal inner class we can access both static and non static members of outer class but from static nested class we can access only static members of outer class.

Example:

```
class Test {
	int x=10;
	static int y=20;
	static class Nested{
		public void methodOne(){
			System.out.println(x);//C.E:non-static variable x cannot be reference from a static context
			System.out.println(y);
		}
	}
}
javac Test.java (Test.class, Test$Nested.class)
```

Compression between normal or regular class and static nested class ?

Normal /regular inner class

1) Without existing outer class object there is no chance of existing inner class object.
   That is inner class object is always associated with outer class object.
2) Inside normal or regular inner class we can't declare static members.
3) Inside normal inner class we can't declare main() method and hence we can't invoke regular inner class directly from the
   command prompt.
4) From the normal or regular inner class we can access both static and non static members of outer class directly.

Static nested class

1) Without existing outer class object there may be a chance of existing static nested class object.
   That is static nested class object is not associated with outer class object.
2) Inside static nested class we can declare static members.
3) Inside static nested class we can declare main() method and hence we can invoke static nested class directly from the
   command prompt.
4) From static nested class we can access only static members of outer class directly

Various possible combinations of nested class &interfaces :

1. class inside a class :
   We can declare a class inside another class
   Without existing one type of object, if there is no chance of existing another type
   of object, then we should go for clas inside a class.

```
class University {
      class Department {
      }
}
```
Without existing University object, there is no chance of existing Department object.
i.e., Department object is always associated with University.

2. interface inside a class :
We can declare interface inside a class

```
  class X {
            interface Y {
                  }
  }
```

Inside class if we required multiple implementation of an interface and these implementations of relevant to a particular class, then we should declare interface inside a class.

```
class VehicleType {
      interface Vehicle {
            public int getNoOfWheels();
        }
  class Bus implements Vehicle {
      public int getNoOfWheels() {
            return 6;
        }
  }
  class Auto implements Vehicle {
   public int getNoOfWheels() {
            return 3;
   }
  }
}
```

3. interface inside a interface :
We can declare an interface inside another interface.

```
      interface Map {
            interface Entry {
                  public Object getKey();
                  public Object getValue();
                  public Object setValue(Object newValue);
            }
      }
```
Nested interfaces are always public,static whether we are declaring or not. Hence we can implements inner inteface directly with out implementing outer interface.

```
interface Outer {
      public void methodOne();
      interface Inner {
            public void methodTwo();
      }
}
class Test implements Outer.Inner {
      public void methodTwo() {
            System.out.println("Inner interface method");
      }
      public static void main(String args[]) {
            Test t=new Test();
```

```
            t.methodTwo();
      }
}
```
Whenever we are implementing Outer interface , it is not required to implement Inner interfaces.

```
class Test implements Outer {
      public void methodOne() {
            System.out.println("Outer interface method ");
      }
      public static void main(String args[]) {
            Test t=new Test();
            t.methodOne();
        }
}
```
i.e., Both Outer and Inner interfaces we can implement independently.

4. class inside a interface :

We can declare a class inside interface. If a class functionality is closely associated with the user interface then it is highly recommended to declare class inside interface.

Example:
```
interface EmailServer {
      public void sendEmail(EmailDetails e);
            class EmailDetails {
                  String from;
                  String to;
                  String subject;
            }
}
```
In the above example EmailDetails functionality is required for EmailService and we are not using anywhereelse . Hence we can declare EmailDetails class inside EmailServiceinterface .
We can also declare a class inside interface to provide default implementation for that interface.

Example :
```
interface Vehicle {
      public int getNoOfWheels();
      class DefaultVehicle implements Vehicle {
            public int getNoOfWheels() {
                  return 2;
            }
      }
}
class Bus implements Vehicle {
      public int getNoOfWheels() {
          return 6;
      }
}
class Test {
      public static void main(String args[]) {
            Bus b=new Bus();
            System.out.println(b.getNoOfWheels());//2
```

```
            Vehicle.DefaultVehicle d=new Vehicle.DefaultVehicle();
            System.out.println(d.getNoOfWheels());//3
      }
}
```
In the above example DefaultVehicle in the default implementation of Vehicle interface where as Bus customized implementation of Vehicle interface.
The class which is declared inside interface is always static ,hence we can create object directly without having outer interface type object.

Conclusions :
1. We can declare anything inside any thing with respect to classes and interfaces.

```
      class A{
            class B{

            }
      }
      class A{
            interface B{//static

            }
      }
      interface A{
            interface B{//public static

            }
      }
      interface A{
            class B{//public static

            }
      }
```
2. Nesting interfaces are always public, static whether we are declaring or not.
3. class which is declared inside interface is always public,static whether we are declaring or not
4. interface declared inside a class is always static,but need not be public.

File(IO Package)
==========
  Agenda:
1. File
2. FileWriter
3. FileReader
4. BufferedWriter
5. BufferedReader

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

Example:
```
import java.io.*;
class FileDemo{
      public static void main(String[] args)throws IOException{
            File f=new File("abc.txt");
            System.out.println(f.exists());//false

            f.createNewFile();
            System.out.println(f.exists());//true
      }
}
```
1st run
=======
```
false
true
```

2nd run
======
```
true
true
```

=> A java File object can represent a directory also.
Example:
```
import   java.io.File;
import   java.io.IOException;

class FileDemo{
  public static void main(String[] args)throws IOException{
      File f=new File("cricket123");
      System.out.println(f.exists());//false

      f.mkdir();//Creates a new directory
      System.out.println(f.exists());//true
 }
}
```
1st run
=======
```
false
true
```

2nd run
======
```
true
```

```
true

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


Constructors of File class
==========================
 File f=new File(String fname)
 File f=new File(String directoryName,String fileName);
 File f=new File(File f,String fileName);

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

2. File f=new File(String subdirname,String name);
    => Creates a File object that represents name of the file or directory present
in specified sub directory.
            eg#1. File f1=new File("abc");
                        f1.mkdir();
                           File f2=new File("abc","demo.txt");
                        f2.createNewFile();

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");
                        f2.createNewFile();

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

Program:
import java.io.*;
class FileDemo{
      public static void main(String[] args)throws IOException{
            File f=new File("demo.txt");
            f.createNewFile();
      }
}
Requirement
=> Write code to create a directory named with IPLTeam in current working directory
and create a  file named with abc.txt in that directory.
             cwd
               |=> IPLTeam
                      |=> abc.txt
Program:
import java.io.*;
class FileDemo{
  public static void main(String[] args)throws IOException{
```

```
      File f1=new File("IPLTeam");
      f1.mkdir();
      File f2=new File("IPLTeam","abc.txt");
      f2.createNewFile();
  }
}
```

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

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

Program:

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

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

Important methods of File class
==========
1. boolean exists();
2. boolean createNewFile()
3. boolean mkdir()
4. boolean isFile();
5. boolean isDirectory()
6. String[] list();
7. long length();
8. boolean delete()

Important methods of file class:

1. boolean exists();
   Returns true if the physical file or directory available.

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

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

4. boolean isFile();
   Returns true if the File object represents a physical file.

5. boolean isDirectory();
   Returns true if the File object represents a directory.

6. String[] list();

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

7. long length();
It returns the no of characters present in the file.

8. boolean delete();
To delete a file or directory

Requirement: Write a program to display the names of all files and directories present in D:\EnterpriseJava
Requirement: Write a program to display only file names.
Requirement: Write a program to display only directory names.

Requirement: Write a program to display the names of all files and directories present
in D:\EnterpriseJava

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

public class TestApp {
	public static void main(String[] args)throws IOException {
		File f=new File("D:\EnterpriseJava");
		String[] s= f.list();
		int count=0;
		for(String s1:s){
			count++;
			System.out.println(s1);
		}
		System.out.println("The no of files are :: "+count);
	}
}
```

Requirement: Write a program to display only file names.

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

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

		File f=new File("D:\EnterpriseJava");
		String[] s= f.list();
		int count=0;

		for(String s1:s){
			File f1=new File(f,s1);
			if (f1.isFile()){
				count++;
				System.out.println(s1);
			}
		}
		System.out.println("The no of Directories are :: "+count);
	}
}
```

Requirement: Write a program to display only directory names

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

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

            File f=new File("D:\EnterpriseJava");
            String[] s= f.list();
            int count=0;

            for(String s1:s){
                  File f1=new File(f,s1);
                  if (f1.isDirectory()){
                        count++;
                        System.out.println(s1);
                  }
            }
            System.out.println("The no of Directories are :: "+count);
      }
}
```

FileWriter
========
Constructors:

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

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

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

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

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

Methods:

1. write(int ch);
   To write a single character to the file.
2. write(char[] ch);
   To write an array of characters to the file.
3. write(String s);
   To write a String to the file.
4. flush();
   To give the guarantee the total data include last character also written to the file.
5. close();
   To close the stream.

eg#1.

```
import java.io.FileWriter;
import java.io.IOException;

public class TestApp {
      public static void main(String[] args)throws IOException {
                  FileWriter fw=new FileWriter("abc.txt");
                  fw.write(73);
```

```
                fw.write("neuron\nTechnology\nPrivate\nLimited");
                fw.write("\n");
                char ch[] ={'a','b','c'};
                fw.write(ch);
                fw.flush();
                fw.close();
      }
}
```

A new file will be created automatically

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

Note:
 => The main problem with FileWriter is we have to insert line separator manually,which is  difficult to the programmer. ('\n')
 => And even line separator varing from system to system.
 =>Represenation of "\n" would vary from system to system.

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

Constructors:
  FileReader fr=new FileReader(String name);
  FileReader fr=new FileReader (File f);

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

2. int i=fr.read();

3. System.out.println((char)i);
      As this method returns unicodevalue , while printing we have to perform type casting.

4. int read(char[] ch);
      It attempts to read enough characters from the file into char[] array and returns
      the no of characters copied from the file into char[] array.

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

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

7. void close();

eg#1.

```
import java.io.FileReader;
import java.io.IOException;

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

eg#2. Reading an array of characters

abc.txt

```
=======
  1000 characters are available
```

Scenario1:

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

Scenario2:

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

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

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

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

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

                  fr.read(ch);

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

                  fr.close();
      }
}
```

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

1. While writing data by FileWriter compulsory we should insert line separator(\n) manually which is a bigger headache to the
     programmer.

2. While reading data by FileReader we have to read character by character instead of line by line  which is not convenient to the
     programmer.

Assume we need to search for a 10 digit mobile no present in a file called "mobile.txt"
  =>Since we can read only character just to search one mobile no 10 searching and to search 10,000 mobile no we need to read 1cr times,
      so performance is very low.

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

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

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

Which of the following declarations are valid?
1. BufferedWriter bw=new BufferedWriter("cricket.txt"); //invalid
2. BufferedWriter bw=new BufferedWriter (new File("cricket.txt")); //invalid
3. BufferedWriter bw=new BufferedWriter (new FileWriter("cricket.txt")); //valid
4. BufferedWriter bw=new BufferedWriter(new BufferedWriter(new FileWriter("crickter.txt")));  //valid

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.

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: 4(newLine())

```
import java.io.*;
class TestApp
{
      public static void main(String[] args)throws IOException
      {
            BufferedWriter bw = new BufferedWriter(new FileWriter("abc.txt"));
            bw.write(73);
            bw.write("neuron");
            bw.newLine();
            bw.write("technology");
            bw.newLine();
```

```
            char ch[] = {'a','b','c'};
            bw.write(ch);


            bw.flush();

            bw.close();

      }
}
```

Note
1.bw.close()// recomended to use
2.fw.close()// not recomended to use
3.bw.close()// not recomended to use
  fw.close()
=> When ever we are closing BufferedWriter automatically underlying writer will be closed and we are not close explicitly.

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

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

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

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

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

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

                  FileReader fr=new FileReader("abc.txt");
                  BufferedReader br=new BufferedReader(fr);
                  String line= br.readLine();
                  while(line!=null){
                        System.out.println(data);
                        line=br.readLine();
                  }
                  br.close();
```

```
      }
}
```
Note:
1.br.close()// recomended to use
2.fw.close()// not recomended to use
3.br.close()// not recomended to use
   fw.close()
=> Whenever we are closing BufferedReader automatically underlying FileReader will be closed it is not required to close explicitly.
=> Even this rule is applicable for BufferedWriter also.

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

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

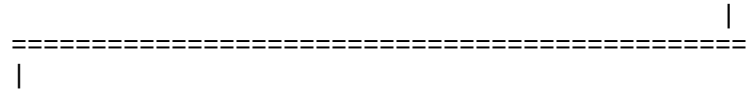
Methods:
```
1. write(int ch);
2. write (char[] ch);
3. write(String s);
4. flush();
5. close();

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);
```

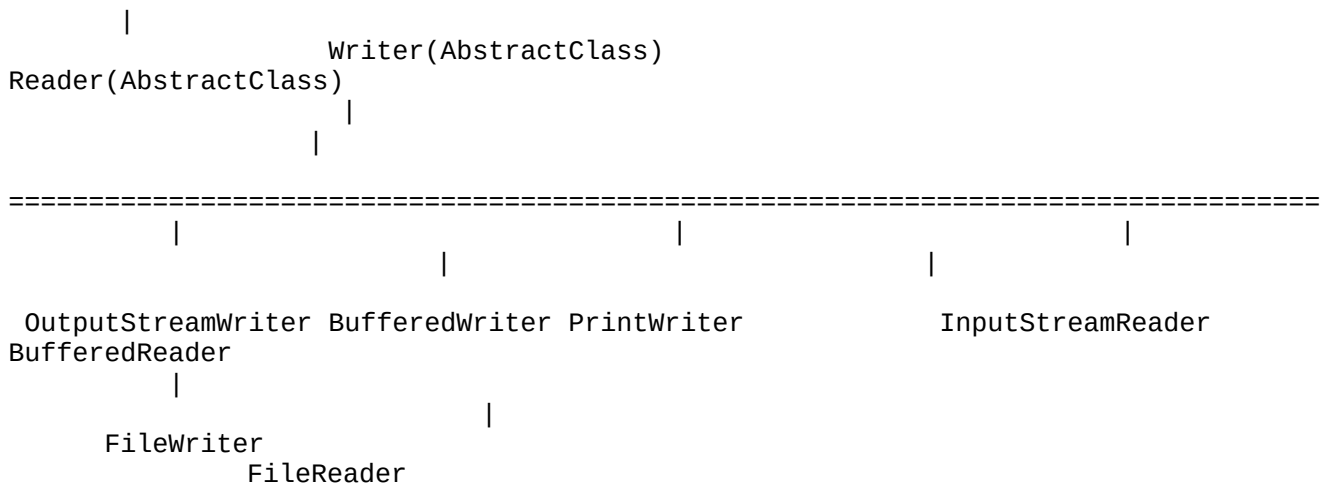
Note 1:
1. The most enhanced Reader to read character data from the File is BufferedReader.
2. The most enhanced Writer to write character data to the File is PrintWriter.

Note 2:
1. In general we can use Readers and Writers to handle character data. Where as we can use  InputStreams and OutputStreams to
    handle binary data(like images, audio files, video files etc).

2. We can use OutputStream to write binary data to the File and we can use InputStream to read  binary data from the File
      Character Data => Reader and Writer
      Binary Data       => InputStream and OutputStream

Note 3:






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

```
import java.io.*;
class TestApp {
	public static void main(String[] args)throws IOException {
		PrintWriter pw =new PrintWriter("file3.txt");

		//copy from file1.txt to file3.txt
		BufferedReader br=new BufferedReader(new FileReader("file1.txt"));
		String line = br.readLine();
		while(line!=null){
			pw.println(line);
			line = br.readLine();
		}

		//copy from file2.txt to file3.txt
		br=new BufferedReader(new FileReader("file2.txt"));
		line = br.readLine();
		while(line!=null){
			pw.println(line);
			line = br.readLine();
		}

		//closing the resources
		pw.flush();
		br.close();
		pw.close();


	}
}
```

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

```
import java.io.*;

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

```
            PrintWriter pw =new PrintWriter("file3.txt");

            //copy from file1.txt to file3.txt
            BufferedReader br1=new BufferedReader(new FileReader("file1.txt"));
            BufferedReader br2=new BufferedReader(new FileReader("file2.txt"));

            String line1= br1.readLine();
            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();
                  }
            }

            //closing the resources
            pw.flush();
            br1.close();
            br2.close();
            pw.close();


      }
}
```

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

```
import java.io.*;

class TestApp
{
      public static void main(String[] args)throws IOException
      {
            PrintWriter pw =new PrintWriter("output.txt");

            //copy from file1.txt to file3.txt
            BufferedReader br1 =new BufferedReader(new FileReader("input.txt"));
            String line = br1.readLine();

            BufferedReader br2 =null;

            while(line!=null)
            {
                  boolean isAvailable = false;
                   br2=new BufferedReader(new FileReader("delete.txt"));
                  String target = br2.readLine();

                  while(target!=null)
                  {
                        if(line.equals(target))
                        {
                              isAvailable = true;
                              break;
```

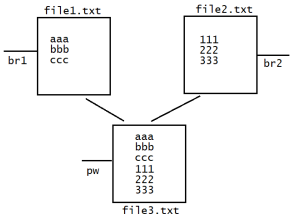
```
                        }
                        target = br2.readLine();
                  }
                  if (isAvailable==false)
                  {
                        pw.println(line);
                        pw.flush();//flush to ensure all data is written to the
file
                  }

                  line = br1.readLine();
            }
            //closing the resources
            br1.close();
            br2.close();
            pw.close();


      }
}
```

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

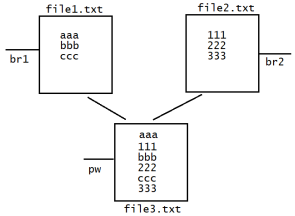
```
import java.io.*;

class TestApp
{
      public static void main(String[] args)throws IOException
      {
            PrintWriter pw =new PrintWriter("output.txt");

            //copy from file1.txt to file3.txt
            BufferedReader br1 =new BufferedReader(new FileReader("input.txt"));
            String line = br1.readLine();

            BufferedReader br2 =null;

            while(line!=null)
            {
                  boolean isAvailable =false;
                  br2 =new BufferedReader(new FileReader("output.txt"));
                  String target = br2.readLine();

                  while(target!=null)
                  {
                        if (line.equals(target))
                        {
                              isAvailable = true;
                              break;
                        }
                        target = br2.readLine();
                  }

                  if(isAvailable==false){
                        pw.println(line);
                        pw.flush();
                  }

                  line = br1.readLine();
```

```
            }
            //closing the resources
            br1.close();
            br2.close();
            pw.close();


      }
}
```

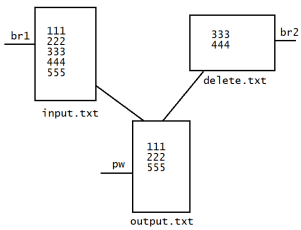
Requirement => file1.txt ,file2.txt copy all the contents to file3.txt



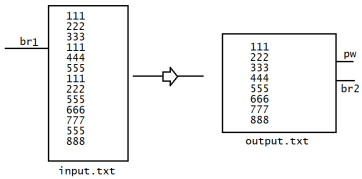

Requirement => file1.txt and file2.txt
copy one line from file1.txt and from file2.txt to file3.txt.

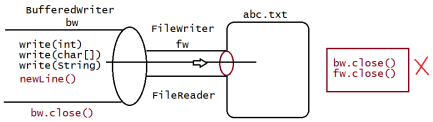


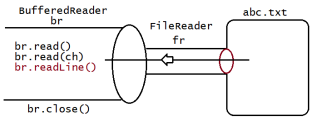

Write a program to perform extraction of mobile no only if there is no duplicates




Write a program to remove duplicates from the file




Bufferedwriter
bw
Filewriter
fw
abc.txt
write(int)
write(char[])
write(String)
newLine()
FileReader
bw.close()
bw.close()
fw.close()


BufferedReader
br
FileReader
fr
abc.txt
br.read()
br.read(ch)
br.readLine()
br.close()

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

Serialization: (1.1 v)
=> The process of saving (or) writing state of an object to a file is called serialization but strictly speaking it is the process of converting an
object from java supported form to either network supported form (or) file supported form.

=> By using FileOutputStream and ObjectOutputStream classes we can achieve serialization process.
|=> writeObject(Object obj)

Ex: using flipkart booking an iPhone and iPhone reaching to the user.

De-Serialization:
=> The process of reading state of an object from a file is called DeSerialization but strictly speaking it is the process of converting an
object from file supported form (or) network supported form to java supported form.
=> By using FileInputStream and ObjectInputStream classes we can achieve DeSerialization.
|=> readObject()

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;
      int j=20;
}

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

            Dog d1=new Dog();

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

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


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

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

      }
}
eg#2.
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.Serializable;

class Dog implements Serializable{
       int i=10;
       int j=20;
}

class Cat implements Serializable{
      int i=100;
      int j=200;
}

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

            Dog d1=new Dog();
            Cat c1=new Cat();

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


            System.out.println("Deserialization started");
            FileInputStream fis=new FileInputStream("abc.ser");
            ObjectInputStream ois=new ObjectInputStream(fis);
            Dog d2=(Dog) ois.readObject();
            Cat c2=(Cat) ois.readObject();
            System.out.println("Deserialization ended");

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

            System.out.println("Cat object data");
            System.out.println(c2.i+"\t" +c2.j);
```

```
      }
}
```

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

Note:
1. We can perform Serialization only for Serilizable objects.
2. An object is said to be Serilizable if and only if the corresponding class implements Serializable interface.
3. Serializable interface present in java.io package and does not contain any 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.if there is a mismatch in order it would result in
"ClassCastException".
5. If we are trying to serialize a non-serializable object then we will get RuntimeException saying "NotSerializableException"

Transient keyword:
1. transient is the modifier applicable only for variables,but not for classes and methods.
2. While performing serialization if we don't want to save the value of a particular variable to meet security constant such type of variable ,
then we should declare that variable with "transient" keyword.
3. At the time of serialization JVM ignores the original value of transient variable and save default value to the file .
4. That is transient means "not to serialize".

eg#1.

```
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)throws
IOException,ClassNotFoundException {

            Dog d1=new Dog();
```

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


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

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

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

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

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

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

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

            Dog d1=new Dog();

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

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


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

```
            FileInputStream fis=new FileInputStream("abc.ser");
            ObjectInputStream ois=new ObjectInputStream(fis);
            Dog d2=(Dog) ois.readObject();

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

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

Transient Vs Final:
1. final variables will be participated into serialization directly by their
values.
     Hence declaring a final variable as transient there is no use.
       //the compiler assign the value to final variable

eg: final int x= 10;
      int y = 20;
      System.out.println(x);// compiler will replace this as System.out.println(20)
becoz x is final.
      System.out.println(y);

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

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


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

            Dog d1=new Dog();

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

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


            System.out.println("Deserialization started");
            FileInputStream fis=new FileInputStream("abc.ser");
```

```
            ObjectInputStream ois=new ObjectInputStream(fis);
            Dog d2=(Dog) ois.readObject();

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

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

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


Declaration output
================
1.
int i=10;
int j=20;
output:: 10   20

2.
transient int i=10;
int j=20;
output:: 0 20

3.
transient int i=10;
transient static int j=20;
output:: 0  20

4.
transient final int i=10;
transient int j=20;
output::  10  0

5.
transient final int i=10;
transient static int j=20;
output: 10 20

Note:
We can serialize any no of objects to the file but in which order we serialized in
the same order only we have to deserialize,
if we change the order then it would result in "ClassCastException".

Example :
Dog d1=new Dog( );
Cat c1=new Cat( );
Rat r1=new Rat( );

FileOutputStreamfos=new FileOutputStream("abc.ser");
ObjectOutputStreamoos=new ObjectOutputStream(fos);
oos.writeObject(d1);
```

```
oos.writeObject(c1);
oos.writeObject(r1);

FileInputStreamfis=new FileInputStream("abc.ser");
ObjectInputStreamois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
Cat c2=(Cat)ois.readObject();
Rat r2=(Rat)ois.readObject();
```

=> If we don't know the order of Serialization then we need to use the following code

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

      Object obj=ois.readObject();/runtime object can be Dog,Cat,Rat
      if(obj instanceof Dog){
            Dog d=(Dog)obj;// parent type type casting
            //perform operation related to Dog
      }
      if(obj instanceof Cat){
            Cat C=(Cat)obj;
          //perform operation related to Cat
      }
      if(obj instanceof Rat){
             Rat r=(Rat)obj;
           //perform operation related to Rat
      }
```

Object graph in serialization:

1. Whenever we are serializing an object the set of all objects which are reachable from that object will be serialized automatically.
   This group of objects is nothing but object graph in serialization.
2. In object graph every object should be Serializable otherwise we will get runtime exception saying "NotSerializableException".

eg#1.

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


class Dog implements Serializable{
      Cat c=new Cat();
}

class Cat implements Serializable{
      Rat r=new Rat();
}

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

```
            Dog d= new Dog();

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

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

            System.out.println("DeSerialization Started");
            FileInputStream fis= new FileInputStream("abc.ser");
            ObjectInputStream ois=new ObjectInputStream(fis);
            Dog d1=(Dog)ois.readObject();
            System.out.println(d1.c.r.i);
            System.out.println("DeSerialization ended");


   }
}
```

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

=> In the above example whenever we are serializing Dog object automatically Cat and Rat objects will be serialized because these are part of object graph of Dog object.
=> Among Dog, Cat, Rat if at least one object is not serializable then we will get runtime exception saying "NotSerializableException".

CustomizedSerialization
=======================
During default Serialization there may be a chance of lose of information due to transient keyword.
example: remember mango and money inside it.

eg#1.

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

class Account implements Serializable{
      String name="sachin";
      transient String password="tendulkar";
}
public class Test {
      public static void main(String[] args)throws
IOException,ClassNotFoundException{
```

```
            Account acc=new Account();
            System.out.println(acc.name +"=====> "+ acc.password);

            System.out.println("Serialization Started");
            FileOutputStream fos= new FileOutputStream("abc.ser");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            oos.writeObject(acc);
            System.out.println("Serialization ended");

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

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

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

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

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

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

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

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

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

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

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

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

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

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

eg#1.

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

class Account implements Serializable{
      String name="sachin";
      transient String password="tendulkar";

      private void writeObject(ObjectOutputStream oos)throws Exception{
            oos.defaultWriteObject();//performing default Serialization
```

```
            String epwd="123"+password;//performing encryption

            oos.writeObject(epwd);//write the encrypted data to file(abc.ser)


      }
      private void readObject(ObjectInputStream ois)throws Exception{
            ois.defaultReadObject();//performing default Serialization

            String epwd=(String)ois.readObject();//performing decryption

            password=epwd.substring(3);//writing the extra data to Object
      }
}
public class Test {
      public static void main(String[] args)throws
IOException,ClassNotFoundException{

            Account acc=new Account();
            System.out.println(acc.name +"=====> "+ acc.password);

            System.out.println("Serialization Started");
            FileOutputStream fos= new FileOutputStream("abc.ser");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            oos.writeObject(acc);
            System.out.println("Serialization ended");

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

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

=>  At the time of Account object serialization JVM will check is there any writeObject() method in  Account class or not.
=>  If it is not available then JVM is responsible to perform serialization(default serialization).
 => If Account class contains writeObject() method then JVM feels very happy and executes that   Account class writeObject() method.
      The same rule is applicable for readObject() method also.

eg#2.

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

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

```
	transient int pin=4444;

	private void writeObject(ObjectOutputStream oos)throws Exception{
		oos.defaultWriteObject();//performing default Serialization

		String epwd="123"+password;//performing encryption
		int epin=1234+pin;//performing encryption


		oos.writeObject(epwd);//write the encrypted data to file(abc.ser)
		oos.writeInt(epin);//write the encrypted data to file(abc.ser)
	}
	private void readObject(ObjectInputStream ois)throws Exception{
		ois.defaultReadObject();//performing default Serialization

		String epwd=(String)ois.readObject();//performing decryption
		int epin=ois.readInt();//performing decryption

		password=epwd.substring(3);//writing the extra data to Object
		pin=epin-1234;//writing the extra data to Object
	}
}
public class Test {
	public static void main(String[] args)throws
IOException,ClassNotFoundException{

		Account acc=new Account();
		System.out.println(acc.name +"=====> "+
acc.password+"======>"+acc.pin);

		System.out.println("Serialization Started");
		FileOutputStream fos= new FileOutputStream("abc.ser");
		ObjectOutputStream oos=new ObjectOutputStream(fos);
		oos.writeObject(acc);
		System.out.println("Serialization ended");

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

		System.out.println("DeSerialization Started");
		FileInputStream fis= new FileInputStream("abc.ser");
		ObjectInputStream ois=new ObjectInputStream(fis);
		acc=(Account)ois.readObject();
		System.out.println(acc.name +"=====> "+
acc.password+"========>"+acc.pin);
		System.out.println("DeSerialization ended");
   }
}
Output
sachin=====> tendulkar======>4444
Serialization Started
Serialization ended
**********************************
DeSerialization Started
sachin=====> tendulkar========>4444
DeSerialization ended

Serialization w.r.t Inheritance
==============================
Case 1:
```

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

That is Serializable nature is inheriting from parent to child.

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

```
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 Serializable{
      int i=10;
}
class Dog extends Animal{
      int j=20;
}

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

      Dog d=new Dog();

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


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

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


   }
}
```

Output

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

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

Note :Object class doesn't implement Serializable interface.

Case 2:
1. Even though parent class does not implements Serializable we can serialize child object if child  class implements Serializable interface.

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

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

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

5. If case of non-serializable parent class then just instance control flow will be performed  and share it's instance variable
     to the current object.

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("Serialization started");
```

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


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

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

```
Output
No arg Animal constructor
No arg Dog constructor
Serialization started
Serialization ended
*****************************
DeSerialization started
No arg Animal constructor
10====> 999
DeSerialization ended
```

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

Externalization : ( 1.1 v )
1. In default serialization every thing takes care by JVM and programmer doesn't have any control.
2. In serialization total object will be saved always and it is not possible to save part of the object , which creates performance problems at certain point.
3. To overcome these problems we should go for externalization where every thing takes care by programmer and JVM doesn't have any control.
4. The main advantage of externalization over serialization is we can save either total object or part of the object based on our requirement.
5. To provide Externalizable ability for any object compulsory the corresponding class should implements externalizable interface.
6. Externalizable interface is child interface of serializable interface.

Externalizable interface defines 2 methods :
1. writeExternal(ObjectOutput out ) throws IOException
2. readExternal(ObjectInput in) throws IOException,ClassNotFoundException

public void writeExternal(ObjectOutput out) throws IOException
This method will be executed automaticcay at the time of Serialization with in this method , we have to write code to save required variables to the file .

public void readExternal(ObjectInput in) throws IOException,ClassNotFoundException
This method will be executed automatically at the time of deserialization with

in this method , we have to write code to save read
   required variable from file and assign to the current object.

At the time of deserialization JVM will create a seperate new object by executing public no-arg constructor on that object JVM will call readExternal() method.
Every Externalizable class should compusory contains public no-arg constructor otherwise we will get RuntimeExcepion saying "InvaidClassException" .
eg#1.

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

class ExternalizableDemo implements Externalizable{
      String i;
      int j;
      int k;

      ExternalizableDemo(String i,int j,int k){
            this.i=i;
            this.j=j;
            this.k=k;
      }

      public ExternalizableDemo(){
            System.out.println("Zero arg constructor");
      }

      //Performing Serialization as per our requirement
      public void writeExternal(ObjectOutput out) throws IOException{
            System.out.println("call back method used while Serialization");
            out.writeObject(i);
            out.writeInt(j);
      }

      //Performing DeSerialization as per our requirement
      public void readExternal(ObjectInput in) throws
IOException,ClassNotFoundException{
            System.out.println("call back method used while DeSerialization");
            i=(String)in.readObject();
            j=in.readInt();
      }

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

      ExternalizableDemo d=new ExternalizableDemo("nitin",100,200);

      System.out.println("Serialization started");
      FileOutputStream fos=new FileOutputStream("abc.ser");
```

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


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

      System.out.println("DeSerialization started");
      FileInputStream   fis=new FileInputStream("abc.ser");
      ObjectInputStream ois=new ObjectInputStream(fis);
      d=(ExternalizableDemo)ois.readObject();
      System.out.println(d.i+"======>"+d.j+"======>"+d.k);
      System.out.println("DeSerialization ended");
   }
}
```

```
Output
Serialization started
call back method used while Serialization
Serialization ended
*****************************
DeSerialization started
Zero arg constructor
call back method used while DeSerialization
nitin======>100======>0
DeSerialization ended
```

1. If the class implements Externalizable interface then only part of the object will be saved in the case output is

```
      public no-arg constructor
       nitin---- 10 ----- 0
```

2. If the class implements Serializable interface then the output is nitin --- 10 --- 20
3. In externalization transient keyword won't play any role , hence transient keyword not   required.

Difference b/w Serialization and Externalization

```
================================================
Serialization
=============
```

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

Externalization

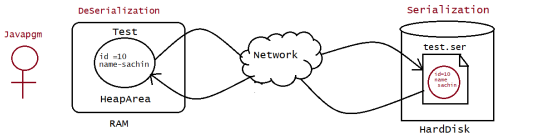
1. It is meant for Customized Serialization
2. Here every thing takes care by programmer and JVM does not have any control.
3. Here based on our requirement we can save either total object or part of the object.

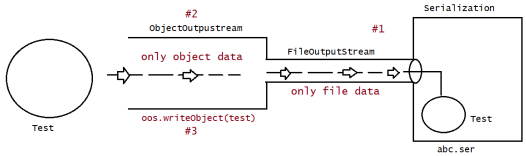
4. Externalization is the best choice if we want to save part of the object.
5. relatively performence is high
6. Externalizable interface contains 2 methods :
    1. writeExternal()
    2. readExternal()
7. It is not a marker interface.
8. Externalizable class should compulsory contains public no-arg constructor otherwise we will get
    RuntimeException saying "InvalidClassException"
9. transient keyword don't play any role in Externalization.

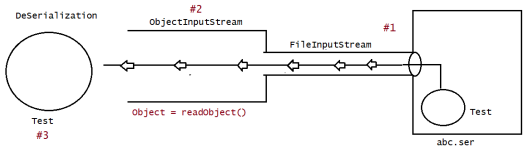
Topics pending
small topic
    a. serialVersionUID
1. Cloneable
    shallow copy, deep copy
2. Different ways of Creating an object
3. Difference b/w ClassNotFoundException vs NoClassDefFoundError
4. Command line arguments

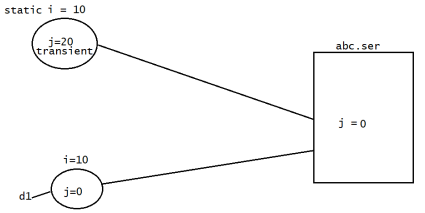
Life of an Object

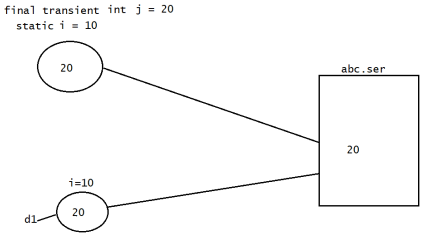
1. As long as the program is executing, the object would be available on RAM, once the program stops its execution then object would be flushed.



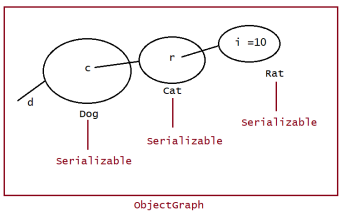



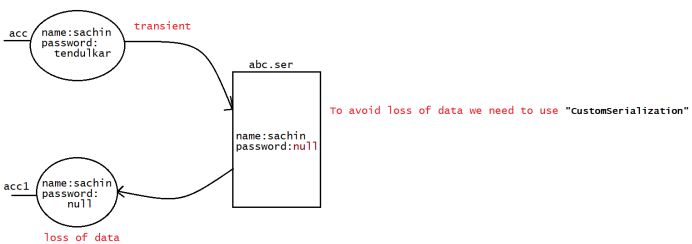





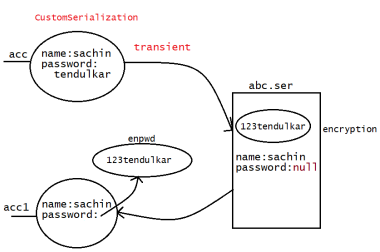



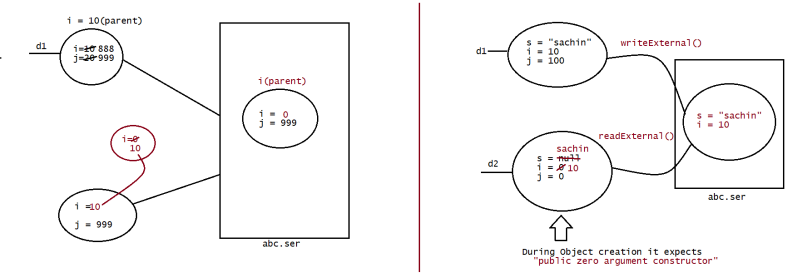

```
class Dog implements Serializable
{
    Cat c =new Cat();
}
class Cat implements Serializable
{
    Rat r = new Rat();
}
class Rat implements Serializable
{
    int i =10;
}
```




To avoid loss of data we need to use **"CustomSerialization"**



Today topics
==========
Small topic
      a. serialVersionUID
  1. Cloneable
       shallow copy, deep copy
  2.  Different ways of Creating an object
  3.  Difference b/w ClassNotFoundException vs NoClassDefFoundError
  4.  Command line arguments
  5.  Singleton class/Design pattern using factory methods
Navin sir => SQL

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

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

Serialization
==========
```
class Dog implements Serializable{
      public static final long serialVersionUID = 1L;
      int i=10;
      int j=20;
}

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

DeSerialization
=============
```
class Dog implements Serializable{
      public static final long serialVersionUID = 1L;
      int i=10;
      int j=20;
}
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");
```

We can solve above problems by configuring our own serialVersionUID .

eg#1.

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

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

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

```
D:\TestApp>javac Dog.java

D:\TestApp>java Sender

D:\TestApp>javac Dog.java

D:\TestApp>java ReceiverApp
10=====>20
```

=> In the above program after serialization even though if we perform any change to Dog.class file we can deserialize object.
=> We can configure our own serialVersionUID both sender and receiver not required to maintain the same JVM versions.
      Note : some IDE's generate explicit serialVersionUID

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.

Signature

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

eg#1.

```
public 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=100;
            t2.j=200;
            System.out.println("Acutal object => "+t1.i+" "+t1.j);
            System.out.println("Cloned object => "+t2.i+" "+t2.j);
   }
}
```

Output

```
Acutal object => 10 20
Cloned object => 100 200
```

KeyPoints about Cloneable interface

==========

=> We can perform cloning only for Cloneable objects.

=> An object is said to be Cloneable if and only if the corresponding class implements Cloneable interface.

=> Cloneable interface present in java.lang package and does not contain any methods.

It is a marker interface where the required ability will be provided automatically by the JVM.

=> If we are trying to perform cloning on non-clonable objects then we will get RuntimeException saying "CloneNotSupportedException".

eg#1.

```
class Cat
{
      int i;
      Cat(int i){
            this.i=i;
      }
}
class Dog implements Cloneable
{
      Cat cat;
      int j;

      Dog(Cat cat,int j){
            this.cat=cat;
            this.j=j;
      }
      public Object clone()throws CloneNotSupportedException{
            return super.clone();
      }
}

public class Test{
      public static void main(String[] args)throws CloneNotSupportedException{
            Cat cat=new Cat(10);
            Dog d1=new Dog(cat,20);
            System.out.println("Acutal object => "+d1.cat.i+" "+d1.j);
```

```
            System.out.println("Perfoming cloning");
            Dog d2=(Dog)d1.clone();
            d2.cat.i=100;
            d2.j=200;

            System.out.println("Acutal object after cloning => "+d1.cat.i+"
"+d1.j);
            System.out.println("Cloned object data          => "+d2.cat.i+"
"+d2.j);
   }
}
Output
Acutal object => 10 20
Perfoming cloning
Acutal object after cloning => 100 20
Cloned object data          => 100 200

Note:
=> Shallow cloning is the best choice , if the Object contains only primitive
values.
=> In Shallow cloning by using main object reference , if we perform any change to
the contained object then those changes will be reflected
      automatically in cloned copy.
=> To overcome this problem we should go for Deep cloning

Deep Cloning :
1. The process of creating exactly independent duplicate object(including contained
objects also)   is   called deep cloning.
2. In Deep cloning , if main object contain any reference variable then the
corresponding Object   copy will also be created in cloned object.
3. Object class clone( ) method meant for Shallow Cloning , if we want Deep cloning
then the  programmer is responsible to implement by
     overriding clone( ) method.

eg#1.
class Cat
{
      int i;
      Cat(int i){
            this.i=i;
      }
}
class Dog implements Cloneable
{
      Cat cat;
      int j;

      Dog(Cat cat,int j){
            this.cat=cat;
            this.j=j;
      }
      public Object clone()throws CloneNotSupportedException{

            Cat c1= new Cat(cat.i);
            Dog d1=new Dog(c1,j);
            return d1;

      }
}
```

```
public class Test{
      public static void main(String[] args)throws CloneNotSupportedException{
            Cat cat=new Cat(10);
            Dog d1=new Dog(cat,20);
            System.out.println("Acutal object => "+d1.cat.i+" "+d1.j);

            System.out.println("Perfoming cloning");
            Dog d2=(Dog)d1.clone();
            d2.cat.i=100;
            d2.j=200;
            System.out.println("Acutal object after cloning => "+d1.cat.i+"
"+d1.j);
            System.out.println("Cloned object data              => "+d2.cat.i+"
"+d2.j);
   }
}
Output
Acutal object => 10 20
Perfoming cloning
Acutal object after cloning => 10 20
Cloned object data               => 100 200
```

Note:
In Deep cloning by using main Object reference if we perform any change to the contained Object those changes won't be reflected to the cloned object.

Example:

```
Test t1=new Test();
Test t2=(Test)t1.clone();
System.out.println(t1==t2); //false
System.out.println(t1.hashCode()==t2.hashCode()); //false
```

Singleton classes :
For any java class if we are allow to create only one object, such type of class is said to be singleton class.
Example:
1) Runtime class
2) ActionServlet
3) ServiceLocator
4) BusinessDelegate

eg#1

```
Runtime r1=Runtime.getRuntime();//getRuntime() method is a factory method
Runtime r2=Runtime.getRuntime();
Runtime r3=Runtime.getRuntime();
.............................................
.............................................
System.out.println(r1==r2);//true
System.out.println(r1==r3);//true
```

Advantage of Singleton class :
If the requirement is same then instead of creating a separate object for every person we will create only one object and
we can share that object for every required person we can achieve this by using singleton classes.

That is the main advantages of singleton classes are Performance will be improved and memory utilization will be improved.

Creation of our own singleton classes:
We can create our own singleton classes for this we have to use private constructor, static variable and factory method.

```
class Test {
            private static Test t=null;
            private Test(){}//to avoid object creation by the user using new
keyword

            public static Test getTest() //getTest() method is a factory method
            {
                  if(t==null){
                        t=new Test();
                  }
                  return t;
            }
}
class Client{
            public static void main(String[] args){
                  System.out.println(Test.getTest().hashCode());//1671711
                  System.out.println(Test.getTest().hashCode());//1671711
                  System.out.println(Test.getTest().hashCode());//1671711
                  System.out.println(Test.getTest().hashCode());//1671711
            }
}
```

We can create any xxxton classes like(double ton,triple ton...etc)

Example:

```
class Test {
            private static Test t1=null;
            private static Test t2=null
            private Test(){}

            public static Test getTest()//getTest() method is a factory method
            {
                  if(t1==null){
                        t1=new Test();
                        return t1;
                  }
                  else if(t2==null){
                        t2=new Test();
                        return t2;
                  }
                  else{
                        if(Math.random()<0.5) //Math.random() limit : 0<=x<1
                              return t1;
                        else
                              return t2;
                  }
      }
}
public class Client{
      public static void main(String[] args){
                  System.out.println(Test.getTest().hashCode());//1671711
```

```
                System.out.println(Test.getTest().hashCode());//11394033
                System.out.println(Test.getTest().hashCode());//11394033
                System.out.println(Test.getTest().hashCode());//1671711
      }
}
```

Factory method:
By using class name if we are calling a method and that method returns the same class object such type of method is called factory method.

Example:

```
Runtime r=Runtime.getRuntime();//getRuntime is a factory method.
DateFormat df=DateFormat.getInstance();
```

If object creation required under some constraints then we can implement by using factory method.

```
Calendar calendar = Calendar.getInstance();//static factory methods
String result = "name".toUpperCase();//instance factory methods
```

Different ways of Creating an Object

1. using new Operator

```
Test t=new Test();
```

2. using newInstance()

```
Class.forName("com.abc.main.Test").newInstance()
```

3. using clone()

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

4. using factorymethods

```
Runtime r=Runtime.getRuntime();
DateFormat df=DataFormat.getInstance();
```

5. using Serialization and DeSerialization

```
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Test t=(Test)ois.readObject();
```

new Vs newInstance( ) :
1. new is an operator to create an objects , if we know class name at the beginning then we can create an object by using new operator .
2. newInstance( ) is a method presenting class " Class " , which can be used to create object.
3. If we don't know the class name at the beginning and its available dynamically Runtime then we should go for newInstance() method

```
public class Test {
      public static void main(String[] args) throws Exception {
             Object o=Class.forName(arg[0]).newInstance( ) ;
            System.out.println(o.getClass().getName( ) );
      }
}
```

If dynamically provide class name is not available then we will get the RuntimeException saying ClassNotFoundException
To use newInstance( ) method compulsory corresponding class should contains no argument constructor ,
otherwise we will get the RuntimeException saying "InstantiationException".

if the constuctor is private then it would result in "IllegalAccessException"

Difference between new and newInstance( ) :
new
===
new is an operator , which can be used to create an object.
We can use new operator if we know the class name at the beginning.
      Test t= new Test( );
If the corresponding .class file not available at Runtime then we will get RuntimeException saying NoClassDefFoundError , It is unchecked.
To used new operator the corresponding class not required to contain no argument constructor

newInstance( )
============
newInstance( ) is a method , present in class Class , which can be used to create an object .
We can use the newInstance( ) method , If we don't class name at the beginning and available dynamically Runtime.
      Object o=Class.forName(arg[0]).newInstance( );
If the corresponding .class file not available at Runtime then we will get RuntimeException saying ClassNotFoundException , It is checked.
To used newInstance( ) method the corresponding class should compulsory contain no argument constructor , Other wise we will get
RuntimeException saying InstantiationException.

Difference between ClassNotFoundException & NoClassDefFoundError :
1. For hard coded class names at Runtime in the corresponding .class files not available we will get NoClassDefFoundError ,
    which is unchecked
            Test t = new Test( );
            In Runtime Test.class file is not available then we will get "NoClassDefFoundError"

2. For Dynamically provided class names at Runtime , If the corresponding .class files is not available then we will get the
    RuntimeException saying "ClassNotFoundException".
            Ex : Object o=Class.forname("Test").newInstance( );
      At Runtime if Test.class file not available then we will get the "ClassNotFoundException" , which is checked exception.

Difference between instanceof and isInstance( ) :
instanceof
instanceof an operator which can be used to check whether the given object is particular type or not We know at the type at
beginning it is available.

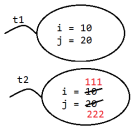
eg: String s = new String("sachin");
System.out.println(s instanceof Object );//true
      //If we know the type at the beginning only.

isInstance( )
isInstance( ) is a method , present in class Class , we can use isInstance( ) method to checked whether the given object is particular type or
not   We don't know at the type at beginning it is available Dynamically at Runtime.

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

```
                Test t = new Test( ) ;

      System.out.println(Class.forName(args[0]).isInstance(t));////arg[0] --- We
don't know the type at beginning
      }
}
java Test Test //true
java Test String //false
java Test Object //true
```

```
class Test
{
      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 =111;
          t2.j =222;
          System.out.println(t1.i + "=====>" +t1.j);
          System.out.println(t2.i + "=====>" +t2.j);
     }
}
```



```
class Parent{
      public Object m1(){}
}
class Child extends Parent{
      public String m1(){}
}

Object
   |
   |
String
```

sir what is significance of  System.out.println()  is asked to me in interview sir

System => It is a predefined class present in java.lang package
out        => It is a static variable which holds the reference of PrintStream class
pritnln => it is a method of PrintStream class

main(String[] args)  jvm does not regcognize without this argument why sir?
 => Any data sent to the main method will be collected by args[] which is of String type
      so jvm recognise only main(String[] args) as the first point of contact for execution

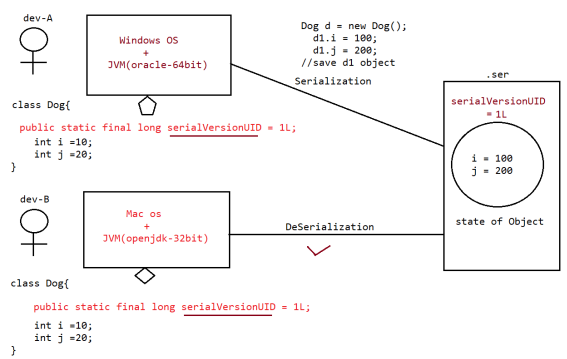
sir what is call by value and call by reference in java? i have come across this question in interview?

```
void add(int  a,int b){

}
int x =10;
int y =20;
add(x,y);//pass by value

void add(Student s1,Student s2){

}
Student std1 =new Student();
Student std2 =new Student();
add(std1,std2);//pass by reference
```

what are all the types of access modifiers??
    11 access modifiers
      private, public,protected
      static,synchronized,strictfp
         final,abstract,native
         transient,volatile

private void readObject() will read that object in the file when we say
      ==> password = empwd.substring(3).
password is a part of object which is readed from the file not the instance variable right?
answer: read from file

private void readObject(ObjectOutputStream oos){} sir here we are not initializing the reference then how we doing oos.read() like this can u explain internally?
Answer:Object of OutputStream is creating and that reference is collected in oos.


dev-A
Windows OS
+
JVM(oracle-64bit)
Dog d = new Dog();
d1.i = 100;
d1.j = 200;
//save d1 object
Serialization
.ser
serialVersionUID
= 1L
i = 100
j = 200
state of Object
class Dog{
public static final long serialVersionUID = 1L;
int i =10;
int j =20;
}
dev-B
Mac os
+
JVM(openjdk-32bit)
DeSerialization
class Dog{
public static final long serialVersionUID = 1L;
int i =10;
int j =20;
}

Today it is QA discussion from 2.05PM IST
  Enterpise Java batch(weekends only)
  FullStackJobGaurateed batch(daily 7.30PM IST to 10.30PMIST)
  contact details
      haider : syedhaider@ineuron.ai
      nitin    : nitin@ineuron.ai

Q>
sir how to take input from the user in the same line? If we write like this
```
Scanner scan= new Scanner (System.in);
System.out.println("Enter the value of a:");
int a=scan.nextInt();
System.out.println("Enter the value of b:");
int b= scan.nextInt();
```
then for b value it will go to next line

```
java FileName   10 20
 Integer  a = (Integer) arg[0]
 Integer  b = (Integer) arg[1];
```

Q> I pass the array to funcation that function smply only return what it will do i got from  one mcq

```
class A {
      psvm(String[] args){
            int arr[] ={1,2,3,4,5};
            m1(arr)

            for(int elem: arr)
                  System.out.print(arr);//1 ,2 ,3,50,5
      }
      public static void m1(int[] arr){
             arr[3] = 50;
             return;
      }
}
```

Q>
 for (int i=0;i<10 ;i+=2 ) sir  in for loop  after boolean it is necessary increment/decrement
   but here  inisilize i+=2

```
for(stm1;stmt2;stm3){
      //body of loop
}
```
stmt1=> any valid java statement but suggested in initalisation one
stmt2 => compulsorily boolean only
stmt3 => any valid java statement but suggested in increment/decrement
                        eg: i++,++i, i = i+1, i+=1,i+=2

Q> sir first statement  we can write sop(" hii")
Answer :yes

Q>how to define 2D ARRAY confusion in brackets?

```
int[] a ={10,20,30};
int[][] a = {
                        {10,20,30},
```

```
                        {40,50,60}
                  }
int[][][] a = {
                        {
                              {10,20,30},
                              {40,50,60}
                         },
                        {
                              {10,20,30},
                              {40,50,60}
                         }
                  };
```

Q>
how to declare array of class?

```
int[] arr =new int[5];//array of 5 integers
Student[] std=new Student[5];//array of 5 Students
```

Q>
what does in.ineuron.folder1.*(implicit import); mean? what I want to ask is, is it going to
add or load all the classes at folder1's level or it adds all the classes
let's say in.ineuron.folder1.abc;(explicit import)
in.ineuron.folder1.xyz;

Q> Stack extends Vector, we have addElement(Object e) from Vector class,
also we have push(Object e) so what is the difference in these and which to use when?
what makes them different?
Answer: Code would result in CompileTime Error.

Q>

```
int i=1,j=10;
      do{
            if(i++>--j)// 4>6
                  continue;
      }while(i<5);
     sop(i+""+j); // 5 6
```

sir what is the output?
JVM
i = 1,2,3,4,5
j = 9,8,7,6

Comparable and Comparator
==========

TreeSet:
 => The Underlying Data Structure is Balanced Tree.
 => Insertion Order is Not Preserved and it is Based on Some Sorting Order.
 => Heterogeneous Objects are Not Allowed. If we are trying to Insert we will get Runtime Exception Saying ClassCastException.
 => Duplicate Objects are Not allowed.
=>  null Insertion is Possible (Only Once).
=>  Implements Serializable and Cloneable Interfaces but Not RandomAccess Interface.

Constructors:
1) TreeSet t = new TreeSet();
      Creates an Empty TreeSet Object where all Elements will be Inserted According to Default Natural Sorting Order.
2) TreeSet t = new TreeSet(Comparator c);
      Creates an Empty TreeSet Object where all Elements will be Inserted According to
      Customized Sorting Order which is described by Comparator Object.
3) TreeSet t = new TreeSet(Collection c);
4) TreeSet t = new TreeSet(SortedSet s);

eg#1.
```
import java.util.TreeSet;
classTreeSetDemo {
      public static void main(String[] args) {
                  TreeSet t = new TreeSet();
                  t.add("A");
                  t.add("a");
                  t.add("B");
                  t.add("Z");
                  t.add("L");
                  t.add(new Integer(10));//ClassCastExcpetion
                  t.add(null);//RE: Exception in thread "main" java.lang.NullPointerException
                  System.out.println(t); //[A, B, L, Z, a]
      }
}
```

eg#2.
```
import      java.util.TreeSet;
class  TreeSetDemo {
            public static void main(String[] args) {
                  TreeSet t = new TreeSet();
                  t.add(new StringBuffer("A"));
                  t.add(new StringBuffer("Z"));
                  t.add(new StringBuffer("L"));
                  t.add(new StringBuffer("B"));
                  System.out.println(t);
       }
}
```
RE: Exception in thread "main" java.lang.ClassCastException: java.lang.StringBuffer cannot be cast to java.lang.Comparable

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.
Hence we are ClassCastException in the Above Exampl

Comparable (I):
Comparable Interface Present in java.lang Package and it contains Only One Method compareTo().

obj1.compareTo(obj2)
Returns –ve if and Only if obj1 has to Come Before obj2.
Returns +ve if and Only if obj1 has to Come After obj2.
Returns 0 if and Only if obj1 and obj2 are Equal.

eg#1.

```
System.out.println("A".compareTo("Z")); //-ve value
System.out.println("Z".compareTo("K")); // +value
System.out.println("Z".compareTo("Z")); // zero
System.out.println("Z".compareTo(null));//NPE
```

Wheneverwe are Depending on Default Natural Sorting Order and if we are trying to Insert Elements then Internally JVM will
Call compareTo() to IdentifySorting 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);
```

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.

Comparator (I):
This Interface Present in java.util Package.
Methods: It contains 2 Methodscompare() and equals().

```
public  int compare(Object obj1, Object obj2);
```

Returns –ve if and Only if obj1 has to Come Before obj2.
Returns +ve if and Only if obj1 has to Come After obj2.
Returns 0 if and Only if obj1 and obj2 are Equal.

```
public boolean equals(Object o);
```

Whenever we are implementing Comparator Interface 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 the 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 Customized Sorting (Descending Order). In this Case the Ouput is [20, 15, 10, 5, 0]

Various Possible Implementations of compare():

```
public int compare(Object obj1, Object obj2) {
            Integer I1 = (Integer)obj1;
            Integer I2 = (Integer)obj2;
            return I1.compareTo(I2);
            return -I1.compareTo(I2);
            return I2.compareTo(I1);
            return -I2.compareTo(I1);
            return +1;
            return -1;
            return 0;
}
```

Output:

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("ponting");
                  t.add("sangakara");
                  t.add("fleming");
                  t.add("lara");
                  System.out.println(t);
```

```
	}
}
class   MyComparator implements Comparator {
	public  int compare(Object obj1, Object obj2) {
			String s1 = obj1.toString();
			String s2 = (String)obj2;
		return s2.compareTo(s1);
		//return -s1.compareTo(s2);
	}
}
```

Write a Program to Insert StringBuffer 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("L"));
			System.out.println(t);
	}
}
class MyComparator1 implements Comparator {
	publicint 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 then Consider their Alphabetical Order:

```
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);
```

```
	}
}
```
Note:
If we are Depending on Default Natural Sorting Order Compulsory Objects should be Homogeneous and Comparable Otherwise we will
get RE: ClassCastException.

If we defining Our Own Sorting by Comparator then Objects Need Not be Homogeneous and Comparable. That is we can
Add Heterogeneous Non Comparable Objects to the TreeSet

When we go for Comparable and When we 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 is satisfied with Default Natural Sorting Order 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 EmployeeId and
Customized Sorting Order is Based on Alphabetical Order of Names:

```
import  java.util.*;
class Employee implements Comparable {
            String name;
            int   eid;
            Employee(String name, inteid) {
                  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("ponting", 14);
            Employee e3 = new Employee("lara", 9);
            Employee e4 = new Employee("flintoff", 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);
      }
}
```

Comparison of Comparable and Comparator:

Comparable
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.

Comparator
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 RuleBaseCollator.

can you please explain covariant return type, i have problem to understand what its actually return.

```
class Parent{
      //returns Object
      public Object getObject(){
            return new Object();
      }
}
class Child extends Parent{

     //returns String
    @Override
    public String getObject(){
            return new String("sachin");
    }

}
```

Covariant Type
==========

```
Object
    |
    |IS-A
    |
String
```

how to take input from user of HAS-A variables

```
    class University{
         //HAS-A
        private  Department dept;
    }
   class Department{
        private Integer deptId;
        private String depName;

       //setXXXX and getXXXX methods
   }
```

sir concrete means in short ??

concrete -> It refers to completeness

abstract -> It refers to incompleteness

Output of the code
==========

```
package Practice;
class Demo1 extends Demo {

      //It is a specialized method
      public void display() {
            System.out.println("train");
      }
}
class Demo3 extends Demo{

         //It is a specialized method
       public void display() {
             System.out.println("In Demo3");
       }
```

```
 }
public class Demo{ //parent class

	private void display() {//parent method
		System.out.println("trainee");
	}
	public static void main(String[] args) {
		Demo d = new Demo1();
		d.display();//trainee
	}
}
```

Note: Since the method is not overriding,JVM will not bind the method call, rather Compiler will bind the call based on the reference type.4
If it is overriding, then JVM will bind the method call based on the run time object, where compiler would just perform type checking.

If we have two default methods in two interfaces, lets assume both the methods have same name.
Child class implements both the interfaces. we can only override one method in child class.
Then how are we beating ambiquity in this case??Please try this in editor.

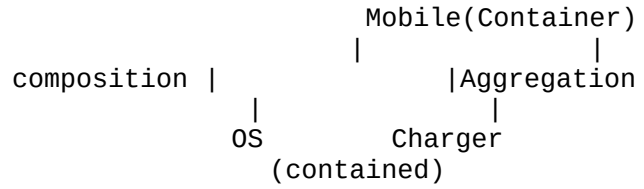
```
interface Left{
	default void m1(){
		System.out.println("Default method from Left");
	}
}
inteface Right{
	default void m1(){
		System.out.println("Default method from Right");
	}
}
class Test implement Left,Right{}//CompileTime Error.

class Test implement Left,Right{
	@Override
	public void m1(){
		//syntax to call interface specific methods
		InterfaceName.super.methodName()

		Left.super.m1();//Default method from Left
		Right.super.m1();//Default method from Right
		System.out.println("Implementation from Test Class");
	}
	p.s.v.m(String[] args){
		Test t1=new Test();
		t1.m1();//Implementation from Test Class
	}
}
```

Dependent object Not ready means what?? means please explain term not ready a

```
  Employee(Target Object)
	|
	|
	|
  Account(Dependent Object)
```

In composition how does target object gets deleted automatically? how dependency

object becomes null? just confused on this.
Composition => Container and Contained Object




Note: Interface static method will not reach to its implementation class,so it is possible to access through child
reference it should be accessed only using interface name.

```
interface X{

   //utility Method
   public static void foo(){
      System.out.println("foo");
   }
}

class Y implements X{
      //static method will not participate in inheritance, so overiding here is not possible
      public static void foo(){
            System.out.println("hello from Y");
      }
}

public class Z
{
   public static void main(String[] args)
   {
      X.foo(); // it is valid
      Y.foo(); //CE

      Y y =new Y();
      y.foo(); //Hello from Y(compiler will bind the method call)

      X x =new Y();//Compiler will bind the call based on reference type
      x.foo();//foo

   }
}
```

Overriding => Compiler duty is to just use the reference and check whether the methods are
available in the respective class or not,and jvm duty is to bind the method call
based on the run time object.

Overloading/Method hiding => Compiler duty is to check the reference and bind the method call based
on the method signature, jvm duty is to just the execute the method which
is binded by the compiler.

```
Compile Time
============
      Compiler will perform TypeChecking
            a. variable type checking
                        byte a = 127;//valid
                        byte b= 300;//CE
            b. reference type checking
                        class Parent{
                                    public void m1(){}
                        }  class Child extends Parent{
                                    public void m2(){}
                        }
                        Parent p=new Child();
                        p.m1();//valid
                        p.m2();//invalid



RunTime
=======
      It creates the Object and perform the desired operation by communicating with
JVM
      class Parent{
            public void m1(){}
      }  class Child extends Parent{
                        @Override
                        public void m1(){}
      }

            Parent p=new Parent();
            p.m1();//parent class m1() executed

            Child c=new Child();
            c.m1();//child class m1() executed


            Parent p=new Child();
            p.m1();//child class m1() executed


abstract class => does it have a constructor? yes
                     Constructor chaining possible ? yes

interface        => does it have a constructor? no
                    Constructor chaining possible? no

Customer/Software Requirement specification
   interface  Remote{
            int minVolume =0;
            int maxVolume =100;

            public String changeChannels();
   }

            1                                                                    M
JAVA  =====SRS(OracleTeam)==================> Database
                                                                        a. MySQL
                                                                        b. Oracle
                                                                        c.
```

PostgreSQL

d. Sybase

infinite for loop syntax
==========
```
for(;;)
   System.out.println("Hello");//infinite hello
```

Association
========
```
                M                                   1
   Employee---------------------------------- Department
```

Ananomyous innner class---------- > interface implemenation

Lambda Expression ----------------> FunctionalInterface implementation

```
interface IFirstSample{
      //few requirements
}

interface ISecondSample extends IFirstSample{
      //few requirements

      //special requirements
}
```

JDBC API
========
```
interface Statement{
      //abstract methods
}
interface PreparedStatement extends Statement{
      //abstract methods
}
interface CallableStatement extends PreparedStatement{
      //abstract methods
}
```

QA is disabled(**Security reason)
---
Tommor topic : Collection continuation including all legacy classes.

sir can you please explain var ARGS one more time
varargs -> Introduced in jdk1.5V
Syntax: datatype ...variable
Note :internally the var arg variable will be used as an array only.

```
class Demo{
	public void add(int... x){//internally will be treated like int a[]
		//it will print the value of the arguments collected
		for(int data: x)
			System.out.println(data);
	}
}
class Test{
	public static void main(String[] args){
			Demo d = new Demo();
			d.add(10,20);
			d.add(10,20,30);
			d.add(10,20,30,40);
	}
}
```

Producer Consumer Problem
---

```
package in.ineuron.producerconsumer.main;

//producer thread operations
class Producer extends Thread {

	// Producer producing the data in StringBuffer
	StringBuffer sb;

	public Producer() {
		// StringBuffer object is created with a default capacity 16
		sb = new StringBuffer();
	}

	@Override
	public void run() {

		synchronized (sb) {

			for (int i = 1; i <= 10; i++) {
				try {
					sb.append(i + ": ");
					Thread.sleep(100);
					System.out.println("appending");
				} catch (InterruptedException e) {
					e.printStackTrace();
				}

			}
			// send the notification to the waiting thread
			sb.notify();
		}

	}
```

```
}

//Consumer thread operations
class Consumer extends Thread {

	// Creating producer object to get the produced data from StringBuffer
	Producer producer;

	// injecting the Producer Object into Consumer
	public Consumer(Producer producer) {
		this.producer = producer;
	}

	@Override
	public void run() {

		synchronized (producer.sb) {

			try {

				//wait till the notification is sent by producer
				producer.sb.wait();

				// consume the data produced by the producer
				System.out.println(producer.sb);

			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}
}

//Effecient way of interthread communication  using wait() and notify()
public class BetterCommunication {

	//Driving code where he start the other thread
	public static void main(String[] args) {

		Producer obj1 = new Producer();
		Consumer obj2 = new Consumer(obj1);

		Thread t1 = new Thread(obj1);// producer thread
		Thread t2 = new Thread(obj2);// consumer thread

		t2.start();// consumer should wait
		t1.start();// producer should start
	}
}


2 Threads wants to share the resource such that data inconsistency should not
happen
			"synchronized"


1st Thread ----> updating an object with some value
```

2nd Thread ----> get the value of the Object (wait)

wait() ----> lock is released from the Object and given to the Thread who wants to update the
value for the Object.

notify() ---> The thread now got the lock ,using the lock update the value for the Object,notify
to the waiting thread that hey u gave the lock i used and i have update plz take the
updated value and use it.

object level -> method should be just synchronized
class level -> method should be static synchornized

native -> code is not from java language, code is from other community languages like
c, c++, python

```
static synchronized {

}

synchronized
{

}
```

Q>
can you please tell what this is error cannot find symbol?

```
String name = "sachin";
System.out.println(name.length);//CE: can't find symbol :: length
```

Q>
In Enum, Why was the constructor called 3 times for-(Fail,Pass, NoResult), when we were only using PASS as reference ?

```
enum EResult{

	// public static final EResult PASS =new EResult();
	// public static final EResult FAIL =new EResult();
	PASS,FAIL;
	EResult(){
		System.out.println("constructor of enum is called");
	}
}
public class Test {
	public static void main(String[] args) {

		EResult eResult =null;
		eResult=EResult.PASS;

	}
}
```

Sir, I have a doubt. When we are creating an object, in the heap it will be created and its instance variables also in heap. But what about the methods present in the object sir? Will it be created only once it is called and those methods will find space in stack? Is this correct sir?
those named constants are refernce of enum type, so only object is created.

Q>
Sir, I have a doubt. When we are creating an object, in the heap it will be created and its instance variables also in heap.
But what about the methods present in the object sir?
Will it be created only once it is called and those methods will find space in stack? Is this correct sir?

method area -> .class file and static varaible memory(jvm will give default value as per the datatype)
stack area -> local variables will get memory followed by the code which needs to executed by JVM
heap area -> Object data(instance variable) will get memory and jvm will give default value as per the datatype

Q>
@FuncationalInterface annotation used in interface its allowed to only one abstract method ,
if more than one abstract method try to add its shows error, like that I had requirement to Interface to allow
only two abstract method , is it possible to create any custom annotation to achieve this?

Answer: yes possible with custom annotations.(too complex)

Q>
Sir I was trying to input values with for each:Source codeimport java.util.*;

```
public class VarargsDemo {
```

```
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		int n=5;
		int nums[]=new int[n];

		//foreach loop should not be used for writing the data to array.
		for(int i:nums) {
			System.out.println("Enter number:");
			i=sc.nextInt();
		}
		//foreach loop is used to read the data from array
		for(int j:nums) {
			System.out.println(j+"\tnumber:"+nums);
		}
		Calc obj=new Calc();
		int result=obj.add(nums);
		System.out.println(result);

	}

}

class Calc{
	int sum=0;
	public int add(int... nums) {
		for(int i:nums) {
			sum=sum+i;
		}
		return sum;
	}
}Output is:Enter number:
11
Enter number:
12
Enter number:
23
Enter number:
34
Enter number:
23
0	number:[I@62ee68d8
0	number:[I@62ee68d8
0	number:[I@62ee68d8
0	number:[I@62ee68d8
0	number:[I@62ee68d8
0What is wrong?
```

Q>
Sir one of the advantage of collection (ArrayList over ) Arrays is it store heterogeneous data but when Generic
was introduced in JDK 1.5 then this advantage is no longer but still we have one advantage of it is it is
dynamic so we can add element to it whereas in Arrays it is fixed size that is if we want to add element to it then we have to create a new Array object
Sir the statement I had wrote is correct?

Answer: Correct

Q>
how to use cmd to extract info about pakages and interface? if i try i got error
    javap fullyqualified classname

      eg: javap java.lang.Object

Q>
what is fill ratio??
  fillration-> upon how much filling the data into the collection, i need to increase my capacity is specified through fillratio.

Q>
Can you explain :: (double colon) operator and use?
      :: -> method reference and also has constructor reference.
      usage method reference can be replace in lambda expression also

```
                eg: ArrayList<integer> al =new ArrayList<Integer>();
                                al.add(0); al.add(10); al.add(5);
                      al.stream().forEach(System.out::print);
```

Innerclass --> required in SpringBoot projects
filehandling---> without knoweldge of this io can't be attained(JDBC)
Serialization,DeSerialization -> Hibernate and JDBC internaly work can't be understood.

```
contact details
============
    haider : syedhyder@ineuron.ai
    nitin     : nitin@ineuron.ai

import java.util.Scanner;
class Two_Sum{
    public  int[] sum(int arr[],int target){
            int[] index;
           looop: for(int i =0 ;i<arr.length;i++){
             for(int j=i+1;j<arr.length-1;j++)
              {
                        int sum=arr[i]+arr[j];
                        if (sum==target){
                              System.out.println("[" + i+ ","+ j+"]");
                          index = new int[2];
                        index[0]=i;
                        index[1] = j;
                              break loop;
                        }
                   }
            return index;
        }
    }
    public static void main(String [] args){
            Two_Sum obj= new Two_Sum();
            int arr[]= new int[10];
            Scanner sc= new Scanner(System.in);
            for(int i=0 ; i<arr.length;i++){
                              arr[i]=sc.nextInt();
            }
            System.out.println("Enter the target element:");
            int target=sc.nextInt();
            obj.sum(arr,target);
    }
}

Why Main method only accept String types of Arguments?

java Test sachin 10 IND
         |
         |
        Test.main(new String[]{"sachin","10","IND"})
If the data is collected in String type, then we can covert the String object to
any type using "Wrapper classes".
class Test{
      public static void main(String[] args){

      }
}


Q>
Example 1:

Input: nums = [2,7,11,15], target = 9
      Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Q> once explain this keyword sir

```
class Student
{
	//instance variables
	String name; int age;

	//local variables are name,age
	Student(String name,int age)
	{
		this.name  =name;
		this.age = age;
	}
}
	Student std = new Student("sachin",10);
```

Note: when name clash occurs between local and instance variable, jvm will give preference only for local variable this concept is called
as Shadowing.
this keyword holds the address of current object, so it resolves the name clash b/w local and instance variables.

From your session on 3rd Nov: Immutability is the main disadvantage of SCP, got it.

But, on 1st Nov, you said creating String literal without newing up is the most efficient way.
Could you please clarify that for me? Thanks in advance!

refer: adhar card application.

Q> What is the difference between static and non-static?

static => common for all the objects of a class.
nonstatic/instance => specific to particular object of a class.

Q>
can you explain about null ...null is belong to which category like object or primitive type or string like that..who is null?
Object  variable we call as reference default value is null.
eg: String name = null;
Integer i = null;

Q> Sir Can you discuss about static control flow.

```
class Test{
	static int i;// memory will be located in method area , it will be loaded at the time of loading .class file
	int j;

	static
	{
		System.out.println("Test .class file is loading");
		i =10;
	}
	//instance block will be executed
	{
			j = 20;
```

```
	}
	//constructor will be executed
	public Test()
	{
		j = 100;
	}
}
Test t = new Test();
1 . load the .class file
2. during the loading of .class file memory for static variable will be given with
default value
3. static block will be executed
4. instance block will be executed
5. constructor will be executed.

Q> when to use static method and when to use non static method?
	static method => helper method/ utility methods, if we want to give
facility of the logic to the user
				without creating the objects of its class then make
such methods as "static".
	non-static-method => normally the methods will be non-static as we need the
object to create and then
					get the service of the method.


Q> explain about pass by value and pass by reference

	int a = 10;
	int b = 20;
	add(a,b);//pass by value becoz we are sending primitive type data

			10    20
public void add(int x, int y)
{
	System.out.println("The sum is :: "+(x+y));
}

pass by reference
===============
   Student s1 =new Student();
   Student s2 =new Student();
	validate(s1,s2);//pass by reference as we are sending the object type

public void validate(Student x,Student y)
{

}

Encapsulation
===========
    Process of keeping the data member private and exposing those value to the
outer world through methods is called "encapsulation".

public class Student
{
	private String name;
	public Student(String name){this.name =name;}
	public void setName(String name){this.name =name;}
	public String getName(){return name;}
```

```
}
```

Good Afternoon guys,
    Post ur Questions in QA i will answer.

contact details
      hyder: syedhyder@Ineuron.ai
      niitn  : nitin@Ineuron.ai

Q> when a String having null value give null point Exception and when it will treat null as a value ?

```
       String name = null;
      System.out.println(name.toUpperCase());// NullPointerException

            vs
      String name = "null";
      System.out.println(name.toUpperCase());//NULL
```

Q>I'm having a small doubt about constructor chaining.

Answer:

```
class Object
{
      Object()
      {

      }
}
class Parent extends Object
{
      Parent()
      {
            super();
      }
}
class Child extends Parent
{
      Child()
      {
            super();
      }
}
new Child();
```

Q>

```
class Account implements Serializable
{
      private String name = "sachin";
      transient private String pwd = "123";

      private void writeObject(ObjectOuputStream oos)
      {
            oos.defaultWriteObject();//name = sachin, pwd = null

            String encypwd = 1234+pwd;
            oos.writeObject(encypwd);// ObjectOuputStream class writeObject() will be called encypwd= "1234123"
```

```
	}

	private void readObject(ObjectInputStream oos)
	{
		ois.defaultReadObject();//name = sachin, pwd = null
		String decrypwd= (String)ois.readObject(encypwd);
		pwd = decrypwd.substring(3);//123
	}

}
	Account acc =new Account();
	ObjectOutputStream oos=new ObjectOutputStream(fos);
	oos.writeObject(acc);// jvm will check whether the Account class contains
writeObject(ObjectOutputStream)
                                    if it is not available in Account class then
jvm will call ObjectOutputStream writeObject().

Q>
String s1=null;
String s2=null;
System.out.println(s1+s2);//System.out.println(null+null); // nullnull
System.out.println(s1.concat(s2));//NPE

Q>
sir in inheritance i was created object to child class and same variable present in
both classes like (int a ,int b)
with diiferent values and one  add() in parent class. when i invoke that method it
will give sum of
variables present in parent class not in child class sum

class Parent
{
	add(){

	}
	add(int a,int b){

	}
}
class Child extends Parent{
	add(int a,int b){

	}
}

Parent p = new Child();
p.add(10,20);// if overriden then based on runtime object method will be called.
                  // if overloaded then based on compile time reference method will
be called.

Q>
Sir there was interview question like In how many ways we can define loose
coupling?
1. Parent ref =new Child();
2. class Demo{
		public void test(Parent p){
			;;;;
			;;;;
		}
```

```
    }
3. class Demo{
       public Parent test(){
                    ;;;;
                    ;;;;
              }
    }
```

Q>

what is the value  ar[s1.charAt(i)]++;  ????

What are the topics for tomorrow's session?
   Tommo topic: navin sir (SQL) for weekend batch enterprise java batch tommo at 9.00AMIST
                        JDBC(if time permits)

Sir cloneable small demo
Cloneable
   => it is a marker interface through which the objects can be cloned.

```
class Demo implements Cloneable
{
      int i;
      Demo(int i){
            this.i =i;
      }

}

Demo d1=new Demo(10);
Demo d2 =(Demo)d1.clone();
```

Q>
sir is interface which implemnets multiple inheritance?

```
   interface IDemo{ void m1();}
   interface ISample{void m2();}

   class SampleImpl implements IDemo,ISample{
            public void m1(){}
            public void m2(){}
   }
```

Q>
sir interface the word which works as srs and the interface which works as implementation of multipleinheritance is that same?
  Yes both are same..

Q>changes for mutable strings happen in existing object or a new object gets created &
     old reference points to a new object ? as you solved snippet with both concept, need clarity

```
Integer x = 400;//wrapper class immutable
Integer y = x; // y = 400
   x++; //x =401
System.out.println(x==y);//false

StringBuilder sb1 =new StringBuilder("sachin");//mutable
      sb1.append("tendulkar");//sb1,sb2 = sachintendulkar
StringBuilder sb2 =sb1;
System.out.println(sb1==sb2);//true
```

Q>Why can't we override a static method sir?
      static methods are dealing the information at class level,so overriding is not possible rather
      if we try to do it results in "Method hiding".

```
Q> interface A{}
class B implements A{}
      A a=new B();
```

Can we say refrence variable of A is the object of B?
 ans. true

Q>sir y cant we create  class as static but inner class as satic
      Outermost class can't be marked as static becoz "static" refers to usage of class without creating an Object.
      where as inner class can be marked as static becoz "inner class" data can be refered without creating the outer class Object.

Q> Sir this is one question interview question. What does null mean in java.
            null in java represents a keyword where we can use it store the default value for reference variables.
                  eg: String name = null;  Date d =null;

Q> foremost advantage of HAS A realationship. we can aslo build without realtionship aslo the output is same know sir?
            HAS-A => communication and navigation of data is made simple
            without HAS-A relationship the complexity of coding would increase.

Q> sir can you explain again why can't we override variables in interface?
      overriding is not applicable for variables,it is applicable only for methods

 Q> Can you give a brief about dependency injection?
      The process of injecting dependent object into target object is called "Dependancy injection".
            It can be done in 2 ways
                  a. constructor
                  b. setter

eg: Flipkart(object)//target object
         |
      DTDC(object)// dependent object

Q> what is tight coupling and loose coupling
            class Animal{}
            class Monkey extends Animal{}

            Monkey monkey = new Monkey();//tight coupling
            Animal animal     =Demo. getAnimal(); //loose coupling

class Demo{
      public static Animal getAnimal(){
            Animal animal =null
            //logic of creating object
            return animal;
      }
}

Q> what is override  toString methos do??
       toString() -> it will be called automatically when we use object reference.
                     Normally in toString() we override to print the object data.


Q>
public enum Enum1 -> Enum1.class, Enum1$1.class, Enum1$2.class,Enum1$3.class
{
      RED(100)
        {
            public   void displaycolor()

```
		{
			System.out.println("your color name is "+this);
		}
	},

	GREEN(200)
	{
		public  void displaycolor()
		{
			System.out.println("your color name is ");
		}
	},

	BLUE(200)
	{
		public  void displaycolor()
		{
			System.out.println("your color name is ");
		}
	};

	private Enum1(int a)
	{
		System.out.println("I am in enum");
	}

	public  abstract  void displaycolor();

}

Ananomyous inner class
====================
final Parent  p =new Child(){

}
abstract class A{}
abstract class B extends A{}
abstract class C extends B{}
class D extends C{}
```

contact details:
hyder: syedhyder@ineuron.ai
nitin : nitin@ineuron.ai

Todays topic for Java Full Stack(7.30PM IST batch)
a. Wrapper classes continuation
b. var-args
c. Different ways of creating an Object
d. Difference b/w ClassNotFoundException vs NoClassDefFoundError
e. instanceof vs isInstance()
f. packages and import statements usage

Q>
```
public static <T> void sort(java.util.List<T>, java.util.Comparator<? super T>);

ArrayList<Integer> al  = new ArrayList<Integer>();
al.add(10);
al.add(20);
al.add(5);
al.add(0);
     Collections.sort(al) ==sort====> <Integer>
```

Q> sir if we create interface serializable and interface of cloneable you gave example i did'nt understand i also rewatched the video

```
class Sample implements Serializable,Cloneable{

}
```

new SampleImpl(); =>Object can be cloned using clone() becoz of Cloneable interface.
Object can be sent over the network store the state of the Object in a file becoz class implements Serializable.

Q>

```
Byte b1 = new Byte((byte) 22);
Byte b2 = new Byte((byte) -128);
Byte b3 = new Byte((byte) +127);
Byte b4 = new Byte((byte) 'a');//97
Byte b5 = new Byte("20");
Byte b6 = new Byte((byte) '7');


System.out.println(b1);//22
System.out.println(b2);//-128
System.out.println(b3);//127
System.out.println(b4);//97
System.out.println(b5);//20
System.out.println(b6);//55
System.out.println();
```

sir proper defination of class , object and interface ??
class : It is template or blue print for which an Instance should be created.
Object : It is an instance of a class
Interface : It refers to SRS or contract b/w client and service provider or 100%abstract class.

contact details
       hyder: syedhyder@ineuron.ai
       nitin  : nitin@ineuron.ai

Retrieving Date value from the database
=======================================
=> For this we can use either simple Statement or PreparedStatement.
=> The retrieved Date values are Stored in ResultSet in the form of "java.sql.Date"
and we can get this value by using getDate() method.
=> Once we got java.sql.Date object,we can format into our required form by using
   SimpleDateFormat object.

Sequence
========
1. Database
       (java.sql.Date)sqldate = rs.getDate(2);

2. Our required String Form
      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-mm-dd");
       String s = sdf.format(sqldate);

3. String s holds the date.
      System.out.println(s);

sir,can you explain this line Integer i=new Integer(1)+ new integer(2);

```
Integer i1 =new Integer(1);
Integer i2 = new Integer(2);
System.out.println(i1);//1
System.out.println(i2);//2
Integer i3 = i1+i2;
System.out.println(i3);//3
```

where we need to use \n  \t  and  \r   sir
   \n => new line
  \t  =>horizontal tab space
  \r  => carriage return(taking the curson back to the same line at the begining)

Sir you have closed statement object but same way I have trying to close its show
me error? Why Sir?
```
import java.beans.Statement;//wrong import so error
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

            if(statement!=null)
                  statement.close();
```
It's showing me error.

Q>
What is the difference between: ` `, ' ', " "
`` => backticks(use when we specify column names of database)
'' => Single quote(use when we work with character)
"" => String(use when we work with String)

Q>
What is .length in array. its not a method, whats this actually?

```
   length => it is a property/fields of Array class.

class [I@...{
      int length; //property of Array class which holds the size of an Array.
}

Q>
stmt-1;

try{
      stmt-2;
      stmt-3;//exception occured
      stmt-4;
}catch(XXXX e){
      stmt-5;
}finally{
      stmt-6;
}
stmt-7;

stmt-1,2,5,6,7

Q>
Can throws keyword can be applied on methods only? or constructo too?
  Answer: yes possible

Q>
Why have you put the constructor of JDBCutil as private? you said that you don't
want to create object of
that class? why that? and how that way? coz already you have put static in the
two methods: getConnection() and closeConnection()?
Answer: Cretion of an object is a costly event in realtime application,so avoid it.
                To avoid object creation make constructor of a class a private.
            make the class utility class, by writing the methods as static.

Q>
Why people are talk about java concurrency? I have obseving nowadays
 Answer: Becoz the realtime applicatioin depends the usage of "MultiThreading".

Q>
How to run jdbc programs from command prompt?
set path and classpath environmental variables as shown below before running from
command prompt.

D:\Wrapper classes>echo %path%
C:\Program Files\Java\jdk1.8.0_202\bin;

D:\Wrapper classes>echo %classpath%
.;D:\jars\mysql-connector-j-8.0.31.jar;

Eg:
unicode values => fixed
eg: A => 65
       B => 66
       ;;;
       ;;;

      a  => 97
      b  => 98
```

;;;;;
;;;;

WHat is the diffrence between object and direct values?
int a = 10;//primitive type data which is holding direct value
memory for a is given as 4 bytes and directly we can touch the memory using variable name.

Integer a = new Integer(10); // primitive data boxed into Object as "Wrapper class".
memory for a is given in heap as it is object and directly touching the value is not possible, it should be
done with the help of methods like toString(),valueOf(),.....

Q>
sir can we write query like "insert into student()values(?,?)"; I mean without giving column
names when we are using preparedStatement and if we write like this ... is it right way to do ?
i am asking this because yesterday one discussion is going on discord regarding this.

answer: insert into student values(?,?),yes possible.

Q> Why JVM always call the main method having String[] args only not other overloaded methods?
Answer: jvm is designed by SUNMS team, to make a call to method with the following signature only.
eg: public static void main(String[] args){}

Q> isInstance () i have a confusion what does he do?
System.out.println("sachin" instanceof String);//type(String) known from begining so use instanceof
isInstance() will work like instanceof only but the rule is the "type" is not known from begining.

Q> Why can't we make a constructor final,static
construcor => means object creation
static and final => not meant for Objects so we can't write.

Q> In what all condition a toString() method is triggered
```
class Test{}
System.out.println(new Test());//new Test().toString()
Test t =new Test();
System.out.println(t);// t.toString() is called
```

Q> what are max no of argument that can be passed in command line to the array?
Answer. No limit

Q> Sir, the OUT parameters is something I'm still a bit confused.
```
DELIMITER $$

USE `enterprisejavabatch`$$

DROP PROCEDURE IF EXISTS `getStudentsById`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `getStudentsById`(IN id INT,OUT stdName VARCHAR(20),OUT stdAddr VARCHAR(20))
BEGIN
            SELECT sname,saddr INTO stdName,stdAddr
            FROM student
            WHERE sid = id;
END$$

DELIMITER ;

call  getStudentsById(7,@stdName,@stdAddr)
select @stdName as 'name'
select @stdAddr as 'saddr'
```

hello sir, is this any situation when I get a waiting state while asking for a connection object. I mean how many connection objects in the connection pool?

Answer: depends on DB vendor

Sir, Please explain JNI error and how it is caused

```
    class Test{}
              |
         javac Test.java(eg: jdk17v)
            |
        Test.class

       java Test(eg: jdk8V)
            |
           output(JNI error)
```

Q>

```
class A {

  public boolean is(Object o) {
          return (o == this);
  }

}

A someA = new A();
A anotherA = new A();
someA.is(someA); // returns true
someA.is(anotherA); // why returns false
```

becoz comparison is on different object, not on same object

Q>

this confusion

```
============
public class MyThisTest {
  private int a;//42

  public MyThisTest() {
    this(42); // calls the other constructor
  }

  public MyThisTest(int a) {
    this.a = a; // assigns the value of the parameter a to the field of the same
name
  }

  public void frobnicate() {
    int a = 1; ;;local variable
    System.out.println(this); //prints MyThisTest a=42 why?
  }

  public String toString() {
    return "MyThisTest a=" + a; // refers to the field a
  }
}
```

Q>

```
interface a{
      void add(int a,String name);
}
```

```
public class LamdaEx1 {
   public static void main(String[] args) {
          a t=(a,name)->{
            String na=name;
            int aa=a;
            System.out.println("hi");
          };
       t.add(10,"sourav");
      }
}
```

now code will work.

```
MyRunnable r = new MyRunnable();
Thread t1 =new Thread();
Thread t2 =new Thread(r);

t1.start() //new thread will be created, and thread class run() will be called
t2.start()// new thread will be created and MyRunnable run() will be called
```

```
ArrayList l = new  ArrayList();
l.add(10);

                                10===> java.lang.Integer
System.out.println(l.get(0).getClass().getName());
the  output is Integer class.

sir explain little bit about servlet exception?
     ServletExeception
            |=> Problem would occur when we work with ServletAPI
                  like class is not under public, constructor is not written, or
some problem with B.L



class Test
{
      static int a = m1();
      static {
            System.out.println("Inside static block");
      }

      static int m1() {
            System.out.println("from m1");
            return 20;
      }
      public static void main(String[] args)
      {
      System.out.println("Value of a : "+a);
      System.out.println("from main");
      }
}

Few days back i asked you, what is the difference between import package and
inheritance.
You said you will discuss it later. Can you explain it now?
      => import
                  getting up the service of some other class to our current class,
where the service provider
                  class in not present in current folder.

      => inheritance
                getting up the service of another class to our class to reduce the
code.

class Person{
      public String name;
      public char gender
      public int age;
}

import java.io.*;
class Student extends Person{
      public void display(){
                  FileOutputStream fos = new FileOutputStream("sample.txt");
      }
}
```

Is there any way to convert .class file to .java?
  yes possible using DeCompiler will be shown using  Lombok API

Also, when will you teach us about cmd prompt and OS?

      OS====> for users it is GUI
      Command promt-> it acts like a os for developers to run the application from command prompt.

```
Sample s =new Test();
   s.m1();
   s.m2();
   s.m3();
```

Exposing the set of services but hiding the method implementation is called "abstraction".