JAVA

Date __/__/__

→ Java is platform independent (platform → It is a software. (OS))

→ A program that runs on different OS then it is Platform independent

→ Java is everywhere.

→ Principle of Java : Write Once, Run Everywhere WORA

→ Java Library is collec^{n} of predefined classes.

→ Java SE (Core Java)

→ Java EE (Advance Java)

→ Java ME (Micro Edition for mobiles)

Java Packages

Related Classes kept in one group is called Package

Features of Java

→ OOPs

→ Distributed [run more than one machine]

→ Interpreted [Interpretor used]

→ Robust [Fault tolerance

→ Secure [Can do money transaction related]

No pointers, so can't point to invalid ~~thing~~ memory loca^{n}

In Java data is Encrypted

→ Portable [Run on any OS]

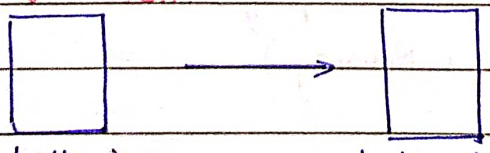
→ Multi threaded [Run Parallel]

→ Garbage Collector [memory released automatically]

Run by JVM, But it is not in C++

Compile & Run

Compile



→ In Java we have only one type of Compiler & it is same for Windows, Linux & other OS. But in C++ we have diff compilers for Windows, ~~Os~~ Linux That's why Java is called platform Independent

Date __/__/__

Java converts code to [Bytecode] [file.class]

→ This Bytecode cannot be understood by any operating System.

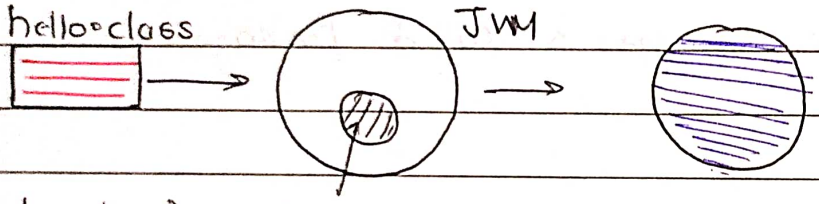
How to Run ?

JVM → It has (Just in Time Compiler) Interpeter.

→ Interpreter reads line by line & converts into format that OS understands

→ Compiler converts whole code into machine language & stores in a seperate file.

→ Interpreter does not store in a file it reads line by line & translates code ~~to~~ that is understand by OS



(Just in time) Interpretor

→ [Interpretor] ~~gir~~ reads one line translates & giv to OS

→ While OS understands, Interpretor is Idle

→ So this leads to Slow execution

So to resolve this we have {Just in time compiler} Interpretor stores in Queue & when OS asks for next line then that translated code is given, So performance ↑.

→ We have different JVMs for windows, Mac, & Linux OS

JDK Java development kit [developing]

JRE Java Runtime Environment [Running]

→ Javac.exe → Compiler
Java.exe → Application Launcher
} These two files are made in C++

JRE → contains JVM & Java Package Classes

→ JVM is sofware is platform Independent

Date __/__/__

→ Case Sensitive means a ≠ A is different

→ Not possible to make a function which is not a member of any class (as we do in C++), means functions should be member of a class then we able to use
→ either write public or don't write anything

```
public class Helloworld
{
    public static void main (String [] args)
    { System.out.println (" Hello " ); }
}
```

// no need to add ; like in C++

→ 4 Access specifier.

→ outerclass cannot be private or protected

→ In this code String & System are predefined classes & Helloworld is user defined

→ (String [] args) is used ~~for~~ ~~to~~ when we pass something thru command line

→ when we use [.] after a Class like "System." then we are accessing [static] members

→ [out] → is [static refrence] variable, it is refers to an object It is like a pointer in C++ (Byte code)

→ javac HelloWorld.java → [Compiles] code & save in [HelloWorld.class]

→ java HelloWorld → To [Run] program.

Data types & Keywords

→ Defines a set of values & set of opera" performed

→ strongly typed → means we can't do char a = 1;

char → 16 Bits Byte → 8 bits int → 32 bits float → 32 bits double → 64

→ String is not a Primitive Datatype, it is a class.

→ long val = 35L; → write L for long

Widening Conversion one datatype will upgrade to other automati

```
int y = 3;
float x = y; // no Error   Int → float ✓
int z = x // Error         float → Int ✗
```

Date / /

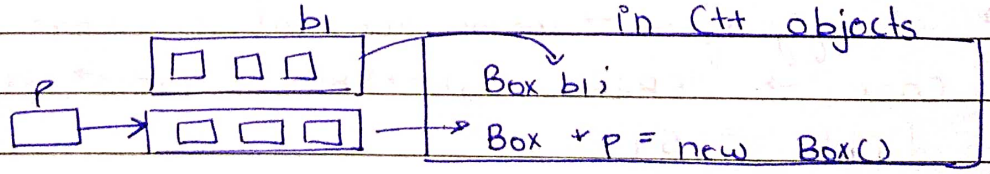
```
float x = 3.4f
int y = x ; // narrowing Conversion , Error , Data loss after .
int y = (int) x // no Error
float R = 3.5 // Narrowing Conv. Error
float R = 3.5f // No error
```

→ char to int, long, float, double
→ int to long, float, double
→ long to float or double
→ float to double

Classes & Objects in Java

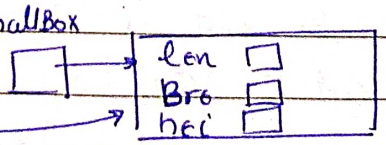
→ Class → user defined Data type. it has properties & methods
Object → real world entity, it consumes memory to hold properties.

```
class Box {
private int len, bre, hei ; } these are properties
public void setDimension (int l, int b, int h) ← this is a method
{ len = l; bre = b ; hei = h ; }
        void
public ^ showDimension () ← this is a method
  { System.out.println (" L = " + len ) ;
                       " ( " B = " + bre ) ;
                       " ( " H = " + hei ) ;
  }
}
```



in C++ objects

```
class Example
{ public static void main ()
  { Box smallBox = new Box();
    // this is refrence Variable.
    smallBox.setDimension ( 12, 10, 5 );
}
```



Date __/__/__

```
public class Example
{ int x; // instance variable
  static int y; // static member variable
  public void fun1() { } // instance member function
  public static void fun2() { } // static member function
// Keep in mind that we cannot make static int a; inside
   functions
// We can have static Inner class
  static class Test { // Public static String variable = " ";
// public static void main (String [] args)
  {   Example ex1 = new Example();
      ✓ Example.y = 5;
      ✓ Example.fun2();
  }   ✓ Example.Test.variable = "INDIA"
// Static member functions can access only static members
   & not instance variables
//  public static void fun2() { x=4; } // Can't Do Like this
                                   y=4; ✓
```

* Static Mem fun^{n} can access only static members
* To call Static member Classname.static_mem_fun^{n};
* Something which is common to all objects then we make static variables / functions.

Wrapper Classes

→ Java cannot be said as 100% oop but nearly.
Why? We use primitive Datatypes int, char, float which are not objects so this makes JAVA 99.9% oops.

→ But we can achieve this using wrapper classes
→ There is a wrapper class for every primitive type.

boolean	→ Boolean	functions
int	→ Integer	▷ ValueOf : static Method.
char	→ Character.	

Date / /

Code:

```
public class Example {

    Public static void main (String [] args)
    {
Integer i = Integer.valueOf ("123");   Passed as String only but
Double d1 = Double.valueOf ("3.14");   Value is stored as datatype.
        = Integer.valueOf ("101011", 2); // Decimal Equivalent stored.
    }
```

2) ParseXxx() Important

```
int a = Integer.parseInt ("123");
```

returns Int value

```
double b = Double.parseDouble ("3.14");
```

3) xxxValue()

→ The is instance method of wrapper class

→ returns corresponding primitive type

xxx → Int, Char, float, double.

```
int c = i1.intValue();
```

(i1) is object made above

Command Line arguments

```
public class Echo {
    public static void main (String [] args)
{     System.out.println (args[0]); // will print hellobhai
    }
}
```

> javac Echo.java

> java Echo hellobhai this is command line argument

```
for (int i=0; i<args.length; i++)
    { System.out.println (args[i]); // Print all
```

//vijeta// > java Echo Aditi Molly Arushi Priyanshi

Date __/__/__

Sum of nos thru command line

```
public class Sum
{
    public static void main (String [] args)
    {   int s=0;
        for (int i=0 ; i< args.length ; i++)
            s = s + Integer.parseInt (args [i]);
        System.out.println ("Sum is : " + s);
    }
}
```

> javac Sum.java // Compiles code

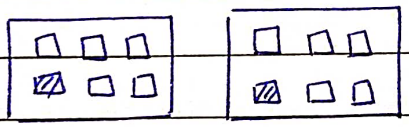
> java Sum 12 34 56 78

output : Sum is (12+34+56+78)

Packages

→ way we organize files into different directories according to their functionality.

→ Ex: Java.io , Java.net are packages.



Package 1 Package 2

→ Advantage :

help us avoid class name collision.

Same package cannot contain two or more same name class

But two diff package can have classes of Same Name

Ease of Maintainance.

→ How to Build Package

```
public class HelloWorld {  // we want this code to be in World Package
    public static void main (String [] args)
    {  }
}
```

→ so add `package world;` at top

```
package world
    Public class Hello ....
```

> Javac `-d .` HelloWorld.java → Compile.

↓ In current folder make a directory.

> java world.HelloWorld → Run

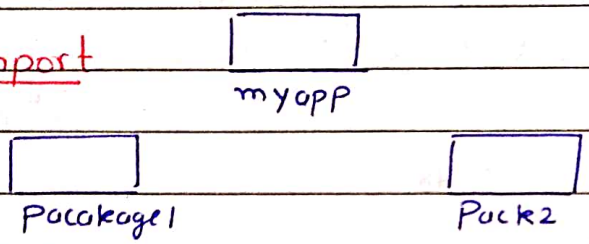
Date / /

→ In 1 Java file only 1 Public file should be there *

→ Name of file should be same as name of Public class

→ If we donot make public class in our file then we can give any name to our file

Use of Import



i) Make Example.java file & Student.java

```
public class Student                    package pack2
{ private int rollno;
  private String name;
  public void setRollno (int r)
    { roll = r; }
  public void setName ( String n)
    { name =n; }
  public int getRollno()
    { return (rollno); }
  public String getName()
    { return (name); }
}
```

```
package pack1                    // pack2.* → import all classes
                                 import pack2.Student;
public class Example
{
    public static void main (String [] args)
    {
        Student s1 = new student ();
        s1.setRollno (100);
        s1.setName ("Saurabh")
    }
}
```

Compilation

> javac -d . Student.java → Student.class

> javac -d . Example.java → Pack1 → Example.class

Date __/__/__

> Java Pack1.Example → Run

Java Access Modifiers

→ Private
→ Protected
→ Public
→ Default

If no access specifier is written then that class belongs to [Default].

→ for Outer Class

write public or don't write. No pvt or protect

```
public class Out {  //outerclass
  public class Inn {}  //inner class
}
```

→ for Inner Class : all 4 are possible.

→ One Java file can contain only 1 ~~file~~ public class

→ Protected : those can be accessed from any class in that [Package] / folder & child Class from other Package

→ Default : Can be accessed with in same package & not by other packages

Constructor in Java

→ Same name as class, no return type, member func^n of class

```
public class Box {
  private int l,b,h;
  public Box()  // Constructor makes object an object. by
  { l=0; b=0; h=0; }  Properly Initializing it
  public static void main(String[]args)
  {  Box b1 = new Box();
  }
```

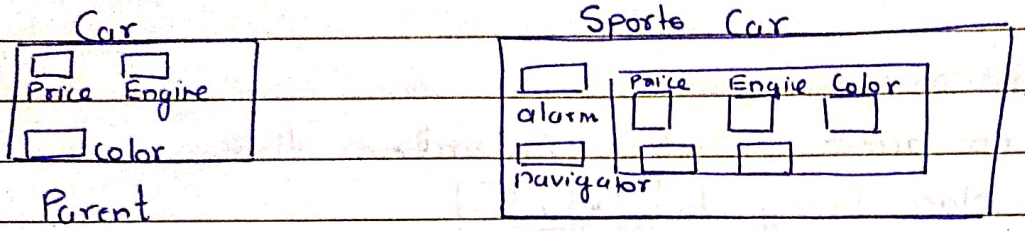
→ If we do not make constructor then compiler makes it.

Date __/__/__

Inheritance in Java

→ is a relationship

→ Make Parent that class that is Generalized



→ class SubClass extends SuperClass

Person.Java

```
public class Person
{ private int age;
  private String name;
  public void setAge(int a)
  { age = a;
  public void setName (String n)
  { name = n; }
  public int getAge () {
      return (age); }
  public String getName()
  { return (name); }
```

Student.java

```
class Student extends Person
{ private int rollno;
  public void setRollno(int r)
  { rollno = r; }
  public int getRollno()
  { return (rollno); }
}
```

On Compiler

> javac *.java # compiles all files
.class file is made

Example.java

```
public class Example {
  public static void main (String [] args)
  { Student s1 = new student();
    s1.setRollno(23);
    s1.setName("Aditi");
    s1.setAge (22);
```

> java Example # to Run main function

→ In Java a Subclass can only have one Parent class unlike C++ where we can have more than 1 Parent class

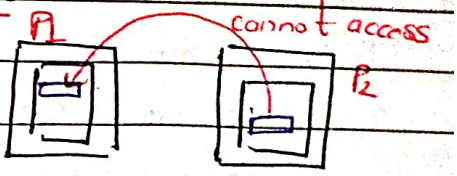
→ No Multiple Inheritance

→ Private mem of Superclass are not accesible by Subclass Object, but the class members can access them.

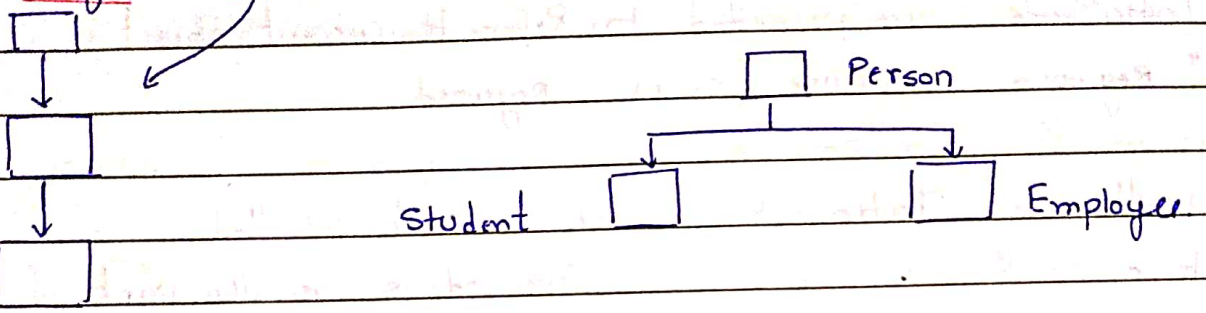
vijeta

→ Members that have Default accessibility in the superclass are not accessible by subclasses in other packages But can be accessed in its own package

Date / /



→ Single Multilevel , Hierarchical Inheritance.



Initialization Block in Java

→ Instance Initialization Block

→ Static Initialization Block

12 07 2002
21 01 2001
33 08 4003

```
public class Test {
  private int x; // Instance Variable
  {                                   // This has no function name, it is a Instance Initialization Block
    System.out.println("Initialization Block : x=" + x);
    x = 5;
  } // This block will run as we make Test object
// we can make more than one Initialization block & compiler converts it into one block
// Constructor me Initialization Block ka code likha hota hai so vo code pehle chalega fir baki jo Constructor me likha.
  public Test() {
    System.out.println("Constructor : x=" + x);
  }
/*
  public static void main(String[] args)
  { Test t1 = new Test(); // Initialization Block x = 0 Constructor x = 5
    Test t2 = new Test(); // Initialization Block x = 0 Constructor x = 5
  }
}
```

→ In Initialization Block there is no return type

Date __/__/__

→ Instance Initializer block is Executed when an Instance of class is created

→ Instance Initializers are permitted to Refer to current object via "this" keyword & to use "super" keyword

→ Whatever written in Initialization block is first outputted then whatever written in constructor. This code is basically part of Constructor only

Static Initialization Block

```
public class Test
{   private static int k;
    static
    {   // This is Static Initializ" Block
        // They can access only Static members of Class
        System.out.println("Static Initializer : k = " + k);
        k = 10;
    }

    public static void main (String [] args)
    {
        new Test(); // Static Initializer : k = 0;
    }                       then k turns 10
```

// Static Block will Execute it code before object Creation

// Static Block will Run only once, even if I make more than one object then also it will execute once only & not like Initializer block that runs whenever new object created

// Static Initializer will Run either we make object or try to access static members like Test.k

→ Static initializer declared in a class is executed when class is initialized, This or Super Keyword cannot be used in Static block, Return also cannot.

// vijeta //

Date __/__/__

Overloading & Overriding in JAVA

→ If two methods of a class (whether both declared in same class, or both inherited by a class or one declared & one inherited with SAME NAME But SIGNATURES are not same (i.e # diff parameters) are called Overloaded

→ Method overloading is a way to implement POLYMORPHISM

→ In C++ if we have two same name functions with diff param & are present in two different classes Parent, Child then it would not be overloading it is called Method Hiding means we could calle childversion but not parent version

→ But in Java it is not like C++, we call it overloading if Child ~~inhe~~ has same name function

```
Class A {
public void f1(int x)
{ System.out.println("ClassA"); }
}

class B extends A
{                    → // This called Method overloading in JAVA, but Method hiding in
  Public void f1(int x, int y)   C++
  { System.out.println("Class B"); }
}


Public class Example1
{ public static void main(String[] args)
  { B obj = new B();
    obj.f1(5); // This would work in Java but not in C++ due to
    obj.f1(3.14); Early Binding, gt would give Error in C++
      ↳ // This would work fine in Both Java & C++
  }
}
```

> Javac Example1.java > java Example1

Date __/__/__

→ In overloading according to the concept it is clear which function to execute.

→ Does not depend on return type just same name diff params is overloading

Overriding

→ Same name same signature

```
Class A  // Car
{  public void fn(int x)  // Gear functionality
   { System.out.println("Class A"); }
}

Class B extends A  // SportsCar
{
   Public void fn(int x)  // Gear functionality but advance logic for SportsCar.
   { System.out.println("Class B"); }
}

Public class Example1
{
   Public static void main(String []args)
   {   B obj = new B();
       obj.fn(5);  // B class fn will run
   }
}
```

→ Why Overriding ?

Suppose we have Car class & SportsCar class so sportsCar will inherit properties of Car class.

Now suppose we have GearShift() functionality in Car

But in SportsCar we have some different way to GearShift

So you might be thinking lets make another function like GearshiftsportCar() & use this. BUT if we create Sports car object then it will be able to access GearShift of Class Car & also the GearshiftSportsCar() so this won't be good. So if we want to make some modification to a Service / function in Parent class then override that fun" in Child Rather than creating new function.

Date __/__/__

Final keyword in JAVA

→ final Instance variable
→ final Static variable
→ final local variable
→ final class
→ final methods

```
public class Example
{  private int x ; // Default value is 0
   Private final int y ; // Black variable.
// we can initialize final instone variable by two methods.
1) Private final int y = 5;   3) Constructor Example() { x = 5 ; }
2) Using Instance Initializing block
   { x = 5 ; }
   public static void main (String [] arg)
   {     Example e1 = new Example() ;
   }
}
```

→ Final members can be Initialized only once ; we canot change x

* final static variable.

```
private final static int y ; // blanck untill initialized.
// initialize by using static block
static { y = 4 ; }
```

* Final local Variable

```
public void fun() {
final int k ; // final local variable must be initialized before its use
// we can initialized it only once & cannot be altered
```

* Final Java Class

final class Parent // now we cannot make child class of this

* final Methods : final void somefunc() { }
// Subclass won't be able to Override.

Date / /

this Keyword in Java

→ In C++ this is a local pointer while in Java its a refrence variable.

```
public
Class Example
{ public static void main (String [] args)
    {
      Box b1 = new Box();   // b1 → caller object
      b1.setDim(12, 10, 15);
    }
}
```

```
class Box {
private int l, b, h;
public void setDim( int x, int y, int z)
  { l = x; b = y; h = z; }
// This refers to the object that calls setDim.
Public void setDim( int l, int b, int h)
  { this.l = l;
  // Instance var  local var
    this.b = b;
    this.h = h;
  }
```

this → refering to caller object i.e b1

→ this is used to refer caller object

Suppose:

```
public void SendBox()
  { GiftTaker gf = new GiftTaker();
    gf.acceptGift(this);
  }
```

we can't write b1 here as that object is not accessible so instead we can write (this) which refers b1 object.

→ This cannot be used in static context

Super keyword

Date __/__/__

→ In heritance, subclass object when call an instance member function of subclass only, function contains implicit refrence variables 'this' & 'super' both refering to current object

→ this refrence variable is of subclass type

→ super refrence variable is of super class type

```
class A
{   public void f1() { }
}

class B extends A {
Public void f1()
{  super.f1(); // to call f1 of A we need super
}
}

class Example
{  public static void main (String [] args)
   {   B obj = new B();
       obj.f1(); // It will call f1 of B then due to super keyword
                    f1 of A runs
```

→ If your function overrides one of its superclass' method, we can invoke the superclass version of the method by using super keyword.

→ It avoids name confict between member variables of Super & sub Class

```
class A {
   int z;
   Public void f1() { }
}
```

```
class B extends A {
int z;
Public void f1() { super.f1(); }
public void f2 () {
int z;
z = 2 ; // this
this.z = 3 ; // to calling object
super.z = 4 ; // z of A class
}}
```

Date __/__/__

Static Members in heritance in Java

→ When a class Inherits from Parent then both static & non static members get inherited.

```
class Parent
{ public static void f1()
  { System.out.println("hello"); }
}
class Child extends Parent { }

Public class Example {
  public static void main (String [] args)
  { Child.f1(); // Child Class ~~ob~~ calls Parent's f1()
  }
}
```

Function hiding : If we make some name & signature static member in Child class, then it hides parent's function.

```
class Parent {
  public static void f1(){ System.out.println("hello"); }
}
class Child extends Parent {
  public static void f1()
  { System.out.println("vo man"); }
}
public class Example {
  public static void main (String [] args) // function hiding
  { Child.f1(); }
}
```

Child's f1 will be called

→ Overriding only happens if functions are Non Static.

→ If functions are static then Method Hiding occurs.

Class P
{ public static void f1() } public void f1() { }
or viceversa
Date __/__/__

Class C
{ public x void f1() } Public static void f1() { }

class public class Example
{ public static void main (String [] args)
{ C.f1() ; → Error will occur.
}

=> Compile time Error will occur is static method hides an instance method

=> Static member variables do not Inherit.

Class Parent
{ static int y = 4; }

Class Child {
static { y = 5 ; }
}
= this is not Parent's y as static variables not inherited

Public class Example
{ public static void main (String [] args)
{ System.out.println ("y = " + Child.y); // 4 prints
}

→ Static memer variables can be hide not inherited.
→ Child's static won't run.
→ It would run if we make remake static int y ; in Child class then static { y = 5 ; } would hide Parent's y

→ Static member variables are not Inherited
→ if we make two same name signature static functions are in Parent & Child class then Child's function hides parent's function & Function Hiding occurs not Overriding

Constructors In Inheritance
→ Constructors are not Inherited
→ Subclass constructor calls Parent class constructor. using super() ;

Class A { Class B extends A Class Example
int a { int b } public static void main(String[] arg
Public A() Public B() { B obj = new B() ; }
{ } { } } // A , B printed.

vijeta > javac Example.java > java Example

Date __/__/__

```
class A //Parent                    class B extends A //Child
{ int a;                            { int b;
  public A(int x)                     public B()
  { a=x;                              { super();
    System.out.println("A"); }          System.out.println("B"); }
}                                   }

Public class Example {
    B obj = new B(); }                public void static main(String []args) {
}
```

=> When we don't write super(); compiler itself calls Parent class Default constructor automatically.

=> The above code will give error caz super() is calling Default constructor & A class does not have default it has parameterised

=> So we can write super(5); in B class Constructor to call A class parameterised constructor.

Implict Constructors in superclass & Subclass → no need to make Constructor
Implicit Cons in Child/Sub B & explict in Paren/Super A → Need to make constr in B
Implict in Parent/Super & Explict in Child/Sub class → no need to make C in B
Explicit constructor in Parent & Child, then its our choice wether to call Parent's Parameterised Const

Constructor Chaining

→ Constructor can call its own constructor or parents, his choice.
→ It can call its another constructor rather than calling parents

```
class A
{ public A(){ System.out.println("A 1"); } }
class B {
  public B(){ System.out.println("B 1"); } }
public class Example {
    public static void main(String []args) {
        B b1 = new B(); }
//vijeta// }
```

This will print A1 & B1 caz b object calls B class constructor & B class constructor calls super(); & A class constructor is called

Date __/__/__

But see below code.

```
class A
{   public A() { System.out.println("A1"); } }
class B {
  public B()
  {                          // B(4) calls B(int)
    this(4); // Instead of calling super this(4) will call its overloaded
    System.out.println("B 1"); // constructor
  }
  public B(int x)  → first super() will be called.
  { System.out.println("B 2"); }
} // output : A1 B2 B1
```

→ this is local refrence variable it points to current calling objct.

→ Constructors can call other constructors of the same class or superclass

→ Constructor call from a constructor must be first step

```
int a;
this(4); → now this won't work as it should appear first
```

→ Such series of invocation of constructors is Constructor Chaining.

```
> javac Example.java
> java Example
```

→ First line of constructor is either super() or this()

→ Constructor never contains super() & this() both.

Date __/__/__

Abstract Class

→ In C++ we make abstract class by making one or more pure virtual functions (Do nothing functions)

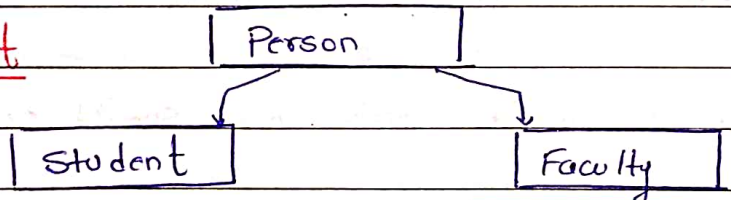
→ In Java we have "Abstract" Keyword

→ Abstract is that ~~call~~ cannot be instantiated, cannot make object of that class

```
abstract class Person
{ private String name;
  private int age;
  public void setName(String n) { name = n; }
  public void setAge(int a) { age = a; }
}

class Example {
    public static void main(String[] args)
    { X Person p = new Person(); } // can't be instantiated
}
```

Why abstract



→ To inherit common properties, Generalization

→ In an institute there is either Student & Faculty, why would we make Person object its useless so we make person class abstract class, & Student & Faculty inherits from Person.

→ Making Person object would be wrong.

→ Java Abstract class are used to declare common characteristics of subclasses

→ It can be used as Superclass for other classes that extend abstract class

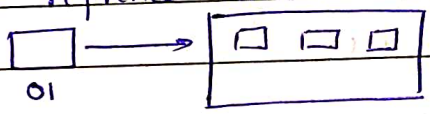
Date __/__/__

→ Abstract class contains attributes & methods like any other class

→ Cannot create object of Abstract class but can make refrence variable

```
abstract class A
{


}
class B extends A { }
public class Example
{ public static void main (String [] args)
    { A o1 = new B();
    } // o1 is refrence variable of A type but points to B object
}
```



Abstract Methods

→ Abstract method is similar to pure virtual function in C++ i.e it has no implementation

→ Abstract Method declaration must end with semicolon rather than block.

```
public abstract void fun();
```

→ Abstract class can include abstract methods

→ If a class has any abstract methods, whether declared or inherited the entire class must be declared abstract

```
class Person { abstract void show(); }
class Student extends Person { }
Class Example { public static void main (String [] args)
                { Student s = new Student(); } X
```

above code gives Error caz if a class has abstract method then that class should be declared with "abstract"

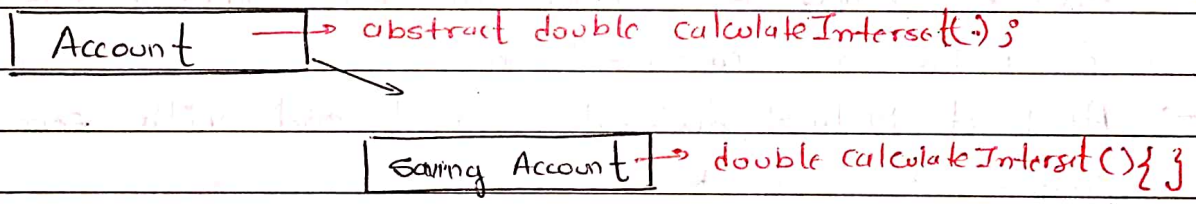
Date / /

→ also as Student is inheriting Person so student must also be made abstract as abstract void show(); will be inherited by student.

→ now as Student is made abstract so we cannot be make student object

```
abstract class Person {
    abstract void show();
}
class Student extends Person {
    void show() { // some code }
} // if we override function then there won't be any problem.
class Example {
public static void main (String [] args)
 {  Student s = new Student();
    S. show();
 } // Everything works perfectly
}
```

Example



→ Override of functions must be done then only object of Saving Account will be made.

Interface in Java

```
Interface Name { }
```

→ Interfaces just specify the method declaration (Implicitly Public & abstract) & can only contain fields (which are implicitly public static final)

```
Interface SomeName {
    int x;
    void someFunction(); }
```

Public static final.

this is public & abstract

Date / /

→ There will be no implementation of functions inside interfaces

→ In interface by default all members are public, we write not write does not matter if function is abstract & if variable is static (means it will get memory only one time)

Code

```
Interface I1 { void someFunctions(); }
class A implements I1 {
    public void someFunctions() { // code } // overriden.
}
```

→ Subclass [extends] Superclass

→ Subclass [implements] Interface

→ Interface cannot be instanticated

→ Interface do not have Constructors

→ Abstract class can have constructors (for accessing objects variable)

→ As Interface does not have instance variables then no need of constructor. - it has static member variables

→ If a class that implements an interface does not define all the methods of the interface, then it must be declared 'abstract' & method definitions should be provided by the subclass that extends abstract class

```
interface A { void fun1();
              void fun2();   } All are public abstract
              void fun3();
            }

~~class~~ [abstract] class B extends { ~~void~~ public void fun1() { }
All methods are not overriden ←       public void fun2() { }
So declare class B as abstract }
class C extends B { public void fun3() { } }
```

Date __/__/__

Why Interface ?

→ It tells functions Prototype

Ex Interface Admission
{ int registration();
int batchAllotment();
int iCardGeneration();
}

Now student Class can use to implement these methods & can provide logic whatever he requires.
Similarly Employee class can implement these according to their own way

→ In Interface Multiple Inheritance is possible : but in classes there is no Multiple Inheritance possible

interface I_1 { } ✓
interface I_2 { } ✓
Interface I_3 extends I_1, I_2 { } ✓

→ Class extends Class
→ Interface extends Interface
→ Class Implements Interface

Class A implements I_3 { ✓
// A should implement all methods of I_1, I_2, I_3
}

Class B extends A implements I_3, I_4 { } ✓

→ You can not create object of any interface but creation of object refrence is possible

→ Object refrence of Interface can refer to any its subtype class.

Date __/__/__

```
interface I1 {
    void f1()
}
interface I2
{
    void f2();
}
class A implements I1, I2
{
    public void f1()
    { }
    public void f2()
    { }
}
class Example {
    public static void main(String [] args)
    {   A obj = new A();
        obj.f1();
        obj.f2();
        obj.f3();
    }
}
```

Object refrence of interface can refer to any its subclass type.

```
I1 obj1 = new A();
```

refrence variable

```
obj.f1();  ✓
obj.f2();  X   } Error
obj.f3();  X
```

Difference Between Abstract & Interface

→ Abstract class can have any access modifiers for members

→ Interface can have only public members.

```
abstract class Person
{ private String name;
  public void setName(String n)
    { nam = n; }
  public String getName()
    { return (name); }
}
```

```
interface Calculation
{ double pi = 3.14;
  int add(int a, int b);
  int sub(int a, int b);
}
```

- all public
- static variable
- abstract methods

} Implicitly

Date __/__/__

→ Abstract class may or may not contain abstract method. ~~Interface can not have define~~

→ Interface can not have defined method.

```
Abstract class Person
{
  private String name;
  public void setName(String n)
  { name = n; }
  public String getName()
  { return(name); }
}
```

```
interface Calculation
{ double pi = 3.14
  int add(int a, int b);
  int sub(int a, int b);
}
```

→ Abstract class can have static or non-static members

→ Interface can have only static member variables

→ Abstract class can have final or no final members

→ Interface can have only final member variables. (initialized only once)

Taking Input in JAVA

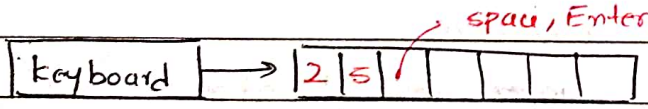
→ Scanner class

→ Scanner is a final class means no child can be made.

→ Scanner class is a part of `java.util` package

→ we can read java input from System.in using Scanner class

→ System.in → refrence varrable.



this is an object

~~new Scanner~~ SC = ~~object~~ new Scanner(System.in);

Scanner class has function nextInt() → take input from buffer & convert into integer

x = sc.nextInt() → take data from buffer until we ~~can~~ get space new line or Enter.

→ Scanner → take input from Buffer.

// vijeta //

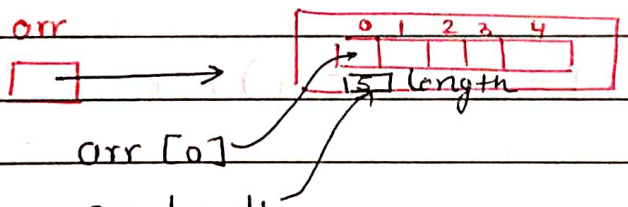
Date __/__/__

- nextInt() → converts Data from buffer to Integer
- nextLine() → a string with spaces
- nextDouble() → converts to Double.
- next() → Take one word string, no space

```
Import java.util.*;
Class Example {
public static void main (String [] args)
{  System.out.println ("Enter your name & age ");
   Scanner sc = new Scanner (System.in);
   String name = sc.next(); ✓
   int age = sc.nextInt(); ✓
}
}
```

Arrays in Java

→ int arr [] ; → refrence variable that refer Array
int arr [] = new int [5];



→ int arr [] = new int [] { 2,4,6,8 }; // Fine ✓
int arr [] = new int [2] { 2,4,6,8,10 }; // Error X
int arr [] = new int [5] { 2,3,4,5,8 }; // Error : you cannot mention size & values together
int arr [] = {2,4,6,8,10 }; ✓ Fine

→ Array is not Blank (there is no garbage value)
Initialized with 0 for int arr []

```
{ int arr [];
  arr [0] = 25;
  arr [1] = 9;
```
} will give error because arr is refrence variable not an object

```
{
  int arr [];
  arr = new int [3];
  arr [0] = 10;
  arr [1] = 20;
  arr [2] = 30;
  // all good.
}
```

Date __/__/__

User Input in Array

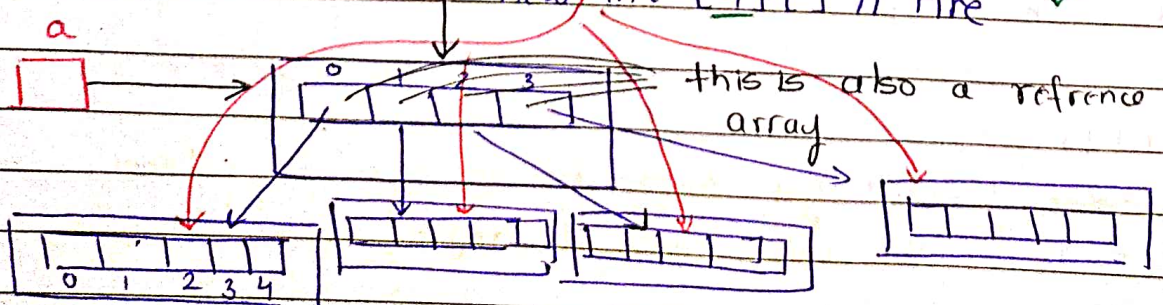
```
import java.util.*;
Class Example {
  public static void main (String [] args)
{
  int arr [] = new int [5];
  Scanner sc = new Scanner ( System.in );
  System.out.println (" Enter 5 no ");
  for (int i=0; i<5 ; i++)
     { arr[i] = sc.nextInt(); }
  for (int i=0; i<5; i++)
  { System.out.println ("arr [" + i + "] = " + arr [i]); }
}
```

2-D Arrays

→ int arr [][]; or int [][] arr; this is not array it is reference variable

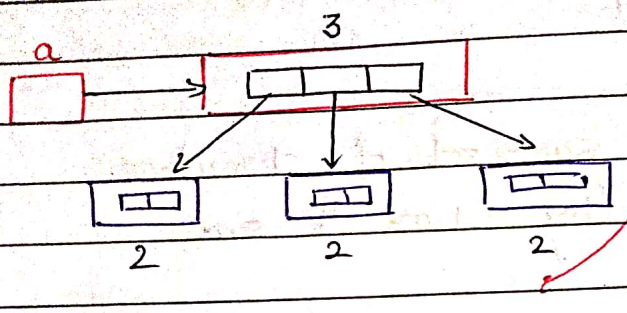
→ int [][] a = new int [4][5]; ✓

But int [][] a = new int [][] // Error X
= new int [][5] // Error X
= new int [4][] // Fine ✓



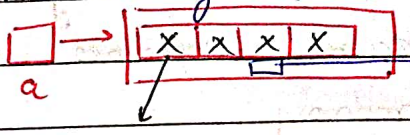
Date

```
int [][] a = new int [][] = { {3,4}, {5,6}, {7,8,9} };
```




```
int [][] a = new int [4][];
System.out.println (a[0][0]);
```

This will give error.



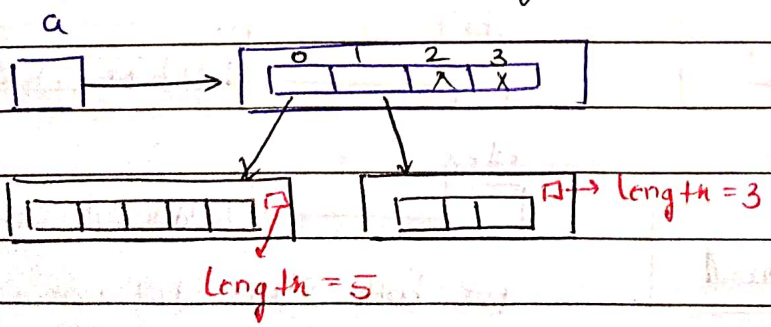
But if we print a[0]
then null will be printed
println(a.length) → 4 printed on screen.
println(a[0].length) → Error
a[0] is not containing anything
it is p null

These refrence are not pointing to any arr so Bydefalt they are null so we are not able to access a[0][0]

Distinct length feature.

→ In Java it is possible to declare 2D array with different length of each array

```
public class Example {
  public static void main ( String [] args )
{   int a[][] = new int[4][];
      a[0] = new int[5];
      a[1] = new int[3];
     System.out.println (" length: " +a[1].length);
}
```



String Class

→ java.lang.String class is final class means its child cannot be made.

Date __/__/__

String Class

→ Immutable Classes, cannot be changed

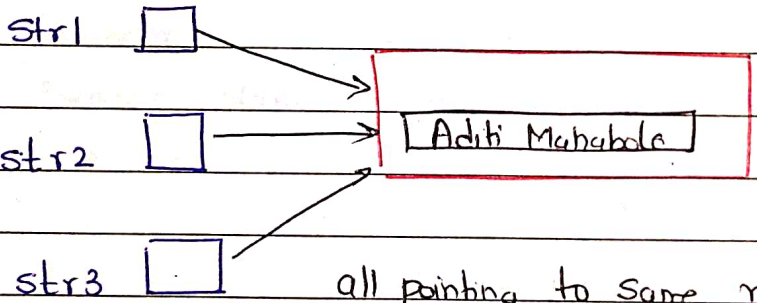
→ String str1 = "My name";

→ If two or more string have same set of characters in the same sequence then they share the same refrence memory

∴ Ex
```
String str1 = "Aditi Mahabole";
String str2 = "Aditi Mahabole";
String str3 = "Aditi" + "Mahabole";
```

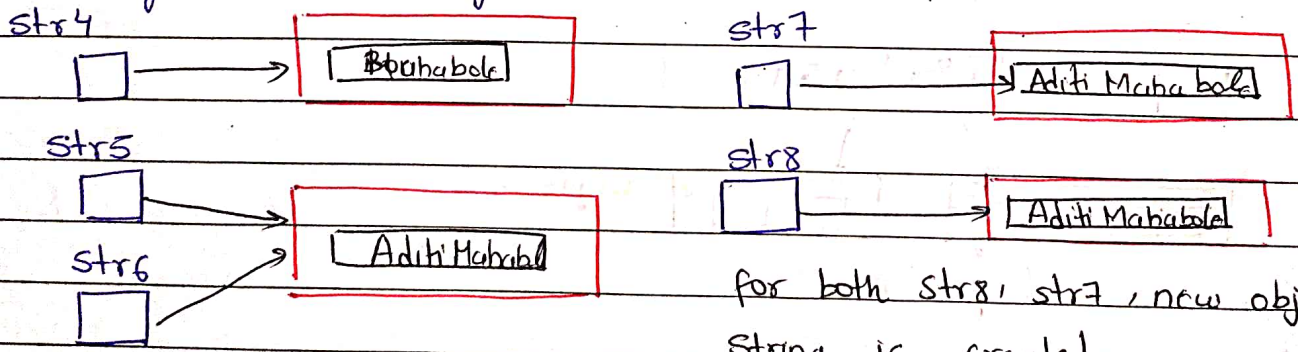
→ str1, str2, str3 denote same String object

→ new object is not made if value is exactly same in variables



all pointing to same memory block

Ex
```
String str4 = "Mahabole";
String str5 = "Aditi" + str4;
String str6 = "My name is Bond";"Aditi Mahabole";
```



for both str8, str7, new object String is created

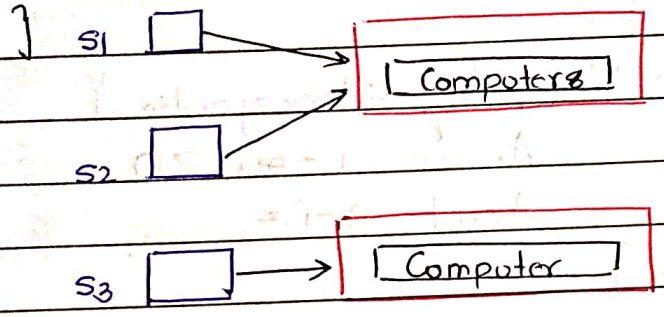
→ String str7 = new String("Aditi Mahabole"); // SYNTAX

→ if we write new then even if character are same then also new object will be created

String str8 = new String("Aditi Mahabole");

Date __/__/__

```
class StringExample {
public static void main (String [] args)
{  String s1 = "Computer";
   String s2 = "Computer");
   String s3 = new String ("Computer");
System.out.println ("Result 1 : " + (s1==s2)); // true
                   ("Result 2 : " + s1.equals(s2)); // true
                   ("Result 3 : " + (s1==s3)); // false
                   ("Result 4 : " + (s1.equals(s3)); //true
}
```



→ s1==s2 checks if they pointing to same memory location (same object)

→ s1.equals(s2) checks is Data same or not thats why for s1.equals(s3) it gave a true value

String Methods

→ toUpperCase() , toLowerCase()

```
String s1 = new String ("Hello")
String s2 = s1.toUpperCase();
```
→ It returns a new string & does not change s1

→ IndexOf (int ch)
indexOf (int ch , int fromIndex)
indexOf (String str)
IndexOf (String str , int fromIndex)
lastIndexOf (int ch)
lastIndexOf (int ch , int fromIndex)
lastIndexOf (String str)
lastIndexOf (String str , int fromIndex)

```
String s = new String ("computer");
int i =s.indexOf('m')
int j = s.indexOf('m',3);
```
3 Ke baad m nhi hai

```
int i = s.indexOf ("ute");  // 4 printed
      = s.indexOf ("ute",4); // 4 printed
```
Lastindex of → searches from right to left

Date / /

// equals() → returns boolean value
→ equalsIgnoreCase (String anotherString)

String s1 = new String ("Computer")
s2 = new String ("Computer")
if (s1.equals(s2)) // false. {}
If (s1.equalsIgnoreCase(s2)) // true. {}

→ compareTo(String s) → matches unicode (ASCII code)
int i = s1.compareTo(s2)
if (i==0) { // Strings are same }
else if (i>0) { // Strings not same } // app to dictionary order }
Computer 77-109 Amar i-ma>0
computer Amit 0=i=
else if (i<0) { // in dictonary order }

→ s1.substring(4); // uter printed
(2,4) // 2 included, 4 not included.

Exception Handling

→ any abnormal unexpected events or extraordinary conditions that may occur at runtime.

→ x/y y=0 this is Exceptional situation.
→ Runtime error message occur due to Exceptions
4 options
Defalt throw ↓ Default catch
Default throw ↓ our catch
Our throw ↓ Default catch
Our throw ↓ our catch

Date / /

→ Koi bhi local variable agar initialized na kiya to vo blanck bata

→ Class instance variables are initialized automatically, but not local members

```
Class Example
{
  public static void main (String [] args)
  {
    String s1; // this is not object & is not initialized as it is blanck
    System.out.println ("First Line");
    System.out.println ("String length: " + s1.length()); // this will give
    System.out.println ("Last Line");                      Compile time error
  }
}
```

→ this is a local variable

if String s1 = null then nullpointerexception raised

→ Throwable Class provides a String variable that can be set by the subclasses to provide a detail message that provides more information of the exception occured

→ All classes of Throwables define a one parameter constructor that takes a string as the detail message.

→ The class Throwable provides getMessage() function to retrieve an exception

Two types of exception in JAVA

→ Checked (Compile time Exceptions) → Unchecked (Run time Exception)

ArrayIndexOutOfBound }
NullPointerException

are subclasses java.lang.RuntimeExcep which is a subclass of the Exception class.

Date / /

24
60
84

```
Class Example {
public static void main (String [] args)
{
    try {
        System.out.println(810);
        System.out.println(" In try ");
    }
    catch ( ArithmeticException e )
{ System.out.println ( e.getMessage ()); }
System. out. println (" hello ");
}
}
```

Explicit throw in JAVA

Syntax throw < throwable Instance >

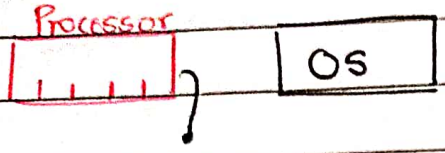
```
Class Example {
  public static void main (String [] args)
{     int balance = 5000;
      int withdrawalAmount = 3000;
Try { if ( balance < withdrawAmount )
        throw new ArithmaticException (" Insufficient Balance ")
    balance = balane - withdrawalAmount ;
    System.out.println (" Transaction Successfully ");
  }
}
```

```
catch (ArithmaticException e)
{   System.out.println ( e.getMessage ()); }
```

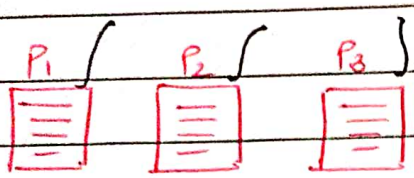
Date __/__/__

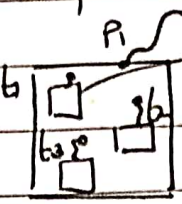
Threading in Java

→ thread is an independent path of execution within a program.



→ Establish connection btw process & processor.





we want all to run parallely
t1, t2, t3 are threads

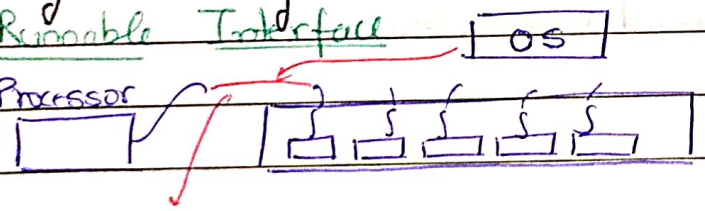
→ Multithreading:
refers to 2 or more tasks executing concurrently within a single program.

→ Every thread in Java controlled by java.lang.Thread class

Two ways to create thread in Java

→ Implement Runnable Interface (java.lang.Runnable)
→ By Extending the Thread class (java.lang.Thread)

1) Runnable Interface



this connection is done by OS

```
Thread t1 = new Thread()
Thread t2 = new Thread()
```

Runnable is predefined Interface

```
class A
{   funcA()
    {  }
}
```

```
{  }
i1 = new A();
```

if i1 is refrence variable of Runnable Interface then we can say A class object in i1

→ A class that implements Runnable interface then its object can be passed in new Thread(✓) here i.e thread's constructor as Thread's constructor will had refrence variable of Runnable Interface

```
Thread(Runnable r1) {r1.run()}
```

Date __/__/__

```
Class A implements Runnable {
    run() ← implementing Runnable Interface's run function
    {       // overriding run function

    }
}
```

Code

```
class A implments Runnable
{   public void run()
    {  int i
    }
}   // we are implementing run of Runnable Intertace in class A & B

class B implements Runnable
{   public void run() {}
}

public class Example {
    public static void main (String [] args)
{       Thread t = new Thread (new A());
        Thread t2 = new Thread (new B());
        t.start();
        t.start()
}
```

OS will give some time to thread A the some time to thread B sot output won't be in sequence