## Java

1) OOps Concept
   - Inheritance
   - Polymorphism:
     - overloading
     - overriding
   - Abstraction
     - interface
     - abstract class
2) Excepⁿ Handling
3) Strings
4) Collections.
5) Keywords: static, final, this, super, Access Modifier
6) Java8 streams & lambda exprⁿ

WD meth
Inter
Locators
~~[illegible]~~
exceptio
Testing

## Selenium: → Locators

→ Webdriver methods & Interaction with WebElements
[alert, window, frame, WebTable
→ Waits
→ TestNG
→ Exceptions Insᵐ & Handling
→ Advancesᵐ: actions, GRID, JavaScriptExecutor

## RealTimeScenarios

## FrameWork:

Java directly dzn't support multiple Inheritance
indirectly it supports in form of interface.

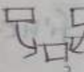
can we overload/override main () method?
overload → yes
override → NO

OOP has 3 pricipals
language ^

1) Encap$^n$: To put one within other
By class, PKg, subpkg.
→ wrapping up data & methods into a single unit (class)

2) Inheritance: process by which one class aquires prop. of other class
→ support Reuseability of code

Multiple Inh: Java supports M.I. in form of interface.



3) Polymorphism: many forms
overloading & overriding

TON$^r$ Diff$^t$

meth sign$^n$ should be same. ↑ same m1(x,y)

All ~~A~~ exactly

TON, RT, AM

overloading → Static Poly → decided at compile time

AM$^r$ & RT Anything

overriding → Dynamic Poly → decided at run time. which method to exec

Order means order of type ✓
Not order of variable ✗

④ Abstractions: process of representing essential features without including Background detail or expl$^n$.
→ hiding the complexity or implement$^n$ details.

↳ Achieved by access specifier, objects.
interface → 100% abstract$^n$
abstract class → partial abstraction (0-100%) ↓ depending on how many abstract methods are there in abst.

class → public & default only
interface → inside or o/s pkg

| methods | var | constr |
→ All 4 → All 4 → All 4

public → less Restrictive (stronger Access)
protected → Inside PKG & also outside PKG but only in child classes.
default → only inside PKG.
private → More restrictive (weaker Access) → Inside class only.

↑ increasing visibility

PPDP

Can we override static methods: → NO, but
11 → we can declare static method with same signature in subclass but it will not behave as overridden method.
12 We can not override static method as they are part of class not object. your code will look like you
1 are overriding static methods but o/p will be diff^t than the expected.
2 Can we overload static methods → Yes.
↳ so main can be overloaded
3 but not overridden.

→ subclass method overrides superclass method.
4 → The version of method that is executed will be
5 determined by the object that is used to invoke the method. if parent object invokes parent method version execute
6 if child → child version of method exeutes.

↳ subclass version of method is used to provide a specific implement^n of a method that is already provided by one of its parent.

✓ → private & static method can not be overridden but same name method can be used in child class.

→ Any decimal value in java is considered double. So to make it float use "f"

→ used to apply some restrictions on var, method or class.

final var → To declare a constant. a var.   final float pi = 3.14f;
we cannot change value of var once assigned.

final method: To prevent overriding.

final class: To prevent inheritance

→ this & super cannot be used in static context.

this → To access the instance var when same name
Keyword.  parameter has been used
or local var.

super(a, b);
default or parameterised

{ $S^r$ class default const$^r$ is automatically called by subclass def. const$^r$ by JVM

1 super : To call $s^r$ class default or parameterised const$^r$ inside subclass const$^r$.
2 To access a method/var of sup$^r$ class inside subclass
which are hidden/overriden.

e.g.
used in user-defined excep$^n$

```
void show()
{   super.show();
}
```

{ super()
super.meth()
super.var

4 → super must be 1st statement inside child const$^r$ of subclass.

Diff b/w final & static
→ meth, var, class
→ meth, var, Block, inside block, method

→ final can be used for local variables but static can't be.

| this | super |
|---|---|
| Its a predefined instance var which holds current obj ref. | points to immediate sup$^r$ cl obj of curr. obj. |
| ① → refers to curr. cl. obj | ① refers to $S^r$. cl. obj. |
| ② In same way this can be used to access all non-static meth/var. this.x, this.show(); | ② In subclass, super.a, super.display(). |
| ③ this() can be used to call oth$^r$ const$^r$ in same class. A() { this(); → must be 1st stat$^t$ } this(a,b) | ③ In subclass, we can call $S^r$ cl. const$^r$ A() { super(); ⇒ added by JVM by default if not added explicitly. super(a,b) } |

```
interface A                    interface B
  {  void sum();                 {  void sum();
  }                              }

class C implements A,B
  {
  public void sum()
      {
        sop("Hi");
      }
  }
```

public → can not be default as in interface by default its public

→ There will be no compil^n error it will work fine

```
→ interface A1 {                  abstract class A2
        void sum();                  {
      }                                 void sum();
                                     }
  interface B1 {                  abstract class B2
        void divide();               {
      }                                 void divide();
                                     }
```

So what is need of interface if same thing can be achieved by abstract class?

Ans. coz interface supports multiple inheritance

```
class xyz implements A1, B1      but   ~~class xyz extends
      {                                    A2, B2
      }                                  {~~
```

if we want to enforce class can hv only public meth → public, protected, default in ab. cl.

only public static/final var → can be anything in ab. cl

then only interface is the option

There cannot be final or private meth (ab. meth) in ab. cl. or interface.

abstract → when some specifier or by default at some places, we can not hv other type there like private, protected etc.

## Abstract class

→ contains abstract & concrete methods both.
but interface contains only abstract method.

→ Can't be instantiated
→ dzn't support MI.

Final class (Vs) abstract Class

| If class | | then method |
|---|---|---|
| ① final | → | indirectly final |
| not necessary final | ← | final |
| ② abstract | → | no necessary abstract |
| must be declared abstract | ← | abstract |

→ we can define an ab. cl without any ab. meth. so what is use of such class. Here you do not allow it to be instantiated on its own.
(directly can't be instantiated)

→ subclass of an abstract class usually provides implementations for all of the abstract methods in its parent class.

However, if it doesn't, then subclass must also be declared abstract. { valid modif for ab. (meth) → public, protected, default
{ Invalid → static, final, private

When to use:
→ you want to share code among several closely related classes.
→ if you want to declare non-static or non-final fields.

→ Interface has only static & final variables.
→ abstract class can provide the implementation of interface (partially or fully)
→ ab. class has usual flavour of class member (variable) like private / protected
→ ab. class has constr even if we can't instantiate an abstract class, constructor is used by child classes.

# Interface.

→ By default public & abstract

No need to override these 2 meth in class which ~~[illegible]~~

→ By default methods are public

since Java 8, it can hv concrete meth as well (default & static methods) keyword

→ Interface defines only abstract methods & public static final variables.

→ supports Multiple Inheritance

→ can't be instantiated, cannot hv constr.

~~[illegible]~~ By default → { public final static } constants

→ A class should implement all the methods of interface.

→ An interface can extend another interface

```
interface B extends A
      {   int a = 10;  ⇒ by default p.s.f
      }   int a;  // compil^n error
class xyz implements B
      {
      } → class has to define code
              for all the methods of B & A

Class xyz implements M, N { }
```

When to use:

1) if you want to take advantage of multiple inheritance.

2) if you want to specify the behaviour of a particular datatype, but not concerned who implements its behaviour

Real life example: Maharaja Burger, McDonalds
→ all franchise need to follow the contract.
interface is kind of contract. → optional feature/flavour can be provided by default methods

→ Abstract class can hv final/non-final, static/non-static variables

→ ~~Ab~~ Interface can't provide the implement^n of abstract class

→ interface methods & variables are public by default array size may be defined while declaration of array.

```
int a[3];            int a[];
    a[0] = 1;        a = new int[3]
    a[1] = 2;
    a[2] = 3;
```

```
(1) int a[] = {1, 6, 7, 5};

    int a[][] = { {2,4,7}, {4,3,2}, {3,4}, {7,8,6} };

                        -> primitive
    int a[][] = new int[4][3];
                         ↓   ↓
                        row  col

All same   int a[]; int ^[]a;   (space)
           int[] ^a;            (space)
```

```
{
   {2,4,7},
   {4,3,2},
   {3,4},
   {7,8,6}
};
```

```
(2) int a[] = new int[5];
       a[0] = 10;
       a[1] = 20;
         :
       a[4] = 50;
```

→ `int a[] = new int[-5];` → Prog. will compile but at run time error will be shown Negative size.

```
(3) String [] st = new String[3];
          st[0] = "ABC";
          st[1] = "xyz";
          st[2] = "PQR";
      for (int i = 0; i < st.length; i++)
              sop(st[i]);
```

## Interface e.g.

```
interface A {
        void a();  // by default, public & abstract
        void b();
        void c();
        void d();
    }

abstract class B implements A {
    public void c() {
                    sop("I am c");
                }

class M extends B
{   p.v. a(){
              sop("I am a");
            }
    p.v. b(){
              sop("I am b");
            }
    p.v. d()
            { sop("I am d")
}
```

Create a class that calls the methods of interface A

```
class Test{
    psvm(s...)
    {   A a = new M();
        a.a();
        a.b();
        a.c();
        a.d();
    }}
```

O/P:
```
I am a
I am b
I am c
I am d
```

```
interface WebDriver{
        manage();
        findElements();
        findElement();
        get();
        close();
        quit(); }
        ------
class ChromeDriver implements WDr
        { == }
    }
```

```
                  → interface
WebDriver dr = new FFD();

dr.manage()
  .findElements()
  .close()
  .quit()
```

⇓

All these are interface methods implemented by FF ✓ Chrome ✓ IE ✓ class

Method Overriding

Access Specifier Rules in Method Overriding:

1) We can not reduce the visibility of inherited method from Parent (inside Child).

```
Class Parent
{
    protected void display()
        {  sop("Hello1");
        }
}
Class Child extends Parent
{
        void display() => compil^n error
        {
            sop("Hello2").
        }
}
```

→ can be same protected or more visible public

→ can not have default or private

| Parent disp() | child disp() |
|---|---|
| if public | only public, can not be protected, default or private |
| if protected | can be public, protected but not default, private |
| if default | can be Public, protected, default but not private |
| if private | → It can be anything (out of 4) {even static} |

since private method can not be overriden

⇒ All the methods of Interface must be overridden with public only.

→ The methods of an Interface must be public, if not mentioned takes by default as public.

→ So subclass which implements the interface should override with the same specifier or more (public) but not less.

Exception → Parent
↓
IOException
↓
FileNotFoundException

→ ~~if a variables~~ This rule is applicable for methods only not for variables, coz its a rule for method overriding, there is no concept of variable overriding in java
if a variable is say protected in Parent then it can be declared anything out of 4 in Child even private.

→ we can not have:

```
public disp()
  {
     Private or Protected int a = 6; X
                  int a=6; ✓
           final int a = 6; ✓
  }
```

Parent meth — Child meth

Checked → No exc^n
Checked → Same or less
Checked → Any Unchecked exc^n.

Uncheck → Any Unchecked
Uncheck → No exc^n
Uncheck → But can not throw Any checked

NoEx^n ↳ same Rule as for Uncheck.

## Exception Rules in Method Overriding:

→ Subclass overridden method can not throw more checked Exc^n ~~Exception than that~~ of superclass method.

→ subclass overridden method may not throw any exc^n, even if S^r class meth throws checked or Unch. exc^n. (No compil^n error)

```
class Parent
  {
     public void display() throws IOException { }
  }
class Child
  {
     public void display() throws Exception
       {
       }
  }
```

superclass of IOEx. ⇒ compile time Error

It can throw unchecked ex^n as well.
It can throw either same (IO Excep^n) or its subclass FileNotFoundException or no need to throw at all.

→ Above rule doesn't apply to Unchecked excep^n. It will compile can be anything, any where.

→

```
char a[] = {'H', 'E', 'Y'}
            String.valueOf(a);
  String s = a.toString();

  String st = new String(s);
        sop(st);      HEY
char ch[] = st.toCharArray()
```

---

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

<u>String to Integer</u> ↗

Integer to string ↘

```
     String s = Integer.toString(123);

                 ar.toString();
```
↙ array of integers.

```
int i = 10;
   String s = String.valueOf(i);
```

```
interface A
{
    void sum(); ——→ Applicable modifiers:
                    public, default (public),
                    static, abstract
                              ↓
                         unnecessary
                         coz
                         method is
                         already abstract
}
        |
        ↓
    abstract void sum();
```

```
public class Vehicle
{
        void horn() { sop("genericharn");
                    }                              void test()
}                                                  {
                                                     sop("test")
                                                   }
public class Car extends Vehicle
{
     void horn()
       {
         sop("car horn");
       }
     void  horn(int a)
        {
          sop ("car horn: " +a);
        }
   psvm (S---a)
   { //case1
       Vehical v1 = new Vehicale();
          v1 horn() → generic horm
          v1.horn(30); → compil^n error
  //case2
      Vehicle v2 = new Car();   v2.test(); ⇒ test
      v2.horn(); → car horn  → if both meth were static ⇒ "generic horn"
                                     then Parent horn() meth
                                     is called as runtime polymorphism
                                     dzn't happen for static methods.
      v2.horn(30); → compil^n error
  //case3
      Car v3 = new Car()          V3.test(); ⇒ test
       v3. horn(). → cam horn
       v3. horn(30); → car horm: 30
//case4 Car v4 = new Parent(); → compil^n error
```

→ method is called as per class which is storing the reference of instance created except in case of overridden method

Wrapper Class: A class whose object wraps primitive datatypes.

→ Used to convert Prim. data types into objects

Data Str like
→ collections (ArrayList, Set etc) store only objects, not the primitive types

| Primitive Data Types | Wrapper Class |
|---|---|
| char | Character |
| int | Integer |
| float | Float |
| double | Double |
| boolean | Boolean |
| byte | Byte |
| short | Short |

$$\begin{tikzcd} \text{Primitive Data Type} \arrow[r, bend left, "\text{Autoboxing (Automatic Conversion) e.g. int to Integer}"] & \text{Wrapper Class} \arrow[l, bend left, "\text{Unboxing e.g Integer to int}"] \end{tikzcd}$$

```
{ char ch = 'a';
{ Character a = ch; // Autoboxing
{ Arraylist <Integer> al = new AL<I>();
            al.add (25) { int a = 10;
                        { Integer x = new Integer(a);

{ Character ch = 'a';
{       char a = ch;   // Unboxing
{   int num = al.get(0);
```

Array: a.length
String: s.length()
Arraylist: al.size()

Can we create Private constructor?

Yes,

if we declare constructor as private we cannot create obj of the class outside class. if tried → compiler error

Where to use:

→ singleton Design Pattern (create only one obj, use it multiple times)

(final class) → It won't allow ~~a~~ class to be subclassed → if tried to use extends → compiler error

→ if all methods are static in class, we can use private const

↳ If no constr is created in a class explicitly (not even default Cr) then compiler provides default constr. → Cr which has no para

s.substring(begin index)

s.substring(b, e)

String s = "Hello"

s.substring(s.length()) (if beginIndex = length of string) → returns emptystring (by design)

s.substring(3, 3) (beginIndex = endIndex) → "

s.substring(-1) ⇒ IndexOutofBoundEx

s.substring(2, 1) → "

Static Keyword in Java → mainly used for memory management. (It saves memory) or It makes our program memory efficient

01 THURSDAY DECEMBER

336-030 • WK 49

⇒ static can be used with

$\boxed{\text{variable, method, block, nested class}}$

- variable ↓ also known as Class variable
- method ↓ " class method
- block ↓ used to initialize static data member. ↓ It is executed before the main method at the time of class loading
- nested class ↓ only nested classes can be static.

1) Static Variable → basically Global variable. → makes prog. mem. eff.

memory is assigned only once to static variable (at the time of class loading)

↳ can be used to refer the common property of all objects/instances (that is not unique for each obj)

e.g. company name of employee (say 500 emp. record)

College name of students

{ String empId, String empName, static String company }

↳ a single copy of var. is created & shared among all objects/instances of class.

↳ we can create static var. at class level only.

To count No. of obj created:

```
classTest {
    static int count=0; // will get mem. only once & retain its value.
    Test () { count++; }
    psvm(S...) {
        Test t1 = new Test();     o/p ⇒ 2
        Test t2 = new Test();
        sop("Total obj created are:" + count);
    }
```

→ if removed, o/p = ① as each time mem. is assigned & value is reset

At any point of time all instances of cl will the updated value if its updated by a same inst.

2016

can be overloaded
but not overridden
belongs to class, not to obj of class.

## 2) Static Method

→ can be invoked without creating an object of class

→ can access only static data $m^r$ (var.) (saves mem. here)
& can change the value of it.

↳ cannot call non-st meth. directly (obj. req)

C.Name.meth()
C.Nam.var

→ this & super cannot be used in static (meth.) context

## 4) Nested Class :

→ class inside class

→ can have non-static or static methods.

↳ A static class cannot access non-static $m^{rs}$ of the Outer Class. It can access only static $m^{rs}$ of $O^r C$.

↳ Nested static class dzn't need ref. of $O^r C$. but nonstatic nested class (Inner class) requires $O^r$. C. reference.

static class obj.

```
O^rClass.NestedStaticClass obj = new O^rC.NSC();
                                 obj.printMsg();
```

```
O^rClass outer = new O^rC();
O^rClass.InnerClass inner = new O^rC.IC();
inner.display();
```

obj of non-st. class

→ non-static method (Instance/obj meth) can call static meth & access static var.

→ abstract method cannot be static in java,
Interface can not have static method in it as all meth are implicitly abstract.

Ab. cl can hv static meth & var.

ClassName.methodName();
It will try to call a meth which is not defined/abstract, so no use of such call.

→ const$^r$ cannot be static in java as c$^r$ is invoked by obj. & static belongs to class not obj.

Alt+F4 To close open things on window

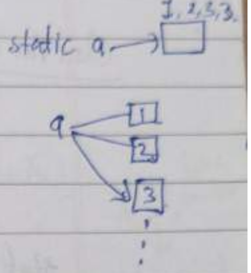
CLASSMATE
Date ____
Page ____

To run the program $\boxed{\text{ctrl + F11}}$ in Eclipse

To put company Name as static as memory are assigned only once to var. if its static.

```
static int a = 1
for(i=0; i<10; i++)
{
    sop(a)
    a++;
}
```

static a → 1, 2, 3,



```
static String college = "BBD";
int rollno;
String name;
```

= ⇒ assignment opr

== ⇒ equality opr

Condnal opr :

if((a==1) && (b==2))

&& → AND

|| → OR

Ternary opr

result = some condn ? value1 : value2 ;

if condn true → value1

if cond false → value2

```
int a=10, b=20;
int result = (a<b) ? a : b;
sop (result); ⇒ 10
```

Variables:

local : inside method

Instance , Inside class but outside method

Class Var or Global → static variables (Inside class, but o...

access ↓ by ↓

ClassName.VarName

→ No need of obj. which require memory

→ Memory assigned only once

static method/variable are used for memory mangement.

→ static methods in java belongs to the class (not an instance of it)

→ static method can not refer to this or super in anyway.

↳ refers to curr. obj

↳ can not be used in static Context.

```
public class A
{
    static int a = 10;
    int b = 20;
    p.s.v m(String args[])
    {
        sop (a); //10
        sop(b); //Error
              ↓
        static method cannot
        use non-static
        vars.
        A ob = new A();
        sop(ob.b); //20
    }
```

→ we can't use "static" for local variables, only final is permitted

```
void sum()
{
    static int i = 1;  X
    - - -
}
```

"Local variables can't be static."

# Getting UserInput : Scanner

```
int i;
float j;
String s;

Scanner in = new Scanner (System.in);
   sop("Enter a string");
     s = in.next();                    in.nextLine()
   sop(" Entered String is :"+s);
        "
     i = in.nextInt();
        "

        "
     f = in.nextFloat()
        "
```

for a line of sentence ↓ in.nextLine()

tiple stances (objects)

Looping — for, Do-while, while

Init^n, cond^n, incre^t

```
for (int i=1; i<12; i++)
  {
     sop(" ");
  }
```

t, cond^n, Incr^t

```
int i=0;
while (i<12)
  {
     sop("Hello");
     i++;
  }
```

Init^n, Incre^t, cond^n

```
int i=0;
do
  {
     sop(" ");
     i++;
  }
while (i<12);
```

Diff b/w while & do-while → do-while st. is executed atleast once even when cond^n is false

$$\begin{tikzcd}
 & & \text{majorly Used} \\
\text{Array} \arrow[r] \arrow[d] & \text{Coll"} \arrow[ur] \arrow[r] & \text{No need to give size, Used majorly Nowadays} \\
\text{Array} \arrow[d] & \text{int a[10]} \arrow[r, "\text{Need to give size}"] & \text{Not used measurely} \\
\text{Obsolete} & &
\end{tikzcd}$$

```
int ar[];
  ar = new int[3]
    ar[0] = 12
    ar[1] = 15
    ar[2] = 11
  for (int i=0; i<3; i++)
      sop(ar[i]);
```

**Enhance for loop:**

Added from Java 5 onwards

↓

```
int num[] = {1, 3, 5, 9, 6}.
for(int i: num)          → can be used
                           in coll"
{
   sop(" Numbers are: " + i);
}
```

**2D Array.**

```
String names [] [] = {
              {"abc", "adc", "xyz"},
              {"mno", "asd"}
              };
for (int i = 0; i<2; i++)
     for (int j=0; j<3; j++)
          sop (names [i] [j]);
```

i++; first value is used then incremented
++i; first value is incremented then used

```
int i = 10;
    sop(i++);   //10
    sop(++i);   //11
```

$\Rightarrow$ (String [] args)  [ ] $\xrightarrow[\text{replaced with}]{\text{can be}}$ ... b/w string & args

→ (String ... [ ])

(String args[ ]) ←

(String args...) X

8th June, 2016

Polymorphism → overloading (static Polymorphism)
→ Overriding (Dynamic Overriding)

P1
↓
P2
↓
P3
↓
C

Parent p = new child ( );

At run time, time its decided which class to refer. → dynami

→ classes can not be private, bcoz it won't be of any use

→ static & final method → can not be overridden
→ can be over loaded

→ So main(S...a) can be overloaded but not overriden

Method ~~without~~ body
~~Contains~~

only extended
↑
Abstraction → Abstract class (Abstract + Concrete)
Abstraction → interface (only abstract methods)

↓ hiding the complexity or implementⁿ details.

interface ↓ {implemented + extended

↳ 100% Absⁿ is achieved using Interface.

→ hv to create/implement all abstract methods by Class (child).

↓ hv to create & implement all methods ~~by class~~

↳ contains only constants (final fields)

→ Java indirectly supports multiple inheritance by interface (not by class)

class A implements B, C

Indirectly no need to write final

→ we can not create instance of interface or abstract class they are inherited / implemented then used.

Encapsulⁿ → data hiding

Encapsulⁿ → Private

Source → Generate G^r & S^r method

```
public class GettersSetters
    {
        private int empid;
        private int empage;
        private String empname;

     public void setEmpid(int empid)
        {
           this.empid = empid;
        }
     public int getEmpid()
        {
           return empid;
        }
        —— — — —
    }
```

→ Interface fields are public, static & final by default, and methods are public & abstract
implicitly

```
public class A
    {
        ps vm(S... A)
        {
            GettersSetters gs = new GettersSetters();
                gs. set Age(19);
                gs. set Empid(420);
                gs. set Empname("Amit");

                Sop(gs.getAge());
                sop(gs.getEmpid());
                SOP(gs.getEmpname());
        }
    }
```

---

Excepⁿ → can be handled [ Checked (Compile time)
                                            Unchecked (Runtime)

Error
↓
can not be handled
→ DB down
→ memory is less

{ ~~Arithmetic Excepⁿ~~
  NullPointer Exⁿ
  Array IndexOutofBound Eⁿ
  String " "
  Number Format excepⁿ

---

1)
```
int a[] = new int[10];

try{
    a[11] = 9;
}
Catch (ArrayIndexOBoundE)
{}
```

2)
```
String s = "abcxyz";
try {
        sop(s.charAt(50));
        sop ("code just After error in try block");   ← will not be printed
}
Catch(StringIndexOutofBoundsExⁿ)
        {}
sop ("rest of codes..");   ← will be printed
```

classmate
Date
Page

But its not a good practice

→ we can have two ^or more classes in a single file
→ there can be only one public class in a file

It will compile easily but while running we have to select one class (file) at a time

→ JDK (JVM, JRE) is platform dept.

→ args: receives any command line arguments present when the prog. is executed

Pollymorphism: one task is performed by diff^t ways.
→ Abstract^n:- Hiding internal details & showing functionality
e.g. Phone call (we don't know the internal processing)
→ Encapsul^n: Binding code & data together in ~~to~~ a single unit.
e.g → Capsule (wrapped with medicines)
→ Class
→ Java Beans

→ Static Keyword in Java mainly used for Memory Management
→ can be used for: variable, method, block, nested class
✓ → Static method can not access non-static data or call non-static method directly → only via object of class can be accessed/called.
→ local var inside st meth need not to be static.
✓ → Static method can be called inside non-static method via classname.methname
→ 5 Key word in Java Excep^n handling: try, catch, finally, throw, throws

3) try { Int c = 10/0; } ⇒ ArithmeticExcep^n: / by 0
4) try { int num = Integer.parseInt ("XYZ"). }
⇓ NumberFormatExcep^n
5) try { String str = null;
sop( str.length()); } → NullPointerExcep^n

works as interface b/w two or more classes

```
public interface showable
    {
        void show();
    }
```

```
interface A
{
    m1();
    m2();
    m3();
}
```

→ 100 classes are Implementing A so if m4() is added in A then we know where to make changes.

→ Mainly 3 reasons to use:

1. used to achieve full abstraction
2. ______________ multiple inheritance
3. It can be used to achieve loose coupling

→ dependency

Tight coupling: when a gp of classes are highly dependent on one another

↳ usually bad because it reduces flexibility & code re-usability.

it make changes more difficult.

impedes testability

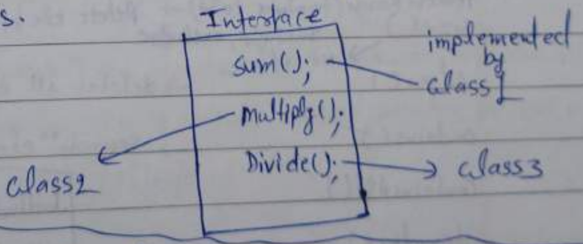
Loose Coupling: reducing dependencies of a class that uses a diff<sup></sup> class directly.

→ more flexible & maintainable, testable

↳ achieved by a design that promotes single-responsibility & separat<sup></sup> of concerns.

↳ A loosely coupled class can be tested independedly

↳ classes can communicate through interfaces rather than other concrete classes.



```
m1()
int x = 5;
  sop(x);
}
```

```
public abstract class DemoAb
    {
        public void disp1()
            {
                sop("I am DKV");
            }
        abstract public void disp2();
    }
```