refers to a language that use objects in programming, they use object as primary source to implement what happen in code. Object are seen by viewer or user.
It aims to implement real-world entities like inheritance, hiding, etc. 22 06 24
Main aim of oop is to bind together the data & func. that operate on them so no other part of code can access data except that function.

OOPs

general notes:-

A class is a named group of properties and functions, if combined into a single entity, it can be done via classes. "starts with a capital letter".

A class is a template of an object and an object is an instance of a class.

- class → logical construct
- object → physical reality // occupies space in memory.

Constructor is a special function, that runs when you create an object and it allocates some variables.

Note:- You cannot access non-static stuffs without referencing their instance in a static ~~constr~~ context.

Hence, ~~static x = 1~~ ~~object~~ [static does not depends on object]

In different packages with protected you will only able to access it in subclasses.

You cannot create object of an abstract class. ally can't create abstract constructors. ally cant → abstract static methods if any method is abstract the class must be abstract.

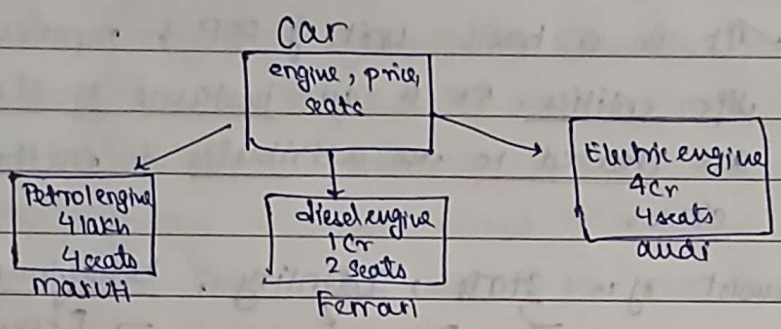
Date 24/06/24
Page:

* Chapter 1 Actual notes:-
* Classes and Objects

A class is a template of an object, and an object is an instance of class.

Class creates new data type that can be used to create objects.

When you declare an object of a class, you are creating an instance of that class. Thus a class is a logical construct and object has a physical reality.
↓
i.e. it occupies space in memory



car = class
maruti, Ferrari, audi = objects

Properties of a class :-

- not real world identity, it's just a template from which objects are created
- not occupy memory
- group of variables of different data types + group of methods
- It contains :- 1. Data member (int class; Node left;) 2. method 3. Constructor 4. Nested class 5. Interface

Class Declaration in Java :-

[public, private, protected, default]

```
access_modifier class class_name {
    data member;
    method;
    Constructor;
    nested class;
    interface;
}
```

e.g.

```
class Student {
    int rno;
    String name;
    float marks;
    psvm {
        Student s1 = new Student();
        sout(s1.rno);
        sout(s1.name);
        sout(s1.marks);
    }
}
```

Creating object → Object

s1.marks → used to use attributes (instance variables)

- **Objects:-** It is a basic unit of OOP & represent real-life entities. It is an instance of class that are created to use attributes & methods of class.

It consist of:- State, Identity, Behaviour

- State: [value from its data type]
- Identity: [distinguishes one object from another]
- Behaviour: [effect of data type operations]

e.g. of object Dog.

Identity	State / attribute	Behavior
" name of Dog	" Breed, Age, color	" Bark, sleep, eat

When we create an object (non-primitive data type), it always allocate on heap memory.

Declaring an object = creating an object of class, the class is ~~called~~ said to be instantiated.

State must be unique for each object, while instances share attributes & behaviour of class (it may be more).

e.g. Dog tuffy;

If we declare a ref. variable → tuffy, it's value will be null until an object is actualy created & assigned to it.

Simply declaring a reference variable does not create object.

- Initializing Java object:-

The dot operator → links the name of object with name of instance variable, although it is referred as dot operator, the formal specification for Java categorizes the . as a separator.

The 'new' keyword dynamically allocates (i.e. allocating at run time) memory for an object & returns a reference to it. This reference is the address in memory of object allocated by new. This reference is then stored in variable. Hence, in Java all class objects are dynamically allocated.

Box mybox; → declare reference to object

mybox = new Box(); → allocate a Box object

The first line declares mybox as a reference to an object of ~~the~~ type Box. At this point, mybox does not yet refer to an actual object.

The next line allocates an object & assign reference to it to mybox.

After second line executes, you can use mybox as if it were a Box object. But in reality, my box simply holds, in essence, the memory address of the actual Box object.

The key to Java safety is that you cannot manipulate references as you can through pointers (not in JAVA). Thus you cannot cause an object reference to a point to an arbitrary memory location.

* not used for int or char, etc. bcoz primitive types are not implemented as objects

A closer look at new :-

```
Classname class_var = new classname();
```

- Classname: name of class that's being instantiated
- class_var: it is a variable of class type being created
- classname(): followed by parenthesis (()) specifies the constructor for class.

A constructor defines what occurs when an object of a class is created.

It is important to understand that new allocates memory during run time.

```
Box b1 = new Box();
Box b2 = b1;
```

b1 & b2 both refer to same object. The assignment of b1 to b2 did not allocate any memory or copy any part of original object. It simply makes b2 refer to same object as does b1 ~~b1 b2~~. Thus any change made to the object through b2 will affect ~~b1~~ the object referred by b1. Hence, when you assign one object reference var. to another obj. ref. variable, you are not creating a copy of object, you are only making a copy of reference.

```
int square (int i) {
    return i * i;
}
```

A parameter is a variable defined by method that receives a value when method is called.

here i = parameter. An argument is a value that is passed to a method when it is invoked.

eg. square (100) passes 100 as an arg. Inside square(), the parameter i receives that value.

Note:- `Bus bus = new Bus();`

- bus: looked by compiler
- 'object' → looked by JVM.

Constructors and "this" keyword :-

Sometimes a method will need to refer to the object that invoked it. To allow this Java defines the "this" keyword. This can be used inside any method to refer to the current object. i.e. this is always a reference to object on which method was invoked.

Final keyword :-

→ A field can be declared as final. Doing so prevents its contents from being modified, making it essentially a constant. This means you must initialize a final field when it is declared.

→ common coding convention to choose all uppercase identifiers for final fields i.e. final int FILE-OPEN = 2;

→ Unfortunately final guarantees immutability when instance variables are primitive types, not reference types.
i.e. If an instance variable of a reference type has final modifier then value of ~~object itself can change~~ instance variable (reference to object) will never change, it will always refer to same object but value of object can change.

The finalize() method :-

→ Sometimes an object will need to perform some action when it is destroyed. To handle such situations, Java provides a mechanism called Finalisation. By using this, you can define specific action that occur when object is just about to be reclaimed by garbage collector.

→ To add finaliser to class, simply define finalize() method. The Java run time calls the method whenever it is about to recycle an obj.

of that class.

→ protected void finalize() {
    // finalization code here
  }

- Constructors: Once defined, the constr. is automatically called when object is created before the new operator completes.
- These are strange coz, they have no return type, not even void.
- This is bcoz, implicit return type of class' constructor is the class itself. type itself.

In the line: Box mybox1 = new Box();
is calling the Box() constructor

- Inheritance & constructors in Java:-

In Java, constructor of base class with no argument gets automatically called in derived class constructor.

e.g.

filename → main.java

```
class Base {
    Base() {
        Sout("Base class constructor called");
    }
}
class Derived extends Base {
    Derived() {
        Sout("Derived class constructor called");
    }
}
public class Main {
    psvm {
        Derived d = new Derived();
    }
}
```

Any class will have a default constructor, does not matter if we declare it in the class or not. If we inherit a class, then derived class must call its super class constructor.
It is done by default in derived class.
If it does not have default constr. in derived class, the JVM will invoke its default constructor & call super class constr. by default.
If we have a parameterized constructor in derived class still it calls the default super class constr. by default.
In this case if super class does not have a default constr., instead it has parameterized constr. then derived class constr. should call the " super class constr.

Date 25 | 06 | 24

OOPs → (Indian Programmer)

(Practical approach)

* Lecture 1

nothing mentioned then 'default' access modifier

```
class Student {            // name of class
    string name;    } Datamembers/attributes of class (access after making objects)
    int rno;

    public void printdata() {
        sout(name);
        sout(rno);
    }

    psvm {
        Student std1 = new Student();
        // [class name] Student, std1 = object name, new = keyword → helps in dynamic (run time) memory allocations, Student() = [constructor]
        std1.name = "Ram";    } [accessing attributes]
        std1.rno = 56;
        std1.printdata();     // [accessing using function]
        std1.sout(std1.name);
        std1.name = "ram";
        sout(std1.name);  → // mutable
```

O/P
Ram
56
Ram
ram

mutable: we say instance variables are primitive variables.
exception: final keyword
if we've use final int rno;
then std1.rno = 56 ✗
std1.rno = 72; ✗
final → immutable

* Lecture 2

- Using static keyword → we can access methods of class without declaring objects.

- Final keyword → ek baar jo value dal gayi wahi hamesha rahegi → immutable
  - define at attributes of class.
  - provides accessibility to keep same name of a variable or fix its name in the entire program.

also we must have to initialize it at data members of class only.

Java Class Methods:-

```
class Dog {
    int License;
    String name;
    public void eat() {
        Sout (name + "eats");
    }
}
```

→ name of method
→ non-parameterized

O/P
Bruno eats.

```
psvm () {
    Dog dog1 = new Dog();
    dog1.name = "Bruno";
    dog1.eat();
}
```

∴ Class has 2 properties for objects i.e. Data members
State, Identity, Behaviour
Behaviour of objects (methods)

* Lecture-3

Constructor → special type of fn./method, which is when whenever we create an object it runs automatically without calling.
→ also called initialiser (to initialize values).

because we already call a constructor while making a new object.

```
Class ABC {
    int id;
    int age;
    ABC (int id, int age) {
        this.id = id;
        this.age = age;
        Sout("Hello World");
    }
public class {
psvm {
    ABC obj = new ABC();
```

Class name should always be equal to constructor name

// shortcut alt+insert → select parameters

5;19

Types of Constructors:-

1. Default 2. Parameterized.
   - Default → [no parameter passed]
   - Parameterized → previous example

just writing ABC obj = new ABC(); ↳ it will pass

Both can be used when needed.

Whenever need to initialize variable then we will use parameterized constr. else we use default constr. depends on our use case.

How to call if both are present?
↳ check constructor types.java file.

Hence, based on no. of arguments, we can make different constructors.

* Lecture-4 (THEORY)

There are 4 pillars of OOPs :-
1. Inheritance 2. Polymorphism 3. Encapsulation
4. Abstraction

Abstraction = providing those only sufficient things necessary for a user. (car ka engine ki kya chal rha hai use koi mtlb nhi, chalana se hai simply).

Hence,

Abstraction is a technique of providing only the essential details to the user by hiding the unnecessary details of entity.

Date : / /
Page :

✓ It provides simple interface to user without asking for complex details to perform an action.

** ✓ It can be achieved via interface & abstract classes.

Advantages:-

✓ helps in reducing operational complexity at user end.

✓ Inheritance = ✓ technique of acquiring properties of another class having feature in common. ✓ (using extends)

✓ Also known as child-parent relationship, where child inherits the properties of parent.

✓ also called is-A relationship (∵ child is-a type of parent).

✓ Advantages:-

✓ increases reusability & decreases duplication.

✓ i.e. agar ek type e.g. 'price' vehicle me already declared hai to car me phir se declare krne ki koi jaruraat nhi we can directly access in car(child) as car.price = 500000.

✓ Encapsulation = binding of data items in a single class.

✓ It is a technique of restricting a user from directly modifying the data members or variables of a class in order to maintain the security / integrity of data.

✓ It can be achieved via ✓ access-modifiers (public, private, protected & default).

↓ Advantage

✓ Polymorphism = poly (multiple) + morphism (forms) = ✓ multiple forms

✓ It allows you to perform an action in multiple or diff. ways

- There are two types of polymorphism :-

1. Compile Time = Method Overloading = Static polymorphism
2. Run time = " Overriding = Dynamic "

Advantages :-

- programmers code can be reused via polymorphism.

* Lecture - 5

- Inheritance in Java

Mechanism of building a new class on existing functionality of [super class.]

- Use cases :- method overriding
  - reuse code & reduce duplication.

Some common terms :-

Sub class = child class

Super class = Parent class

Inherits = to inherit / take things

extends = (syntax)

* 'Extends' keyword → used to access parent class from child class

```
class childClass extends parentClass {
    // child data members
}
```

∴ using child class objects can access both classes.

* Lecture - 6

- Types of Inheritance :-

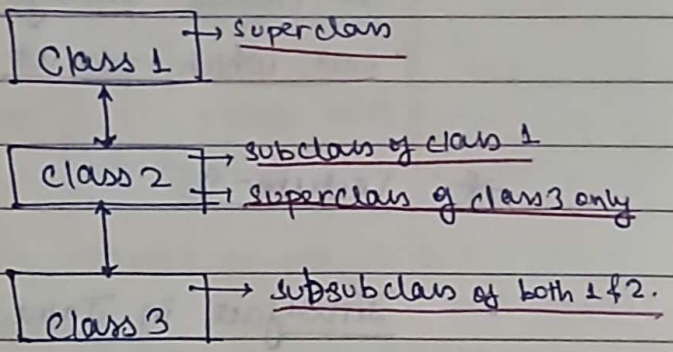
Single Inheritance = simplest type

Just 1 parent class & 1 child class.

previous example of vehicle & car in Intellij is an example of Single Inheritance.

* Lecture - 7

Multilevel Inheritance



- eg. GrandParent
  ↓
  Father
  ↓
  You

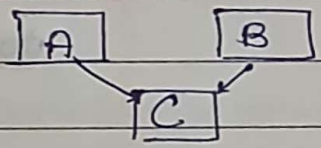
- refer code.

Note :- class of A cannot access class B & C & by B can't access from class C.

* Lecture - 8

class cannot access multiple classes

Multiple Inheritance



→ Not possible in Java.

⇒ Since, ek child ko 2 parent nhi ho skte.

e.g.
```
class A {
    void show () {
        sout (" class A method ");
    }
}
class B {
    void show() {
        sout (" B method ");
    }
}
class C extends A, B {
    void run () { ...
    }
}
psvm {
    C obj = new C ();
    obj.show ();   // ambiguity "which 'show' to call"
                   if diff. name then only also we can't do it.
```

is called "Diamond Problem in Java"

This problem is solved by Interface.

↓

helps in achieving multiple inheritance.

Q. Why Multiple Inheritance is not directly possible in Java?
→ It creates ambiguity & the compiler gets confused to call which parent.

* Lecture-9

- **Interface in Java**

= It is a set of abstract methods you would want your class to implement. → (mtlb if a class is implementing an interface then it can access those methods (abstract) from interface) these methods are public and abstract by default. ∴ no explicit use of abstract keyword.

and any class implementing your interface will need to provide implementation of those methods.
↓
mtlb

in interface we only declare method, we do not put body in it so class which will implement this interface will write body on own.

Interfaces are use to achieve abstraction & implement multiple inheritance.

Car ant engine → no use how it works

e.g.
```
interface PizzaRecipe {
    public void makeDough();
    "    "  proof    ();
    "    "  prepareTopping();
    "    "  shapeDough();
    "    "  bakePizza();
}
```

shapeDough(){ }

not required bcoz it's there by default in interface.

1. default access modifier of interface is public (not default)

Jab bhi 2 class → 1 interface ☆ implement karega then overriding must hoga, since both class will provide body to that name of method of interface only.

Date : / /
Page :

- How to declare & use an interface in Java?

⇒ 'Implements' keyword

eg. interface A {
    public void method1();
}

→ class B implements A {
    public void method1() {
        sout("xyz");
    }
}
(provider → Body)

can't define body in interface.

class C implements A {
    @Overriding
    public void method1() {
        sout("abc");
    }
}
"method overriding"

Note :- Object kabhi bhi Interface ☆ nhi banta hai sirf class ☆ banta hai.

here, A obj = new C(); ✓ ∵ C implements A
(name of class, constructors, interface)

A obj = new A(); ✗

Interview Pov

some points to remember for interface :-

- It is used when we want to achieve 100% abstraction, whereas abstract classes can be used to achieve anything b/w 0-100% abstraction.
- It can't have constructors bcoz we cannot create objects of an interface
- If you want a class to achieve multiple inheritance.
- If an interface is made private, or if the methods in it are made private or protected; then compilation error will be thrown.

Advantages :-

- used to enforce a specification that classes must implement certain methods to use interface.
- multiple inheritance
- used to achieve loose coupling.
    - demerit: when a system is dependent on another system

(not adv.)

Learn Packages from Java

* Disadvantages :-

1. Interface expose their member variables to user since they must be public.
2. If any methods are declared in Interface then it becomes a must do situation for a class to implement all methods (neccessary or not neccessary).

Lecture - 10

Q How multiple inheritance problem solved ?

→
```
interface A {
    public void run ();
}
interface B {
    public void run ();
}
class C implements A, B {
    public void run () {
        System.out ("C class run method");
    }
psvm {
C obj = new C();
obj.run;
```

- earlier was ambiguity bcoz last time A also had a body for it said ("A runs") & B ("B runs") so C gets confused whom to call now since they both dont have any body ∴ C method runs.

* Lecture - 10

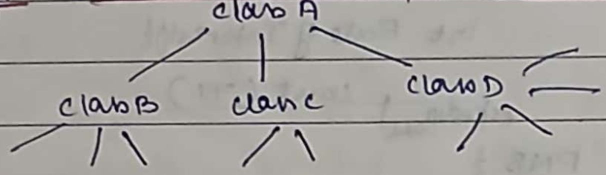
- super keyword :- 1. To invoke parent class variable
  2. " " " " method
  3. " " " " constructor

subclass ← child class

- if any (extended) base class is there with parent class then jab tak super keyword use nhi hoga tab tak parent class ka result nhi aayega in base class.

* Lecture-11

- Hierarichical Inheritance = ek class multiple class ko inherit karegi. (opposite of multiple inheritance).



* Lecture-12

- Method Overloading

3 main conditions of method overloading :-

1. same name of function.
2. Diff. no. of args.
3. Same no. of args but different datatypes.

eg:

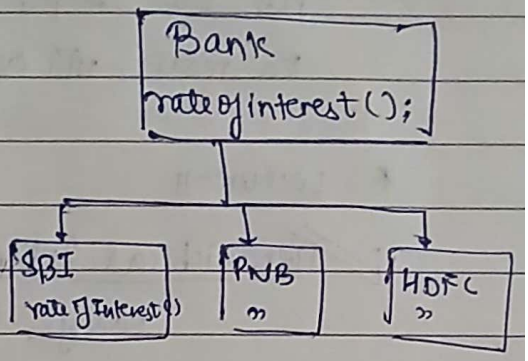
```
public static int sum (int a, int b) {
    return (a+b);
}

public static int sum (int a, int b, int c) {      // diff. no. of args
    return (a+b+c);
}

public static String sum ( String x, String y) {      // diff. data type
    return x+y;
}
```

no ambiguity

- Advantage :- for large program you can use same func. name to remember.

* Lecture-13

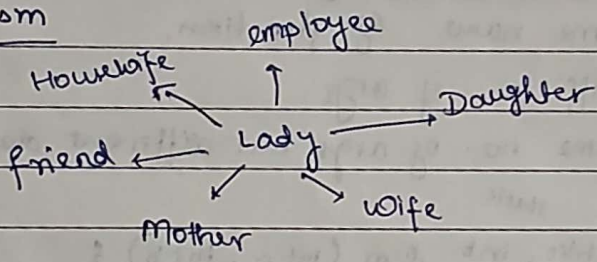
Method Overriding




```
class Bank {
        int RateofInterest(){
}
class SBI extends Bank {
        int RateofInterest(){
        sout(7%)
}
class PNB extends Bank {
        int " " " {
        sout(6%)
}
```

* Lecture-14

Polymorphism



* eg.1 Method Overloading code
* eg.2 " Overriding code

* Lecture-15

Types :-

1. Runtime → Polymorphism checked at run time - ∴ Slow

   ⇓

   Method Overriding ⇒ dynamic binding ⇒ Late binding

2. Compile time → " " " compile time

   ⇓

   Method Overloading ⇒ static binding ⇒ early binding

   ∴ Fast

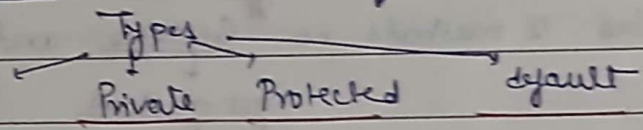
Date :
Page :

- Check differences b/w types in screenshot/whatsapp.
- Note :- we can directly use method overloading without making objects in psvm, not in case of overriding.
  ∴ Inheritance is must in case of runtime polymorphism.

* Lecture-16

Access Modifiers

use this in attributes of class



	Public	Private	Protected	Default
1. Same Class	✓	✓	✓	✓
2. Same Package subclass	✓	X	✓	✓
3. Same Package non-subclass	✓	X	✓	✓
4. Diff. Package subclass	✓	X	✓	X
5. Diff. Package non subclass	✓	X	X	X

- Note :- if no access modifier is mentioned than it is default Modifier.

* Lecture-17

- **Encapsulation**

(class — socho! private को focus कर रहा hai)

- Binding all data members in one (class) is called encapsulation
  (data members: variables + methods of class)
- Secure ∵ binded in one class only.
- contains 2 methods i.e. *Getters and *Setters.
  - Getters ↓ get method ↓ provide value, return value or print it.
  - Setters ↓ set method ↓ initialise value

e.g.

```
class Student {
    private String name;
    private int age;
    private float marks;

    public void get-details () {
        print (name);
        print (age);
        print (marks);
    }

    public void set-details (String name, int age, float marks) {
        this.name = name;
        this.age = age;
        this.marks = marks;
    }
```

(public — to access private in main)

* Lecture-18

- **Abstraction**
  - Data abstraction is a process of hiding certain details & showing only essential info. to user
  - achieved via abstract classes or interfaces

- Note:- We cannot make an object of abstract class.
  abstract class has its own special method called
  abstract method → which has no body in it. → same as interface
  ↳ only declared, eg. abstract public void print();

- Tip:- How ArrayList class is declared :-
  ArrayList <Integer> arr = new ArrayList <>();

- for (Employee employee : employeeList) {
  this is called for each loop
  ⇓⇓
  for (Employee employee ; employee < employeeList ; employee++) {

  eg. for (int i; n) {
  ⇓
  for (int i; i<n; i++) {

→ Remaining topics to learn in OOPs are → static keyword, Generics, enums, framework, lambda expressions.

* Static keyword in Java

Why? ek method/variable ko bar-bar chalane ke liye baar baar memory allocate karni pad rahi hai (obj. bana ke).

```
e.g.  class XYZ {
          print() {}   → memory taken here ✓
      }

      psvm
      XYZ obj = new XYZ();
      obj.print();   → memory taken here also (again)
      }                (new memory allocated)
```

∴ using static :-

```
XYZ {
    static print() {}
}
```

print is acting as a data member of XYZ directly.

definition

```
XYZ.print();
```

→ directly accessed without making new object memory

Definition → ✓ Static keyword in Java is used for memory management.

✓ We can apply static keyword with variables, methods, blocks and nested classes.

✓ Static keyword belongs to class than an instance of class.

3 types → 1. Static Variables
2. " methods
3. " Blocks.

- **Java Static Variable**

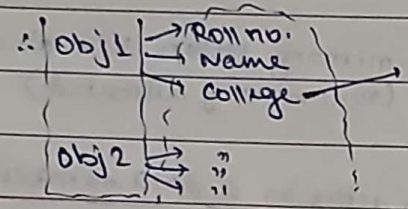
→ used to refer common property of all objects. (e.g. name of college for all students).

→ it gets memory only once in class area at time of class loading.

Advantage → makes your program memory efficient.

- problem:-

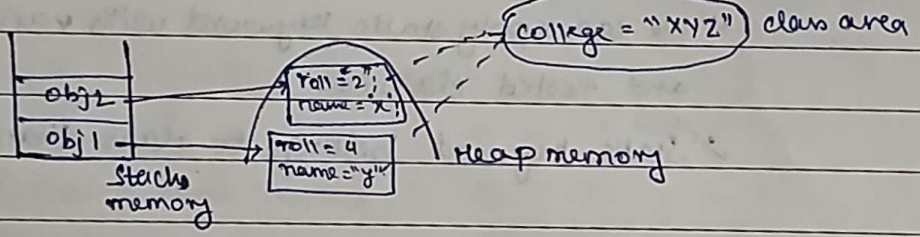
```
class student {
    int rollno;
    String name;
    String college = "XYZ";
}
```



roll & name are unique but why there's need of college extra space via obj1 & obj2 ∵ its common for both.

Solution:- (no need of allocating memory to same college again)

```
class Student {
    int roll;
    String name;
    static String College = "XYZ";
}
```



- Java Static Method:-

→ static with method = static method.

→ Static method belongs to class rather than object of class.

→ " " can access static data member & can change value of it.

```
class Hello {
    print void hello() {
        sout("Hello from non-static");
    }
    psvm {
        hello(); → error
    }
}
```

↳ [Note:- same class]

non-static method can't be referrenced from static context.

Solution:-

class Student {
    public static void hello(){
        sout ("hello");
    }
    psvm{
        hello();
    }
}

∴ we have psvm as static & to use sout, etc.

O/P
hello

- Two main restrictions for static methods :-
  → static method can't use non-static data member or call it directly.
  → this and super cannot be used in static context.

- Why main method is static ?
  ↳ public static void main (String args[]){

1. in class xyz {
        psvm {
        ------ ↳ no need for making a new obj of xyz to access it. we can directly access via static method.
        }
   }

- Java Static Block

anything inside static will execute first (before main also) & fix its value in memory.

```
class Main {
    static {
        sout (" static ");
    }
    psvm{
    sout ("main");
}}
```

] → representation of a static block.

O/P
static
main