

refers to a language that use objects in programming, they are 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. 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 stuff without referencing their instance in a static context.

Hence, static \rightarrow [static does not depends on object] object]

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

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

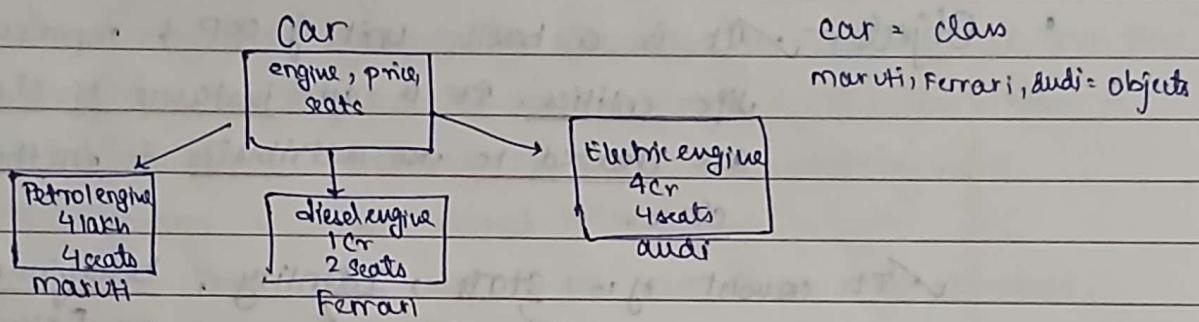
* Chapter 1 Actual notes:-

* 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



* 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 2. Method 3. Constructor

4. Nested class 5. Interface



* Class Declaration in Java :- capital (starting)

[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 (SL, rno);

sout (sl. name);

, sout (SL. 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 consists of :- state, Identity, behaviour

[value from its]

[data type]

distinguishes one

object from another.

Effect of data type

L Operations

~~Veg. of object Dog.~~

Identity

Name of Dog

State / attribute

Breed, Age, color

Behavior

Bark, sleep, eat

✓ 1 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, its value will be null until an object is actually 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 (ie. 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, mybox 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~~



Date: _____

Page: _____

A closer look at new:-

classname class-var = new classname();

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

✓ 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~~. 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.

'object' → looked by JVM.

Note:- Bus bus = new Bus();

looked by compiler

Constructors and "this" keyword :-

→ mark

Sometimes a method will need to refer to the object, 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, marking it essentially a constant. This means you must initialise 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
}`

~~of~~

✓ 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 coz, 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 Box {  
    Box() {  
        System.out.println("Base class constructor called");  
    }  
}
```

```
class Derived extends Base {
```

```
    Derived() {
```

```
        System.out.println("Derived class constructor called");  
    }  
}
```

```
public class Main {
```

```
    public static void main() {
```

```
        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."~~



OOPs → (Indian Programmer)

(Practical approach)

Lecture 1

```
class Student {  
    string name;  
    int rno;
```

nothing
mentioned
then 'default'
access modifier

name of class

access after making objects

} Data members / attributes of class

```
public void printData() {
```

sout(name);

} sout(rno);

run time

dynamic

helps in memory allocations

```
psvm {  
    object name  
    keyword → [constructor]  
    Student std1 = new Student();
```

class name

std1.name = "Ram";

} [Accessing attributes]

std1.rno = 56;

O/P

Ram

56

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

✓ [Accessing
using
function]

std1.printData(); ✓

stat sout(std1.name);

std1.name = "ram";

sout(std1.name); → // mutable

* 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() {  
        System.out.println(name + " eats");  
    }  
}
```

```
public static void main() {
```

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

✓ O/P

Bruno eats.

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

✓ Lecture - 3

Constructor → special type of fn/method, which is run when 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;  
        System.out.println("Hello World");  
    }  
    public static void main(){  
        ABC obj = new ABC();  
    }  
}
```

|| Shortcut alt+insert & select parameters

✓ O/P
Hello World

✓ Types of Constructors:-

1. Default ✓ Parameterized.

↳ previous example

✓ [No parameter passed]

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

✓ Both can be used when needed.

✓ When we 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.

if we pass arguments then parameterized constructor will be called else default will be called

✓ 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 or engine ki kya chal rha hai user koi mtlb nhi, chalne se har sing).

Hence,

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

- ✓ 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 & eliminates duplication.

i.e. agar ex type e.g. 'price' vehicle it already declared hai to car & phir k dedate kme ki koi jarurat nahi 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).

poly morphism
 " "

Poly morphism = multiple + forms

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

- There are two types of polymorphism:-

✓ Compile Time = Method Overloading = Static polymorphism

✓ 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

✓ Inherit = 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

✓ Multi-level Inheritance

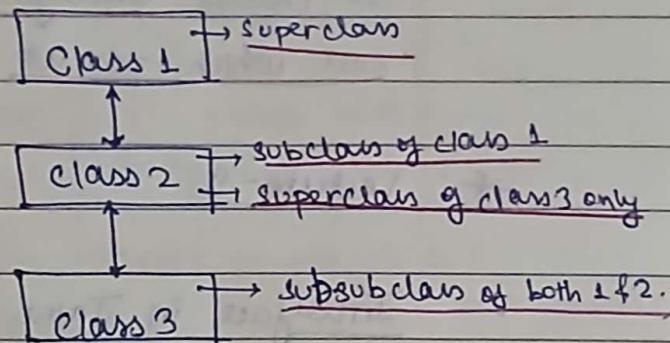
• e.g. GrandParent



Father



You



• refer code.

~~✓ Note :-~~ class of A cannot access class B & C + bly B can't access from class C.

* Lecture - 8

✓ Multiple Inheritance

→ Not possible in Java.

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

✓ e.g. class A {

void show () {

 System.out.println("class A method");

class B {

void show () {

 System.out.println("B method");

class C extends A, B {

void run () {

}

public

 C obj = new C();

 obj.show(); // ambiguity "which show to call"

If diff. name then only we can't do it.

is called "Diamond Problem in Java".

✓ This problem is solved by Interface.

helps in achieving multiple inheritance.

Ques. 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. → (mtib 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.

↓
mtib

✓ 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.

current engine → no use how it works

✓ Interfaces are used to achieve abstraction & implement multiple inheritance.

✓ e.g. Interface Pizza recipe {

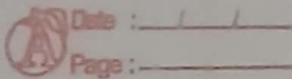
ShapeDough ()

```
    public void makeDough ();
        " " " Proof " " ();
        " " " prepareTopping ();
        " " " shapeDough ();
        " " " bakePizza ();
```

✓ not required
bcz it's there
by default in
interface.

∴ default access modifier of
interface is public (not default)

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



- How to declare & use an interface in Java?

⇒ 'Implements' keyword ✓

e.g. interface A {

```
    public void method1();
```

→ class B implements A {

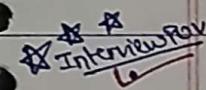
provider public void method1() {
Body ← [sout("xyz");]
}

can't define body in interface.

class C implements A {
 @Override public void method() {
 System.out.println("abc");
 }
}

Note :- Object kabhi bhi Interface ~~ki~~ nahi banta hai sir class ~~ki~~ banta hai.

here, ~~A~~ \boxed{A} obj = new C(); \checkmark : c implements A
A obj = new A(); X



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 be interface.
 - multiple inheritance
 - used to achieve loose coupling.
 - ↓
 - demerit: when a system is dependent on another system

* Disadvantages:-

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

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.println("C class run method");

} sum {

 C obj = new C();

 obj.run();

- ↑ earlier was ambiguity bcoz last time A had a body
 for also
 it said ("A runs"). Why, 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

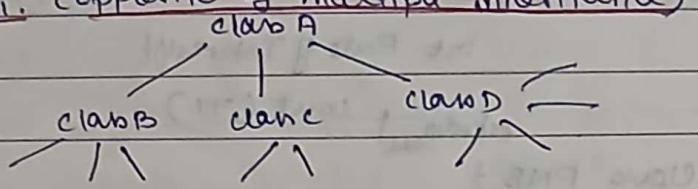
2. " " " " " method

3. " " " " " constructor

✓ if any (extended) base class is there with parent class then jab tak super keyword m nahi hogा tab tak parent class ka result nahi aayega in base class.

* Lecture-11

~~ii~~ Hierarchical Inheritance = ek class multiple class ~~at~~ inherit karegi. (opposite of multiple inheritance).



** Lecture-12

Method Overloading

3 main conditions of method overloading :-

✓ Same name of function.

✓ Diff. no. of args.

✓ Same no. of args but different datatypes.

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

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

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

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

* lecture-13

~~Method Overriding~~

class Bank {

 int Rate of Interest () {

 extends Bank ; }

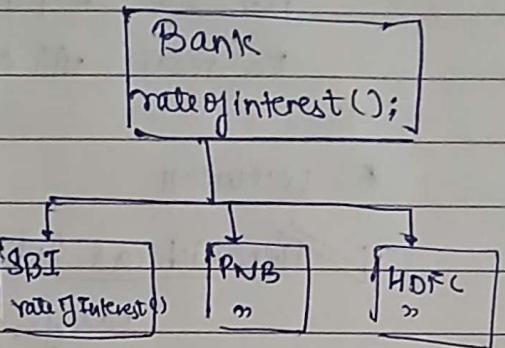
class SBI {

 int Rate of Interest () {

 extends Bank ; cout (7%)

class PNB {

 int " " " {
 cout (6%)
 }

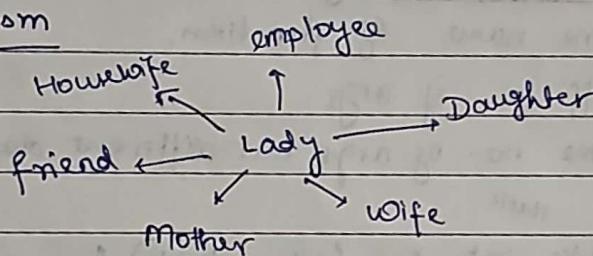


*

lecture-14

~~✓~~

Polymorphism



~~★~~

{ e.g. 1 Method Overloading code

e.g. 2 " overriding code

→

lecture-15

Types:-

Runtime → Polymorphism checked at run time -

Slow

↓

Method Overriding → dynamic binding ⇒ Late binding

2. Compile time → " " " compile time

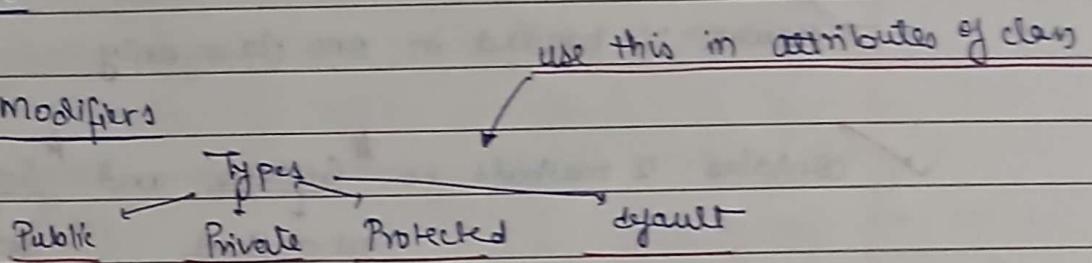
∴ Post

Method Overloading → static binding ⇒ early binding

- ✓ 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



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

✓ Note:- if no access modifier is mentioned then it is default modifier.

* Lecture-17

• Encapsulation

✓ binding all data members in one class is called encapsulation

variables
methods of
class

✓ secure & binded in one class only.

✓ contains 2 methods i.e. Getters and Setters.

↓
get method
provide value, return
value or print it.

↓
set method
initialise value

e.g. class Student {

private String name;
private int age;
private float marks;

✓ to access
private members
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; } }

* 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, e.g. abstract public void print();

- Tip!- How ArrayList class is declared? :-

ArrayList < Integer > arr = new ArrayList <>();

* for (Employee employee : employeeList) {
 ↓↓↓
 for (Employee employee ; employee < employeeList ; employee++) {
 (cause now)
 (gradient approach)
 loop
 }

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

* Static keyword in Java

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

e.g. class XYZ {

 print(); → memory taken here ✓
}

pcvm

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

∴ using static :-

XYZ {

 Static print();
 }

definition
 }

feature

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

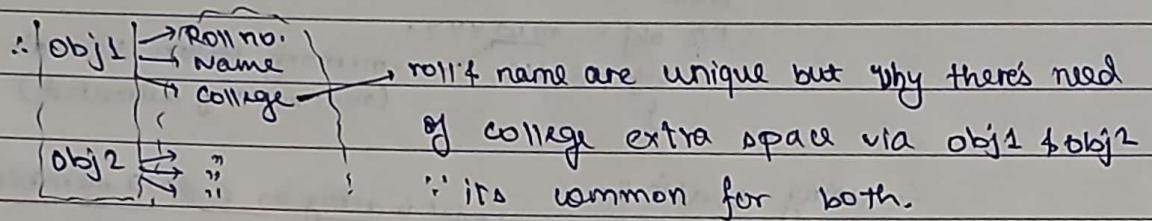
→ used to refer common property of all objects. (e.g. name of college for all students).

→ 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";
```



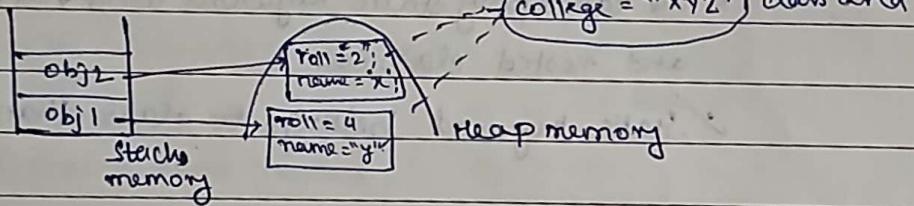
Solution:- (no need of allocating memory to same college again
class student)

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 {

~~W~~ Problem:- print void hello() {

```
sout("Hello from non-static");
```

Psum {

hello(); → error

3
4

↳ [Note:- same class]

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

Date: _____
Page: _____

Solution:-

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

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[]) {
 ...
}

in class XYZ {
 Psum {

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 {
 System.out.println("Static");
 }
 public static void main(String args[]) {
 System.out.println("main");
 }
}

representation of a block.

O/P
Static ✓
main