OOPS-1

Good Morning

Content

01. Programming paradigms
02. Procedural programming
03. Object Oriented programming
04. Access modifiers

Programming Paradigm

↳ Style or standard way of writing a program.

Why to follow programming paradigm?

- → less structured
- → Hard to read & maintain
- → Reusability
- → Easier to handle in multiple-developer environment.
- → Security standard.

* Types of programming paradigms?

01. Imperative Programming

→ Tell the computer to perform the task by giving instructions line by line

```
int a = 10
int b = 20
int sum = a+b;
print (sum);
int diff = a-b
print (diff);
```

02. Procedural programming

→ split the entire program into small procedures or functions.

```
int a = 10;
int b = 20;
addTwoNumbers(a,b);
subtractTwoNumbers(a,b);

void addTwoNumbers(a, b)
{
    int sum = a+b;
    print(sum);
}

void subtractTwoNumbers(a,b)
{
    int diff = a-b
    print(diff)
}
```

03 Object Oriented Programming

→ Build the entire program on the basis of classes & objects.

04 Declarative programming

→ We are not going to tell computer how to do it.

Instead, we will only tell what to do.

Eg:- Select * From TABLE

05 Functional Programming

→ Higher order functions

→ First class functions

→ State of variables doesn't change.

* Procedural Programming

→ Split the entire code into small functions

```
int as = sum(a,b)

void sum(int p, int q){
    return p+q;
}
```

* Issues with procedural programming.

⇒ We are studying

= Yogesh is teaching

= We are listening

Sentence = Subject + Action verbs

Visualise in real world
- → Someone is doing something
- = Entities that are performing some action

* Procedural programming, I have to go & print details of one student.

```
printStudent ( String name, int age, String Gender){
    print (name);
    print (age);
    print (Gender);
}
```

Is there any way to combine these set of attributes ↓

Struct / Structure

```
Struct Student {
    String name;
    int age
    String Gender;
}
```

→ In Java, it is not allowed to create functions in struct.

→ All variables inside struct are public

```
printStudent (Struct st) {
    print (st.name);
    print (st.age);
    print (st.Gender)
}
```

something → someone

In real world → we try to visualise everything as someone doing some action

But in procedural programming,

→ Action is being done on the entity

Cons of Procedural programming

01. Difficult to understand as we humans visualize things differently

02. We can't restrict the access.

* Object Oriented programming

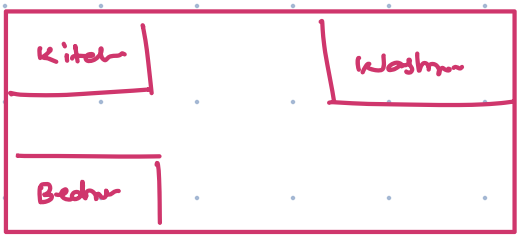
→ Entities are core in OOPs

→ Every entity will have some attribute or behavior.

⇒ Build our entire program through classes & objects (Entity)

* Class → Blueprint of object

→ Representation of idea



} structure of idea

```
class Student {
    String name;
    int age;
    String batch;
    double PSP;
    changeBatch();
    PauseCourse()
    SolveProblems()
    giveMockInterviews()
}
```

Properties

→ Class doesn't take any memory.

→ Not a real entity, just an idea.

→ Multiple instances of class.

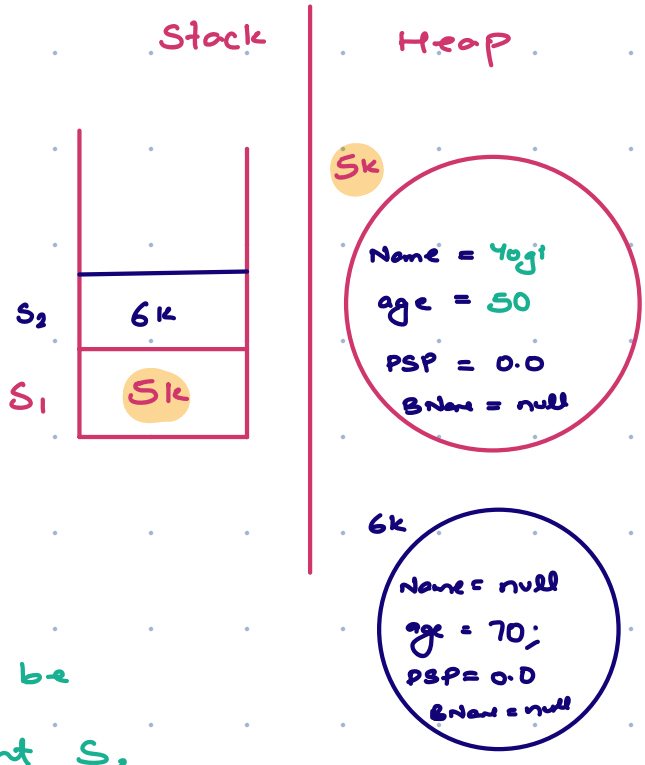
* Object = Real instance of the class.

```
class Student {
    String name;
    int age;
    String BatchName;
    double PSP;

    void changeBatch(String newBatch)
    {
        BatchName = newBatch;
    }

    void giveMockInterview()
    {
        System.out.print("Have to give to be DSA certified");
    }
}
```

```
PS v main() {
    Student S1 = new Student();

    S1.name = "Yogi";
    S1.age = 50;

    Student S2 = new Student();
    S2.age = 70;

    Student S3 = null;

}
```

object reference

Memory will not be allocated to student S_3



7:56 AM → 8:07 AM

When you write `object obj = null;` in most programming languages like Java or C#, you are declaring a variable `obj` of type `object` and initializing it to `null`. This statement itself does not allocate memory on the heap for an object. Instead, it simply sets up a reference variable that has the potential to point to an object. Initially, this variable does not point to any memory location in the heap because it is null.

In managed languages like Java and C#, objects are indeed allocated on the heap. However, the memory for these objects is allocated only when you create an instance of an object using the `new` keyword or equivalent object creation mechanisms in the language. For example, `obj = new Object();` would allocate memory on the heap for an instance of `Object` and assign its reference to `obj`.

Thus, the statement `object obj = null;` merely sets up a reference that could later point to an object in the heap, but does not itself cause any memory allocation in the heap.

* Principle & pillars of OOPS

Abstraction

Principle → Fundamental foundation / Concept

Pillars → Support the principle or hold things for principle

Encapsulation

Inheritance

Polymorphism

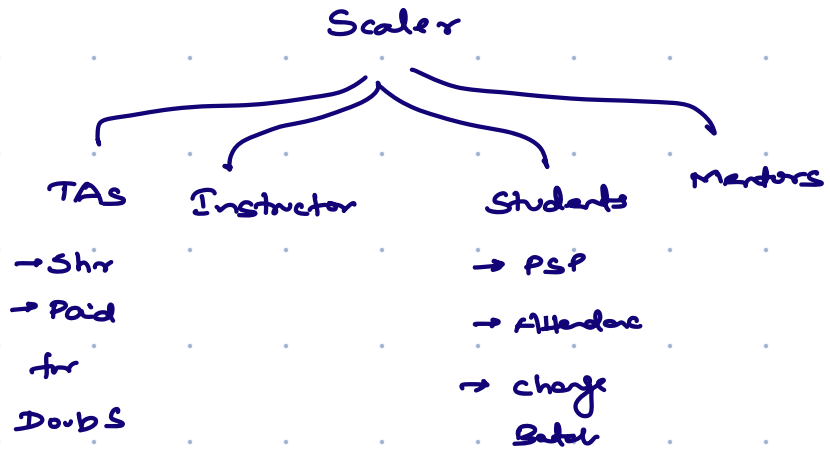
Principle → I'm a good person

Pillars → kind
helping
Honest / Good behavior
→ Attending every class

* Abstraction

→ Representing in terms of ideas.

Scaler
↓
→ Yogesh
→ Anshuman
→ Tanay ?



* Purpose of Abstraction

→ Share the relevant information, without sharing the implementation.

* Encapsulation

Capsule

Purpose of Encapsulation
- → Purpose of capsule is to hold medicine together.
- → protects the medicine from outside world

To hold attributes & behaviour together → Class

To protect attribute & behaviour from outside world → Access modifiers

Access modifier

01. Public → can be accessed by everyone
02. Private → can be accessed only inside the class
03. Protected → can be accessed inside the same package

Different package → only the child class can access it.

04. Default (Not defining public, private or protected)
    ↳ Every can access it inside the same package.

```
package P1;
    class Student {
        public int age;
        private String name;
        protected double PSP;
        String BatchName;
    }

    class abc {
        Student S3 = new Student();
        S3.name = X
        S3.PSP = ✓
    }
}
```

```
Package P2;
  class xyz {
      Student st = new Student();
      print(st.age); ⟶ ✓
      print(st.name); ⟶ X
  }
}
```

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

* This keyword

↳ current instance of that class

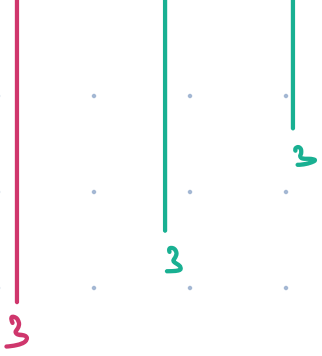
↳ Helps in distinguishing b/w class variable & method variable

```
class student {
    String name;
    int age;
    student(int age){
        this.age = age
    }
}
```

Student S1 = new Student(20);

```
package mypackage;

public class Student {

    public int id = 10;

    private int age = 25;

    protected String name = "Yogi";

    double PSP = 0.0;

    public static void main (String [] args)
    {
        Student st = new Student();

        print (st.id);   ⟶ 10

        print (st.age);  ⟶ 25

        print (st.name); ⟶ Yogi

        print (st.psp);  ⟶ 0.0

    }

}
```

```
package user;

class user{

    void printStudent(){

        Student st = new Student();

        print (st.id);   ✓

        print (st.age);  ✗
```



print (st.name); ✗

print (st.psp); ✗

* Static keyword

→ Class level variables or methods.

01. Static variables → Initialised when class is created / loaded & variable is present for all the objects.

→ Shared among all instances

02 Static methods → Shared among all instances

→ To invoke a static method, we don't have to create instance of an class. You can directly invoke it through class name.

```
class myclass {
    static int var = 0
    int instvar = 0;
    public myclass (int v) {
        instvar = v;
        var++;
    }
}
```

Static memory

var = 0 → 1 → 2

```
public static void main (String [] args)
{
    Myclass O1 = new Myclass (10);
    Myclass O2 = new Myclass (20);

    print (var); ——→ 2
    print (O1. instvar); → 10
    Print (O2 . instvar); → 20
}
```

O1 → instvar = 10

O2 → instvar = 20

```
class abc {
    Public static void main (String [] args )
    {
        int a = 10;
        int b = 20;
            sum (a,b);
    }

    Public static int sum ( a , b ) {
        int sum = a+b;          → method scope
        if (a<b) {
            int c= a-b          → block scope
            return c;
        }
        return sum;
    }
}
```

* Scope of variables

01. Class / Static scope

→ Variable can be accessed anywhere inside the class & the objects of that class can also access it.

02. Instance scope

↳ Declared inside the class but outside the function

→ Accessible inside the object.

03. Method variables

→ Variables declared inside a function can only be accessed inside that function.

04 Block scope { }

```
if (x > 10) {
    int sum = 0;
    sum += x;
}
```

sum variable can be accessed only inside if block.

Doubts

```
if (true) {
    int [] arr = new int [n];
    ==
    =
    -
}
else {
    int [][] arr = new int [n] [n]
    = ~
    = ~
    = -
    - -
}
```

SC →
Worst case scenario

- ~~O(n)~~ $O(n^2)$