

OOPS 1 : Introduction

AGENDA

- Programming Paradigm
- Procedural programming
- Object oriented programming
- Access modifiers

PROGRAMMING PARADIGMS

Q what is a programming paradigm?

↳ Style or Standard way of programming

Eg:- Way of writing code. For Example:- Burger can be made in many way but each brand following defined structure so that the taste is always same.

⇒ without programming paradigms what will happen

- (1) Less structured
- (2) Hard to read & understand
- (3) Hard to test
- (4) Difficult to maintain etc.

Different types of programming paradigm

(1) Imperative Programming :-

It tells the computer how to do the task by giving a set of instructions in a particular order i.e. line by line.

Ex:-

```
int a = 10;
int b = 20;
int c = a+b;
print(c);
int d = b-a;
print(d);
```

(2) Procedural Programming :-

It splits the entire program into small procedures or functions (Section of code that perform a specific task) which are reusable code blocks.

```
int a = 10;
int b = 20;
```

```
addTwoNumbers(a, b);
subtractTwoNumbers(a, b);
```

```
void addTwoNumber(a, b) {
```

```
    int sum = a+b;
    print(sum);
}
```

```
void subtractTwoNumbers(a, b) {
```

```
    int dif = a-b;
    print(dif);
}
```

(3) Object oriented Programming

↳ It build the entire program using classes & object.

(4) Declarative Programming

In this paradigm, you specify "what" you want the program to do without specifying "how" it should be done.

Eg:- `select * from Customer;`
[SQL query]

Based on current programming we are doing which paradigm we are using?

↳ Procedural programming.

PROCEDURAL PROGRAMMING

↳ It splits the entire program into small procedures or function [section of code that perform a specific task] which are reusable code blocks.

Procedure old name → Function

Execution of program starts from a specific procedure, what is it?

↳ Main Function()

For eg:- void addTwoNumbers(a, b) {

 |
 int sum = a + b;
 |
 | print(sum);
 |
 |}

void addThreeNumbers(a, b, c) {

 |
 int sum = a + b;
 |
 | addTwoNumbers(sum, c);
 |
 |}

void main() {

 |
 addThreeNumbers(10, 20, 30);
 |
 |}

↳ Problems with Procedural Programming

Q what you are doing right now?

↳ We are attending Lecture.

↳ I am having dinner.

↳ Chamandeep is teaching.

Generally

→ Subject + Verb (Someone is doing something)
i.e. entities perform action

Eg:-

PrintStudent (String name, int age, String gender) {

 |
 Print(name)
 | Print(age)
 | Print(gender)
 |
 |}

Q So is there any way in procedural programming to combine set of attributes?

↳ yes using struct or class

Eg:-

Public class Student {

 String name;
 int age;
 String gender;

}

Something

Someone

Print Student C Student st) {

}

print (st.name);
print (st.age);
print (st.gender);

So, something is happening on someone.

↳ Problem with Procedural programming

→ It's unnatural, because in real life entities perform action, but in procedural programming language action is performed on entity.

Procedure

OOPs

→ Procedure (Student)

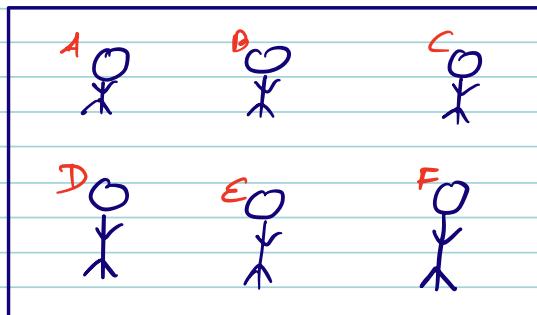
→ student . Print();

↳ Something is being done on someone

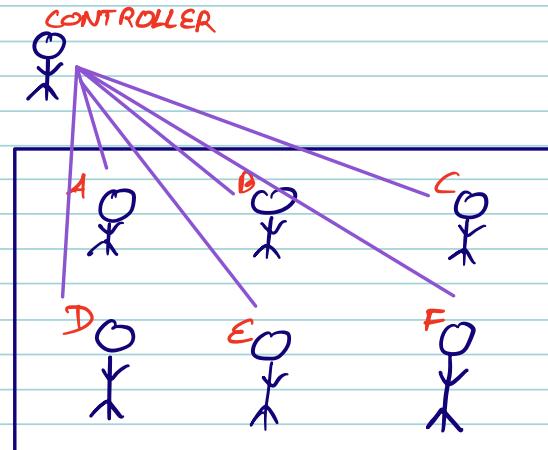
↳ Someone is doing something.

* Privacy is a major challenge in Procedural programming.

⇒ Real Life Example



Object oriented world

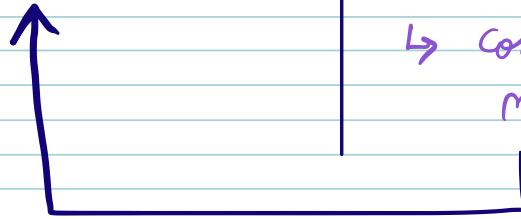


Procedural Programming world.

↳ Here each individual have a control in hand.

↳ controller controls everyone. [PUPPET SYSTEM]

↳ Controller can manipulate the entities



⇒ OOPs Example

Eg:-

class Student {

 string name;

 int age;

 string gender;

 void print () {

 Print (name);

 Print (age);

}

 Student st = new Student();

 st. print();

⇒ Cons of Procedural Programming

- Difficult to make sense.
- Difficult to Debug.
- Spaghetti of code i.e unstructured

OBJECT ORIENTED PROGRAMMING [OOPS]

- Entities are core in OOPs
- Every Entity have some attributes & behaviour.
- ↳ In OOPs using class we will build the program objects (entity).

⇒ CLASS

- Blueprint of an Idea.
- Ex:- Floor plan of building
- class represents structure of Idea.

Eg:- class Student {

 int age;

 String name;

 double PSP;

 Pause Course () { } }

 change batch () { } }

 give Mock Interview () { } }

→ class take no space in memory

→ Not a real entity

→ multiple instances of same class

⇒ OBJECTS

↳ The Real Instance of class

Qn 1 :- Will object of a class occupy memory?
→ Yes

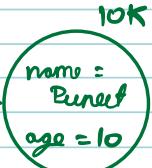
⇒ Creation of class & objects.

```
class Student {  
    int age;  
    String name;  
    String batchName;  
    void changeBatch (String newBatch) {  
        batchName = newBatch;  
    }  
    void giveMockInterviews () {  
        System.out.println ("Giving Mock Interview");  
    }  
}
```

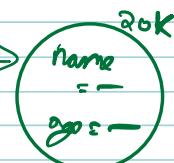
// Object Creation -

```
public static void main (String [] args) {
```

```
    Student st1 = new Student ();  
    st1.name = "Puneet";  
    st1.age = 10;
```



```
    Student st2 = new Student ();  
    st2.name = "Vibas Yadav";  
    st2.age = 20;
```



PILLARS OF OOPs

→ 3 Pillar :- Support to hold thing together

→ 1 Principle :- Fundamental Foundation/Concepts

Ex:-

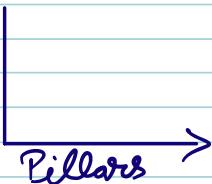
Principle :- I will be a good person



- (1) I will be truthful
- (2) I will do hardwork
- (3) I respect everyone

Similarly In oops

Principle of oops → ABSTRACTION



ways to implement
Abstraction

- (1) Inheritance
- (2) Polymorphism
- (3) Encapsulation

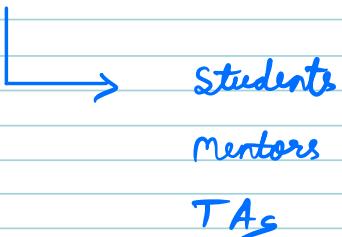
SOURCE:- Java: The Complete Reference

ABSTRACTION

- ↳ Hiding or Privacy? [Not Exactly]
- ↳ Representing in terms of Idea

Ex:-

SCALER



Q Do they have behaviour?

↳ yes, all of them have behaviour.
like, Student can take course, pausetc

So, Abstraction is an idea of representing complex software system in terms of Ideas, because ideas are easy to understand.

⇒ Purpose of Abstraction ?

↳ others don't need to know details of data.

Ex:- ① Dynamic Array

② Taking a left turn.

Q what needed to represented in terms of ideas?

↳ Data

↳ Anything that has behaviour.

PILLARS

① ENCAPSULATION :-

Q → Does the word Encapsulation contains the word which you are aware of?

↳ Capsule [Hold things together]

Q → what we really store in Programming?

↳ Attributes & behaviours

Q2 :- where do we store attributes & behaviour together? what is the technical term for that?

↳ CLASS

② ACCESS MODIFIERS

We got to know that Encapsulation have two Advantages.

- ① One is to hold data & attributes together
- ② Second is it protect members from illegitimate access. You can't access the data from class unless allow you to.

- ↳ First point can be done using class.
- ↳ Second point can be implemented using Access Modifiers.

Quiz :- Which one of these is not an access modifier?

↳ OPEN.

⇒ Types of Access Modifiers :-

- ① Public
- ② Private
- ③ Protected
- ④ Default

Eg:-

```
class Student {  
    public int age;  
    private String name;  
    protected double PSP;  
    String batchname;
```

① Public access modifiers

↳ A public attribute or method can be accessed by everyone.

② Private Access Modifier

↳ A private attribute or method can be accessed by no one, not even the child class.

Q:- If no one can access then what's its purpose?

↳ Privacy / Security etc. . .

③ Protected Access modifier

↳ Protected methods or Attributes can be accessed only from the same package. And subclass of different package.

Package A

class xyz {

 protected int age;

}

Package B

class student extends xyz

 // will be using age

3

④ Default access modifiers → within same package.

	class	package	subclass (same pkg)	subclasses (diff pkg)	world
Public	✓	✓	✓	✓	✓
Protected	✓	✓	✓	✓	
Default	✓	✓	✓		
Private	✓				

Ques 4 :- which is most Restricted Access Modifier
↳ Private.

Ques 5 :- which is most open Access Modifier?
↳ Public

THIS KEYWORD

↳ Used to refer current instance of class or object.

↳ Helps in differentiating instance variables & local variable or method parameter with same name, & to access instance members within method.

modify

Eg:-

```
class Person {  
    String name;  
    void changeName (String name) {  
        this.name = name;  
    }  
}
```

Person P = new Person();
P. changeName ("Vikas");
10K

name =
Vikas

Person P₀ = new Person();
P. changeName ("Puneet");
20K

name =
Puneet

STATIC KEYWORD

→ In Programming languages like Java & C++ is used to Declare class - level members or methods.

which is associated with class itself

① Static Variable

→ when you declare a variable as "static" within a class, it becomes a class variable. These variables are shared among all instances of the class. They are initialized only once when the class is loaded, & their values are common to all objects of the class

Ex:-

```
Public class MyClass {  
    static int Var = 0;  
    int instance Variable; → [IV]
```

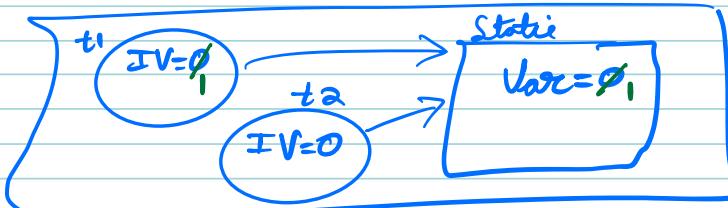
Constructor ←

```
Public MyClass () {  
    instance Variable=0;
```

```
Public void Test () {
```

```
    instance Variable++; Var ++;
```

Now



My Class t₁ = new My Class();

Print (t₁. instance Variable); → 0

Print (t₁. Var); → 0

t₁. Test();

Print (t₁. Var); → 1

My Class t₂ = new My Class();

Print (t₂. instance Variable); → 0

Print (t₂. Var); → 1

Print (My Class.Var); → 1

Observations

① Static Variable is created once for all classes.

② Static variable can be accessed directly from class Name.

② Static method (class Methods)

→ when you declare a method as "static" within a class, it becomes a class method. These methods are invoked on the class itself, not on instances of class. They can access static variables & perform operations that doesn't require access to instance-specific data.

Eg:- Public Test {

 Public static void Main() { } }

 Public static void Swap() { }

Test. Swap();

Scope of a Variable

① Class / Static Scope

↳ Variables declared as static within a class have class level scope. They are associated with class rather than instance. Can be accessed with class name & shared among all instances.

② Instance Scope

↳ Variables defined in class but outside any method / constructor have instance level scope. Associated with a instance of class & each instance has its own copy of these variables.

③ Method / Local Scope :-

Variables declared within a method of class. Can be accessed within specific method.

④ Block Scope :-

Variables defined within {} [Curly braces]. can be accessed within same {}.

class Dummy {

 static int age = 10; → Static Level
 int name = "Chamandeep"; → Instance Level

 PrintCounting () {

 int start = 1; → Method Level.

 for (i=0; i<start; i++) {
 int temp = i+1; → Block Level
 SOP(temp);

