OOPS intro

1) Programming paradigms
2) Procedural programming
3) OOP   Object Oriented Programming
4) Access modifiers

Contest Reattempt till 31 Dec

Note: Concept >>> syntax

5 Jan Contest
Recursion, Math, OOPS

Programming paradigms

Style / Standard way of writing code

Types

1) Imperative → Line by Line instruction

```
a = 10
b = 20
sum = a+b
print (sum)        // 30
```

2) Procedural → Split program into small procedures / functions

```
a = 10                    int add (a, b) {
b = 10                    |    return a+b
x = add (a, b)            }
print (x)
```

3) Object Oriented → we see today

4) Declarative → Tell what you want, NOT how you want it

eg SQL   Select * from users;

● Procedural → Split program into small procedures / functions

```
a = 10
b = 10
add (a, b)
```

```
int add (a, b) {
    return a+b
}
```

} reusable blocks of code

Execution of any code starts which function?
⟹ main()

Issues with Procedural programming

printStudent (String name, int age, String gender)

```
struct Student {
    String name
    int age
    String gender
}
```

Student S₁
S₁.name = "Omansh"
S₁.age = 24
S₁.gender = "Male"

Issues :

1) In Java, struct cannot have methods / function

2) All members of struct are accessible at all times to anyone who wants to access

print(S₁.age)
print(S₁.gender)

Class ⇒ Blueprint of an idea

Eg - Floor plan of a house

```
class Student {
    int age
    String name
    String gender
    void changeBatch()
    double psp
    void pauseCourse()
}
```

```
Student S1 = new Student()
S1.name = "Omansh"
S1.age = 24
S1.gender = "male"
Student S2 = new Student()
S2.name = " ABCD "
S2.age = 28
S2.gender = "female"
```

Principle of OOP → Abstraction (hide the details)

3 Pillars ⇒ 1) Inheritance
2) Polymorphism
3) Encapsulation

Today ⇒ Abstraction & Encapsulation

Rest ⇒ LLD module

● Abstraction (hide the details)

Eg ⇒ turn on car ( )
drive Car ( )
steer Car ( )

1) Relevant data
2) Relevant behaviours

Do you know how this done ? No
You just know what you need to do .
Rest is handled internally .

● Encapsulation ⇒ Capsule
→ Holds powder together
→ Avoids mixing of multible types of powders
→ protect medicine from outside envirornment

What do we store in programming ?
data & behaviors
Where can we do this ?
Class

How does a class protect from outside environment

⇒ Access modifiers

Access Modifiers

1) Public → anyone can access
2) Protected → same package can access
3) Private → Only the class itself can access, no one else

	Class	Package	Subclass	World
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Private	✓	✗	✗	✗

Package
Collection of similar files

Sub class
Class which derives data/ behaviour from another.

scaler . placements . history

A/B/C/D/ scaler/ placements/history / ----→ class x
↳ class y

placements / students

placements / f3 /- - - - - .

Subclass $\longrightarrow$

```
class Bird {
    feathers
    run ()
    fly ()
    color
}
```

$$\begin{tikzcd}
 & \text{Bird} \arrow[dl] \arrow[dr] & \\
\text{Pigeon} & & \text{Crow}
\end{tikzcd}$$

```
class Pigeon : public Bird {
        beak
        feet
}
```

chirping

Static → variable/method common for all objects
keyword of a class

```
class Myclass {
    static int count = 0
    int n

    public Myclass (int val) {
        this . n = val
          ↳ current object
        count ++
    }
}
```

```
Myclass obj1 = new Myclass (10)          obj1.n = 10
Myclass obj2 = new Myclass (20)          obj2.n = 20
print (Myclass . count)    = 2
```

```
class Student {                          Student S1 = new Student()
    int age                              S1.name = "Omansh"
    String name                          S1.age = 24
    String gender                        S1.gender = "male"
    void changeBatch ()                  Student S2 = new Student()
    double psp                           S2.name = " ABCD "
    void pauseCourse ()                  S2.age = 28
}                                        S2.gender = "female"
```

name
age
S1
name
age
S2

- Scope ⇒ region where code is accessible & can be used

Static ⇒ class-level scope
Instance ⇒ class variables . each variable belongs to an instance
Method ⇒ function level scope

```
int factorial (int n){
    if(n==1)   return 1
    ans = n * factorial(n-1)
    return ans
}
```

Block ⇒ for {}   if{} else{}   try catch {}
while {}   switch case

```
int a=10
if(a==10){
    int b=20
    print (a+b)
}
```

a ⇒ [✓ In scope] / out of scope

b ⇒ In scope / [✓ out of scope]

{done}

$\sqrt{n}$ $\qquad\qquad$ $n \log \log n$

$n \log n$

N

1 numbel

$a_0 \quad a_1 \quad a_2 \quad - - - . \quad a_{n-1}$

$\longleftrightarrow$

± 2 $\qquad$ even → even

odd → odd

k $\quad$ even

$n-k$ $\quad$ odd

even → odd $\qquad\qquad$ odd → even

k $\qquad\qquad\qquad\qquad$ $n-k$

$\min(k, n-k)$