Programming Paradigm

Procedural Programming

Object Oriented

Access Modifiers

Linked List | Trees

Programming Paradigm

Standard way of writing a program

Without programming paradigm, code :

- hard to read and understand
- hard to test / maintain
- Less structured

Types of programming paradigms

1. Imperative Programming -

   Telling computer how do task by giving a set of instructions in a particular order i.e. line by line

```
// For eg:
int a = 10;
int b = 20;
int sum = a + b;
print(sum);
int dif = a - b;
print(dif);
```

BASIC, COBOL

2) Procedural programming : C

Split the program into multiple function (section of code that performs a specific task) which are reusable code blocks

```
// 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);
}
```

3. Declarative Programming : SQL

Specify what you want program to do, without specifying how should it be done

Select * FROM Student

4. Object Oriented Programming

Builds the entire program using classes and objects

C++, Java, JS, Python, C#

Procedural programming C/C++

Procedure → function / Method

Each fn can call other fns.

Execution starts from procedure
↓
main()

Problems with procedural programming

1. Tushar / Kalu is losing OOPs
2. Ayush is making notes
3. Apurva is teaching OOPs
4. Amit is watching live lecture.

Subject + Verb
↓ ↓
Entity action

Someone is doing something

```
print Student (int age, String name, char gender) {
    print (age)
    print (name)
    print (gender)
}
```

Any way in procedural programming to combine set of attributes?

struct or structure

Student s

s.name = "Shazi"

like a class but not exactly a class

struct Student {

int age

String name

char gender

}

① Java, a struct has no methods

C/C++ → struct or class can have methods

name . var_name

name → var_name

② All variables in struct are public, i.e. visible to everyone

Something SomeOne

printStudent (Student s) {

print (s.age)

print (s.name) // s → name

print (s.gender)

}

In procedural programming → Action is performed on entity

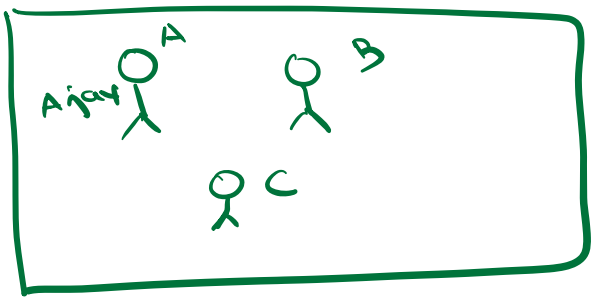
printStudent (Student s) // Procedural

vs

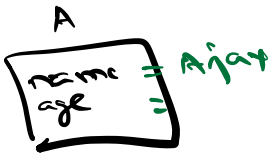
s.print() → ✓ // OOP

① Not close to real life

② Data privacy



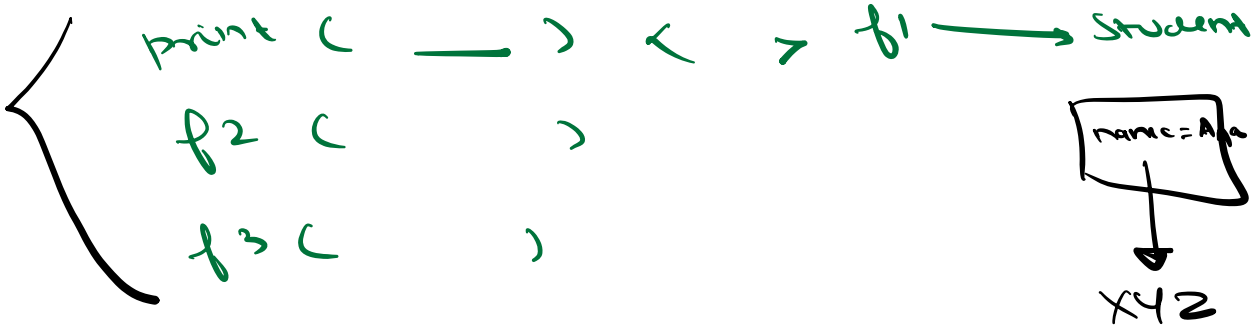

```
Struct student {
    name
    age
}
```




```
Student A =
    new student
A.name = "Ajay"
```

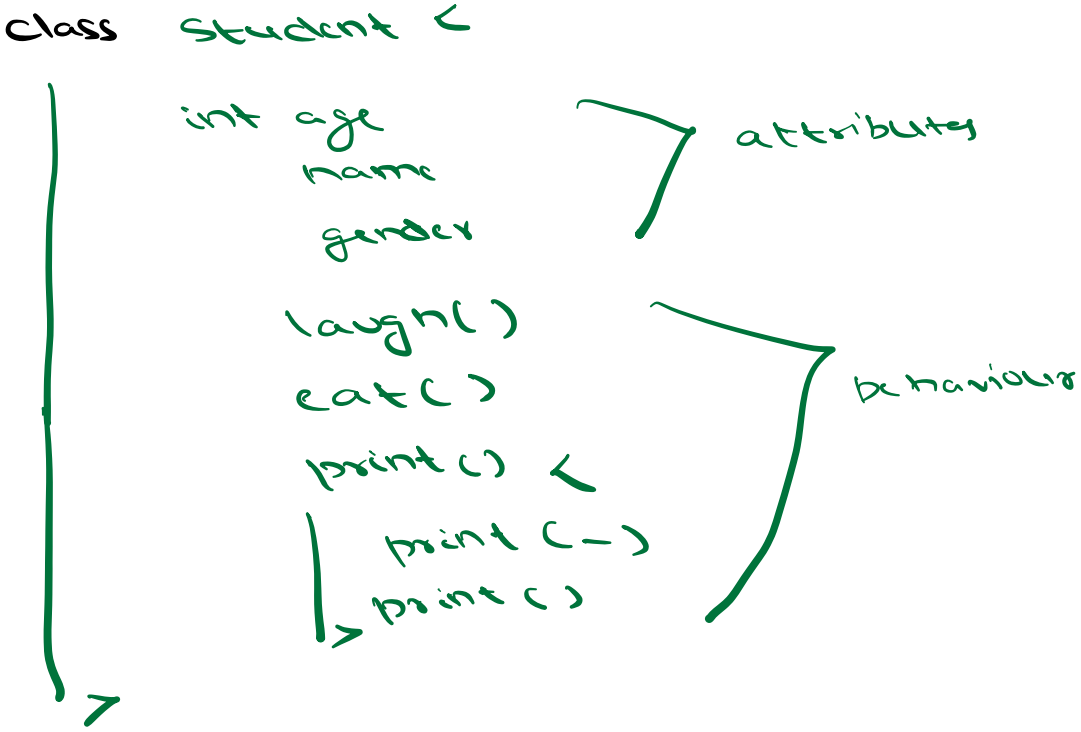

```
Student B =
    new student
B.name =
```

Cons of Procedural programming



OOPS → object oriented programming

Entities → action



OOPS : Entity in core in OOPs

Entity → Attribute

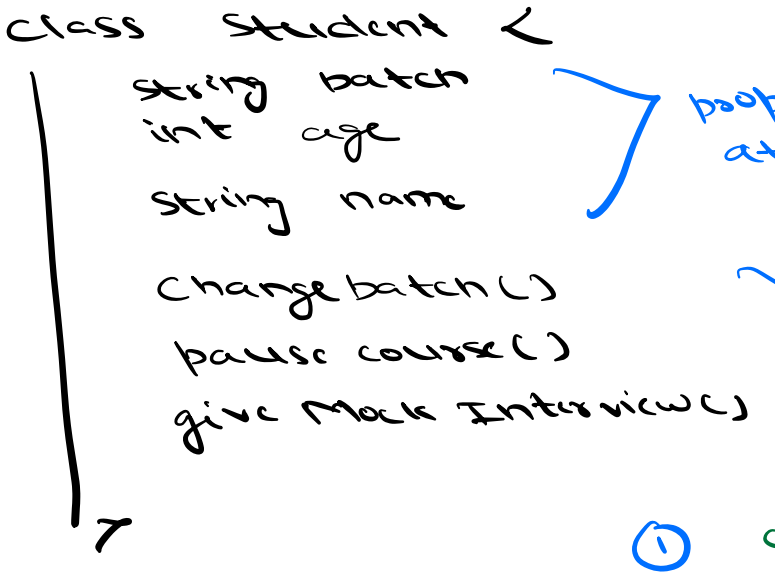
→ Behaviour

Class : Blueprint / template of entity

↓

Blueprint / Floor plan of house

Object : Real life entity / class instance



Object → Real instance of class

↓

occupy memory

① class → not a real entity

② class takes no space in memory

③ Multiple instances of same class

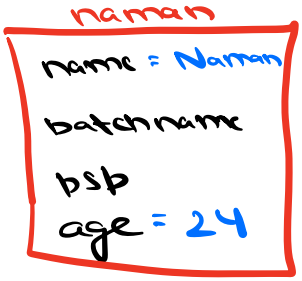
```java
public class Student {
    String name;
    1 usage
    String batchName;
    int age;
    double psp;

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

    void giveMockInterview() {
        System.out.println("Giving mock interview");
    }
}
```

Student naman = new Student()

naman.name = "Naman"
naman.age = 24



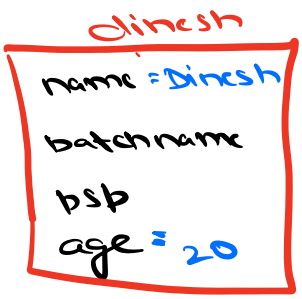
Classname objname = new Classname()

Student dinesh = new Student()

dinesh.name = "Dinesh"

dinesh.age = 20

dinesh.mockinterview()



Pillars of OOPS -

1. Principle - 1 → abstraction → Foundation / Concept
2. Pillars - 3 → Support to hold foundation together

1. Inheritance
2. Polymorphism
3. Encapsulation

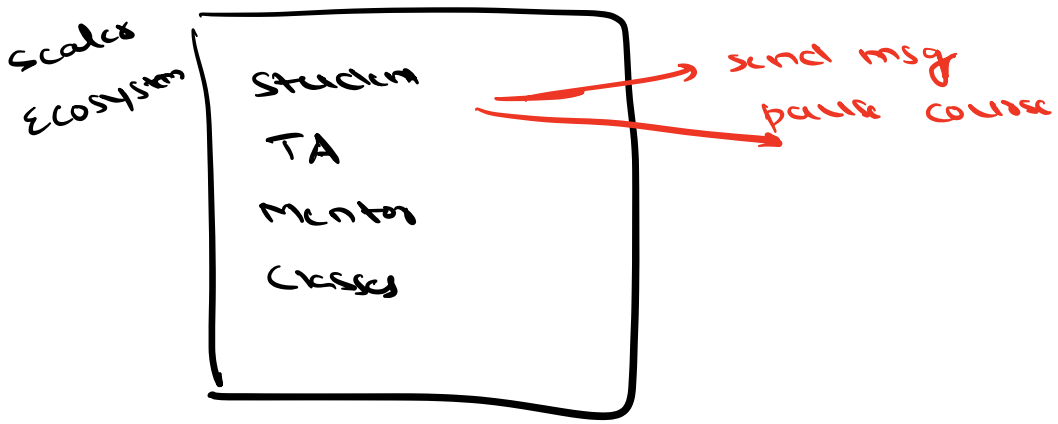
Principle - I want to be a good person

Pillars - I'll be honest
work hard
respect everyone

Java : The complete reference

Abstraction → Representing in terms of ideas

Idea → attribute and associated behaviour



Main purpose of abstraction :

Don't need to know details of idea

- Data
- Behaviour

Encapsulation → OOPs

↓

Capsule

- It will flow away, hold medicine powder together
- Protects the medicine from outside world
- multiple medicine powders are present in capsule, helps to avoid mixing them together

Attribute and Behaviour together

↓

class {
  attributes
  behaviour
}

class Student {
  int age
  private mock()
}

Student s

s.age = 23

Access Modifiers

1) Public : Can access everywhere

2) Private : Can be accessed only inside class definition

3) Protected : Can be accessed only from classes of same pkg or subclasses can access

* Default : Can be accessed in same class and same pkg

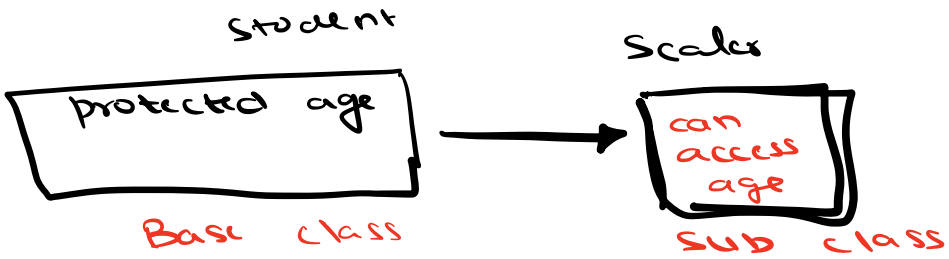
s.setName

Student s

→ s.name = "Name"

→ s.name = "123"

Class Student {
  private String name
  public setName(n) {
    name = n
  }
}



	Class	Package	subclass (same pkg)	subclass (diff pkg)	World
public	✓	✓	✓	✓	✓
protected	✓	✓	✓	✓	✗
No modifier	✓	✓	✓	✗	✗
Private	✓	✗	✗	✗	✗

Most restricted access modifier → Private

Most open access modifier → Public

This keyword → refer to current object / class instance

1) distinguish b/w object instance variable and local variable method parameter inside a fn

2) To avoid variable name ambiguity

Example 1

```
public class Person {
    private string name

    public Person (string name) {
        this.name = name
    }

    public void introduce () {
        print ("Hello, I am" + this.name)
    }
}
```

```
Person p1 = new Person()
```

p1

p2


```
main () {
    Person p1 = new Person ("Alice")
    Person p2 = new Person ("Bob")

    p1.introduce()   // Hello I am Alice
    p2.introduce()   // Hello I am Bob
}
```

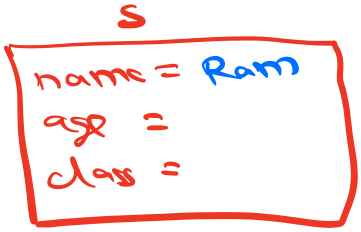
Ex 2

```
class Student {
    String name
    int age
    String class

    public Student (string n) {
        name = n
    }
}
```

```
String s = new Student ("Ram")
```

s



Example of Access Modifiers

```java
package mypackage;

public class AccessModifierExample {
    public int publicVariable = 10; // Public access

    private int privateVariable = 20; // Private access

    protected int protectedVariable = 30; // Protected access

    int defaultVariable = 40; // Default (package-private) access

    public void publicMethod() {
        System.out.println("This is a public method.");
    }

    private void privateMethod() {
        System.out.println("This is a private method.");
    }

    public static void main(String[] args) {
        AccessModifierExample example = new AccessModifierExample();

        System.out.println("Public variable: " + example.publicVariable);
        System.out.println("Private variable: " + example.privateVariable);
        System.out.println("Protected variable: " + example.protectedVariable);
        System.out.println("Default variable: " + example.defaultVariable);
    }
}
```

```java
package otherpackage;

import mypackage.AccessModifierExample; // Import the class from a different package

public class AnotherClass {
    public static void main(String[] args) {
        AccessModifierExample example = new AccessModifierExample();

        System.out.println(example.publicVariable); // Accessing publicVariable is v
        System.out.println(example.defaultVariable); // Error: Cannot access default

        example.publicMethod();
        example.privateMethod(); // Error: Private method is not accessible outside
    }
}
```

Static → used to declare class level data members or methods, info is associated with class

Static variables: These variables are shared among all objects. They are initialized once when the class loads, their value is common to all objects.

Static methods: Methods invoked by the class. They can access static variables and don't require access to object variables

```java
public class MyClass {
    // Static variable
    static int staticVar = 0;

    // Instance variable
    int instanceVar;

    public MyClass(int value) {
        this.instanceVar = value;
        staticVar++;
    }

    public static void main(String[] args) {
        MyClass obj1 = new MyClass(10);
        MyClass obj2 = new MyClass(20);

        System.out.println("Static Variable: " + staticVar); // Output: Static Varia
        System.out.println("Instance Variable (obj1): " + obj1.instanceVar); // Outp
        System.out.println("Instance Variable (obj2): " + obj2.instanceVar); // Outp
    }
}
```

print (staticVar) → 0

2 ← 10 ← 20

MyClass
staticVar = 0 ✗ 2

MyClass n = new MyClass()

n → instanceVar

MyClass.staticVar

obj1
instanceVar = 10

obj2
instanceVar = 20

Scope of a variable

↳ region / area / context within your code where a specific variable can be used or accessed

depending on where variable was declared

1. Class / Static scope
2. Instance scope
3. Method scope
4. Block scope

```java
public class ScopeExample {
    // Class-level variable (static scope)
    static int classVar = 10;

    // Instance variable (instance scope)
    int instanceVar = 20;

    public void exampleMethod() {
        // Method-level variable (method scope)
        int methodVar = 30;

        if (true) {
            // Block-level variable (block scope)
            int blockVar = 40;
            System.out.println(classVar + instanceVar + methodVar + blockVar);
        }
        // The 'blockVar' is out of scope here.
    }
    print(methodVar) → out of scope
    public static void main(String[] args) {
        ScopeExample obj = new ScopeExample();
        obj.exampleMethod();
        // The 'methodVar' and 'blockVar' are out of scope here.
    }
}
```

Doubts

4, 6

5th

Am magical no.

4, 6, 8, 12, 16, 18, ...

4 → 1 till 12 | 6 → 1 till 12

$\frac{12}{4} = 3$ → 4, 8, 12 | $\frac{12}{6} = 2$ → 6, 12

1 → 21

$\frac{21}{4} = 5$

1 ——— 12

5 - 0

4/6

4, 6, 8, 12, 16, 18, 20, 24, ...

24 → 8th magical no

4 → 1 to 24 | 6 → 1 to 24

$\frac{24}{4} = 6$ | $\frac{24}{6} = 4$

6 + 4 = 10 - 2

= 8

LCM (4, 6)

12 → 1 to 24

$\frac{24}{12} = 2$

Q6

n → 5th magical no.

$$\frac{n}{4} + \frac{n}{6} - \frac{n}{LCM(4,6)}$$

a / b

Nth

s

min(a,b)

c

N x min(a,b)

Nth magical no

a = 2

b = 1000

2, 4, 6, 8, 10 , 1000

5th magical no. 10

Min Difference

Lower bound(5) = 4

0	1	2	3	4	5	6	7	8
1	2	4	7	8	10	20	40	80

(x)

s	e	m		
0	8	4	8 > 5 mid > x	left
0	3	1	2 < 5 mid < x	ans = 2 right
2	3	2	4 < 5 mid < x	ans = 4 right
3	3	3	7 > 5	left
3	2	break		

if mid = x return mid