

# OOPS 1 : Introduction

## AGENDA:

- Programming Paradigm
- Procedural Programming
- Object Oriented Programming
- Access Modifiers

Rajat Sharma
sharath r
Saurabh Ruikar
Khushi Raj
Shrikanth
Madhan Kumar M S
Akansh Nirmal
Vishal Mosa
Pankaj
Rajendra
Purusharth A
Vasanth
Rajeeb Kumar Nath
suyash gupta
Sushant Patil
Shani
Venkata Sribhavana Nandiraju
Shradha Srivastava
Gowtham Sankaran
Abhishek Sharma
Vimal
Sanket Giri
Nikhil Pandey

Avg  
~ 64% → next target  
70%

100%  
**GREAT JOB!**

## Programming Paradigms

what is a programming paradigm ?

structured way of writing code.

Eg → Dominos pizza have a structured way of making pizza .

what would happen w/o programming paradigm ?

- \_\_\_\_\_ Difficult to maintain
- \_\_\_\_\_ Hard to test
- \_\_\_\_\_ Difficult to read etc.....

## Types of Programming Paradigms

### 1> Imperative Programming

→ You tell the computer how to do a task

Eg

```
int a = 10  
int b = 20  
int total = a+b  
print(total)
```

### 2> Procedural Programming

Splitting the entire logic into small procedures,

methods  
functions

Eg

```
int a = 10  
int b = 20  
int total = add(a,b)  
print(total)
```

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

### 3) Declarative Programming

You tell what to do , but not how to do it .

Eg — SQL

```
select * from user ;
```

### 4) Object Oriented Programming — OOP

Later today .

## Procedural Programming

Splits entire code into small procedures or functions

### QUIZ

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

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

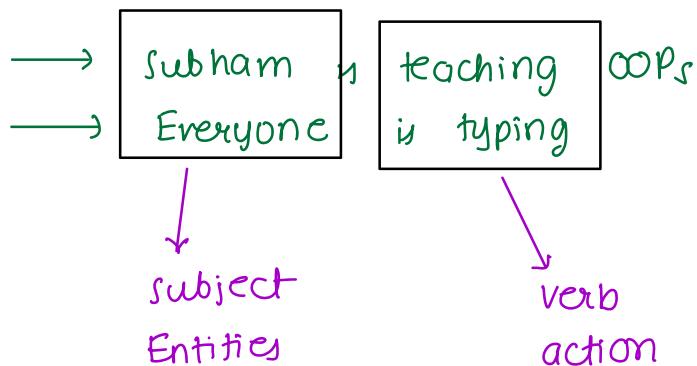
```
int sub( int a, int b) {  
    |  
    | return a-b  
|  
|}
```

```
main() {  
    |  
    | int x = add( 10, 20);  
    | int y = sub ( 20, 40);  
|  
|}
```

## Problems with procedural programming

QUIZ

Type one line on what's going on in class right now?



In real world Entities perform action.

```
void printStudent ( String name,  
                    String batch,  
                    double psp )  
    {  
        print( name )  
        print( batch )  
        print( psp )  
    }  
combine them  
in a struct.
```

```
struct student {  
    String name  
    String batch  
    double psp  
}
```

```
void printStudent ( student student) {
```

```
    print( student.name )  
    print( student.batch )  
    print ( student.psp )
```

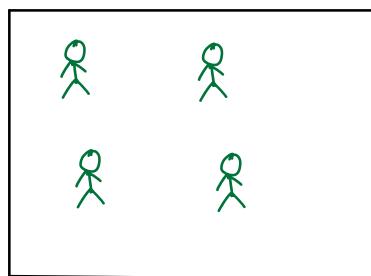
3

action is being performed on student.

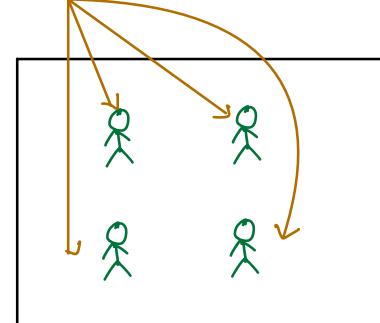
Issues with struct

- all the properties are public
- structs donot support methods.

Real



controller



How a student looks in OOP world ?

```
class Student {  
    string name  
    string batch  
    double psp
```

```
    void print () {
```

```
    print ( name )
    print ( batch )
    print ( psp )
}

void solve() {
    .....
}
```

cons of procedural programming ?

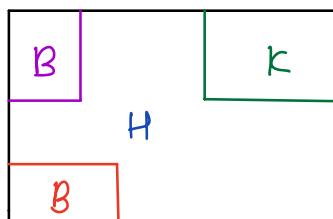
- Not secure { everything is public }
- less maintainable & reusable.
- hard to debug .

## Object Oriented Programming

Entities + behaviour.

QUIZ

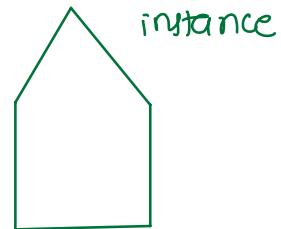
what is a class ?  
Blueprint of an idea.



Blueprint.

} doesn't take space  
significant space  
present in  
method area of  
JVM

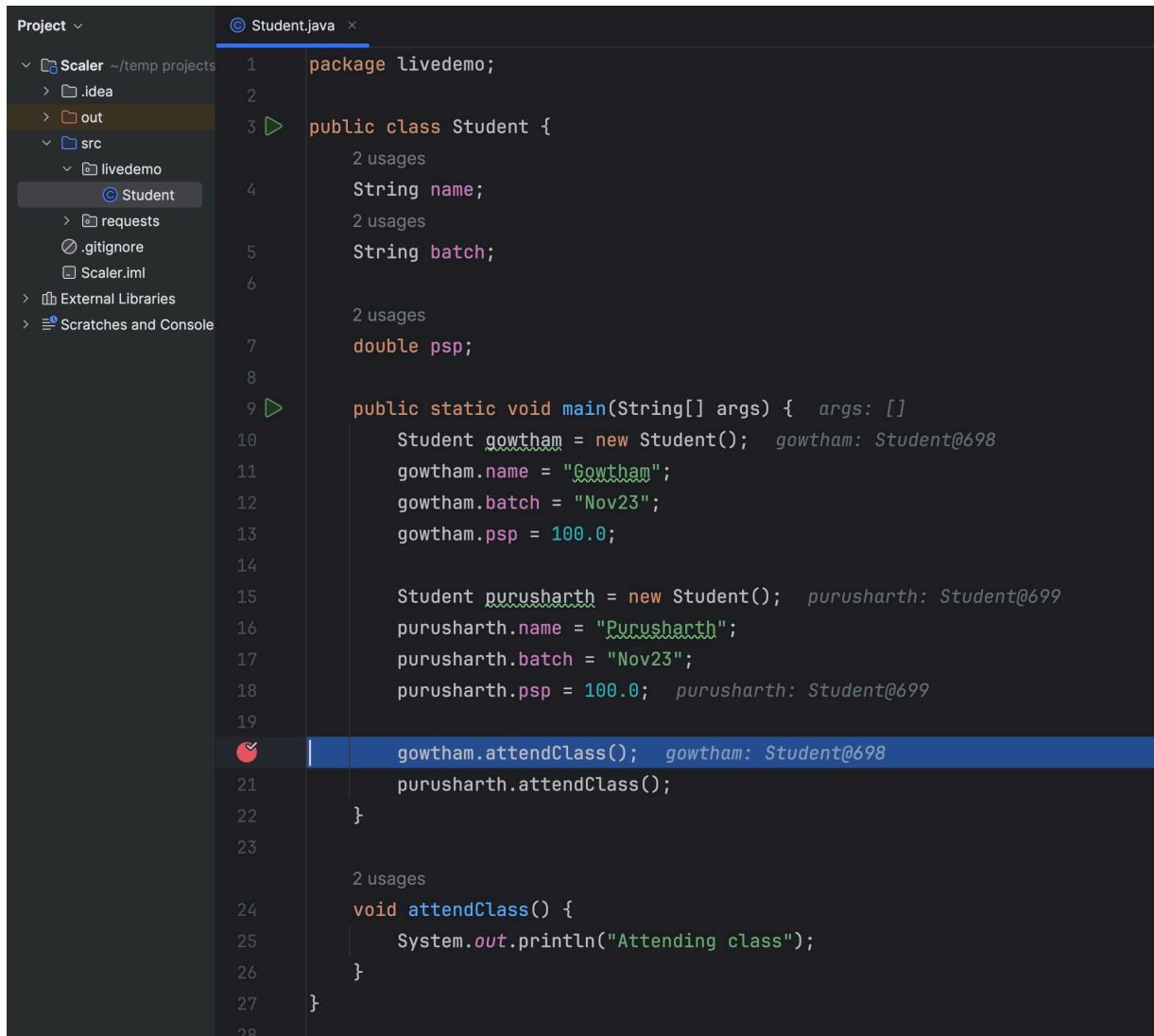
Object — Instance of a class



takes space in heap memory.

A class can have many instances(s)

## Classes & Objects { live demo }



The screenshot shows a Java code editor with the following code:

```
Project ▾ Student.java x
Scalor ~/temp projects .idea out src livedemo Student requests .gitignore Scaler.iml External Libraries Scratches and Console
package livedemo;
public class Student {
    String name;
    String batch;
    double psp;
}
public static void main(String[] args) {
    Student gowtham = new Student();
    gowtham.name = "Gowtham";
    gowtham.batch = "Nov23";
    gowtham.psp = 100.0;
}
Student purusharth = new Student();
purusharth.name = "Purusharth";
purusharth.batch = "Nov23";
purusharth.psp = 100.0;
gowtham.attendClass();
purusharth.attendClass();
}
void attendClass() {
    System.out.println("Attending class");
}
```

The code defines a `Student` class with attributes `name`, `batch`, and `psp`. It contains a `main` method that creates two `Student` objects, `gowtham` and `purusharth`, and calls their `attendClass` methods. The `attendClass` method prints "Attending class". The code editor shows syntax highlighting and code completion suggestions.

Break : 10:25

## Pillars of OOPs

QUIZ:

How many pillars of OOPs are there?

There are 3 pillars and 1 principle.

Principle — Fundamental foundation

Java : The  
Complete Reference.

Pillar — Support for holding thing together

principle of a student? → discipline

How to be a disciplined student.

→ solving Q on time

→ attending class & contest

Encapsulation

Inheritance

Polymorphism

## Abstraction

**QUIZ:** what do you understand by abstraction ?

→ Representing in terms of an idea.

Design Scaler → Ideas

Student, Mentor, Instructor

→ What all thing we will represent in the programming word ?

class → attributes and behaviors / methods

## Encapsulation

**QUIZ:** Does encapsulation contain a word we use in daily life ?

capsule → Holds various types of medicine together  
→ protects the medicine from outside environment.

**QUIZ:** What do we really store in programming ?

→ attributes  
→ methods } class

```
class student {  
    string name  
  
    void tellName {  
        }  
    }
```

How to protect class from outside env ?

## Access modifier

## Types of Access Modifiers ?

- Public — can be accessed by every one
  - Private — can be accessed by no one.  
outside of class.
  - Protected — can be accessed by classes in same package as well as **sub class**. child class.
  - Default — can be accessed within same package

class	package	sub class (same package)	sub class diff package	world
public	✓	✓	✓	✓
protected	✓	✓	✓	✓
default	✓	✓	✓	
private	✓			

## this keyword

```

class student () {
    String batch;
    double psp;
    void changeBatch (String batch) {
        this.batch = batch
    }
}

void compare (student s2) {
    this.psp compare with s2.psp.
    >=
}

```

## Examples

```
1 package livedemo;
2
3 public class AccessModifierExample {
4     3 usages
5         public int publicVar = 10;
6         2 usages
7             protected int protectedVar = 20;
8
9             2 usages
10                int defaultVar = 30;
11
12                1 usage
13                    private int privateVar = 40;
14
15
16                    1 usage
17                        private void privateMethod() {
18                            System.out.println("Private method");
19
20                            3 usages
21                                public void publicMethod() {
22                                    System.out.println("Public method");
23
24                                2 usages
25                                    protected void protectedMethod() {
26                                        System.out.println("Protected method");
27
28                                    2 usages
29                                        void defaultMethod() {
30                                            System.out.println("Default method");
31
32
33
34                                    2 usages
35                                        accessModifierExample.accessModifierExample();
36                                        System.out.println(accessModifierExample.publicVar);
37                                        System.out.println(accessModifierExample.protectedVar);
38                                        System.out.println(accessModifierExample.defaultVar);
39
40                                        accessModifierExample.publicMethod();
41                                        accessModifierExample.protectedMethod();
42                                        accessModifierExample.defaultMethod();
43                                        accessModifierExample.privateMethod();
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139 }
```

```
1 package livedemo;
2
3 ▶ public class OutsideClass {
4 ▶     public static void main(String[] args) {
5         AccessModifierExample accessModifierExample = new AccessModifierExample();
6         System.out.println(accessModifierExample.publicVar);
7
8         // Child class as well as any class in the same package can access protected variables
9         System.out.println(accessModifierExample.protectedVar);
10
11        // within the same package, default variables can be accessed
12        System.out.println(accessModifierExample.defaultVar);
13
14        // CTE since private variables can only be accessed within the same class
15        // System.out.println(accessModifierExample.privateVar);
16        // accessModifierExample.privateMethod();
17
18        accessModifierExample.publicMethod();
19        accessModifierExample.protectedMethod();
20        accessModifierExample.defaultMethod();
21        // accessModifierExample.privateMethod();
22    }
23}
24|
```

```
1 package outside;
2
3 import livedemo.AccessModifierExample;
4
5 ▶ public class OutsidePackageClass {
6 ▶     public static void main(String[] args) {
7         AccessModifierExample accessModifierExample = new AccessModifierExample();
8         System.out.println(accessModifierExample.publicVar);
9
10        // Child class as well as any class in the same package can access protected variables
11        // System.out.println(accessModifierExample.protectedVar);
12
13        // within the same package, default variables can be accessed
14        System.out.println(accessModifierExample.defaultVar);
15
16        // CTE since private variables can only be accessed within the same class
17        // System.out.println(accessModifierExample.privateVar);
18        // accessModifierExample.privateMethod();
19
20        accessModifierExample.publicMethod();
21        // accessModifierExample.protectedMethod();
22        accessModifierExample.defaultMethod();
23        // accessModifierExample.privateMethod();
24    }
25}
26|
```

## Static keyword

→ static variables — If a variable is associated with class rather than instance of a class.

Eg : Integer. MIN\_VALUE.  
                  static .

→ static methods — No need to create instance of a class to call static methods.

Eg : Math. pow      System.out      Math. max

```
class staticEx {  
    static int cnt = 0;  
    static void increase() {  
        cnt++  
    }  
}
```

common action all instances of this class.

```
staticEx.increase() // cnt = 1  
staticEx.increase() // cnt = 2
```

## Scope of a variable

Class level / static scope — Common across all instances of class

Instance scope — variables whose scope is limited to instance of a class

Method scope — scope level is within the method

Block scope — {} variables scope is within braces.

```
1 package livedemo;
2
3 ▶ public class ScopeDemo {
4     // Class level variable
5     1 usage
6     public static int classVar = 10;
7
8     // Instance level variable
9     4 usages
10    int instanceVar = 20;
11
12    }
13
14 ▶ public static void main(String[] args) {
15     // Block level
16     if (true) {
17         int blockVar = 40;
18         System.out.println("Block level variable: " + blockVar);
19     }
20
21     // Accessing class level variable
22     ScopeDemo scopeDemo1 = new ScopeDemo(); // instance 1
23     ScopeDemo scopeDemo2 = new ScopeDemo(); // instance 2
24     scopeDemo1.setClassVar(100);
25     scopeDemo1.instanceVar = 200;
26     scopeDemo2.instanceVar = 300;
27     System.out.println(scopeDemo1.instanceVar);
28     System.out.println(scopeDemo2.instanceVar);
29
30     scopeDemo1.method();
31
32    }
33
34    1 usage
35    public void method() {
36        // Method level variable
37        int methodVar = 30;
38        System.out.println("Method level variable: " + methodVar);
39    }
40
41 }
```