OOPS 1 : Introduction

Agenda

- Programing Paradigms
- Procedural Programing
- Object Oriented Programing
- Acess Modifiers

Programing Paradigms → Standard way of writing the program.

Types →

Imperative Programming → Tell the computer how to do a task in a particular order.

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

Procedural Programming → Divide the code into small procedures or functions.

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

```
void addTwoNumbers(a, b){
    int c = a + b
    print(c)
}
```

Declarative Programming → Tell the program what to do instead of how to do it.

eg → SQL

```
select * from Users;
```

Procedural Programming

Adv → Reusable code blocks

Procedures → functions / Methods

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

void main(){    // code starts from here
    int a=10
    int b=20
    addTwoNumbers(a,b)
}
```

Real World

We are studying
Ketan is learning
Nipun is teaching
We are drinking water

Someone is doing something

Subject + verb

```
printStudent(String name, int age, double psp){
    print(name)
    print(age)
    print(psp)
}
```

to combine set of attributes → struct / structure

```
struct Student {
    string name;
    int age;
    double prof;
}
```

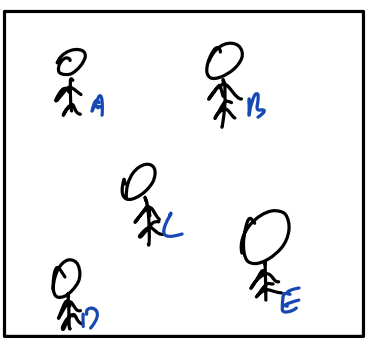
everything is public

```
}
```

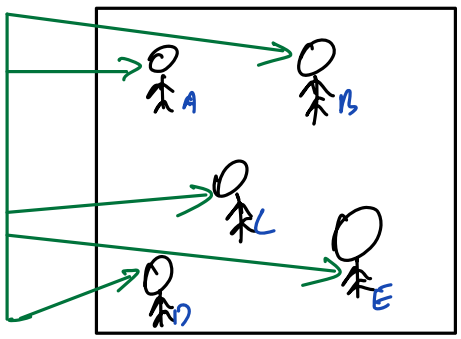
```
print Student(Student st) {
    print(st.name)
    print(st.age)
    print(st.prof)
}
```

something — someone

something is being done on someone.



Real world

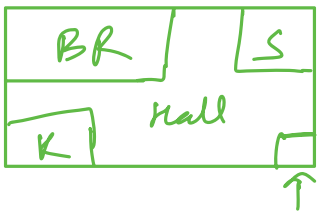


Procedural world

Actual main problem is privacy

Summarize ⇒ 1. Difficult to make sense w.r.t real world
2. Difficult to debug
3. Also difficult to track

Object Oriented Programming



1. Class → Blueprint of an idea

   eg → floor plan of the apartment

```
class Student {
    string name
    int age
    double psp
    :
    giveMockInterview() {...}
    joinClass() {...}
    :
}
```

1. Class takes no space in memory
2. Not a real entity
3. Multiple entities are possible for the same class.

Real Entity → Object
→ occupy memory

```
void main() {
    Student S1 = new Student();
    S1.name = "Kavita"
    S1.age = 27
    S1.psp = 90
```

Student S2 = new Student(); → constructor

keyword

⋮

}

datatype variable name

Student S2;

int a;

Pillars of OOPs

- Abstraction → Principle
- Encapsulation
- Inheritance
- Polymorphism

Principle → fundamental concept / foundation

Pillar → support to hold things together

Abstraction

Representing in terms of ideas.

Student S1 = new Student()

S2

⋮

Main purpose is not to understand details of the idea.

Drive car → Press break → Stop the car

~~How?~~

Encapsulation

→ capsule

Hold things together

Protect the medicin from outside

↓

Acces modifiers

Attributes & Behaviours
variables function

class

Acces modifiers

1. Public → Accessed by everyone
2. Private → Accessd by noone [accessd by class only]
3. Protected → Accessd by class & subclass
4. Default (not defined) → Accessd within package

class → subclass in a different package

→ protected ✓

→ default ✗

"this" keyword ["self" in python]

↓

current instance of the class/object

```
class Student ( ){
  double psp;
    :
    int comparePSP(Student s2){
      return(this.psp - s2.psp);
    }
}

Student s1 = new Stude ();
  s1. comparePSP(s2);
```

Example of access modifier

```
package mypackage;

public class Student {

    public int id = 10;
    private int age = 25;
    protected String name = "Salman";
    double psp = 0.0;

    public static void main (String[] args) {
        Student st = new Student();
        print (st.id); ✓
        print(st.age); ✓
        print(st.name); ✓
        print(st.psp); ✓
    }
}

class User {
    void printStudent() {
        Student st = new Student();
        print (st.id); ✓
        print(st.age); X → compile time error
```

print(st.name); ✗
print(st.psp); ✓

3 3

Static → Keyword used to declare class level members / methods.

1. static variable → shared among all instances of the class.

They are initialized once the class is loaded.

2. static method → These methods can be invoked on the class itself.

They can access static variable & perform operation on it.

Note: To access non-static variable, we need object.

```
class Student {
    String name;
    static int cnt;
    double psp;
    :
    static void addCount() {
        cnt++
    }
}
```

```
cnt=0;
Student st = new Student();
Student.addCount();

Student s2 = new Student();
Student.addCount();

// cnt → 2
```

Scope of a variable

1. Class / Static scope : variable created on class level & shared among all instances of class.

2. Instance scope : variable defined in class, but not in any function. Accessed in a specific instance.

3. Method / Local scope : defined inside a function. accessed only in ten function.

4. Block scope: define inside { ... }

accessed only in {...}

```
public Class Student() {
    static int classVar;
    int instanceVar;
    public void function() {
        int methodVar = 10;
        if (condition) {
            int block var;
            :
        }
    }
}
public static void main() {
    Student s = new Student();
    Student.classVar ✓
```

5. instance Var ✓

5. method Var ✗

5. block Var ✗