Agenda :

1. Introduction to OOPs
2. Pillars / Principles of OOPs
3. Classes / Objects
4. Access modifiers.

Introduction to OOPs

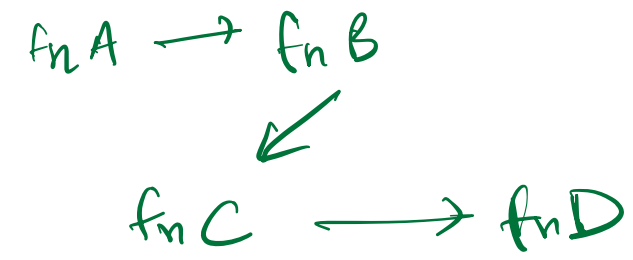
Object Oriented Programming -

Programming Paradigm :

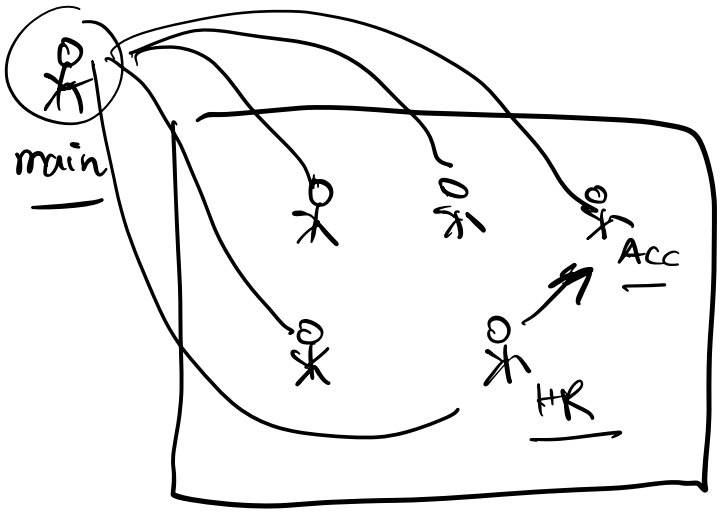
Procedural (C) & OOPs (Java, Python, C++, C#)

Procedural :→ Procedure → Set of instructions
↳ functions / methods.

```
fn A {          fn B {          fn C {
  _ _             _ _             _ _
  _ _             _ _             _ _
  _ _             _ _             _ _
}               }               }
```



* functions don't relate to anything or each other.



printStudent (name, id, batch, age ...) {

}

Student

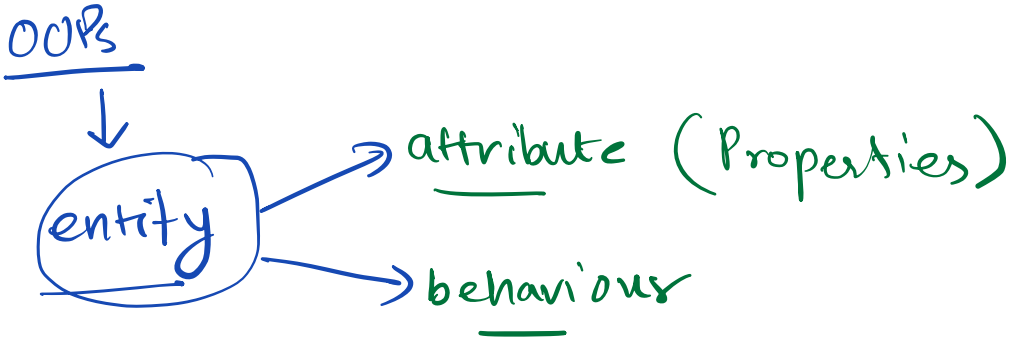
Akash is teaching
Student are learning
People are thinking of a line
Some people are sleeping.

subject + verb

somebody is doing something

Procedural	OOPs
printStudent(s)	student.printStudent();
Something on someone.	somebody doing something.

1) Less Readable
2) less Understandable
3) Less Reusable
4) Less extensible/maintainable.



Car

Attributes:
1. wheels
2. # doors
3. Engine
4. color
5. brand
6. size

Behaviours:
1. accelerate()
2. brake()
3. start()
4. turning()
5. Horn()

7. Capacity

Student

Attributes

name,
email,
DOB,
age,
batch,
id
no.

Behaviour

attendClass()
solveProblem()
takeNotes()
takeContests()

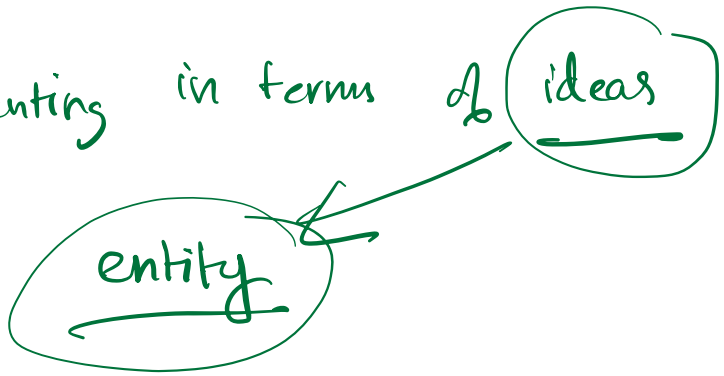
Pillars of OOPs

1. Abstraction → Principle
2. Encapsulation
3. Polymorphism
4. Inheritance

Pillars.

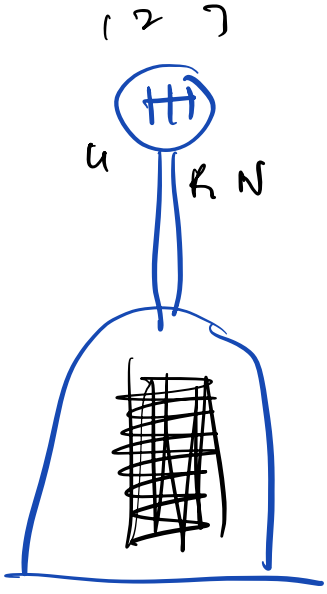
Abstraction

- representing in terms of ideas



Bird

Attributes :	Behaviours
# wings	fly()
weight	eat()
color	walk().
curly	



sort() comparator

3rd Party libraries

Abstraction :

1) representing complex ideas in terms of entities.
2) Others's don't need to know the details.

Encapsulation :

Capsule :

1) Containing everything.
2) Protect it against outside environment.

1) Binding together our attributes & behaviour
   ↳ class.
2) Protecting against illegal access
   ↳ Access Modifiers.

class → • Blueprint of an entity
• custom datatype.

```
Class Student {

String name;
int id;
int age;
String batch;

void printData() {

}

void attendClass() {
```

attributes

behaviour.

3 3

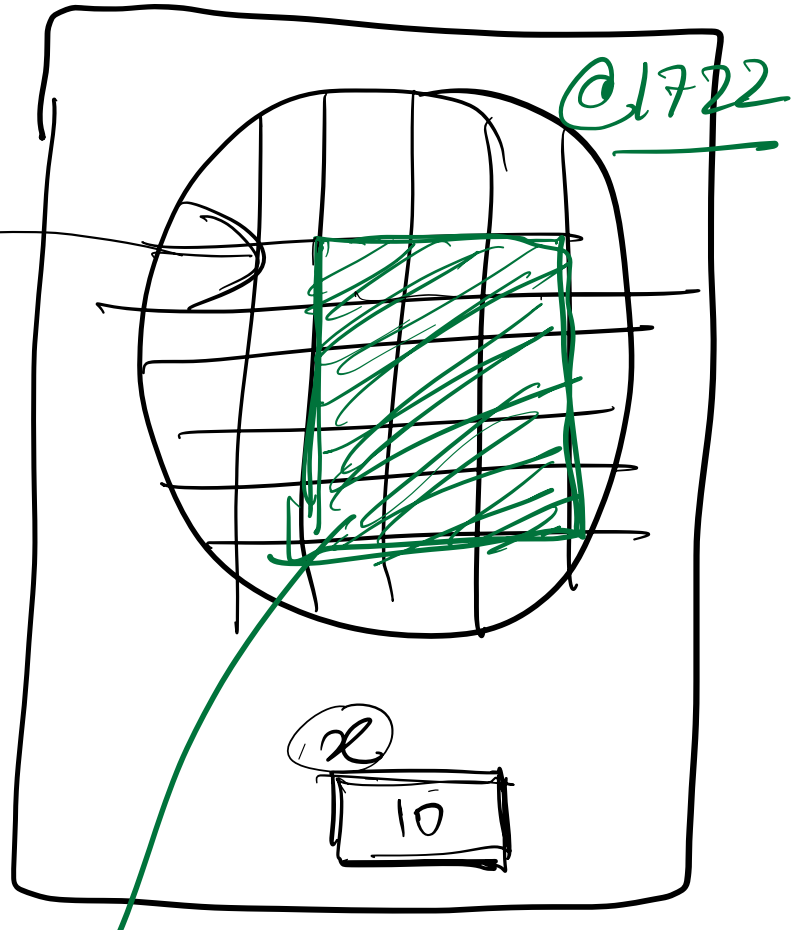
Object is when class in brought into existence

reference variable

Class Name

```
Student x = new Student();
```

- Student → datatype
- x → reference variable
- new → new operator
- Student() → constructor



x

@1722

Primitive data types:

int, long
float, double
boolean, char

• Operator (Dot)

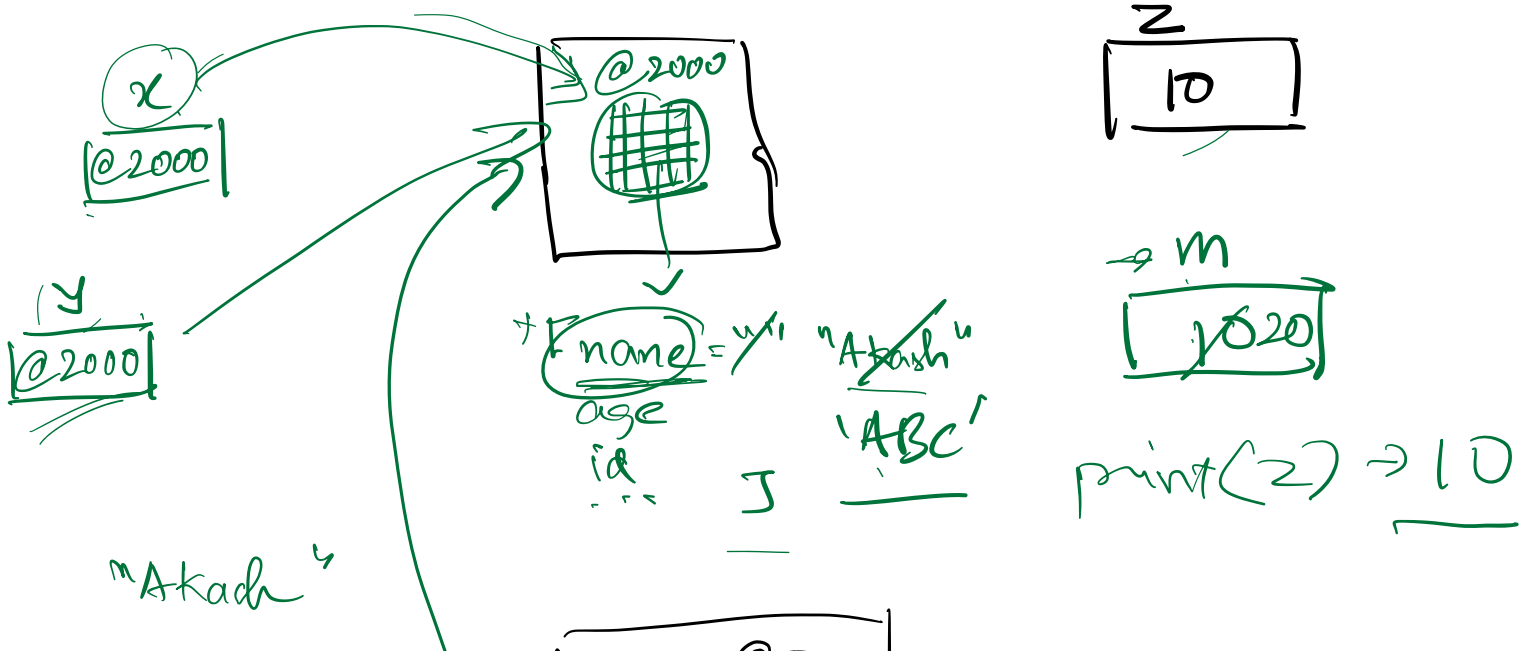
```
} name = "Akash"
  id = 0;
  age = 0;
  branch = "";
}
```

x.name = "Akash";

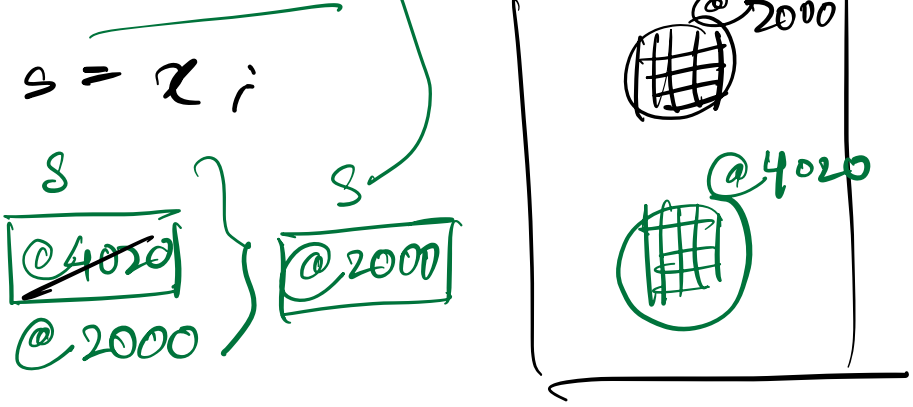
Break till 8:33

```
Student x = new Student();
Student y = x;
y.name = "Akash";
print (x.name);
Student s = new Student()
```

```
int z = 10;
int m = 2;
m = 20;
print (z);
```



print(z) → 10



Access Modifiers :

keywords which restricts accen of the variable

public → can be access anywhere

private → can be accessed only inside that class & no where else.

default → can be accessed everywhere within the package.

protected → can be accessed everywhere within the package
+
outside package can also be accessed only by subclass/childclass

Access Modifier Table :

	Within Same Class	Within same package	Outside the package- (Subclass)	Outside the package- (Global) (without Subclass
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes (only to derrived class)	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No