Programming Paradigms

Style or standard way of writing program.

without → less structured
Hard to read & understand
Hard to test.....

Types

1) Imperative Programming → line by line set of instructions to follow.

```
int a = 2;
int b = 5;
print (a + b);
```

2) Procedural Programming → Splits the program into small procedures / functions which are reusable blocks.

```
addTwoNumbers (a, b) {
    return a + b
}

x = addTwoNumbers (10, 20)
y = addTwoNumbers (30, 40)
```

3) Object Oriented Programming

4) Declarative Programming → Here we specify "what" to do instead of "how" it should be done.

Eg → SQL

```
select * from Users;
```

and so on...

Procedural Programming

Eg → C, C++ etc

```
addTwoNumbers (a, b) {
    return a + b
}

main () {
    int a = 10 , b = 50
    x = addTwoNumbers (a , b )
    y = addTwoNumbers (30, 40)
    print (x)
    ⋮
}
```

Problems

We are studying
Utkarsh is teaching
Karan is enjoying etc.

Sentence → subject + verb
(entities perform action)

```
printStudent ( string name, int age ) {
    print (name)
    print (age)
}
```

combine set of attributes → struct / class

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

1) No methods in some programming languages like Java.
2) All variables are public

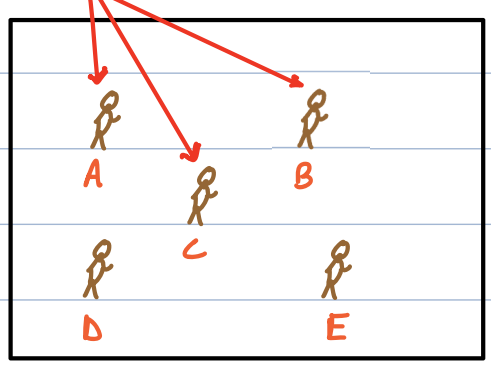
someone → student
something → printStudent()

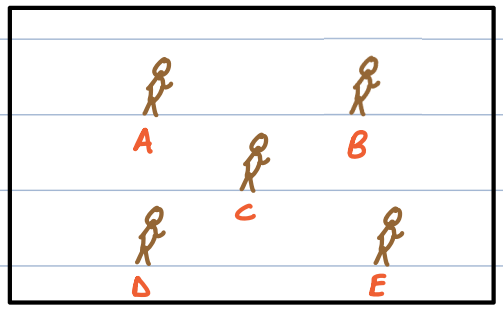
something someone
printSudent (Student st)
action is performed on entity

printSudent (Student st)

OOPS
st. printStudent()
entity is performing action.

Controller (full control)





Object Oriented Programming

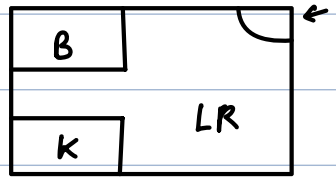
Class → Blueprint of an idea.

Eg → floor plan of an apartment.

Object → Real instance of the class

Eg → actual apartment

occupy memory




```
class Car {
attributes { int model
             String name
             ⋮
methods { drive () {
              ⋮
          }
          turnOnAC () {
              ⋮
          }
          ⋮
}
```

```
// int x = 10;
Car c1 = new Car();
    c1.model = ____
    c1.name = ____
    ⋮
```

each object occupy memory.

```
Car c2 = new Car();
    c2.model = ____
    c2.name = ____
```

Pillars of OOPS

✓ Abstraction → ✓ Principle: Fundamental foundation / concept.

✓ Encapsulation
Inheritance
Polymorphism
} Pillar: Support to hold things together.

Abstraction → Representing in terms of ideas.

Eg → Driving a car

a) Blow horn → press a button

b) Stop the car → press brake

we don't know because its not required.

Encapsulation

capsule

→ hold things together

→ protects the medicine

hold attributes & behaviours // Class

Access Modifiers

1) Public → Accessed by everyone.

2) Private → Cannot be accessed outside class.

3) Protected → Accessed by classed of same package.
+ subclass in different package

4) Default (if nothing specified)

Accessed by classed of same package.

"this" Keyword

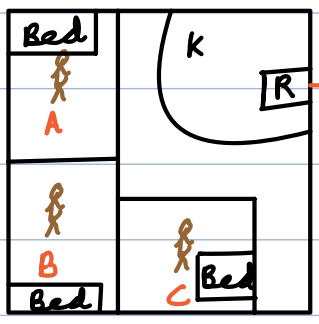
```
class Person {
    private String name;   ①

    Person() {   // constructor → Method with same
        :                        name as class used
    }                            to initialise the object.

    Person(String name) {   ②
        ① this.name = name ②
            refers to current instance of class
    }
}

Person p1 = new Person("Mourika");   ②
print(p1.name)   // Mourika
  ①
```

```
class Person {
    private String name;
    public int id;
    protected int age;
            double height;

    printData() {
```

```
        name ✓          age ✓
        id ✓            height ✓
    }
}

// different package
class Student () extends Person () { // subclass
    printData () {
        Person p1 = new Person ();
        name ✗          age ✓
        id ✓            height ✗
    }
}
```

"static" keyword (Java / C++ etc)

used to declare class level members/variables or methods. (shared among all instances / objects)



→ associated with apartment

Static variables are created when we load a class.

```
class Student {
    static int count = 0; // static variable
    int marks; // instance variable

    Student () {
        count ++; ✓✓✓
```

```
    }
    :
}

Student s1 = new Student();          s1.marks ✓
Student s2 = new Student();
Student s3 = new Student();
print ( Student.count ) // o/p → 3
```

Scope of a variable

1) Class / Static scope → associated with class
2) Instance scope → associated with object
3) Method scope → Variable declared within a method can only be used within same method.
4) Block scope →

```
if (___) {
    int age = ...
}
```

can we use "age" variable → ✗
