Agenda :-

- Programming Paradigms
- Procedural programming
- Object Oriented Programming
- Access Modifiers

Programming Paradigms :-

↓

Style or standard way of writing program.

Without programming paradigm the code will be:

- Less structured
- Hard to read and understand
- Hard to test
- Difficult to maintain, etc.

different types of Programming Paradigms ?

1) Imperative Programming.

- **Imperative Programming** - It tells the computer how to do the 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);
```

2) Procedural Programming

- **Procedural Programming** - It splits the entire program into small procedures or functions (section of code that perform a specific task) which are reusable code blocks.

```
// For eg:
int a = 10;
int b = 20;
addTwoNumbers(a, b);
subtractTwoNumbers(a, b);

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

void subtractTwoNumbers(a, b) {
    int dif = a - b;
    print(dif);
}
```

3) Object Oriented Programming.

↳ classes & Objects.

4) Declarative Programming:-

Declarative Programming - In this paradigm, you specify "what" you want the program to do without specifying "how" it should be done.

e.g, select * from customers;

(Sql)

← Procedural Programming →

Procedure → function (old name)

```
// 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);
}
```

Problems with Procedural Programming.

→ Oman is teaching

→ we all are attending lecture

→ I am having dinner.

SomeOne is doing Something

→ Subject + verb

entities perform action.

```
printStudent(String name, int age, String gender) {
    print(name);
    print(age);
    print(gender);
}
```

Struct or class.

class or

```
// For eg:
struct Student {
    String name;
    int age;
    String gender;
}
```

Student st = ——
PrintStudent(st);

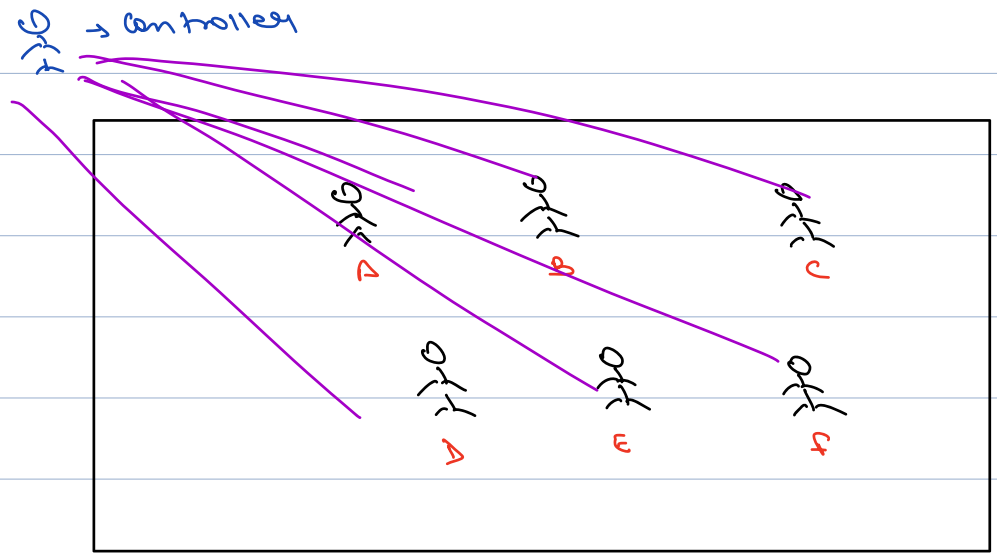
something

```
printStudent(Student st) {
    print (st.name);
    // In some programming languages like C, it's st->name
    print (st.age);
    print (st.gender);
}
```

someone

Something is happening on someone

Procedural	OOP
PrintStudent (Student)	Student . PrintStudent()



So, does the controller person has all info? ---> Yes

So technically any perocedure can play around with entities attributes, and entities have no free will. Its a puppet system. The controller is a puppter holder and each entity is a puppet. This doesn't sound like real world, that is exactly why OOP came into picture.

OOPS :-

```
class student {
    private String name;
    private int age;
    private String gender;

    void print() {
        print (name);
        print (age);
    }
}
```

Student st = ——

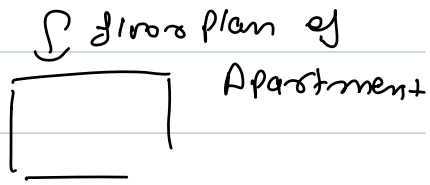
st.print();

Cons of Procedural programming:

- Difficult to make sense
- Difficult to debug and understand
- Spaghetti code i.e. unstructured and needs to be tracked form multiple locations.

Oops :-

class :- Blueprint of an Object.




```
class Student {
    int age;
    String name;
    double psp;
    changeBatch() {-}
    pauseCourse() {-}
    giveMockInterview {-}
}
```

Objects :- They are real instance of class.

They occupy real memory.

e.g.

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

    changeBatch() { batch -> }
    pauseCourse() { - }
    giveMockInterview { - }
}
```

```
Student st1 = new Student();
st1.name = 'Puneet'
st1.age = 10;
```

10k

10k

age = 10;
name = Puneet

```
Student st2 = new Student();
st2.name = 'Vikas Yadav'
st2.age = 20;
```

20k

age = 20
name = 'Vikas Yadav'

← Pillars of OOP →

→ 3 pillars → support to hold things together.

→ 1 principle → fundamental Concept / foundation

Principle :- I will be a good person.

↳ pillars
1) I will be truthful
2) I will do hardwork
3) I respect everyone.

Principle of OOP :-

↳ Abstraction

But how do we implement abstraction?

1) Inheritance
2) Polymorphism
3) Encapsulation.

Source :- Java: The Complete reference.

Abstraction

↳ hiding or privacy.

↳ Representing in terms of ideas.

Scaler → don't need to think about Anshuman Aman etc,

↳ Students, Mentors, TA's etc.

↓

Send msg
Cause pause

So abstraction is an idea of representing complex software system interms of ideas, because ideas are easy to understand.
So, its an concept of making something abstract.

Purpose of Abstraction ?

↳ others don't need to know details of idea.

Abstraction is way to represent complex software system, in terms of ideas.

What needed to be represented in terms of ideas?

- Data
- Anything that has behaviours

← Encapsulation →

What we really store in Programming ?

↳ Attributes & behaviours

↓ variable ↓ function

To protect attributes & methods from outer environment i.e, other classes don't have access to it.

Access Modifiers

We got to know that Encapsulation has two advantages,

- ONE is it holds data and attributes together and → class
- SECOND is it protect members from illegitimate access. You can't access the data from class unless the class allows you to.

→ access modifier.

Generally 4 access modifiers :-

1) Public
2) Private
3) Protected
4) default

```
class Student {
    Private String name;
    public int age;
    Protected double PSP;
    String Batchname;
}
```

1) Public access Modifier :-

A public attribute or method can be accessed by everyone.

2) Private access Modifier :-

A private attribute or method can be accessed by no one, not even the child class.

3) Protected access Modifier :-

A protected attribute or method can be accessed only from the classes of the same package.

& subclasses of a different package.

```
Package a;

class xyz {
        protected int temp;
}

class test2 {
}
```

```
Package b;

class test inherits xyz {

    access temp;
}
```

4) default access modifier (within package)

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
`public`	+	+	+	+	+
`protected`	+	+	+	+	
no modifier	+	+	+		
`private`	+				

'This' Keyword → current Object.

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name; // "this" refers to the current instance of the class
    }

    public void introduceYourself() {
        System.out.println("Hello, I am " + this.name); // Using "this" to access the instance variable
    }

    public static void main(String[] args) {
        Person person1 = new Person("Alice");
        Person person2 = new Person("Bob");

        person1.introduceYourself(); // Output: Hello, I am Alice
        person2.introduceYourself(); // Output: Hello, I am Bob
    }
}
```

```
class Person {
    String name;
    void changeName (String name) {
        this.name = name;
    }
}
```

3

3

```
Person p = new Person();
p.changeName("Vikas");
Person p2 = new Person();
p2.changeName("Puneet");
```

p → 10k

10k: name = Vikas

p2 → 20k

20k: name = 'Puneet'

-: Static keyword :-

The **static** keyword in programming languages like Java and C++ is used to declare **class-level members or methods**, which are associated with the class itself rather than with instances (objects) of the class.

Static Variables (Class Variables): When you declare a variable as "static" within a class, it becomes a class variable. These variables are shared among all instances of the class. They are initialized only once when the class is loaded, and their values are common to all objects of the class.

Static Methods (Class Methods): When you declare a method as "static," it becomes a class method. These methods are invoked on the class itself, not on instances of the class. They can access static variables and perform operations that don't require access to instance-specific data.

```
Public class Myclass {
    static int staticVar = 0;
    int instanceVariable;
    public Myclass() {
        instanceVariable = 0;
    }
    Test() {
        instanceVariable ++
        staticVar++;
    }
    static Test2() {
```

static Var = ~~0~~ 1;

1ok

inst var = ~~0~~ 1

```
Myclass t1 = new Myclass();
print (t1.instvar); -> 0
print (t1.static var); -> 0
t1.Test();
print (t1.staticVar); -> 1
```

2ok;

inst var = 0

```
Myclass t2 = new Myclass();
print (t2.static var) -> 1
print (Myclass.static var);
-> Student.Test2();
```

```
public Test () {

    public static void Main() {

        swap();
    }

    Public static void swap() {

    }

}
```

Scope of a Variable :-

1) Class / Static Scope :-

Class/Static Scope: Variables declared as `static` within a class have class-level scope. These variables are associated with the class itself rather than with instances (objects) of the class. They can be accessed using the class name and are shared among all instances of the class.

2) Instance Scope :-

. **Instance Scope:** Variables declared within a class but outside any method or constructor have instance scope. These are often referred to as instance variables, and they are associated with specific instances (objects) of the class. Each object has its own copy of these variables.

3) Method / Local Scope :-

Method/Local Scope: Variables declared within a method or a block of code have method or local scope. These variables are only accessible within the specific method or block where they are defined. They go out of scope when the method or block's execution is complete.

4) Block Scope :-

Block Scope: Variables declared within a pair of curly braces `{}` have scope limited to that block. These variables are only accessible within the block in which they are defined.

```
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.
    }
```

```
    public static void main(String[] args) {
        ScopeExample obj = new ScopeExample();
        obj.exampleMethod();
        // The 'methodVar' and 'blockVar' are out of scope here.
    }
}
```

e.g

$$\begin{array}{cc} & 2 \\ 0 & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \end{array}$$

0	1	2
1	4	7

row

0	1	2
7	8	9

```
for (i = 0; i < m; i++) {
    for ( j = 0; j < m; j++) {
        (mat[i] = = mat[j])
```

2

0

0 0

0 1

→ 2 0

3 3 3

```java
public class Solution {
    public int[] solve(int[][] A) {
        int n = A.length, m = A[0].length;

        final int inf = 1000000000 + 7;
        int []mi = new int[n], mx = new int[m];
        for (int i = 0; i < n; i++){
            mi[i] = inf;
        }
        for(int i = 0; i < m; i++){
            mx[i] = -inf;
        }

        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                mi[i] = Math.min(mi[i], A[i][j]);
                mx[j] = Math.max(mx[j], A[i][j]);
            }
        }

        ArrayList<Integer> res = new ArrayList<>();
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(mi[i] == mx[j]){
                    res.add(mi[i]);
                    break;
                }
            }
        }

        Collections.sort(res);
        int si = res.size();
        int []ans = new int[si];

        for(int i = 0; i < si; i++){
            ans[i] = res.get(i);
        }

        return ans;
    }
}
```