Programming Paradigms

Standard way of writing a program.

Types → 1) Imperitive Programming → It tells the computer how to do a task by giving set of instructions in a particular order.

Eg →
```
a = 10
b = 5
c = a + b
```

2) Procedural Programming → It splits the program into small procedures / functions.

Eg →
```
add (x, y) {        // reusable
    print (x + y)
}
```

3) Object Oriented Programming

4) Declarative Programming → Specify 'what' to do instead of 'how' it should be done.

Eg → SQL
```
select * from Users ;
```

Procedural Programming

Execution starts from → main ()

Problems with Procedural Programming

We are studying
Utkarsh is teaching
subject + verb (someone is doing something)
(entities perform action)

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

Way to combine attributes
Structure / class.

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

Eg → Java

Difference b/w struct & class →

1) A struct has no methods.
(in C++, struct can have methods)

2) All variables are public in struct. → privacy issue
(in C++, it can be made private but default is public)

---

action entity

```
printStudent (Student s1) {
    print (s1.name)
    print (s1.age)
}
```

something is happening to someone/
action is being performed on entity.

`printStudent (Student s1);` | `s1.printStudent();`

OOPS → Entity perform action.

Class → Blueprint of an idea
Eg → Floor plan of an apartment.
Does not occupy space in memory.

Object → Real instance of a class
Eg → Actual apartment.
Can create multiple objects of the same class.

Pillars of Object Oriented Programming

✓ Abstraction ← Principle (fundamental foundation)

Inheritance
Polymorphism } Pillar (support to hold things together)

✓ Encapsulation

Abstraction

What is the purpose of abstraction?

The main purpose is that others dont need to know the details of the idea.

Suppose you are driving a car, and you want it to turn left and speed up, and you steer the steering to left and press the acceleration pedal. It works right? Do you need to know how does this happen? What combustion is happening, how much fuel is used, How steering wheel turned the car?
No right? This is what we call as abstraction.

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

No one else need to know the details about the ideas.

Encapsulation

If the capsule breaks away, what will happen?

- It will flow away. So first purpose is to hold the medicine powder together.
- Then there are multiple powders are present in the capsule, it helps them to avoid mixing with each other.
- Third purpose is it protects the medicine from the outside environment.

This is exactly the purpose of Encapsulation in OOP.

Access Modifiers

Types of Access Modifiers

There are four access modifiers in most of the programming languages, they are:

- **Public**
- **Private**
- **Protected**
- **Default** (if we don't use any of the above three, then its the default one)

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

`this` Keyword

- In programming, "this" is a keyword used to refer to the **current instance of a class or object**.
- It's typically used to distinguish between instance variables and local variables or method parameters with the same names, and to access or modify instance members within methods.
- This enhances code clarity and prevents naming conflicts in object-oriented languages like Java and C++.

```
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
    }
}
```

Example of Access Modifiers

```
package mypackage;

public class AccessModifierExample {
    public int publicVariable = 10; // Public access

    private int privateVariable = 20; // Private access

    protected int protectedVariable = 30; // Protected access

    int defaultVariable = 40; // Default (package-private) access

    public void publicMethod() {
        System.out.println("This is a public method.");
    }

    private void privateMethod() {
        System.out.println("This is a private method.");
    }
```

```
    public static void main(String[] args) {
        AccessModifierExample example = new AccessModifierExample();

        System.out.println("Public variable: " + example.publicVariable);
        System.out.println("Private variable: " + example.privateVariable);
        System.out.println("Protected variable: " + example.protectedVariable);
        System.out.println("Default variable: " + example.defaultVariable);
    }
}
```

```
package otherpackage;

import mypackage.AccessModifierExample; // Import the class from a different package

public class AnotherClass {
    public static void main(String[] args) {
        AccessModifierExample example = new AccessModifierExample();

        System.out.println(example.publicVariable); // Accessing publicVariable is valid
        System.out.println(example.defaultVariable); // Error: Cannot access defaultVariable from a different
package

        example.publicMethod();
        example.privateMethod(); // Error: Private method is not accessible outside the class
    }
}
```

`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.

1. **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.
2. **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.

The **static** keyword is often used for utility methods and constants that are relevant at the class level rather than the instance level. It allows you to access these members without creating an object of the class.

Static variable is created when we load a class.

```
public class MyClass {
    // Static variable
    static int staticVar = 0;

    // Instance variable
    int instanceVar;

    public MyClass(int value) {
        this.instanceVar = value;
        staticVar++;
    }
```

```
    public static void main(String[] args) {
        MyClass obj1 = new MyClass(10);
        MyClass obj2 = new MyClass(20);

        System.out.println("Static Variable: " + staticVar); // Output: Static Variable: 2
        System.out.println("Instance Variable (obj1): " + obj1.instanceVar); // Output: Instance Variable
(obj1): 10
        System.out.println("Instance Variable (obj2): " + obj2.instanceVar); // Output: Instance Variable
(obj2): 20
    }
}
```

Constructor

Class → Blueprint of entity.

Object → Instance of class.

```
class Student {
    String name;
    int age;
    double psp;
}
```

```
Student st = new Student();
int a = 10;
```

Default constructor creates the object of the class & set default values of the attributes.

Eg → int = 0,
String = null,
double = 0.0,
boolean = false, etc.

```
class Student {
    String name;
    int age;
    double psp;
    String univName;

    Student() {
        name = null;
        age = 0;
        psp = 0.0;
        univName = null;
    }
}
```

No return type.
Same name as the class.
Its public. (default constructor)

```
public class Student {
    String name;
    private int age = 21;
    String univName;
    double psp;

    public Student (String studentName, String universityName) {
        name = studentName;
        univName = universityName;
    }
}
```

psp = 0.0

```
public class Client {
    public static void main(String[] args) {
        Student st = new Student(); //ERROR
}
```

No default constructor if we have manual constructor.

```
public class Client {
public static void main(String[] args) {
    Student st = new Student("Utkarsh", "JIIT");
}
```

✓

Copy Constructor

Used to create copy of an existing object.

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

    Student() {
        name = null;
        age = 0;
    }
    Student(Student st) {
        name = st.name;
        age = st.age;
    }
}
```

copy constructor

```
Student st1 = new Student();
st1.name = "Utkarsh";
st1.age = 27;

Student st2 = new Student(st1); // Copy Constructor
```

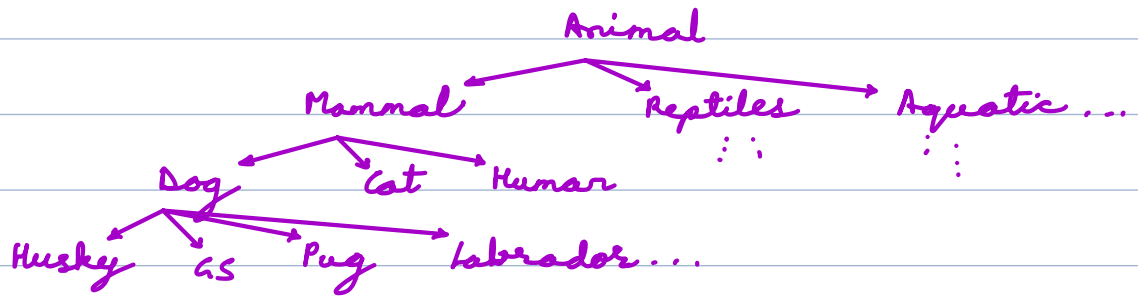
New object is created.

// Object reference for same object (no new object).

```
Student st2 = new Student (st1);   // Deep Copy
st2.name = "Bharath"
print (st1.name)   // Utkarsh

Student st3 = st1;   // Shallow Copy
st3.name = "Rakesh"
print (st1.name)   // Rakesh
```

Inheritance

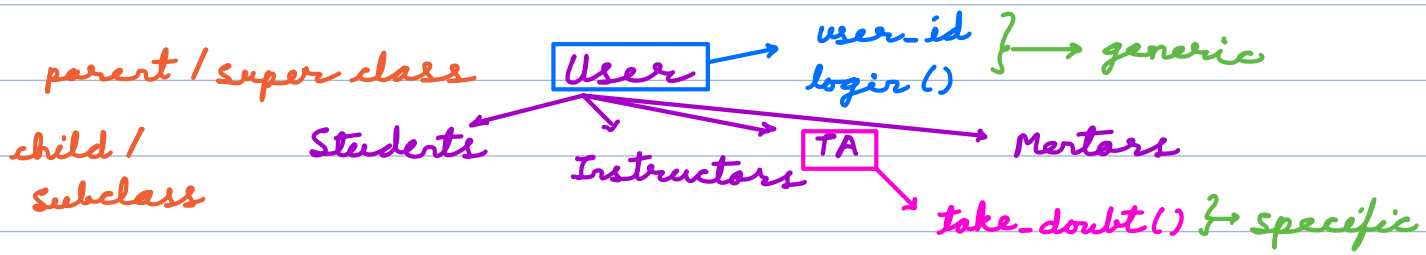


Animals can move ✓
⇒ Mammal can move
⇒ Dog can move
⇒ Pug can move

Representation of hierarchies in classes is inheritance.

parent → child

parent / super class
child / subclass



A child class inherits all the members of the parent class & may or may not add their own members. ✓

```
class User {
    String userName;

    void login() {
        ...
    }
}
```

Java → class Instructor extends User

Python → class Instructor (User)

C++ → class Instructor : public User

C# → class Instructor : User

⋮

child parent

```
class Instructor extends User {
    String batchName;
    double avgRating;

    void scheduleClass() {
        ...
    }
}
```

Constructor chaining

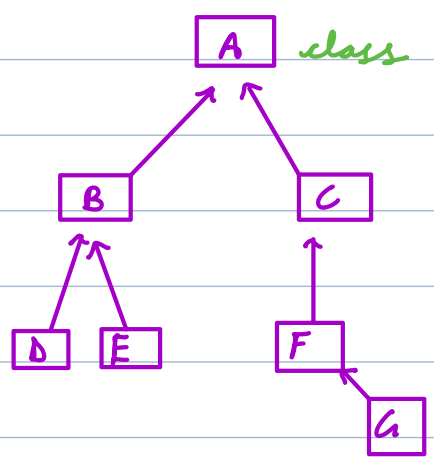
Instructor i = new Instructor();

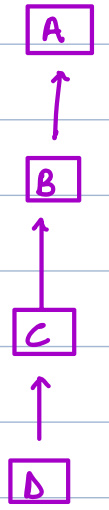
i.avgRating = 4.9;

i.userName = "Utkarsh";

How is userName initialized?

Using constructor of parent class which is called within constructor of child class.





D d = new D();

constructor of D is called — completion 4

⇒ calls constructor of C — 3

⇒ calls constructor of B — 2

⇒ calls constructor of A — 1

```
public class C extends B {
    C() {
        System.out.println("Constructor of C");
    }
    C(String a) {
        System.out.println("Constructor of C with params");
    }
}
```

D d = new D();

```
public class D extends C {
    D() {
        super ("Hello"); // This must be the first line
        System.out.println("Constructor of D");
    }
}
```

calls parent class constructor

Polymorphism

many - forms

Animal a = new Dog(); ✓

Dog d = new Animal(); ✗

```
class A {
    int age;
    String name;
}
```

```
class B extends A {
    String uni;
}
```

```
class C extends A {
    double psp;
}
```

```
A a = new C();
a.psp = 50.0;  // gives error
```

HW → [typecast to C & use attributes of class C.]

How to access attributes of C.

Types of Polymorphism

1) Compile Time → Method overloading

(methods with same name but different parameters.)

```
int add(x, y) {
    return x+y;
}
int add(x, y, z) {
    return x+y+z;
}
```

count / datatype / order ✓

method signature

2) Run Time → Method overriding

```
Class A {
    void doSomething(String a) {
        ...
    }
}
```

String

```
Class B extends A {
    String doSomething(String c) {
        ...
    }
}
```

// overriding parent class function