OOPS

OBJECT ORIENTED PROGRAMMING

Classes and Objects

entities in the real world

group of these entities

class → attributes (properties) + functions (behaviour)

Eg:

```
class Pen {

    String color;
    int tip;

    void setColor (String newColor) {
        color = newColor;
    }

    void setTip (int newTip) {
        tip = newTip;
    }
}

main {
    Pen p1 = new Pen();
    p1.color = "Blue";
}
```

• → dot operator used

Access Modifiers :

AM :	within class	within package	outside package by subclass only	outside package
PRIVATE	Y	N	N	N
DEFAULT	Y	Y	N	N
PROTECTED	Y	Y	Y	N
PUBLIC	Y	Y	Y	Y

Eg :

```
class BankAccount {

    public String username;
    private String password;

    default void setPw (String pwd){
        password = pwd;
    }

}
```

Getters and Setters :

- Getters → to return the value
- Setters → to modify the value

this keyword is used to refer to the current object.

Encapsulation:

↳ Encapsulation is defined as the wrapping up of data and methods under a single unit. It also implements data hiding.

Constructors:

↳ Constructor is a special method which is invoked automatically at the time of object creation.

→ Constructors have the same name as class.

→ Constructors dont have a return type (not even void)

→ constructors are only called once, at object creation.

→ Memory allocation happens when constructor is called.

TYPES OF CONSTRUCTORS:

1. Non parameterized: takes no parameter.
2. parameterized: takes parameter.
3. Copy constructor: used to copy properties of object 1 into object 2.

Eg code :

```
class Student (){
    String name;
    int rollno;
    String pwd;

    Student (){
        Syso("Non parameterized constr is called");
    }                                          //npc

    Student (String name, int rollno){
        this.name = name;
        this.rollno = rollno;                  //pc
    }

    Student (Student s1){
        this.name = s1.name;
        this.rollno = s1.rollno;               //cc
    }
}

main {
    Student s1 = new Student();
    Student s2 = new Student("Shubh", 3290);
    Student s3 = new Student(s2);
}
```

Shallow copy & Deep Copy :

- reference copy
- change in obj1 will reflect in obj2

- new structure is created
- change in obj1 will not reflect in obj2.

// Shallow copy constructor :

```
Student (Student s2){
    marks = new int[3];
    this.marks = s1.marks;
}
```

// Deep Copy constructor :

```
Student (Student s1){
    marks = new int[3];
    for(int i=0; i<marks.length; i++){
        this.marks[i] = s1.marks[i];
    }
}
```

Destructor :

- Garbage collector

Java does the job for you!!

Inheritance :

- Inheritance is when properties and methods of base class are passed onto a derived class.

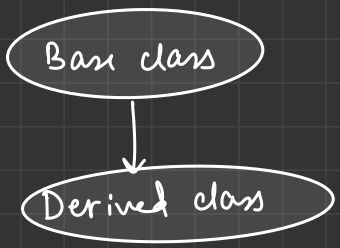
Eg:

```
class Animal{
    // Base class
}
```

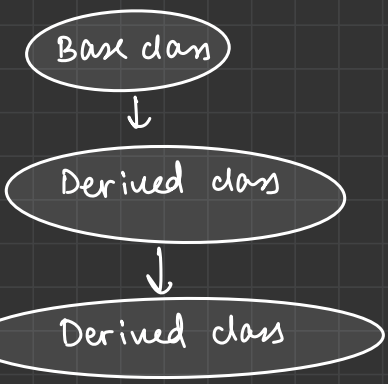
```
class Fish extends Animal{
    //Derived class
}
```

Types of Inheritance :

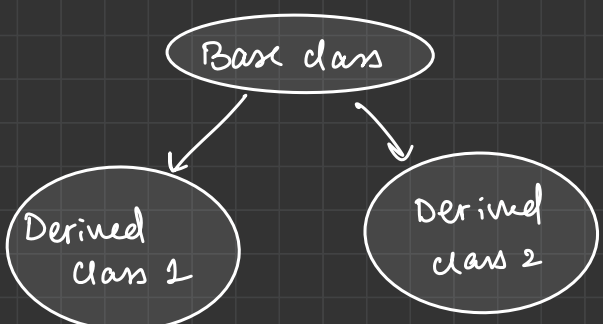
① Single level Inheritance



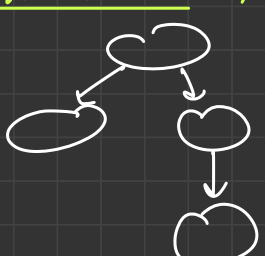
② Multi-level Inheritance :



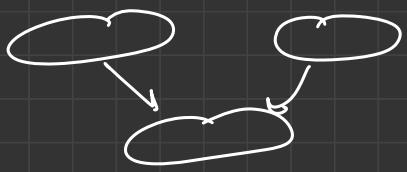
③ Hierarchial Inheritance :



④ Hybrid Inheritance : combination of mix.



(3) Multiple Inheritance : by interfaces in Java.



Polymorphism :

(1) Compile Time polymorphism (static)
↳ method overloading

(2) Run Time polymorphism (dynamic)
↳ method overriding

Method Overloading :

→ Multiple functions with the same name but different parameters.

Eg: Calculator {

sum (int a, int b)
sum (float a, float b)
sum (int a, int b, int c)

}

Method Overriding

Parent and child classes both contain the same function with a different definition.

```
class Animal {
    void eat() {
        syso("Eats Anything");
    }
}

class Deer extends Animal {
    void eat() {
        syso("Eats grass");
    }
}

main() {
    Deer d1 = new Deer();
    Syso(d1.eat());
}
```

→ Eats grass // output

Packages in JAVA :

↳ Package is a group of similar types of classes, interfaces and sub-packages.

- In-built → `java.util.Scanner;`
- User-defined → `package myPackage;`

Abstraction :

↳ Hiding all the unnecessary details and showing only the important parts to the user.

- Abstract classes
- Interfaces

(1) Abstract Class :

→ cannot create an instance of abstract class

→ can have abstract / non-abstract methods

→ can have constructors

Eg:

```
abstract class Animal {          // → cannot create obj of this class
    void eat() {
        syso("eats");
    }
    abstract void walk();        // no implementation
}
```

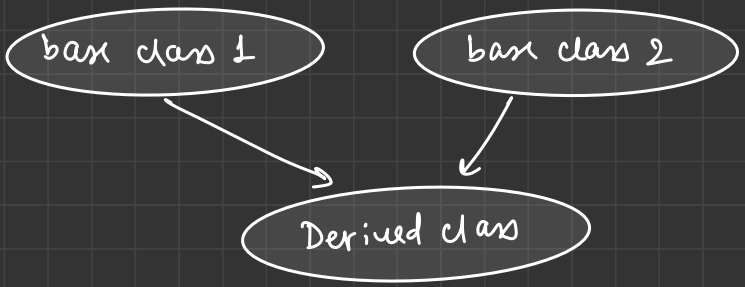
```
class Horse extends Animal {
    void walk() {                          // compulsory to implement
        syso("walks");
    }
}

class chicken extends Animal {
    void walk() {                          // compulsory to implement.
        syso("chicken walks");
    }
}
```

⇒ when a child class object is initialized → first its parent class's constructor is called then its own constructor

(2) Interfaces :

↳ Interface is a blueprint of a class.

→ multiple interface :



→ total abstraction : 100 %

Class → extends

Interface → implements

→ All methods are public, abstract & w/o implementation
→ used to achieve total abstraction.
→ Variables in the interface are final, public and static.

Eg:

```
interface ChessPlayer {
    void moves();
}

class Rook implements ChessPlayer {
    public void moves() {
        System.out.println("up, down, left, right");
    }
}
```

Eg 2 : multiple inheritance :

```
interface Herbivore {
    void grass();
}

interface Carnivore {
    void meat();
}
```

```
class Bear implements Herbivore, Carnivore {

    public void grass() {
        syso ("Eats grass");
    }
    public void meat() {
        syso ("eats meat");
    }
}
```

Static Keyword :

↳ Static keyword in Java is used to share the same variable or method of a given class.

- → properties can be static
- → functions can be static
- → Blocks can be static
- → Nested classes can be static

Super Keyword :

↳ super keyword is used to refer immediate parent class object.

- → to access parent's properties
- → to access parent's functions
- → to access parent's constructor

Eg:

```
class Animal {
    Animal() {
        syso("Animal constructor");
    }
}

class Horse extends Animal {
    Horse() {
        super();
        syso("Horse constructor");
    }
}
```

⇒ In Java, no class can be private or protected

JAVA :	private	default	protected	public
Class	No	Yes	No	Yes
Nested class	Yes	Yes	Yes	Yes
Constructor	Yes	Yes	Yes	Yes
method	Yes	Yes	Yes	Yes
Field	Yes	Yes	Yes	Yes

⇒ child object ko parent reference se bana sakta hai.

Vehicle ↓ Car ≡ Car c = new Car(); ✓
Vehicle V = new Vehicle(); ✓
Vehicle V = new Car(); ✓

Constructor Chaining:

↳ sequence of invoking constructors upon initializing an object. It is used when we want to invoke a no. of constructors, one after the another by using only an instance.

- same class → use this keyword
- from base class → use super keyword

Eg 1:

```
class ConstruvborChain {

    ConstructorChain() {
        this ("Javatpoint");
        Syso(" Default constructor");
    }

    Constructorchain (Strig str) {
        syso ("Parameterized");
    }
}

main {
ConstructorChain cc = new Construvchain();

}
```

//output : Parameterized
Default Constructor

Eg 2 :

```
Class Demo{

    Demo(){
      this(80,90);
      Syso("Default Constructor Parent");
    }

    Demo(int x, int y){
        syso("Parameterized Parent");
    }
}

class Prototype extends Demo{

    Prototype(){
        this("Java","Python");
        Syso("Default constructor child");
    }

    Prototype(String str1, String str2){
        super();
        syso("Parameterized child");
    }

}
```

```
main {
    Prototype p1 = new Prototype();
}
```

// output: Parameterized parent
Default constructor parent
parameterized child
Default constructor child