OOPS

* Atmost 1 public class is allowed in Java (0 or 1 public class)

Object → entities in real world

Class → blueprint to create object

Classes → attributes (properties)
↓
functions (behaviour)

objects are created in heap memory

Access modifier

→ Private → default → Protected → Public

* In Java protected & private class all unusable (object can't be created)

Getters → returns value → Nested class can be private

Setters → Sets the value (modify)

* this → keyword having current object

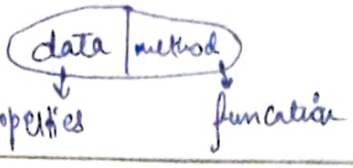
Pillars of OOPS

(i) Encapsulation (ii) Inheritance (iii) Abstraction

(iv) Polymorphism

→ Encapsulation :-

defined as wrapping up of data & methods under a single unit. It also implements data hiding.



__/__/____

useless / sensitive → private / protected {Data hiding}

→ **Constructor:**
is a special method which is invoked automatically at time of object creation

- constructor has same name as of class
- don't have a return type (not even void)
- Constructors are only called once at object creation
- Memory ~~also~~ allocation happens when constructor is called

* Whenever constructor is not created by user, Java automatically creates its constructor
→ but constructor created by Java has no initialization

* **3 types of Constructors**
→ Non-parameterized
→ Parameterized
→ Copy Constructor

→ Copy constructor → copying function of an object

* Shallow & Deep Copying
↓ ↓
References new memory is allocated

* In shallow copy changes reflect
* In deep copy changes don't reflect as new memory is allocated

Destructor:

In Java we don't write destructors bez garabage collection automatically delets all the non usable variable etc.

* Inheritance:

Inheritance is when properties & methods of base class are passed on to derived class.

Example

```
Animal {
    eat()
    breathe()
}
```

Parent class / base

```
Fish {
    Animal class
        +
    other properties
}
```

Derived class

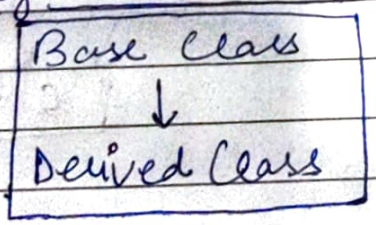
Code ↓

```
Class Animal {        // Base class
    String color;
    void eat() {
        Sout ("eats");
    }
    void breathe() {
        sout ("breathes");
    }
}

Class Fish extends Animal {      // Derived class
                ↑ keyword
    int fins;
    void swim () {
        Sout ("swims in water")
    }
```

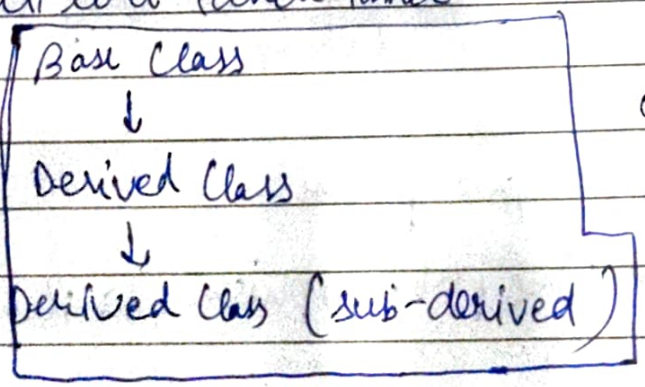
* Type of inheritance :

→ Single level inheritance:



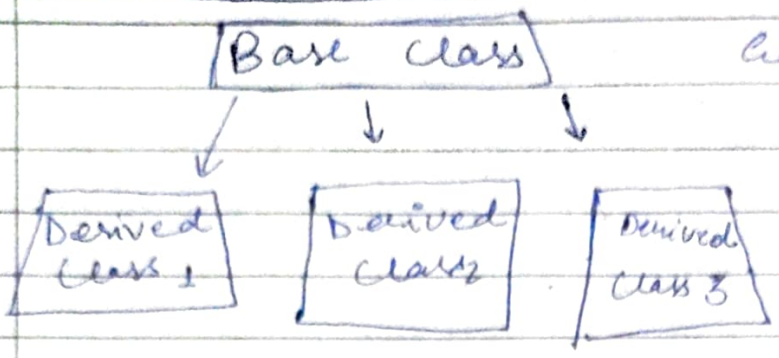

```
Class Mammal extends Animal {
    int legs
}
```

→ Multi level inheritance




```
class Dog extends Mammal {
    String breed;
}
```

→ Hierarchial Inheritance →



eg cat() breathe()
swim() flyll walk()

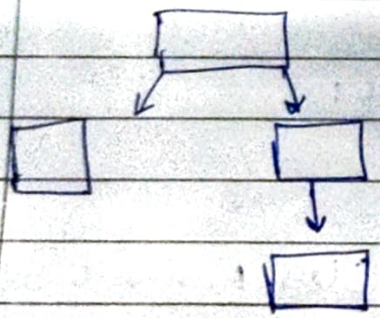
Class fish extends Animal {	Class Bird extend Animal {	Mammal Class the extend Animal {
void swim	void fly	void walk
}	}	}

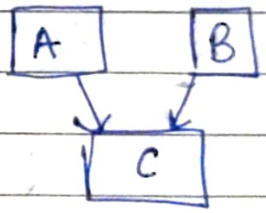
→ Class vehicle { } that are properties of vehicle *

Class car extends vehicle { } → vehicle v = new Car() → True

→ Car v = new vehicle → false

→ Hybrid Inheritance: (combination). → Multiple Inheritance (Not in Java, in C++)





many

Polymorphism: We try to same thing in multiple forms.

forms

- Type of polymorphism

→ Compile time polymorphism (static) → Runtime polymorphism (Dynamic)

→ Method overloading

- Multiple funcations with same name but diffe parametres.

```
Calculator {
sum(int a, b)
sum(~~int~~ float a, b)
sum(int a, b, c)
}
```

* either datatype of parametre are different or number of parametre are different

* Method overloading is compile time overloading

→ Method Overriding

- Runtime polymorphism
- Parent & child class both contain same funcation with a different defination.

```
Class Animal {
    Void eat() {
        Sout("Eats Anything")
    }
}

Class Deer extends Animal {
    Void eat() {
        S.out("eats grass");
    }
}
```

```
psvm {
Deer d = new Deer()
d.eat();
output → eats grass
}
```

* Packages

→ Is a group of similar types of classes, interfaces and sub-packages.

* Inbuilt package (eg → Scanner object is from java.util)
* User defined

* Abstraction:

- Hiding all the unnecessary details and showing only the important part to the user.

* Data ~~hiding~~ hiding but important things anthaon

→ Implement by 2 ~~no~~ ways (i) Abstract classes (ii) Interfaces

→ Abstract classes:
- Cannot create an instance of abstract class (object)
-
- Can have abstract & non abstract function
- Can create constructors

```
abstract class Animal {
    void eat() {
        Sout("eats")
    }
    abstract void walk();
}
```

(* Since it is abstract it doesn't have implementation)

```
class Horse extends Animal {
    void walk() {
        Sout("On 4 legs")
    }
}
```

* until walk method is not created it will show error. It is compulsory to make walk method.

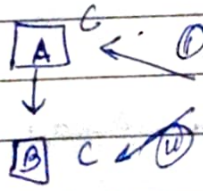
* walk doesn't depend on parent class it depends on child class horse.

* Abstract class provides idea not implementation.

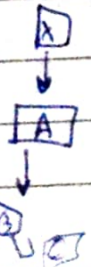
→ Animal a = new Animal();

* this will not work as we cannot make objects of abstract class.

→ Constructor doesn't only initialise its variable but also variables for child classes.



→ first ~~Abstract~~ A's constructor will be called the B's

→ 

→ first A constructor then A → B → C

* Interface

- Interface is a blueprint of class.

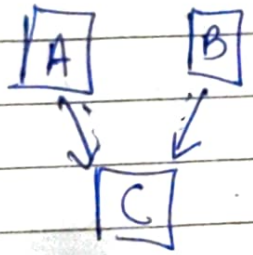
Car [wheel, speed, engine] {Interface}

Blueprint ↑ ↓

Maruti 800 {Class}

Blueprint → Car 1 (M 800) Car 2 (M 800) Car 3 (M 800)

- for multiple inheritance we use interface



- To achieve total abstraction

- Interface → implement , Class → extend

→ All method are public, abstract & without implementation
→ Used to Achieve total abstraction
→ Variable in Interface are final, public & static

__/__/____

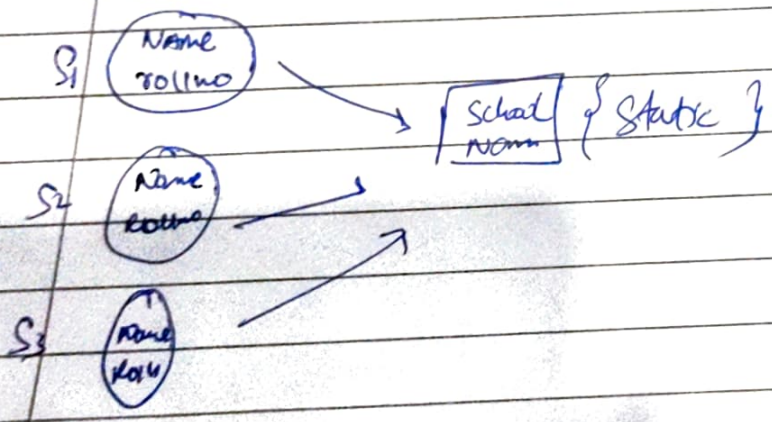
* Static (keyword) :-

- Keyword in java is used to share the same variable/method of a given class.

→ properties → function → block → Nested classes can be made static.

→ Change in one object also change other objects

In Heap :



Eg → main function is made static

* Super keyword

Super keyword is used to refer immediate parent class object