OOP's terminology

1) Inheritance.
2) Polymorphism.
3) Encapsulation.
4) Abstraction.

1) Inheritance :-

- A subclass / child class inherits / acquires prop. from the super class / parent class this is known as inheritance.

- The creation of class by deriving it from another class (super class)

  Rikshaw ⇒ E-rikshaw

  Phone ⇒ Smartphone.

Like for creating a smartphone we don't need to start from scratch we will just inherit / absorb / use the properties of phone class and add properties of smartphone class.

super / parent class

```
class Phone{
.....
}
```

derived / child class.

```
class Smartphone extends Phone{
.....
}
```

"extends" keyword is used to inherite the properties from another class / super class.

- Every class has 'Object' as their superclass.
- child class can access everything from parent class but parent class do not access anything from the child class.

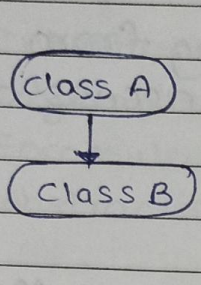
- Super keyword : this keyword is used to reffer the parent class.

e.g.
```
class A{
  int number = 10;
}

class B extends A{
  int number = 50;

  PVSM(String[]args){
    sout(super.number); // this will reffer to its parent class number value.
```

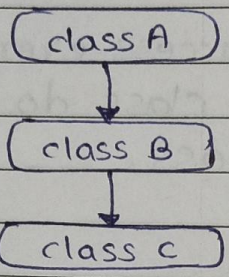
- Using super(); calls the constructor of the parent class.

! In static methods 'super' keyword do not work or can't be used.

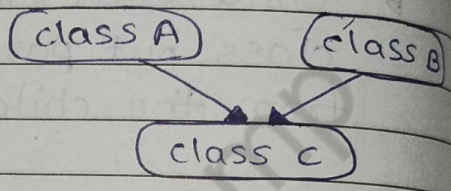
✱ Types of inheritance.



Single inheritance

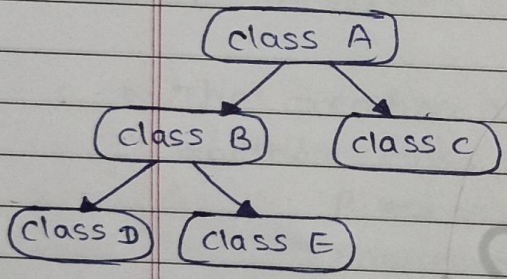


Multi-Level inheritance

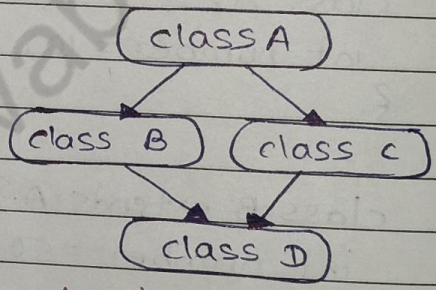


Multiple Inheritance.

(! Not allowed in java)



Hierarchical Inheritance

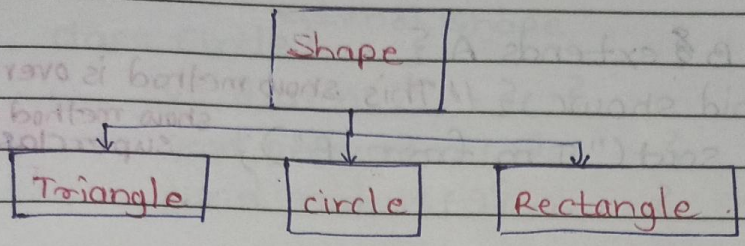


Hybrid inheritance
(Multiple + Hierarchical)
(! Not allowed in Java)

2> Polymorphism

Polymorphism
- many ← ways to represent/structure

▲ This is achieved by
1. Method overriding
2. Method overloading.



⊛ Types of polymorphism.

1) Compile time / static polymorphism.
Achieved via method overloading.

Same name but types, arguments, return types, ordering can be different.

Ex. Multiple constructors.

```
A ao = new A();
A ab = new A(3,4);
```

! Which method to run is checked at compile time so this is called as compile time poly.

2) Runtime / Dynamic Polymorphism.
Achieved by method overriding.

Name, arguments, return type, order everything is same but the only body of the method is diff.

Ex.
```
class A{
    void show(){
        sout("I'm from A");
    }
}
```

static methods can't be overridden bec they loads when class get loaded. They can be inherited only.

This will load when object for B ~~meth~~ class is created.

Hence this will override parent method.

```
class B extends A {
    void show() { // This show method is overriding show method of his superclass
        sout("I'm from B");
    }
}
```

working of overriding !!!

```
Parent obj = new Child();
```

Here, which method will be called depends on

This is known as upcasting

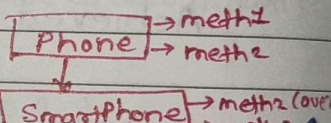
Dynamic method dispatch.

eg. senario 1

```
Phone obj = new SmartPhone();
```

senario 2 // not allow

```
SmartPhone obj = new Phone()
```




```
obj.meth2();
obj.meth3(); // not allow
```

```
Parent obj = new Child();
```

Reff taken

obj which will be called.

This tells us which ~~method~~ should be accessed.

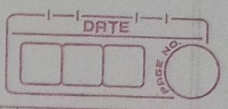
~~This~~ Java determines which version of the method to call based on the type of the object at the time this call happens. Hence this determination is done at run time.

! If we are reffrancing from parent class to the method of child then the method which is being overridden should be present in super class.

eg.

```
class Shape {
    void area() {....}
    void perimeter() {....}
}
```

!!!! By def every class in java extends object class.




```
class Circle extends Shape {
    void area(){.....}
    void perimeter(){.....}
    void radius(){.....}
}
```

Here if we do reff of shape to circle obj.

```
Shape obj = new Circle();
obj.area(); // valid
obj.perimeter(); // valid
obj.radius(); // invalid
```

bec radius(); is not in parent class/shape. As we are reffrencing it from parent class that must be in parent class also.

```
Circle c0 = new Circle();
c0.radius(); // now this is valid.
```

* Final Keyword.

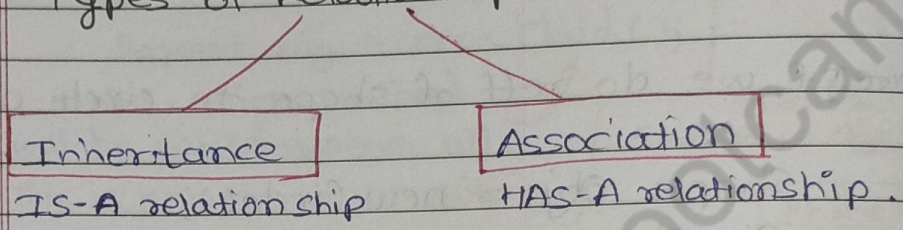
! This can be used to prevent methods overridden. final methods can't be overridden. but it can be overloaded. It will help to improve performance

! If the class is final then by defaultt all the methods in that class becomes final

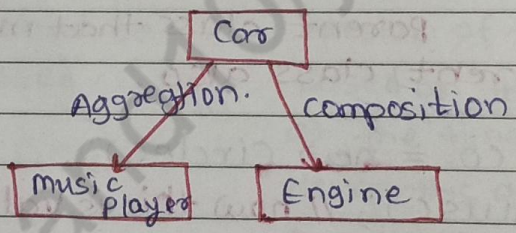
- When it is used to declare the data member of a class it becomes cons.
e.g. It's value can not be change hence value must be provided at the time of declaration.

6/12/2021

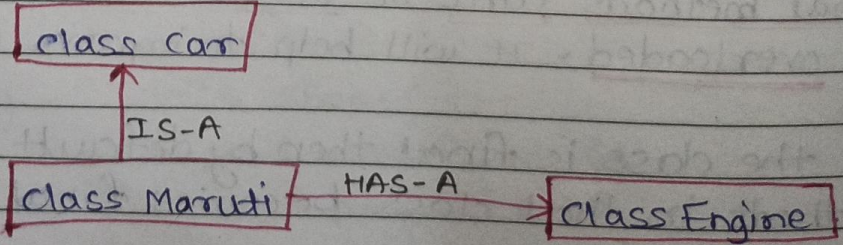
Types of Relationship.



* Association: (HAS-A rel^n)
Have two parts :-
1> Aggregation weak bond.
2> Composition. strong bond.



* Inheritance: (IS-A rel^n)
- It is based on inheritance.
- e.g. Apple is a fruit, Car is a vehicle, etc.



* Abstraction
- Hiding unnecessary details and sharing valuable information/required info only.
E.g. We don't need to know mechanics of the car to run the car.

- It is a process of gaining the information and hiding unwanted info

* Encapsulation:
▷ Wrapping different things in a single unit.
▷ Like class A{
variables
methods
}
Here we have wrapped different variables and methods in a class A
▷ It is used beccuse it provides security like no other object from other class can acess the details from another class except its class.

31/01/22
* Access Modifires.

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World (diff pkg & not sub class)
Public	+	+	+	+	+
protected	+	+	+	+	
No modifier	+	+	+		
private	+				

9/11/2022

Aggregation :

- Aggregation represents HAS-A relationship, which means when a class contains reference to another class known to have aggregation.

- Aggregation refers one way relationship bet^n two objects.

- Aggregation is based on usage rather than inheritance. Class A has-a relationship with class B, if class A has a reference to an instance of class B.

Ex.
```
Class Address{
    int id;
    String city;
    String state;

    Address(int id, String city, String state){
        this.id = id;
        this.city = city;
        this.state = state;
    }
}
```

```
public class Emp{
    String Emp-name;
    Address a;

    Emp(String name, Address a){
        Emp-name = name;
        this.a = a;
    }

    Void show(){
        sout(Emp-name);
        sout(a.city);
                ↑ used as reffrence to access
        sout(a.state);
        sout(a.country);
    }

  psvm(String[] args){

    Address a0 = new Address("malegaon", "MH", "India");
                                value passing in def const.

    Emp e0 = new Emp("Anand", a0);

! If we use class datatype then we have a
  pass object of that class as parameter.

    e0.show();
  }
}
```

O/P : Anand
malegaon
MH
India

✱ Abstract class & Abstract Method

✱ Abstract class:

⊙ The abstract class in Java can not be instantiated (we can't create objects of the abstract classes).

⊙ abstract keyword is used to create/declare an abstract class / methods.

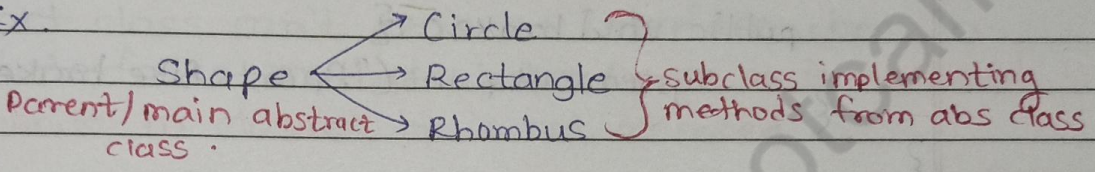
```
e.g. abstract class Parent {
        // fields and methods
     }

     // If we try to create an object of parent
     // then it will throw an error
✗    Parent po = new Parent();
```

⊙ Abstract class can have both methods like abstract method and non abstract method.
Don't have body — Have body / normal methods.

```
abstract class Parent {
        // abstract method
        abstract void son();

        // regular / non abstract method
        void daughter() {
            // body
        }
```

- Abstract class must contain one abstract method.
- when an abstract class is subclassed, the subclass usually provides implementations for all the methods in parent class. If it dosen't then it must be declared abstract.

Ex.



Key Pts:
- Possible to create an reference of abstract class.
- Not possible to create an object of abstract class.
- We can assign reference of an abstract class to the object of a concrete subclass.

abstract class → `A ao = new B();` ← subclass

> multiple inheritance is not allowed.

- Abstract Method. `abstract void method();`

- The abstract method don't have body.
- There is only declaration is done.
- Abstract methods are ment to be overridden. As they don't have body so for working / functioning body should be required so we override them.

But it is not mandatory.

- If class contain abstract method then class must be abstract class.

this will throw an error

```
class ClassNew{
    abstract void method();
}
```

Example:-

```
abstract class MotorBike{
    abstract void brake();
}

class SportsBike extends MotorBike{
    //Implementation of abstract class.
    public void brake(){
        System.out.print("Sports Bike brake.");
    }
}

Class Main{
    psvm(String[] args){
        SportsBike SO = new SportsBike();
        SO.brake();
    }
}
```

O/P: Sports Bike brake.

Recap

Engine car = new Car();

Type → Engine car — what to access

new Car() — which one to access; object

Engine	Car()
start(), stop()	stop(), start(), acc(), light()

As the type is Engine and the object is of car the car can only access methods in/of Engine
car.start(); car.stop()