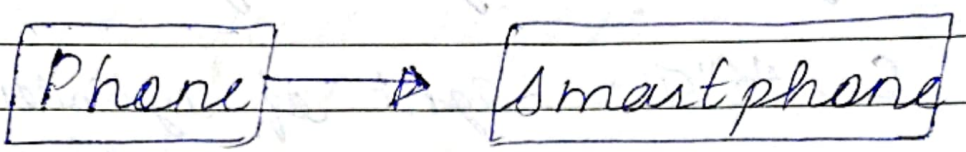
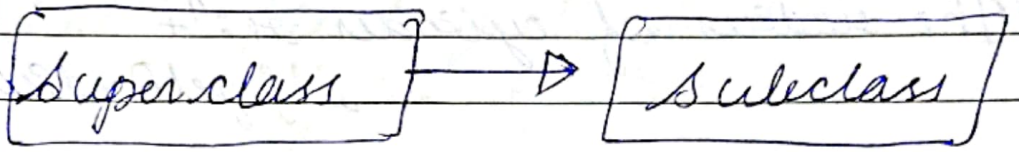
INHERITANCE

→ You might have heard people saying your nose is similar to your father etc. Or formally we say you've inherited genes from your parents.

→ The same phenomena of inheritance is also valid in programming

→ Inheritance is used to borrow prop. & methods from existing class, It Inc. reusability of code
E.g.





subclass extends superclass

Imp: Terminologies:

1) Parent class / Super class:

The class from which a class inherits methods & attributes is known as parent class.

2) Child class / Subclass:

The class that inherits some other class's methods & attributes is known as child class.

Extends keyword in Inheritance

→ The extends keyword is used to inherit a subclass from a superclass.

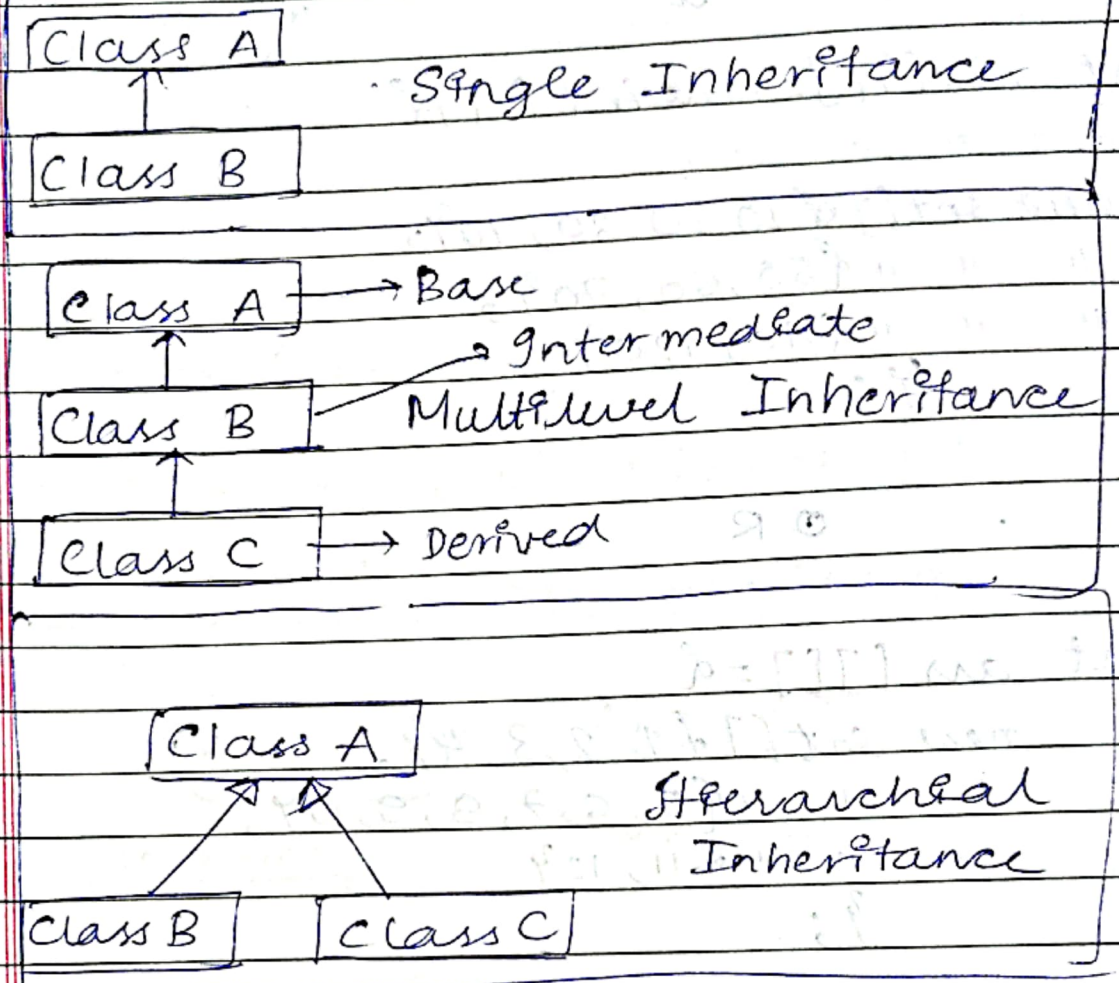
Syntax:

```
class Sub-class-name extends superclass name
{
    // methods & fields
}
```

Eg.

```
public class dog extends Animal {
    // code
}
```

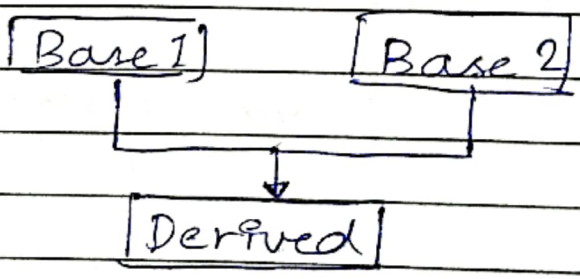
Types of Inheritance



Multiple Inheritance

→ "Multiple Inheritance" refers to the concept of one class extending (or inheriting) more than one base class.

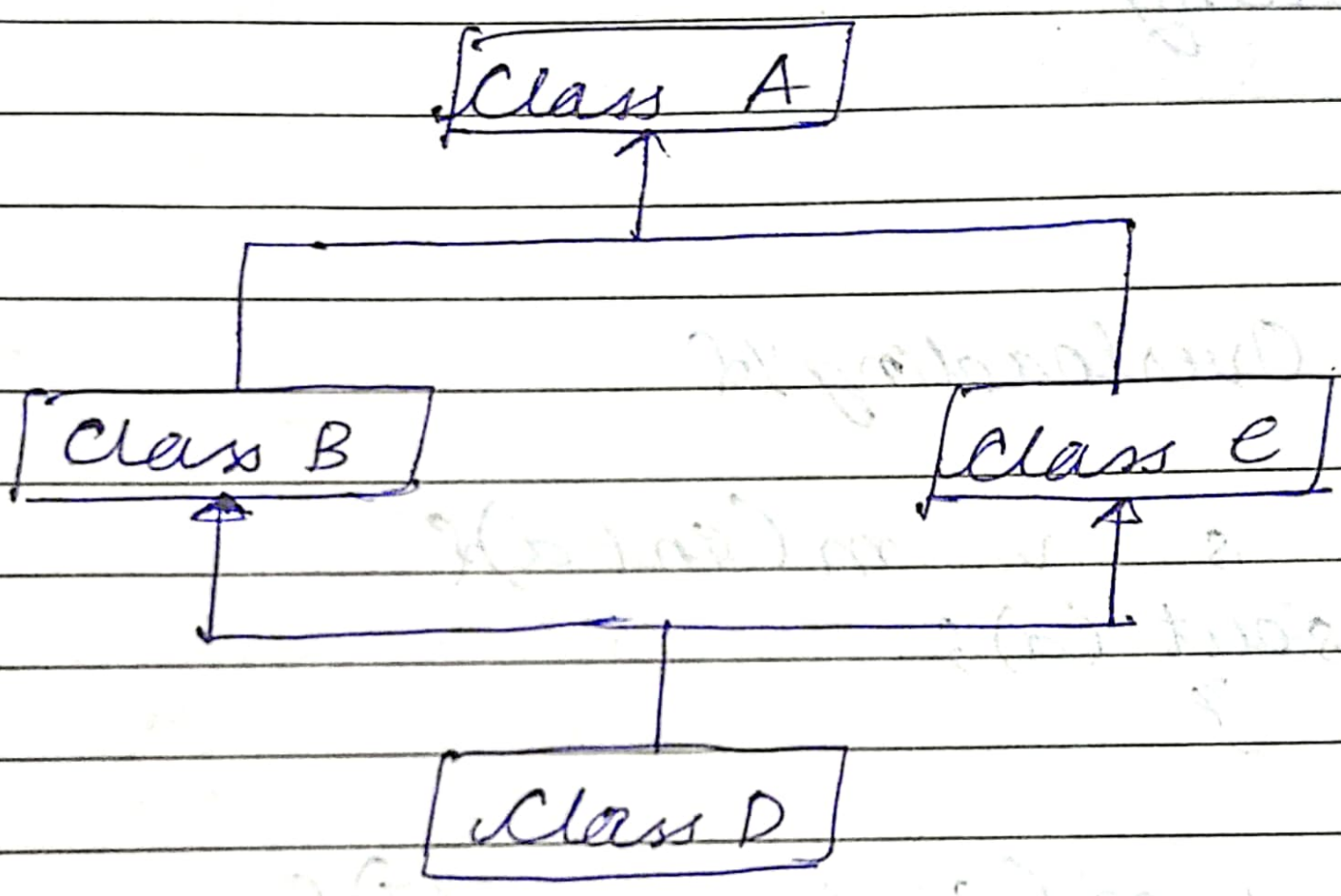
→ The problem with "multiple inheritance" is that derived class will have to manage the dependency on 2 base classes.



Hybrid Inheritance

→ Combination of single & Multiple Inheritance

→



RELATIONSHIPS-

→ Has-A means an instance of one class "has a" reference to an instance of another class or another instance of same class.
→ It has "composition" & "aggregation".
→ We implement HAS-A relationship with "new" keyword.

COMPOSITION

→ Without existence of container object, if there is no chance of existence of contained objects then container & contained objects are said to be strongly associated & this strong associated is known as composition.

E.g. → A university has several departments. Without existence of university there is no chance for departments to exist.
→ Hence university & department are strongly associated & this strong association is called composition.

→ The stronger form is called Composition
→ In Composition the containing object is responsible for creation & life cycle of the contained object (either directly or indirectly)

→ When one object contains another object & outer object can't exist without the other.

E.g.

```
public class Engine {
    final int engineCapacity;

    public Engine(int engineCapacity) {
        this.engineCapacity = engineCapacity;
    }
}

public class Car {
    private final Engine engine;

    public Car(int engineCapacity) {
        this.engine = new Engine(engineCapacity);
    }

    public int getEngineCapacity() {
        return engine.engineCapacity;
    }
}
```

AGGREGATION

→ Without existence of container object, if there is a chance of existence of contained objects then container & contained objects are said to be loosely associated & this strong associated is known as aggregation.

→ E.g. A department has several professors. Without existence of departments there is good chance for professors to exist.

→ Hence professors & department are loosely associated & this loose association is called aggregation.

→ When objects depend on each other, but they have their own independent existence.

E.g.

```
public class Car {
  private final Engine engine;

  public Car(int engineCapacity) {
     this.engine = new Engine(engineCapacity);
  }

  public int getEngineCapacity() {
        return engine.engineCapacity;
  }
}
```

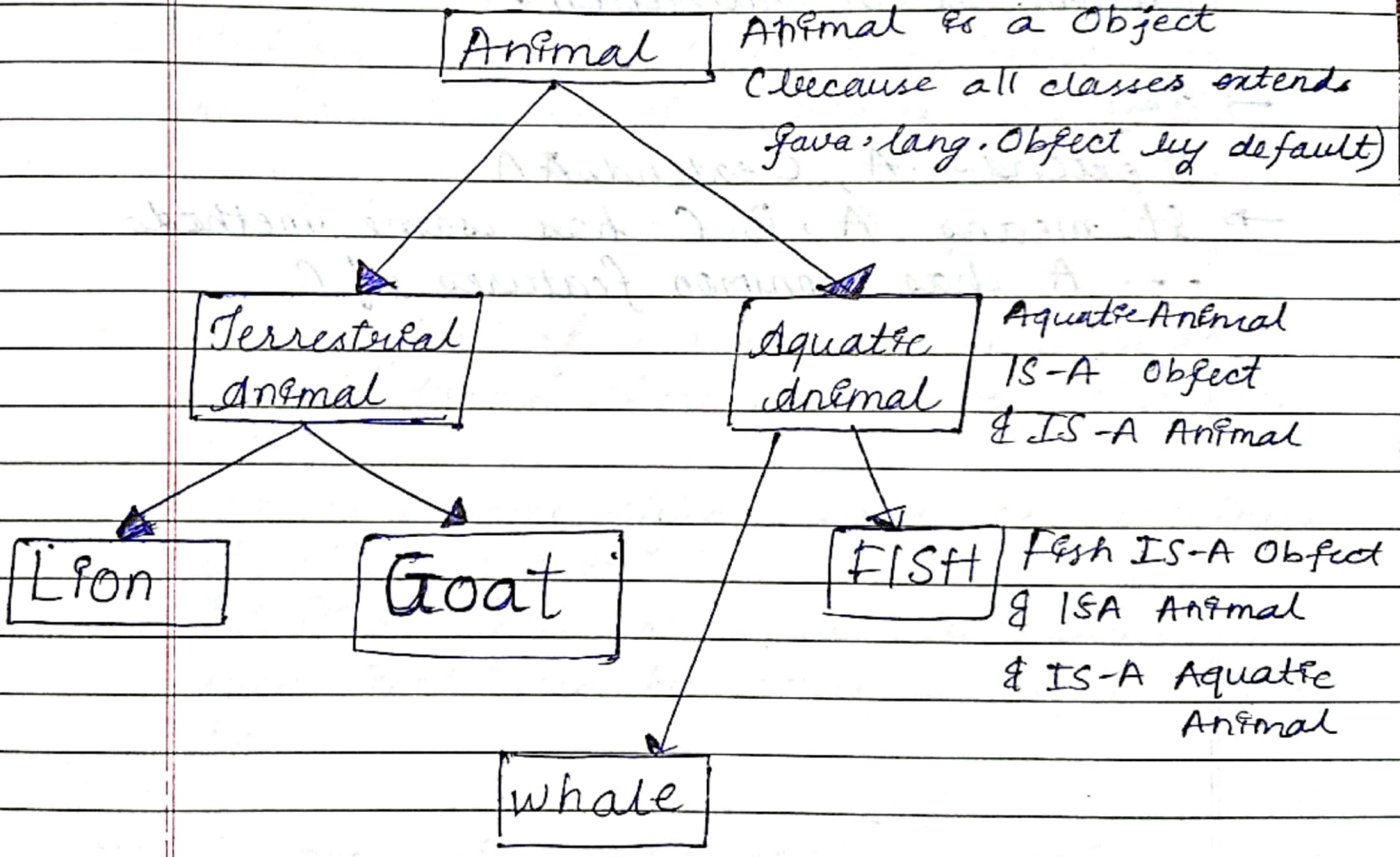
```
public class Person {
    final String name;
    Car car;

    public Person(String name) {
        this.name = name;
    }

    public void buyCar(Car newCar) {
        this.car = newCar;
    }
}
```

IS-A RELATIONSHIP

→ It denotes that "one object is type of another".
→ IS-A rel" denotes inheritance methodology.
→ Inheritance can be implemented with extends (for class) & implements (interface).



It is of 2 types:

- Generalization:

→ Moving up in a hierarchy, looking for general features from bottom to up is "Generalization".

E.g.
class B extends A, C extends A
→ It means A, B, C has same methods ---.
C has all features of A

Specialization

→ Moving Down in a hierarchy looking for common features from up to bottom is 'Specialization'.

→ E.g.
B extends A, C extends A
→ It means A, B, C has same methods
--- A has common features of C

Note: Java doesn't support multiple inheritances, i.e. 2 classes cannot be superclass for a subclass.

```
package.com.company

class Base{
    public int x;

    public int getX(){
        return x;
    }

    public void setX(int x){
        System.out.println("I am in base & setting x now");
        this.x = x;
    }
```

```
public void printMe(){
    System.out.println("I am a constructor");
}
}

class Derived extends Base{
    public int y;

    public int getY(){
        return y;
    }

    public void setY(int y){
        this.y = y;
    }
}

public class Inheritance{

    p s v m (  ){
    // Creating an object of base class

    Base b = new Base();
    b.setX(4);
    Sout(b.getX());

    // Creating an object of derived class
    Derived d = new Derived();
    d.setY(43);
    Sout(d.getY());
    }
}
```

I am in base & setting x now

4
43

Constructors in Inheritance

When a derived class is extended from the base class, the constructor of the base class is executed first followed by the constructor of the derived class.
For following the ~~th~~ inheritance hierarchy, the constructors are executed in the order.

1) C1 - parent
2) C2 - child
3) C3 - grandchild

Constructor during constructor overloading:
→ When there are multiple constructors in the parent class, the constructor without any parameter is called from the child class
→ If we want to call the constructor with parameters from the parent class, we can use the super keyword.
→ super(a, b) calls the constructor from the parent class which takes 2 variables.

this & super keyword in Java

this: THIS KEYWORD

→ this is a way for us to reference an object of the class which is being created / referenced.

→ It is used to call the default constructor of the same class

→ this keyword eliminates the confusion between the parameters & the class attributes with the same name.

For e.g.

```
class cwh {
    int x;

// getter of x
    return x;
} public int getx() {
    return x;
}
// Constructor with a parameter

    cwh(int x) {
        x=x;
    }
// Call the constructor
    p s v m( ) {
    cwh obj1 = new cwh(65);
    Sout (obj1.getx());
}

}
```

o/p → 0

In the above e.g. expected o/p is 65, because we have passed x=65 to the constructor of the cwh class. But the compiler fails to diff. bet" the parameter 'x' & class attribute 'x'. Therefore, it returns 0.

→ Now let's see how can we handle this situation with the help of this keyword:

```
class cwh{
    int x;

  // getter of x
  public int getx(){
        return x;
        }

// Constructor with a parameter

  cwh(int x){
      this.x=x;
      }

// call the constructor

  p s v m(  ){
      cwh obj1= new cwh(65);
        Sout(obj1.getx());

      }
  }
```

o/p = 65

SUPER KEYWORD

→ A reference variable used to refer immediate parent class object.
→ Can be used to refer immediate parent class instance variable
→ Can be used to invoke the parent class method

```
package com.company;
import java.Print.Doc;

class Eklass{
 Int a;
public Int getA(){
  return a;
    }

BEk class{
 Eklass(Int a){
 this.a=a;
    }

public Int returnno(){
    return I;
    }
}
```

```
class Doclass extends Ekclass { Doclass(int c)
  {
    super(c);
    Sout("I am a constructor"); }
  }

public class super {

    p s v m ( ) {
    Ekclass e = new Ekclass(65);
    Doclass d = new Doclass(5);
     Sout(e.getAC());
          }
        }
```

O/p
I am a constructor
65

~~Method overriding~~

METHOD OVERRIDING

→ If the child class implements the same method present in the parent class again, it is known as method overriding.
→ Method overriding helps us to classify a behaviour that is specific to the child class.
→ The subclass can override the method of the parent class only when the method is not declared as final.

```
class A {
  public void meth1(){
      Sout("I am method 1 of class A");
  }
}

class B extends A{ @Override
  public void meth1(){
      Sout("I am method 1 of class B");}}
public class CWH{
   ps v m( ){
     A a=new A();
     a.meth1();

     B b = new B();
     b.meth1();
     }
}
```

O/P

I am method 1 of class A

" " " " B

DYNAMIC METHOD DISPATCH

→ Dynamic method dispatch is also known as run-time polymorphism.

→ It is a process through which a call to an overriden method is resolved at runtime.

→ This technique is used to resolve a call to an overriden method rather than compile time.

→ To properly understand the Dynamic method dispatch in java it is imp. to understand the concept of upcasting, because dynamic method dispatch is based on upcasting.

Upcasting

→ It is a technique in which a superclass reference variable refers to the object of the subclass.

E.g.

~~class Animal{};~~
class Animal{}
class Dog extends Animal{}

Animal a = new Dog(); //upcasting

Here we have taken the reference of parent class Animal (superclass) & created & object of child class Dog (subclass)

E.g.

```
class Phone{
  public void showTime(){
      Sout("Time is 8am");
  }
  public void on(){
     sout("Turning on Phone.....");
  }
}
```

```
class Smartphone extends Phone {
    public void music() {
        sout("Playing music.....");
    }
    public void on() {
        sout("Turning on Smartphone..");
    }
}

public class CWH {
    p s v m ( ) {

Phone obj = new Smartphone(); // allowed
// Smartphone obj2 = new Phone(); //
                                not allowed

obj.showTime();
obj.on();
// obj.music(); // Not allowed
```

o/p Time is 8 am
Turning on Smartphone

Here Phone is the parent class & Smartphone is the child class.

method on() of the parent class is overriden in child class

When obj.on() will be executed, It will call the on() method of the ~~Smartp~~ Smartphone() class : reference variable obj is pointing towards Smartphone();

Note: The data members cannot achieve run time polymerization

Practice set 10

Q2 Create a class circle & use Inheritance to create another class cylinder from it

```
package com.company

public int radius;
circle(){
    Sout("I am a cons. of circle(n-p)");
}

Circle(int r){
    sout("I am a cons. of circle(p)");
    this.radius=r;
}

public double area(){
    return Math.PI*4.0*radius*radius;
}

public double volume(){
    return 4*Math.PI*radius*radius*radius/3;
}
}

class Cylinder1 extends Circle{
    public int height;
```

```
Cylender (int r, int h){
    super(r);
    Sout("I am a constructor of cylender");
    this.height=h;
    }

public double cvolume(){
        return Math.PI*radius*radius*height;
        }
    }

public class Inheritance{

    p s v m ( ){

Circle c = new circle(5);
 Cylender1 cy = new Cylender1(5,7);

  Sout("The area of circle is:" + c.area();
Sout("The volume of cylender is:"+
                        cy.cvolume());
Sout("The volume of circle is:"+
                        c.volume());

        }
    }
```

O/p

I am constr. of circle(param)!

" ——— " ———

——— " ——— Cylender1

The area of circle is: 78.539

The volume of cylender is: 549.778

—" ——— " ———: 523.598