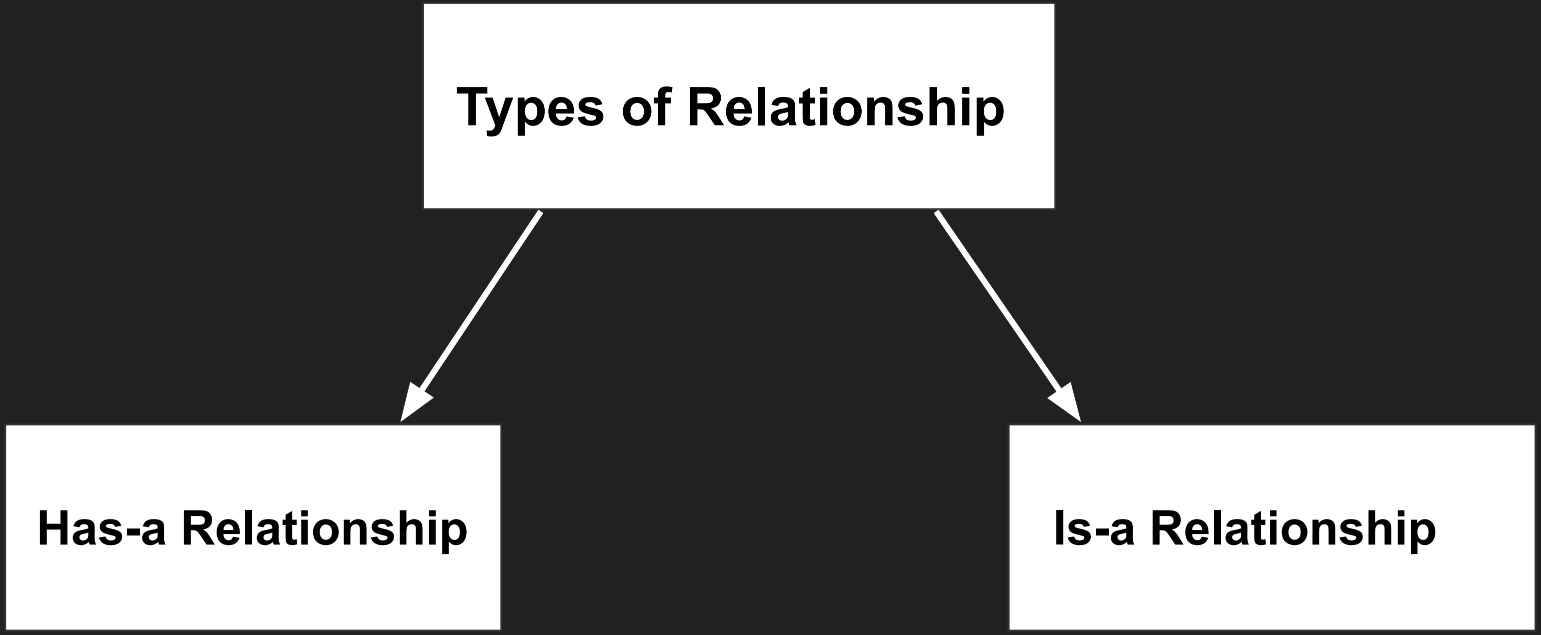
RELATIONSHIP

Relationship

The connection or association between two objects is known as the ‘Relationship’.



Types of Relationship

Has-a Relationship

Is-a Relationship

Has-a Relationship

❖ If one object is dependent on another object it is known as has-a relationship.

❖ It is a unidirectional relationship,i.e, one way relationship.

❖ Based on the level of dependency has-a relationship is classified into two types.

  * Composition

  * Aggregation

AGGREGATION

The dependency between two objects such that one object can exist without the other is known as aggregation.

EXAMPLE : Cab-Ola, Train-Online ticket booking, Bus-Passenger, etc,...

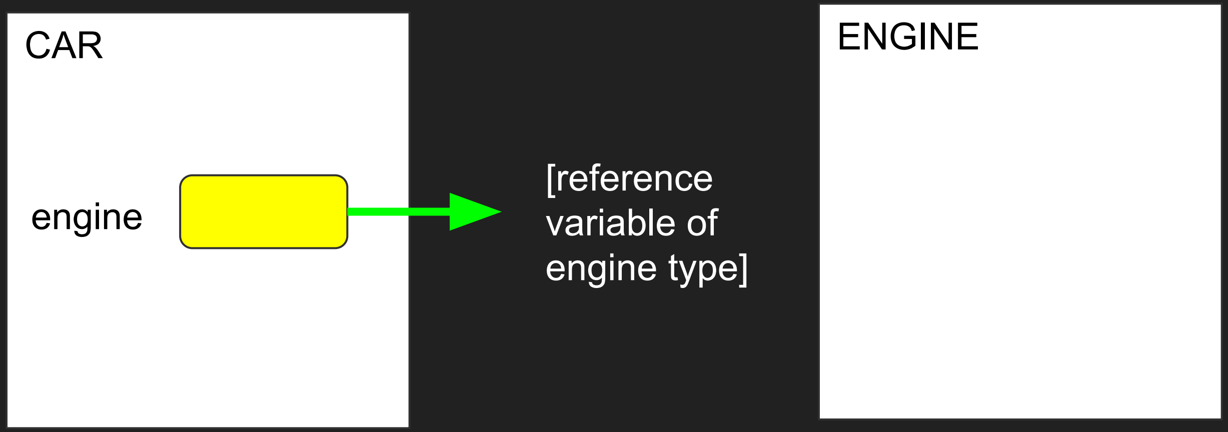
COMPOSITION

The dependency between two objects such that one object can’t exist without the other is known as composition.

EXAMPLE : Car-Engine, Human-Oxygen, etc,...

In java we can achieve has-a relationship by creating the reference variable of one object inside another object.

E.g. Let us consider 2 objects ‘car’ and ‘engine’, ehre the car object is dependent on engine object.



★ The above design can be achieved by creating a non static variable of engine type inside the car class

★ The instance of engine object can be created in 2 different ways by using different design techniques

a) Early instantiation

b) Lazy instantiation

EARLY INSTANTIATION

- If the instance of a dependent object is created implicitly it is known as early instantiation.

- This design can be achieved with the help of an initializer.

STEPS TO ACHIEVE EARLY INSTANTIATION

STEP 1: Create a dependent class.

STEP 2: Create another class and place the reference of the dependent object variable inside the class.

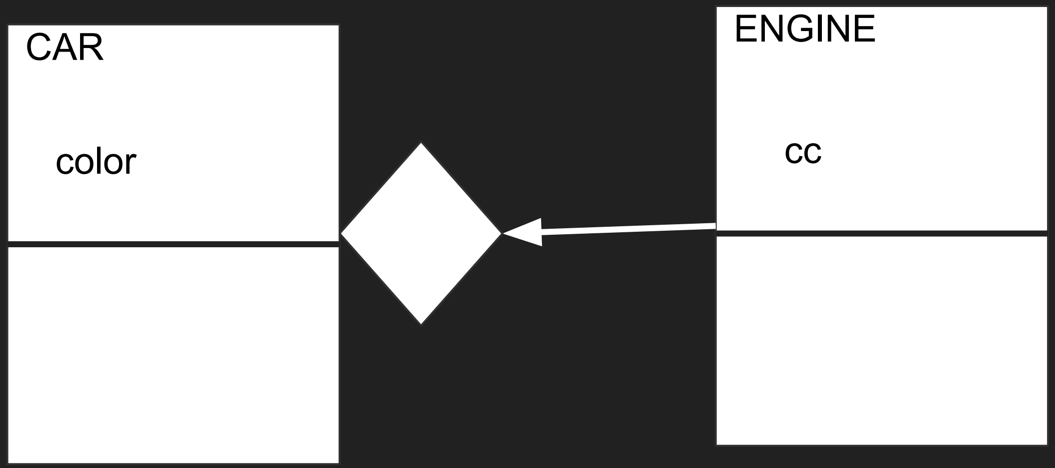
STEP 3: Create a constructor for the class which also accepts the dependent type object.

STEP 4: Create the object for a class so the object of a dependent object is also created.

EXAMPLE :

Whenever we buy a car, the engine is by default mounted inside the car. So the engine object should be created inside the car object.

```
 class Engine
{
int eno = 123;
}
class Car
 {
Engine e = new Engine();
 }
Car c = new Car();
S.o.pln(c.e.eno);//getting engine no by
                           using car object reference
```



LAZY / LATE INSTANTIATION

- In this design, the instance of the dependent object is created only when it is required (It is not implicitly created).
- We can achieve this design with the help of a method, it can be called a helper method.

STEPS TO ACHIEVE LATE / LAZY INSTANTIATION

STEP 1: Create a dependent class.

STEP 2: Create another class and define a parameterized method that will accept the reference of the dependent object and inside that method initialize the dependent object.

STEP 3: Create the object for a class and call a method by passing dependent type object reference so that we can achieve a late/lazy instantiation

EXAMPLE

Mobile is dependent on sim card , but without sim card also a Mobile phone can be used by a user.

```
class Sim
{
        Long simNo;
        Sim(int simNo)
        {
           this.simNo=simNo;
          }
}

Class Mobile
{     Sim s;
       Void insertSim(Sim s)
        {
            this.s=s;
         }
}
```

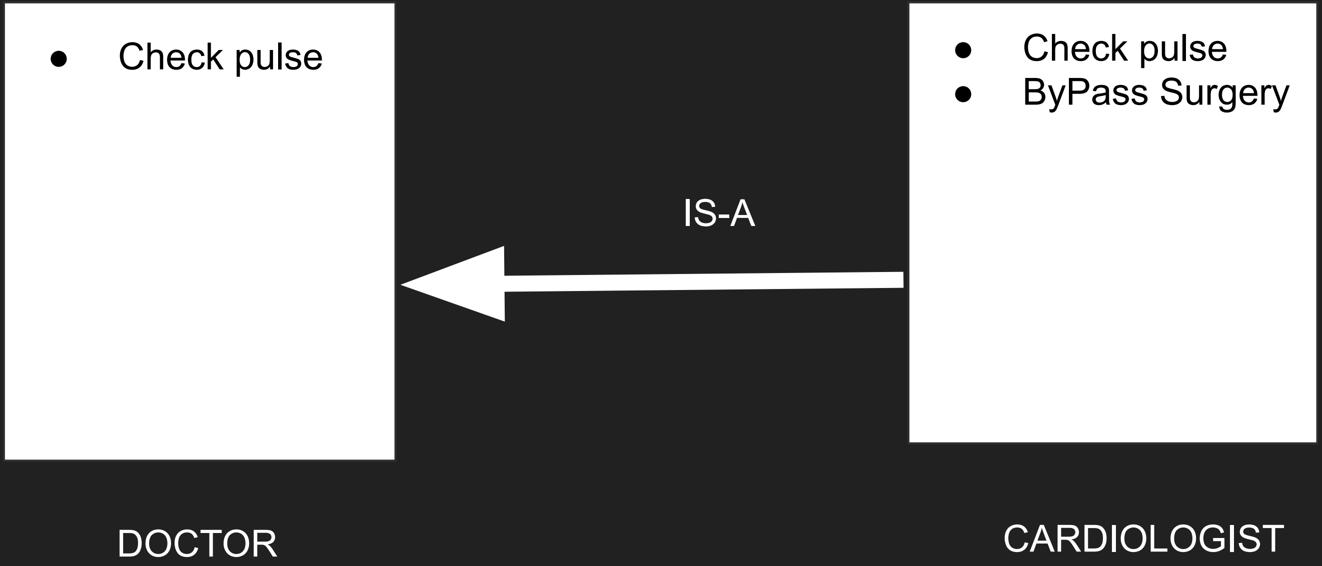
```
class  MobileDriver
{
  public static void main(String[] argos)
   {
      Mobile m1=new Mobile();
      m.inserSim(new Sim(76543287));
    }
}
```

Here we can call the insertSim() method whenever we want to insert the Sim.

IS-A RELATIONSHIP

The association between two objects similar to parents and child is known as ‘Is-a’ relationship.

Here, Doctor is a generalized object /parent object and Cardiologist is an specialized object/child object



- Check pulse

DOCTOR

IS-A

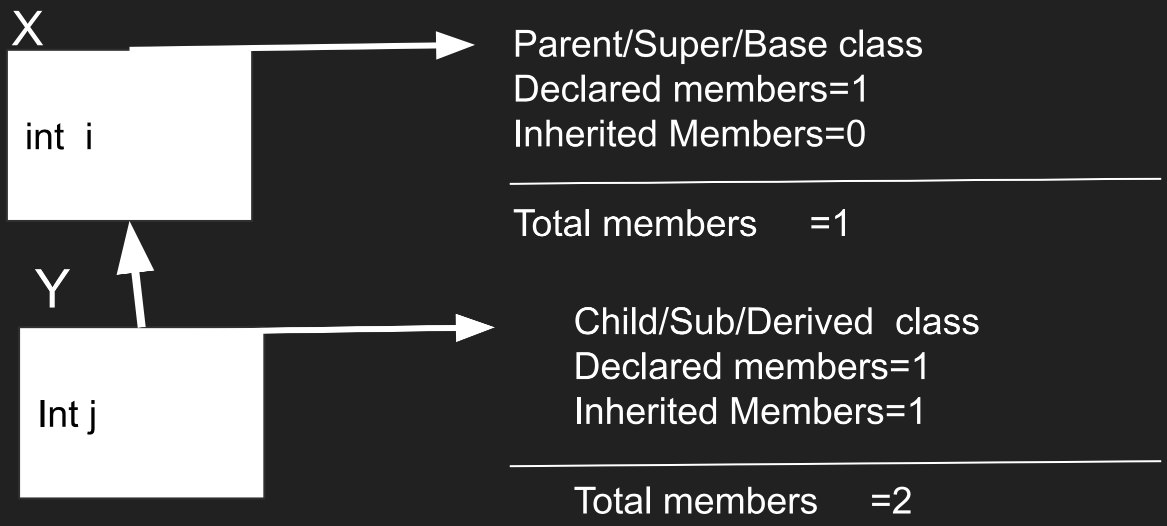
- Check pulse
- ByPass Surgery

CARDIOLOGIST

- Both the object belongs to the same type.Hence both can be called as a family.
- In, Is-A relationship the child object will have all the properties of the parent object and some extra properties of the child.
- Therefore, Parent can be a generalized type and child can be a specialized type.
- Is-A relationship can be achieved with the help of ‘INHERITANCE’.

INHERITANCE

- It is a process of obtaining all the properties of one object into another object.
- The object which is receiving the properties is termed as child/sub/derived objects(classes).
- The object which is providing the properties is known as parent/super/base objects(classes).



Parent/Super/Base class
Declared members=1
Inherited Members=0

Total members =1

Child/Sub/Derived class
Declared members=1
Inherited Members=1

Total members =2

We can achieve Inheritance in Java with the help of the following Keywords:-

1. extends
2. implements

extends:-

a) extends is a Keyword.
b) It is used to achieve inheritance between two classes or between two interfaces.

USING extends FOR TWO CLASSES

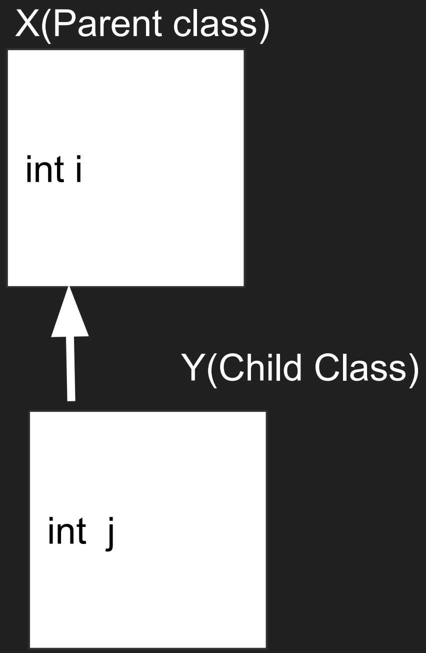
extends keyword can be used only with the child class.

SYNTAX TO ACHIEVE INHERITANCE BETWEEN TWO CLASSES

```
class    ChildClassName      extends   ParentClassName
```

Eg;

```
  class X
{
     int i;
}
class  Y extends X
{
    int j;
}
```



X(Parent class)

int i

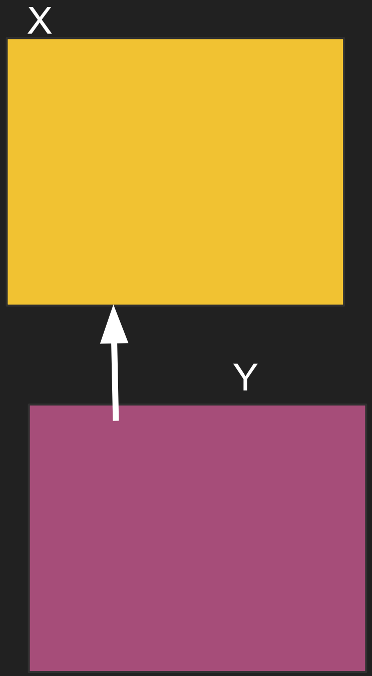
Y(Child Class)

int j

- Except private members and initializers of a class every member is inherited
- With the help of object reference variable we can access member of same class and its parent’s class but we cannot access members of its child class.
- With the help of sub class name we can use all the static members of subclass and its parents/super class.
- With the help of subclass object reference variable we can use both static and non static members of subclass as well as static and non static members of parent/superclass.
- With the help of the superclass reference type we cannot use member of its child or subclass.

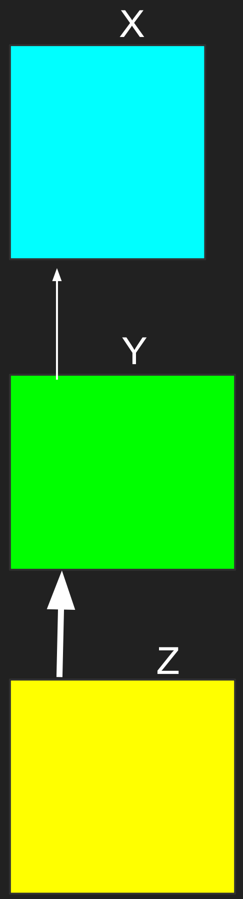
TYPES OF INHERITANCE

1.SINGLE LEVEL INHERITANCE :- Inheritance of only one level is known as single level inheritance.



Y Inherits X

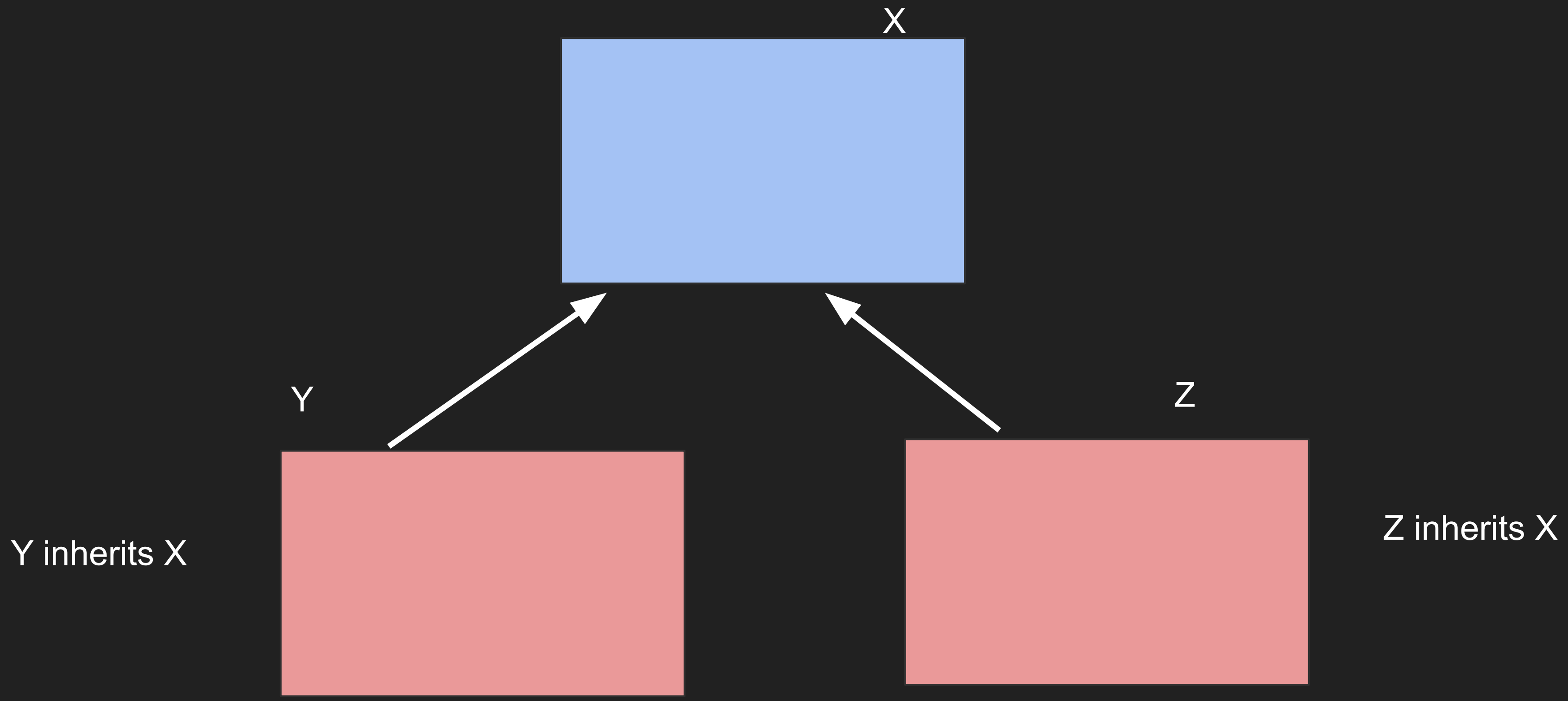
2. MULTI LEVEL INHERITANCE :- Inheritance of more than one level is known as multi level inheritance.



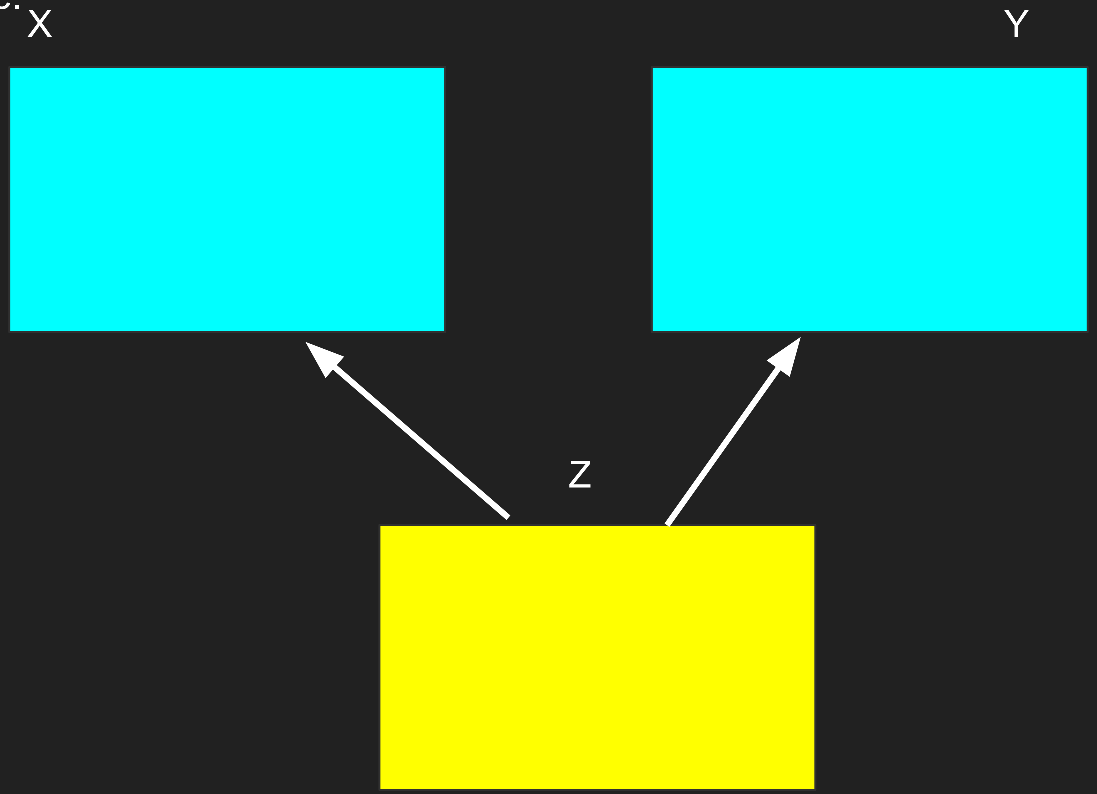
Y inherits X

Z inherits Y and X

3. HIERARCHICAL INHERITANCE :- If a parent (super class) has more than one child (subclass) in the same level then it is known as hierarchical inheritance.



MULTIPLE INHERITANCE :- If a subclass (child) has more than one parent (Super class) then it is known as multiple inheritance.

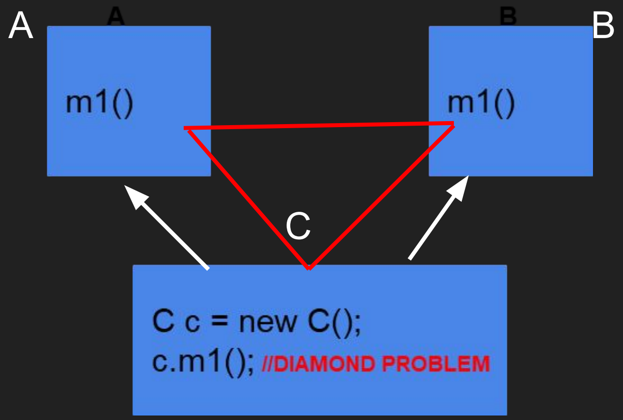


NOTE :

- Multiple inheritance has a problem known as diamond problem.
- Because of diamond problem , we can’t achieve multiple inheritance only with the help of class.
- In java we can achieve multiple inheritance with the help of an interface.

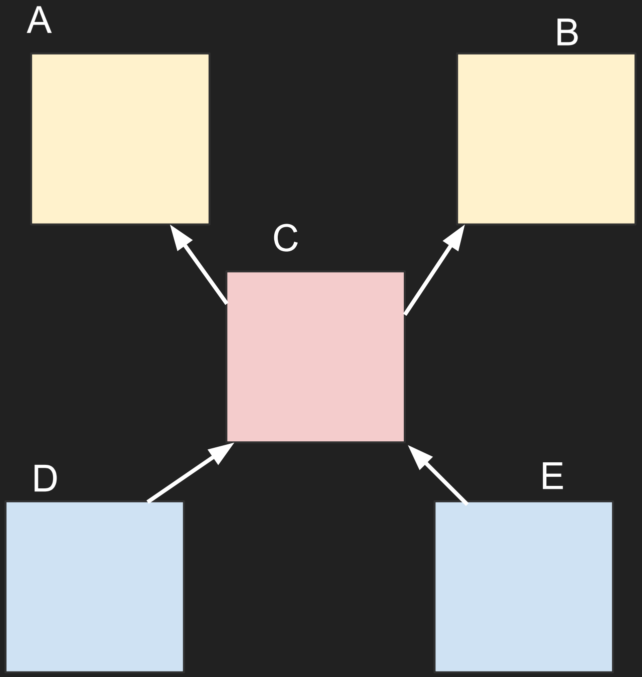
DIAMOND PROBLEM :

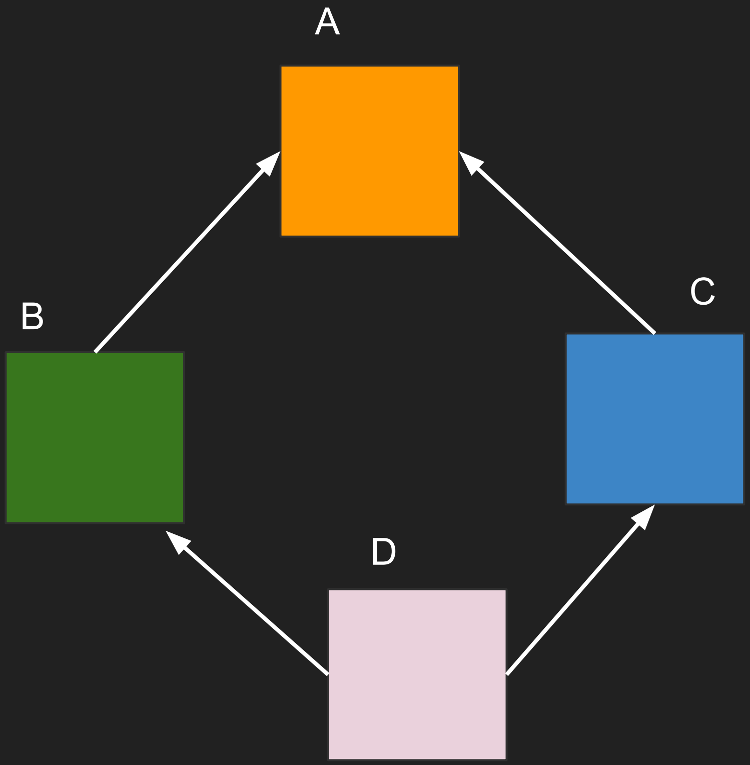
Assume that two classes A and B having the method with same name with same signature. If class C inherits A and B then these two methods are inherited to C (both are having a method with same signature). Whenever we create a object for c and tries to call that inherited method then which method will get executed ? This problem is known as diamond problem.



```
C c = new C();
c.m1(); //DIAMOND PROBLEM
```

HYBRID INHERITANCE :- The combination of multiple inheritance and hierarchical inheritance is known as hybrid inheritance.





super() CALL STATEMENT :

- super is a keyword, it is used to access the members of super class.
- super() call statement is used to call the constructor of parent class from the child class constructor.

PURPOSE OF SUPER() STATEMENT :

- When the object is created, super call statement helps to load the non static members of the parent class into the child object.
- We can also use the super() call statement to pass the data from subclass to parent class.

RULE TO USE SUPER() STATEMENT

- super() call statement should always be first instruction in the constructor call.
- If a programmer doesn't use the super() call statement, then the compiler will have no argument super call statement into the constructor body.

Difference between this() and super() statements

1) this() is used to call the constructor of the same class whereas super() is used to call the constructor of the parent class(Super class).
2) this() is used to represent the instance of child class whereas super() is used to represent the instance of parent class.