INHERITANCE (IS-A) VS. COMPOSITION (HAS-A) RELATIONSHIP

Description

One of the advantages of Object-Oriented programming language is code reuse. There are two ways we can do code reuse either by implementation of inheritance (IS-A relationship), or object composition (HAS-A relationship). Although the compiler and Java virtual machine (JVM) will do a lot of work for you when you use inheritance, you can also get at the functionality of inheritance when you use composition.

IS-A Relationship:

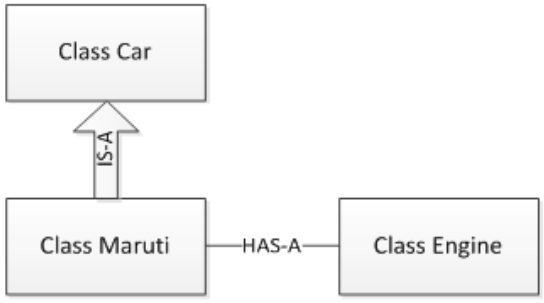
In object oriented programming, the concept of IS-A is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. It is just like saying "A is a B type of thing". For example, Apple is a Fruit, Car is a Vehicle etc. Inheritance is uni-directional. For example House is a Building. But Building is not a House.

It is key point to note that you can easily identify the IS-A relationship. Wherever you see an extends keyword or implements keyword in a class declaration, then this class is said to have IS-A relationship.

HAS-A Relationship:

Composition (HAS-A) simply mean use of instance variables that are references to other objects. For example: Maruti has Engine, or House has Bathroom.

Let’s understand these concepts with example of Car class.




```
1.  package relationships;
2.
3.
4.
5.  class Car {
6.
7.      // Methods implementation and class/Instance members
8.
9.      private String color;
10.
11.     private int maxSpeed;
12.
13.
14.
```

```
15.     public void carInfo(){
16.
17.         System.out.println("Car Color= "+color + " Max Speed= " + maxSpeed);
18.
19.     }
20.
21.
22.
23.     public void setColor(String color) {
24.
25.         this.color = color;
26.
27.     }
28.
29.
30.
31.     public void setMaxSpeed(int maxSpeed) {
32.
33.         this.maxSpeed = maxSpeed;
34.
35.     }
36.
37.
38.
39. }
```

As shown above Car class has couple of instance variable and few methods. Maruti is specific type of Car which extends Car class means Maruti IS-A Car.

```
1.  class Maruti extends Car{
2.
3.      //Maruti extends Car and thus inherits all methods from Car (except final and static)
4.
5.      //Maruti can also define all its specific functionality
6.
7.      public void MarutiStartDemo(){
8.
9.          Engine MarutiEngine = new Engine();
10.
11.         MarutiEngine.start();
12.
13.         }
14.
15.     }
```

Maruti class uses Engine object's start() method via composition. We can say that Maruti class HAS-A Engine.

```
1.  package relationships;
2.
3.
4.
```

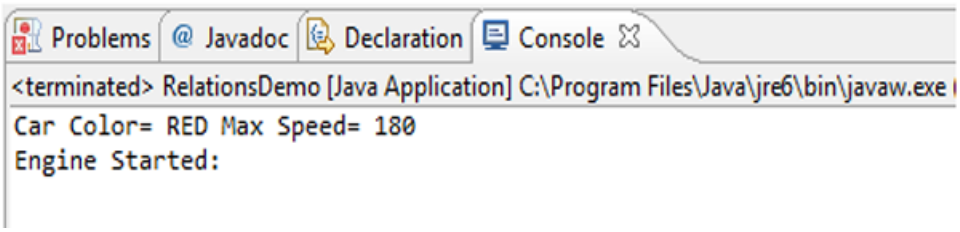
```
5.  public class Engine {
6.
7.
8.
9.      public void start(){
10.
11.         System.out.println("Engine Started:");
12.
13.     }
14.
15.
16.
17.     public void stop(){
18.
19.         System.out.println("Engine Stopped:");
20.
21.     }
22.
23. }
```

RelationsDemo class is making object of Maruti class and initialized it. Though Maruti class does not have setColor(), setMaxSpeed() and carInfo() methods still we can use it due to IS-A relationship of Maruti class with Car class.

```
1.  package relationships;
2.
```

```
3.
4.
5.  public class RelationsDemo {
6.
7.
8.
9.      public static void main(String[] args) {
10.
11.         Maruti myMaruti = new Maruti();
12.
13.         myMaruti.setColor("RED");
14.
15.         myMaruti.setMaxSpeed(180);
16.
17.         myMaruti.carInfo();
18.
19.
20.
21.         myMaruti.MarutiStartDemo();
22.
23.     }
24.
25.
26.
27. }
```

If we run RelationsDemo class we can see output like below.



Comparing Composition and Inheritance

- It is easier to change the class implementing composition than inheritance. The change of a superclass impacts the inheritance hierarchy to subclasses.
- You can't add to a subclass a method with the same signature but a different return type as a method inherited from a superclass. Composition, on the other hand, allows you to change the interface of a front-end class without affecting back-end classes.
- Composition is dynamic binding (run time binding) while Inheritance is static binding (compile time binding)
- It is easier to add new subclasses (inheritance) than it is to add new front-end classes (composition), because inheritance comes with polymorphism. If you have a bit of code that relies only on a superclass interface, that code can work with a new subclass without change. This is not true of composition, unless you use composition with interfaces. Used together, composition and interfaces make a very powerful design tool.

- With both composition and inheritance, changing the implementation (not the interface) of any class is easy. The ripple effect of implementation changes remain inside the same class.
    - **Don't use inheritance just to get code reuse** If all you really want is to reuse code and there is no is-a relationship in sight, use composition.
    - **Don't use inheritance just to get at polymorphism** If all you really want is polymorphism, but there is no natural is-a relationship, use composition with interfaces.

Summary

- IS-A relationship based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance.
- Has-a relationship is composition relationship which is productive way of code reuse.

Source: http://www.w3resource.com/java-tutorial/inheritance-composition-relationship.php