

JAVA - INHERITANCE

Inheritance can be defined as the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.

When we talk about inheritance, the most commonly used keyword would be **extends** and **implements**. These words would determine whether one object IS-A type of another. By using these keywords we can make one object acquire the properties of another object.

IS-A Relationship:

IS-A is a way of saying : This object is a type of that object. Let us see how the **extends** keyword is used to achieve inheritance.

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

Now, based on the above example, In Object Oriented terms, the following are true:

- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say:

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence : Dog IS-A Animal as well

With use of the extends keyword the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass.

We can assure that Mammal is actually an Animal with the use of the instance operator.

Example:

```
public class Dog extends Mammal{

    public static void main(String args[]){

        Animal a = new Animal();
        Mammal m = new Mammal();
        Dog d = new Dog();

        System.out.println(m instanceof Animal);
        System.out.println(d instanceof Mammal);
        System.out.println(d instanceof Animal);
    }
}
```

```
}
```

This would produce the following result:

```
true  
true  
true
```

Since we have a good understanding of the **extends** keyword let us look into how the **implements** keyword is used to get the IS-A relationship.

The **implements** keyword is used by classes by inherit from interfaces. Interfaces can never be extended by the classes.

Example:

```
public interface Animal {}  
  
public class Mammal implements Animal{  
}  
  
public class Dog extends Mammal{  
}
```

The instanceof Keyword:

Let us use the **instanceof** operator to check determine whether Mammal is actually an Animal, and dog is actually an Animal

```
interface Animal{}  
  
class Mammal implements Animal{}  
  
public class Dog extends Mammal{  
    public static void main(String args[]){  
  
        Mammal m = new Mammal();  
        Dog d = new Dog();  
  
        System.out.println(m instanceof Animal);  
        System.out.println(d instanceof Mammal);  
        System.out.println(d instanceof Animal);  
    }  
}
```

This would produce the following result:

```
true  
true  
true
```

HAS-A relationship:

These relationships are mainly based on the usage. This determines whether a certain class **HAS-A** certain thing. This relationship helps to reduce duplication of code as well as bugs.

Lets us look into an example:

```
public class Vehicle{}  
public class Speed{}  
public class Van extends Vehicle{  
    private Speed sp;  
}
```

This shows that class Van HAS-A Speed. By having a separate class for Speed, we do not have to put the entire code that belongs to speed inside the Van class., which makes it possible to reuse the Speed class in multiple applications.

In Object-Oriented feature, the users do not need to bother about which object is doing the real work. To achieve this, the Van class hides the implementation details from the users of the Van class. So basically what happens is the users would ask the Van class to do a certain action and the Van class will either do the work by itself or ask another class to perform the action.

A very important fact to remember is that Java only supports only single inheritance. This means that a class cannot extend more than one class. Therefore following is illegal:

```
public class extends Animal, Mammal{}
```

However, a class can implement one or more interfaces. This has made Java get rid of the impossibility of multiple inheritance.