Object oriented design

MAY 25, 2015

Encapsulation

Suppose you want the variable `small` to be always $1/3^{rd}$ of `big`.

But in the below code as the variable `small` is `public`, any code can access it and assign any arbitrary value thus violating the requirement of `small` being $1/3^{rd}$ of `big`. This happens **due to lack of encapsulation**.

```
class Foo {
    public int big = 9;
    public int small = 3;
    public void setBig(int num) {
        big = num;
        small = num/3;
    }
    // other code here
}
```

The problem can be solved by changing the access modifier of `small` to `private` so that each and every code will go through `setBig()` method to set the value of `big` and `small`. This is nothing but a small **practical example of encapsulation**. Here we encapsulated the variable `small` by making it private and providing *getters* and *setters* for code to access it. *(We should follow this practice always for all variables)*

Inheritance

A class inheriting public/protected properties of another class by using the keyword `extends` is called *inheritance in java*.

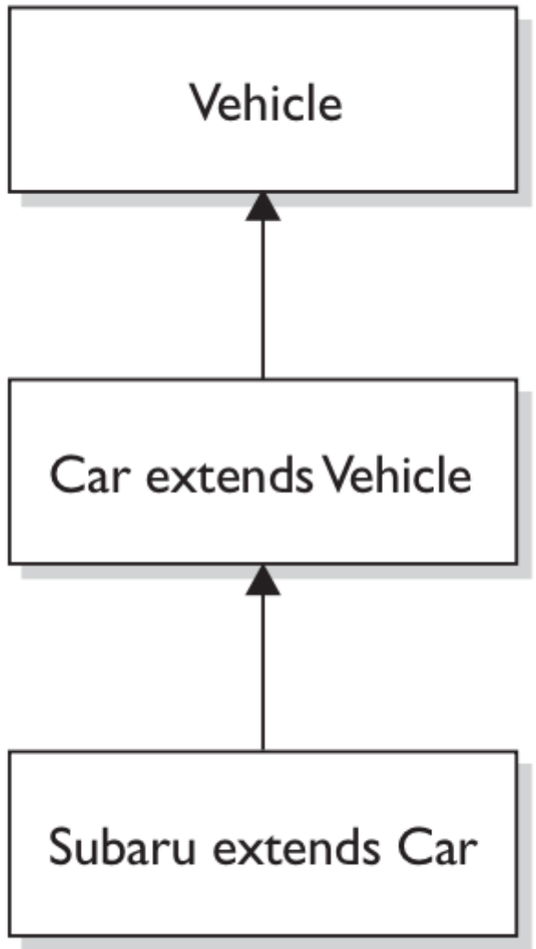
The two most common **reasons to use inheritance**:

- To promote code reuse.
- To use polymorphism.

IS-A relationship

In OO, the concept of *IS-A* is based on **class inheritance or interface implementation**. *IS-A* is a way of saying, “this thing is a type of that thing.”

For example in the below figure:



“Car extends Vehicle” means “Car IS-A Vehicle.”
“Subaru extends Car” means “Subaru IS-A Car.”

HAS-A relationship

HAS-A relationships are *based on usage, rather than inheritance*. In other words, class A *HAS-A* B if code in class A has a reference to an instance of class B. For example, you can say the following, A Horse *IS-A* Animal. A Horse *HAS-A* Halter. The code might look like this:

```
public class Animal { }

public class Horse extends Animal {
    private Halter myHalter;
}
```

Polymorphism

Any Java object that can pass more than one *IS-A* test can be considered **polymorphic**. Other than objects of type `Object`, all Java objects are polymorphic in that they pass the *IS-A* test for their own type and for class `Object`.

The only way to access an object is through a reference variable:

- A reference variable can be of only one type, and once declared, that type can never be changed (although the object it references can change provided its not declared `final`).
- A reference variable's type determines the methods that can be invoked on the object the variable is referencing.
- A reference variable can refer to any object of the same type as the declared reference, or **to any subtype of the declared type**.
- A reference variable can be declared as a class type or an interface type. If the variable is declared as an interface type, it can reference any object of any class that implements the interface.

More on polymorphism in Overriding & Overloading in Java.

Access Control

« Previous

Overriding

Next »