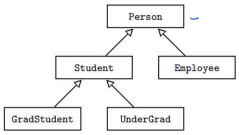
Inheritance

Saturday, February 4, 2023 11:38 AM

What is **superclass** and **subclass**



For any of these classes, an arrow points to its superclass. The arrow designates an inheritance relationship between classes, or, informally, an *is-a* relationship. Thus, an `Employee` *is-a* `Person`; a `Student` *is-a* `Person`; a `GradStudent` *is-a* `Student`; an `UnderGrad` *is-a* `Student`. Notice that the opposite is not necessarily true: A `Person` may not be a `Student`, nor is a `Student` necessarily an `UnderGrad`.

Every subclass inherits the public or protected variables and methods of its superclass (see p. 135). Subclasses may have additional methods and instance variables that are not in the superclass. A subclass may redefine a method it inherits. For example, `GradStudent` and `UnderGrad` may use different algorithms for computing the course grade, and need to change a `computeGrade` method inherited from `Student`. This is called *method overriding*. If part of the original method implementation from the superclass is retained, we refer to the rewrite as *partial overriding* (see p. 135).

class Dog

HSQ "is a" Dog

class HSQ extends Dog

Notice 0: Inheritance is a "is a" relationship, which means that the subclass object is a super class object. For example, 哈士奇 is a 狗

THE `extends` KEYWORD

The inheritance relationship between a subclass and a superclass is specified in the declaration of the subclass, using the keyword `extends`. The general format looks like this:

```
public class Superclass
{
    //private instance variables
    //other data members
    //constructors
    //public methods
    //private methods
}

public class Subclass extends Superclass
{
    //additional private instance variables
    //additional data members
    //constructors  (Not inherited!)
    //additional public methods
    //inherited public methods whose implementation is overridden
    //additional private methods
}
```

Notice 1: you can freely use public variable and method from superclass in the subclass, but you cannot use private variable or method from superclass.

CONSTRUCTORS AND `super`

Constructors are never inherited! If no constructor is written for a subclass, the superclass default constructor with no parameters is generated. If the superclass does not have a default (zero-parameter) constructor, but only a constructor with parameters, a compiler error will occur. If there is a default constructor in the superclass, inherited data members will be initialized as for the superclass. Additional instance variables in the subclass will get a default initialization—0 for primitive types and `null` for reference types.

A subclass constructor can be implemented with a call to the `super` method, which invokes the superclass constructor. For example, the default constructor in the `UnderGrad` class is identical to that of the `Student` class. This is implemented with the statement

```
super();
```

Be sure to provide at least one constructor when you write a subclass. Constructors are never inherited from the superclass.

Notice 2: The constructor in the subclass must have a super statement in the first line to initialize variables in the super class. If you didn't write constructor in subclass, you will use default constructor super();

Notice 3: *super* is a keyword like *this* which is used to refer public **variables or method in the superclass**

Notice 4: Method override means to write a method in the subclass with exactly the same name and signature as that method in the super class

Vehicle v = new Vehicle(); v.run(). The vehicle is running.

Car c = new Car(); c.run() The car is running.

Notice 5: When you create object for the subclass, you can either use subclass or super class as the reference type. What determines the object is the thing after "="

NOTE

1. If `super` is used in the implementation of a subclass constructor, it *must* be used in the first line of the constructor body.
2. If no constructor is provided in a subclass, the compiler provides the following default constructor:

```
public SubClass()
{
    super();      //calls default constructor of superclass
}
```

△ Notice 6.

如果用super class作为subclass object 的reference type，那么则不能调用subclass specific的方法，只能调用在subclass里被override过的方法，这种情况下是调用subclass的override version。

Car 里有，vehicle 里没有的东西

Vehicle v1 = new Car(); v1 不能用

Car c = new Car();

v1.run();

↳ The car is running

https://www.geeksforgeeks.org/referencing-subclass-objects-subclass-vs-superclass-reference/

Consider the example below:

```java
public class Vehicle {
    public int speed;
    private String color;

    public Vehicle() {
        this.speed = 0;
        this.color = "";
    }

    public Vehicle(int speed, String color) {
        this.speed = speed;
        this.color = color;
    }

    public void run() {
        System.out.println(x: "The Vehicle is running!");
    }

    public void run(String tag) {
        // this is an example of method overload
        System.out.println("The Vehicle is running! " + tag);
    }

}
```

```java
public class Car extends Vehicle{

    private int num_wheels;

    public Car() {
        super();
        this.num_wheels = 4;
    }

    public Car(int speed, String color, int num_wheels) {
        super(speed, color);
        this.num_wheels = num_wheels;
    }

    public void run() {
        // this is an example of method override
        System.out.println(x: "The Car is running");
    }

    public int getSpeed() {
        return super.speed;
    }

    public void setSpeed(int speed) {
        super.speed = speed;
    }

    public int accesstest() {
        // you cannot access the private variable of the super class
        super.color;
    }
}
```

```java
class Main {
    Run | Debug
    public static void main(String[] args) {
        Vehicle v = new Vehicle(speed: 2, color: "blue");
        Vehicle v1 = new Car();
        Car c1 = new Car();
        Car c2 = new Vehicle(); // This is illegal, since vehicle is not a car

        // guess what is the output of the following statement

        System.out.println(c1 instanceof Vehicle);
        System.out.println(v1 instanceof Vehicle);
        System.out.println(v1 instanceof Car);
        System.out.println(v instanceof Car);

        c1.run();
        v1.run();
        c1.run(tag: "good");

        System.out.println(c1.getSpeed());
        System.out.println(v1.getSpeed());
    }

}
```

```
true
true
true
false
The Car is running
The Car is running
The Vehicle is running! good
0
```