Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

---

extends Keyword

extends is the keyword used to inherit the properties of a class.
In the following example, you can observe two classes namely `Calculation` and `My_Calculation` .

Using ***extends*** keyword, the `My_Calculation` inherits the methods `addition()` and `subtraction()` of `Calculation` class.

> **Example file 1: Calculation.java**

```
class Calculation {
    int z;

    public void addition(int x, int y) {
        z = x + y;
        System.out.println("The sum of the given numbers:" + z);
    }

    public void subtraction(int x, int y) {
        z = x - y;
        System.out.println("The difference between the given numbers:" +
z);
```

```
    }
}
```

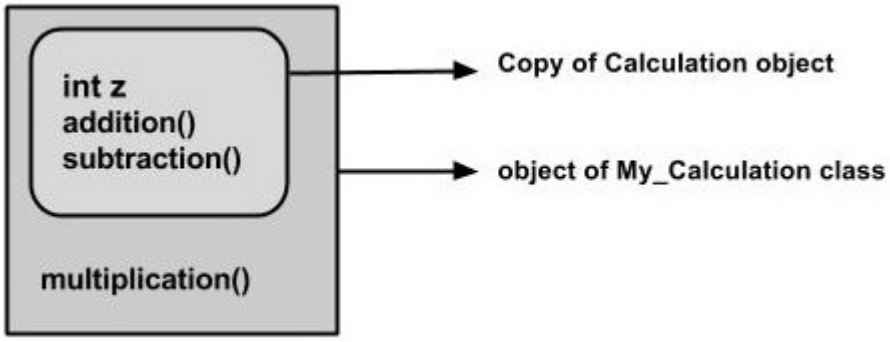
> **Example file 2: My_Calculation.java**

```
public class My_Calculation extends Calculation {
    public void multiplication(int x, int y) {
        z = x * y;
        System.out.println("The product of the given numbers:" + z);
    }

    public static void main(String args[]) {
        int a = 20, b = 10;
        My_Calculation demo = new My_Calculation();
        demo.addition(a, b);
        demo.subtraction(a, b);
        demo.multiplication(a, b);
    }
}
```

> **Output:**
> The sum of the given numbers:30
> The difference between the given numbers:10
> The product of the given numbers:200

In the given program, when an object to My_Calculation class is created, a copy of the contents of the superclass is made within it. That is why, using the object of the subclass you can access the members of a superclass.



The Superclass reference variable can hold the subclass object, but using that variable you can access only the members of the superclass, so to access the members of both classes it is recommended to always create reference variable to the subclass.

If you consider the above program, you can instantiate the class as given below. But using the superclass reference variable (cal in this case) you cannot call the method multiplication(), which belongs to the subclass My_Calculation.

```
Calculation cal = new My_Calculation();
demo.addition(a, b);
demo.subtraction(a, b);
```

> **Note**
> A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

The super keyword

The `super` keyword is similar to `this` keyword. Following are the scenarios where the `super` keyword is used.

1. It is used to differentiate the members of superclass from the members of subclass, if they have same names.
2. It is used to invoke the superclass constructor from subclass.

If a class is inheriting the properties of another class., and if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

```
super.variable
super.method();
```

> **Example (Sub_class.java):**

```
class Super_class {
int num = 20;

// display method of superclass
public void display() {
    System.out.println("This is the display method of superclass");
}
```

```
}

public class Sub_class extends Super_class {
    int num = 10;

    // display method of sub class
    public void display() {
        System.out.println("This is the display method of subclass");
    }

    public void my_method(Sub_class sub) {
        // Invoking the display() method of sub class
        sub.display();

        // Invoking the display() method of superclass
        super.display();

        // printing the value of variable num of subclass
        System.out.println("value of the variable named num in sub class:"
+ sub.num);

        // printing the value of variable num of superclass
        System.out.println("value of the variable named num in super
class:" + super.num);
    }

    public static void main(String args[]) {
        Sub_class obj = new Sub_class();
        obj.my_method(obj);
    }
}
```

In the given program, you have two classes namely Sub_class and Super_class, both have a method named display() with different implementations, and a variable named num with different values. We are invoking display() method of both classes and printing the value of the variable num of both classes. Here you can observe that we have used super keyword to differentiate the members of superclass from subclass.

> **Output:**
> This is the display method of subclass
> This is the display method of superclass
> value of the variable named num in sub class:10
> value of the variable named num in super class:20

Overriding

If a class inherits a method from its superclass, then there is a chance to override the method provided that it is not marked final.

The benefit of overriding is: ability to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

> **Example (TestDog.java):**

```
class Animal {
    public void move() {
        System.out.println("Animals can move");
    }
}

class Dog extends Animal {
    public void move() {
        System.out.println("Dogs can walk and run");
    }
}

public class TestDog {

    public static void main(String args[]) {
        Animal a = new Animal(); // Animal reference and object
        Animal b = new Dog(); // Animal reference but Dog object

        a.move(); // runs the method in Animal class
        b.move(); // runs the method in Dog class
    }
}
```

> **Output**
> Animals can move
> Dogs can walk and run

In the above example, you can see that even though `b` is a type of Animal it runs the ***move method in the Dog class***. The reason for this is: In compile time, the check is made on the

reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

Therefore, in the above example, the program will compile properly since Animal class has the method move. Then, at the runtime, it runs the method specific for that object.

Execution Sequence while loading a child class

Let's run the following code to find out the execution sequence when loading a child class.

> **Example (ChildClass.java):**

```
public class ChildClass extends ParentClass {
    static {
        System.out.println("Child class static instance block");
    }
    {
        System.out.println("Child class instance block");
    }

    public ChildClass() {
        System.out.println("Child class constructor");
    }

    public static void main(String[] args) {
        new ChildClass();
    }
}

class ParentClass {
    private static int value = 10;
    static {
        System.out.println("Parent Class static innitialization block : " +
value);
    }
    {
        System.out.println("Parent Class innitialization block");
    }

    protected ParentClass() {
```

```
            System.out.println("Parent Class consutructor");
        }
    }
```

> **Output**
> Parent Class static innitialization block : 10
> Child class static instance block
> Parent Class innitialization block
> Parent Class consutructor
> Child class instance block
> Child class constructor

So,

1. The first thing that is executed while loading(the object is not yet fully created) a class is static variable initialization and the static blocks in parent class followed by child class static variable initialization and the blocks.
2. The next thing is the parent class initialization block followed by the parent class constructor.
3. The last thing is the child class initialization blocks followed by child class constructor.

Assignment

Create several pairs of Parent and Child classes. For example, Vehicle to Airplane or Car. Don't forget to use `this` and `super` to test.

Further reading

Java Inheritance - how2examples.com