Statics

AUGUST 8, 2015

Static is one of the non-access modifier which has a huge role in Java. It can be used with both variables and methods. It is used when we want some variable or method not to be a part of any specific object of a class rather common for all objects of the class.

For example, keeping count of all the instances created for a particular class we can make the count variable `static` or when we have a method which returns the same result irrespective of the instance invoking it like `getRandom()` returning any random number or `getMax(int x, int y)` returning the larger number between `x` and `y` we can make the methods `static`.

Ways to access static variables/methods

As static variable/methods doesn't belong to any particular object of a class, you need not create any object to access static variable/method. You can access directly with the help of classname and dot `'.'` operator but you can also access normally like any other instance variable/method i.e, via object reference and dot `'.'` operator. See below:

```java
class Frog {

    static int frogCount = 0; // Declare and initialize
                              // static variable
    int frogSize = 0;

    public Frog() {
        frogCount += 1;  // Modify the value in the constructor
    }

    public static int getFrogCount() {
        return frogCount;
    }

    public int getFrogSize() {
        return frogSize;
    }

    public static void main(String[] args) {
        Frog f = new Frog();
        System.out.println(f.getFrogSize()); // Access instance
                                             // method via object reference f

        System.out.println(Frog.frogCount);  // Access static
                                             // variable via class

        System.out.println(f.frogCount);  // Access static
                                          // variable via object reference f

        System.out.println(Frog.getFrogCount());  // Access static
                                                  // method via class

        System.out.println(f.getFrogCount());  // Access static
                                               // method via object reference f
    }
}
```

Some points to remember here

- Static method **cannot access** a instance(non-static) variable directly.

```
public static int getFrogSize() {
    return frogSize; // compiler error
}
```

But they can access through a object reference.

```
public static int getFrogSize() {
    Frog obj = new Frog();
    return obj.frogSize; // ok
}
```

- Static method **cannot access** a instance method directly.

```
public static int getFrogSizeSum() {
    int size = getFrogSize(); // compiler error
    return size * frogCount;
}
```

But they can access through a object reference.

```
public static int getFrogSizeSum() {
    Frog obj = new Frog();
    int size = obj.getFrogSize(); // ok
    return size * frogCount;
}
```

-

Static method **can access** a static variable or static method.

-

Static methods **cannot be overridden**. This is an important point. Consider the below program to understand:

```java
class Animal {
    static void doStuff() {
        System.out.print("a ");
    }
}

class Dog extends Animal {

    // it's a redefinition,
    // not an override
    static void doStuff() {
        System.out.print("d ");
    }

    public static void main(String[] args) {
        Animal[] a = {new Animal(), new Dog(), new Animal()};
        for (int x = 0; x < a.length; x++)
            a[x].doStuff();
    }
}
```

The output of the above program is `'a a a'`. If the `doStuff()` method have been overridden then the output would have been `'a d a'`. Because in case of method overriding polymorphism comes to play, according to which the actual object type determines which method is to be invoked but in this case the object reference determines which method is to be invoked (`doStuff()` of `Animal` class is invoked all the time as the object reference is of `Animal` type) i.e, no overriding and no polymorphism.

In other words, it perfectly makes sense as only inherited methods are overridden and `static` methods are never inherited as they don't belong to any object, so they can't be overridden.

- Static methods **can be overloaded** in the same class as well in the child classes. Read method overloading to be more clear.

```java
public class Foo {
    public static void doStuff(int x) { }

    public static void doStuff(int x, int y) { }  // Valid overload as DIFFERENT ARGUMENT L1
                                                  // (and methods can be overloaded
                                                  // in the same class or in subclass)
}

class Bar extends Foo {

    public static void doStuff(String x) { }  // Valid overload as DIFFERENT ARGUMENT LIST
                                              // (and methods can be overloaded
                                              // in the same class or in subclass)
}
```


Constructors

« Previous

Nested Classes

Next »