Object oriented programming

Using *static* and *final*

Along previous sections, we've been using *static* and *final* words in some parts of our code. It's now time to explain what they exactly mean.

1. Static elements

static modifier defines elements that belong to the class itself, and not to any specific object of the class. For instance, if we define a static attribute:

```
public class MyClass
{
    static int count = 0;
}
```

Then every object that we create of that class will share this attribute with the rest of the objects of that class. So a static attribute is something like a *shared variable*.

Regarding methods, a static method is a method that can be called without creating any object of the class:

```
public class MyClass
{
    ...
    public static void myMethod()
    {
        ...
    }

    public void myOtherMethod()
    {
        ...
    }
}
...
MyClass.myMethod();          // OK
MyClass.myOtherMethod();     // ERROR

MyClass mc = new MyClass();
mc.myMethod();               // OK (but not necessary)
mc.myOtherMethod();          // OK
```

Keep in mind that in a static method you can only use other static elements (attributes or methods), but not any non-static method existing outside of the method. For instance, this could not be done, since `myOtherMethod` is not static:

```
public static void myMethod()
{
    myOtherMethod();
}

public void myOtherMethod()
{
    ...
}
```

If we want to use this method, then we need to instantiate a `MyClass` object inside the static method, and then call this other method:

```
public static void myMethod()
{
    MyClass mc = new MyClass();
    mc.myOtherMethod();
}

public void myOtherMethod()
{
    ...
}
```

You can use many other static methods existing in Java. For instance, `Math` class has a lot of them, such as `Math.pow(...)`, or `Math.sqrt(...)`.

1.1. Instance elements vs class elements

At the beginning of this unit we've talked about *instance variables* referring to attributes. We can talk about *instance elements* to refer to any element in a class (either an attribute or a method) that belongs to a given object, this is, we need to instantiate an object to use or access this element.

On the other hand, we have *class elements* (either class attributes or methods), this is, elements that don't belong to a particular object, so we don't need to instantiate any object to use them. This is what we get when we use *static* in our classes.

2. Final elements

final modifier lets us define elements that can't be modified. If we apply this modifier to an attribute or variable, we are defining a constant: the element can't be re-assigned to any other value:

```
final int number = 3;

// ERROR
number = 6;
```

We can also apply this modifier to methods, or even classes. If we apply the modifier to a method, we are indicating that this method can't be overriden by any possible child class. Regarding classes, a final class can't be inherited.