Static/Class Methods

Source	http://www.leepoint.net/notes-java/flow/methods/50static-methods.html

There are two types of methods.

- **Instance methods** are associated with an object and use the instance variables of that object. This is the default.
- **Static methods** use no instance variables of any object of the class they are defined in. If you define a method to be static, you will be given a rude message by the compiler if you try to access any instance variables. You can access static variables, but except for constants, this is unusual. Static methods typically take all they data from parameters and compute something from those parameters, with no reference to variables. This is typical of methods which do some kind of generic calculation. A good example of this are the many utility methods in the predefined `Math` class. (See Math and java.util.Random).

Qualifying a static call

From outside the defining class, an instance method is called by prefixing it with an *object*, which is then passed as an implicit parameter to the instance method, eg, `inputTF.setText("");`

A static method is called by prefixing it with a *class name*, eg, `Math.max(i,j);`. Curiously, it can also be qualified with an object, which will be ignored, but the class of the object will be used.

Example

Here is a typical static method.

```
class MyUtils {

    . . .

    //================================================ mean
    public static double mean(int[] p) {
        int sum = 0;  // sum of all the elements
        for (int i=0; i<p.length; i++) {
            sum += p[i];
        }
        return ((double)sum) / p.length;
    }//endmethod mean

    . . .
}
```

The only data this method uses or changes is from parameters (or local variables of course).

Why declare a method `static`

The above `mean()` method would work just as well if it wasn't declared `static`, as long as it was called from within the same class. If called from outside the class and it wasn't declared static, it would have to be qualified (uselessly) with an object. Even when used within the class, there are good reasons to define a method as static when it could be.

- **Documentation**. Anyone seeing that a method is static will know how to call it (see below). Similarly, any programmer looking at the code will know that a `static` method can't interact with instance variables, which makes reading and debugging easier.
- **Efficiency**. A compiler will usually produce slightly more efficient code because no implicit object parameter has to be passed to the method.

Calling static methods

There are two cases.

Called from within the same class

Just write the static method name. Eg,

```
// Called from inside the MyUtils class
double avgAtt = mean(attendance);
```

Called from outside the class

If a method (static or instance) is called from another class, something must be given before the method name to specify the class where the method is defined. For instance methods, this is the object that the method will access. For static methods, the class name should be specified. Eg,

```
// Called from outside the MyUtils class.
double avgAtt = MyUtils.mean(attendance);
```

If an object is specified before it, the object value will be ignored and the the class of the object will be used.

Accessing static variables

Altho a `static` method can't access instance variables, it can access `static` variables. A common use of `static` variables is to define "constants". Examples from the Java library are `Math.PI` or `Color.RED`. They are qualified with the class name, so you know they are `static`. Any method, `static` or not, can access static variables. Instance variables can be accessed only by instance methods.

~~~ End of Article ~~~