Summary 6 – Static methods and fields

- What is this?

*A **static method** is a method that is not invoked on an object.*

For example, here are two method calls, one for a static method and another for a non-static method:

```
Math.sqrt(8.5)
"Hello, World".replace('o', '*')
```

The call to the static method sqrt has a single explicit parameter, 8.5.

The call to the non-static replace method has two explicit parameters, the characters 'o' and '*'. ***It also has an implicit parameter, namely, the String "Hello, World" on which the replace method acts.***

As the example shows, we put the class name (here, Math) in front of the method call for static methods, instead of putting an instance of a class. (All methods in Java belong to *some* class – no orphans!)

Recall that *main* is a static method. All the methods of the Math class are static.

One way to think of static methods is that they are "non-object-oriented" methods.

*A **static field** is a field that is shared by all instances of the class.*
For example, here are two fields of the Eye class in JavaEyes, one static and three that are not static:

```
private static final int DEFAULT_RADIUS = 25;
protected EyeBall eyeBall;
protected Color eyeColor;
protected int eyeRadius;
```

The above says that all Eye's share the same *default* radius (namely, 25), but each Eye has its own eyeball, eye color and *actual* eye radius.

As you would expect, static methods cannot refer to non-static fields.

As an aside, the *final* keyword in the first example above is a signal to the compiler to forbid any statement that might change the DEFAULT_RADIUS variable after its initialization to 25. It is commonplace to see static final variables used as constants. As the example shows, the style is to name such constants using all CAPS and underscores.

- For further study:
  - *Big Java*, section 8.6 *Static Methods*
  - *Big Java*, section 8.7 *Static Fields*
  - This summary's *author:* David Mutchler
  - See also the Summary on:
    - *The ThisKeyword*