Unit 8: Class Methods

After this unit, students should:

- understand the differences between instance methods and class methods
- be able to define and use a class method
- know that the `main` method is the entry point to a Java program
- the modifies and parameters required for a `main` method

Let's suppose that, in our program, we wish to assign a unique integer identifier to every `Circle` object ever created. We can do this with the additions below:

```
1  class Circle {
2    private double x;  // x-coordinate of the center
3    private double y;  // y-coordinate of the center
4    private double r;  // the length of the radius
5    private final int id; // identifier
6    private static int lastId = 0; // the id of the latest circle instance
7
8    /**
9     * Create a circle centered on (x, y) with a given radius
10    */
11   public Circle(double x, double y, double r) {
12     this.x = x;
13     this.y = y;
14     this.r = r;
15     this.id = Circle.lastId;
16     Circle.lastId += 1;
17   }
18
19   /**
20    * Return how many circles have ever existed.
21    */
22   public static int getNumOfCircles() {
23     return Circle.lastId;
24   }
25 }
```

- On Line 5, we added a new instance field `id` to store the identifier of the circle. Note that, since the identifier of a circle should not change once it is created, we use the keyword `final` here.
- On Line 6, we added a new class field `lastId` to remember that the `lastId` of the latest circle instance. This field is maintained as part of the class `Circle` and is initialized to 0.

- On Line 15 and 16, as part of the constructor, we initialize `id` to `lastId` and increment `lastId`. We explicitly access `lastId` through `Circle` to make it clear that `lastId` is a class field.

Note that all of the above are done privately beneath the abstraction barrier.

Since `lastId` is incremented by one every time a circle is created, we can also interpret `lastId` as the number of circles created so far. On Line 22-24, we added a method `getNumOfCircles` to return its value.

The interesting thing here is that we declare `getNumOfCircles` with a `static` keyword. Similar to a `static` field, a `static` method is associated with a class, not to an instance of the class. Such method is called a *class method*. A class method is always invoked without being attached to an instance, and so it cannot access its instance fields or call other of its instance methods. The reference `this` has no meaning within a class method. Furthermore, just like a class field, a class method should be accessed through the class. For example, `Circle.getNumOfCircles()`.

Other examples of class methods include the methods provided in `java.lang.Math`: `sqrt`, `min`, etc. These methods can be invoked through the `Math` class: e.g., `Math.sqrt(x)`.

The `main` method

The most common class method you will use is probably the `main` method.

Every Java program has a class method called `main`, which serves as the entry point to the program. To run a Java program, we need to tell the JVM the class whose `main` method should be invoked first. In the example that we have seen,

```
1  java Hello
```

will invoke the `main` method defined within the class `Hello` to kick start the execution of the program.

The `main` method must be defined in the following way:

```
1  public final static void main(String[] args) {
2  }
```

You have learned what `public` and `static` means. The return type `void` indicates that `main` must not return a value. We have discussed what `final` means on a field, but are not ready to explain what `final` means on a method yet.

The `main` method takes in an array (`[]`) of strings as parameters. These are the command-line arguments that we can pass in when invoking `java`. `String` (or `java.lang.String`) is another class provided by the Java library that encapsulates a sequence of characters.