

Class Methods

All the methods defined in the last two sections (“Defining Classes” and “Extending Classes”) are “instance methods” — they are called via instances (objects) of a class. However, a class definition can also include “class methods” (also called “static methods”). A class method is not called through an instance of a class. Class methods provide useful services related to the class, but they do not require an object of the class to be instantiated to call the method.

We have met several classes already that include class methods. In fact, the `Math` class only contains class methods. We do not call the methods through `Math` objects (a `Math` object cannot be instantiated!); rather, they are called simply prefixing the method by the name of the class, for example

```
double x = Math.sqrt(25.0);
```

The `sqrt()` method is a class method defined in the `Math` class, and it provides a useful service — computing the square root of a value.

The `String` class includes class methods *and* instance methods. The instance methods are called through instances of the `String` class, for example

```
String s1 = "Java is fun!";  
int x = s1.indexOf("!"); // indexOf() is an instance method
```

The object `s1` is an instance of the `String` class, and the method `indexOf()` is an instance method of the class. Because it is an instance method, it is called through an instance of the `String` class, in this case, `s1`.

The `String` class also includes “class methods” that provide useful services related to strings. They are called by prefixing the method with the name of the class, for example

```
String s = String.valueOf(25.0); // valueOf() is a class method
```

The `valueOf()` method is a class method that receives a numeric argument and returns a string representation of the argument. This is a useful service related to strings, but the method is not called through an instance of the `String` class. The prefix “`String.`” simply identifies to the compiler the name of the class in which the method definition is found.

A class method is defined using the `static` modifier. Of course, we see an example of this in every Java application, because the `main()` method must be declared `static`. We also used the `static` modifier in other programs — to define additional methods useful for a program’s task. For example, the `NumberStats` program presented in Chapter 6 included a class method called `mean()` with the following signature:

```
public static double mean(double n[], int length)
```

Since `mean()` is a “`public static`” method, it can be used in other programs, without copying the source code from `NumberStats.java`. In this sense, it is just like the class methods of the `Math` or `String` classes just described. We use the method simply by prefixing it with the name of the class in which it is defined. For example, to compute the mean of the first 15 elements in a `double` array named `scores`, we could use the following statement:

```
double z = NumberStats.mean(scores, 15);
```

This statement will work in any program, provided the `NumberStats.class` file is in the same directory as the program.

The `mean()` method above is an example of a method that provides a "useful service". In fact, it's the sort of method we might like to use frequently, in a variety of different Java programs. Now, instead of having to remember where each and every such method is located, it might be worthwhile to define a class to hold these methods. Figure 1 shows a partial listing of a class called `Misc` (for "miscellaneous") to hold class methods like `mean()`.

```
1 public class Misc
2 {
3     public static double min(double[] n) { ... }
4
5     public static double min(double[] n, int length) { ... }
6
7     public static double max(double[] n) { ... }
8
9     public static double max(double[] n, int length)
10
11    public static double mean(double n[]) { ... }
12
13    public static double mean(double n[], int length) { ... }
14
15    public static double sd(double[] n) { ... }
16
17    public static double sd(double[] n, int length) { ... }
18
19    public static String formatInt(int i, int len) { ... }
20
21    public static String formatDouble(double d, int len, int dp) { ... }
22
23    public static double trim(double d, int dp) { ... }
24 }
```

Figure 1. `Misc.java` (partial listing)

The method definitions are omitted in Figure 1 to keep the listing short. The source code for each was simply copied from demo programs presented earlier (see `NumberStats` and `NumberStats2` in Chapter 6, and `TableOfSquareRoots` in Chapter 5).

With all these methods now in a convenient location, it is possible to use them with the prefix "`Misc.`". So, for example, the code fragment

```
double x = 10.0;
double y = 3.0;
double z = x / y;
System.out.println("Answer is " + Misc.trim(z, 2));
```

generates the following output:

```
Answer is 3.33
```

The `trim()` method, if you recall from Chapter 5, trims a `double` to two decimal places of precision. If the code for the `trim()` method is placed in the `Misc` class, as in Figure 1, the method is called simply by prefixing it with "`Misc.`", as shown above.

Note as well in Figure 1 that method overloading is taking place. Several methods appear twice with the same name. There are two versions of `mean()` for example. One computes the mean of all elements in an array (line 11), and the other computes the mean of the first `length` elements (line 13). Because the number of arguments differs, it is straightforward for the compiler to determine which to use.