CS 106A
Summer 2013

Handout 16
July 08, 2013

Methods

Based on a handout by Eric Roberts

Chapter 5 introduces the concept of methods in Java, which are analogous to those you created in Karel. In contrast to the topics of expressions and control statements, the material in Chapter 5 involves far fewer details. On the other hand, the ideas introduced in this chapter are far more important. If you plot overall importance on a scale of 1 to 10, the **`switch`** statement probably weighs in somewhere around 2 (and I'm being very generous here... poor old **`switch`** statement); methods are definitely a 10.

The common idea that links methods in Karel and Java is that both provide a service to other, higher-level parts of the program and therefore act as tools. In both languages, the **`run`** method can call subsidiary methods to accomplish parts of the overall task. Those methods in turn call other methods that perform simpler operations, and so on. The caller views the method in terms of the effect it accomplishes. The method supplies all the details about how that operation is done. By hiding the details of complex operations, methods simplify the conceptual structure of a program considerably and allow you as a programmer to view it at varying levels of detail.

The fundamental difference between methods in Karel and their counterparts in Java is that Java makes it possible for data to pass back and forth between the caller and the method. Callers supply information to the method by supplying **arguments;** methods give information back to their callers by **returning results.** The entire process of passing this data between the two levels is in many respects the most important issue for you to understand in Chapter 5. In particular, you should take note of the following:

- Methods can be applied to other objects. In this case, the syntax of the call is

  receiver`.`*name*`(`*arguments*`)`

- Arguments in the calling method are assigned to the corresponding formal parameters in the callee according to their position in the argument list. Thus, the first argument is assigned to the first parameter name, the second to the second, and so on. The names of the variables are completely irrelevant to this process.
- Arguments are copied rather than shared. If you change the value of a formal parameter, the corresponding actual argument—even if it is a variable with the same name—is unaffected.
- The **`return`** statement causes a method to return immediately to its caller and also indicates the value to be returned as a result.

Methods can return values of any of the types you have encountered so far. Most of you will have little trouble with methods that return numeric data because you are familiar with this concept from high-school algebra. For some reason, methods that return objects or Boolean data seem harder, although the basic idea is precisely the same. Methods that return Boolean values, which are usually called **predicate methods,** are extremely important to programming.