Methods in Java – Method Overloading

Parameters & Data Types

When using a method, we generally expect that the data passed (e.g., `int`, `boolean`, `char`) will match the *parameter list* of the method (i.e., the definition of the parameters of the method).

Some data types, however, are compatible with others. When dealing with compatible data types, Java will automatically make any necessary changes. For example, there are actually four different integer data types: `byte` (8 bits), `short` (16 bits), `int` (32 bits), and `long` (64 bits). It should be obvious that any of the smaller integer data types will easily fit into a larger type, with the long data type able to accommodate any of the others. This is known as *widening*.

Most, but not all, of the compatible data types are fairly intuitive. The following table summarizes the permissible (automatic) conversions. No data will be lost in making these conversions.

From	To
`byte`	`short, int, long, float, double`
`short`	`int, long, float, double`
`char`	`int, long, float, double`
`int`	`long, float, double`
`long`	`float, double`
`float`	`double`

Changing the Number and Type of Arguments

In some cases, it is desirable to have a method that will accept a wide variety of arguments. Changes in the arguments may involve more dramatic changes in data types, or even in the number of arguments passed to the method. We have already worked with an example of such a method: The `System.out.println` method performs correctly regardless of the number or type of argument we give it. This is an example of *method overloading*.

One of the most common forms of method overloading is to allow a different number of parameters.

Example 1 – A Method to Simulate Rolling a Single Die

1. `public static void rollDie()`

   This method accepts no parameters. It simply outputs the result of a single roll from 1 to 6.

2. `public static void rollDie(int numRolls)`

   This method accepts a single parameter. It will output the results the specified number of rolls of the single, 6-sided die.

3. `public static void rollDie(int numRolls, int numFaces)`

   The number of rolls is the same as the previous method. The `numFaces` specifies the number of faces on each die (traditionally 6, but any positive integer is possible).

Methods in Java – Method Overloading

When an overloaded method is called, the Java compiler performs *signature matching*. This is the process where the compiler searches for a match in the method *identifier*, the *number of parameters*, and the *type(s) of parameters*.

On the first pass, the compiler will look for an exact match. If this fails, the compiler will also consider matches that are automatically convertible by data type (widening). The method with the same number of parameters that requires the least amount of widening will be used. If these criteria cannot be met, the call is invalid (and will produce some sort of error message by the compiler).

It is also possible to have multiple matches. In such a case, the method call is *ambiguous*, and is invalid. An error will be generated.

Exercises

1. Research the Java `Math` Class and find an example of an overloaded method.

2. Complete the definitions of the methods from Example 1 and write a main method to test them. If rolls is less than one or faces is less than 4, the methods should produce an error message.

3. Suppose that two method definitions have these headers:

   A: `public static void foo (int m, long n)`

   B: `public static void foo (int i, int j)`

   For each of the following calls, state, with reasons, whether or not the call is valid and, if it is valid, which version of `foo` would be called.

   (a) `foo(8, 4);`
   (b) `foo('A', 5);`
   (c) `foo(4, 0.5);`
   (d) `foo('x', 'y');`

4. Write methods called average, each have two arguments, that will return the average of those two values. One method should have integer arguments, and return an `int` value. The other should accept floating point numbers and return their average as a `double`. Write a main method for testing.