

1 Input, Output, Formatting

One of the most painful parts of any programming experience is always that of input and output. This is painful only because in doing I/O one has to interact with the operating system and the file system, and thus one has to interact with things outside the sphere of one's own particular program in execution in memory. Since the OS and the files change over time, one has to be very careful if one is going to write code that runs correctly. Also, in interacting with a human user, one has to bulletproof the input so that no matter how unusual the input is, one can correctly determine if it makes sense.

The simplest way to do input from the console, and the one we will use pretty much exclusively in this course, is the `Scanner` class. A standard use of this class is shown in `Main1` on the website, and is shown below. A nice thing about the `Scanner` class is that one can test for the presence of a data input of the correct type rather than write error-handling code. This is simply easier, and it is highly recommended.

The simplest way to do output to the console is with the `System.out.println()` statement. This in essence causes the variables, regardless of type, to be converted into strings of a nature appropriate for printing out (integers for integers, numbers with decimal points for doubles, etc.). When one is simply debugging code, the `println` may be the easiest way to get debugging output printed.

When one needs more formal output, however, the `System.out.printf()` command is much to be preferred. This is a language construct that is part of Java 5.0 and higher, but is not part of Java 1.4, so if you wind up using an early version of Java you will get compiler errors. The syntax for this is `System.out.printf("format string", variable names separated by commas)`; The format string needs to have formatting information that lines up in type and number with the type and number of the variables whose names appear separated by commas. The format string can also have text included, and blanks included in the format string are significant in that they cause blank spaces to be produced in the output.

The basic formatting rules include the following

- `%d`, for integers printed with however many spaces are needed
- `%10d`, for integers printed in a field exactly ten spaces wide

- `%12d`, for integers printed in a field exactly 12 spaces wide
- `%f`, for doubles printed with however many spaces are needed
- `%10.2f` for doubles printed with a 10-space field and two decimal places
- `%24.20f` for doubles printed with a 24-space field and 20 decimal places
- `%e` for doubles printed in scientific notation
- `%10.2e` for doubles printed in scientific notation in a field ten spaces wide with two decimal places
- `%24.20e` for doubles printed in scientific notation in a field 24 spaces wide with 20 decimal places

Note that when printing doubles, the number is rounded to fit the number of decimal places asked for in the format. This can be misleading. If you think the number is supposed to be larger than 90.20, for example, but because of roundoff the number in the machine has become 90.1999999, it will print as 90.20 in a `%10.2f` format, but it will fail a test `X > 90.2`.

Note also that you have to allow for the decimal point as a space-filling item, and for the minus sign, exponents in scientific notation, etc.

Note that the `printf` statement, unlike the `println` statement, does not automatically cause a linefeed to be generated. Two `println` statements will cause two lines to be printed. The linefeed is included as the `%n` at the very end of the `printf`. This allows you, if you've got complicated output lines, to print the different parts of an output line with several successive `printf` statements, or to print variable length output lines.