# PrintStream and printf

You have already seen multiple time a basic way to print text and messages in programs:

```
System.out.println("hello!");
```

If we deconstruct this, we see three parts:

- `System.out` is the object that provides a service;
- `println` is the service provided, to print a message;
- `"hello!"` is the argument to the service, the message to be printed.

Next to `println`, `System.out` provides other services to *print out* values towards the user.

`System.out` is an object of type `PrintStream`. The services offered by `System.out` are provided by any object of the type `PrintStream`.

## Services provided by `PrintStream`

The full documentation of the services offered by `PrintStream` is available in the JDK manual; for the time being, you need only remember the following:

**`println(...)`** Print a message, then begin a new line of text afterwards. You knew this already.

**`print(...)`** Print a message, but do not begin a new line of text afterwards. The next use of `print` or `println` will append its message to the same line of output. For example:

```
System.out.print("hello");
System.out.print("world");
```

will print "helloworld" on a single line.

**`printf(...)`** Format zero or more values according to a format specification. See below for details.

## Formatted output

A common task to perform in programs is "alignment": print data in columns, ensuring that values under each other are properly aligned. Another common task is to format numbers according to some application-defined standard, for example "all grades should be printed with one decimal of accuracy" to say the program should print "7.0" and not simply "7".

Unfortunately, `print` and `println` do not know anything about alignment, decimal places and the like. To perform these additional tasks, the service `printf` is provided.

It is defined as follows:

```
printf(String format, ...)
```

This syntax indicates `printf` takes at least one argument, something that evaluates to a string, and then zero or more additional arguments, which can be of any type.

The semantics of `printf` are as follows:

1. `printf` begins processing at the start of the format string;
2. if the current character in the format string is not a "`%`" character, it is printed to the output as-is (unchanged);
3. otherwise, `printf` looks after the `%` sign until it encounters one of the following:
   - a "`d`": the next argument in the remaining argument list is printed out in decimal;
   - a "`s`": the next argument in the remaining argument list is printed as a string;
   - a "`c`": the next argument in the remaining argument list is printed as a character;
   - a "`f`": the next argument is printed as a number with a decimal point;
   - a "`e`": the next argument is printed as a decimal number in scientific notation;
   - a "`%`": the character "`%`" is printed;
4. if there are more characters in the format string, then `printf` moves to the next character and proceeds at step #2 above.

Some examples:

| **Program text** | **What is printed** |
|---|---|
| `printf("")` | (nothing) |
| `printf("hello")` | `hello` |
| `printf("hi %d", 123)` | `hi 123` |
| `printf("hi %d, again", 123)` | `hi 123, again` |
| `printf("%d - %d:  %d", 4, 8, 12)` | `4 - 8:  12` |
| `printf("hi %s!", "bob")` | `hi bob!` |
| `printf("%s%s", "your", "world")` | `yourworld` |
| `printf("score = %d%%.", 13)` | `score = 13%.` |
| `printf("pi = %f", 3.14)` | `pi = 3.140000` |
| `printf("10^3 = %e", 1000)` | `10^3 = 1.000000e+03` |

As you can see, each combination of `%` followed by a character in the format string becomes a "hole" to be filled in by one of the remaining arguments. This group formed by a `%` followed by a special character is called a *format specifier*. The last character in a format specifier (`s`, `c`, `d`, `f`, etc) is called the *format conversion*. There exist other conversions, but in this course you will mostly use `%d`, `%s`, `%c`, `%f` and `%%`.

## Format specifier

The definition above says about specifiers, "`printf` looks after the `%` sign *until* it encounters one of the following". So there it is possible to express additional information between the "%" sign and the conversion, the special character that decides the type of value to convert. This extra information is used to control *how the formatting is done*.

In this course, you will use use the following format specifiers:

| Conversion | Meaning | Example | Prints |
|---|---|---|---|
| `%f` | Print the data as-is | `printf(":%d:", 12)` | `:12:` |
| `%Nf` | Print at least N columns, align right | `printf(":%5d:", 12)` | `:  12:` |
| `%.Mf` | Print M decimals (for floats only) | `printf(":%.3f:", 3.14)` | `:3.140:` |
| `%N.Mf` | Align right to N columns, print M decimals | `printf(":%5.1f:", 3.14)` | `:  3.1:` |
| `%0Nf` | Print at least N columns, align right, fill with 0 | `printf(":%05d:", 12)` | `:00012:` |
| `%-Nf` | Print at least N columns, align left | `printf("%-5d:", 12)` | `:12  :` |

## New line special character

`printf` behaves like `print` in that it never starts a new line in the output by itself. So if you have the following code:

```
System.out.printf("hello %d", 123);
System.out.printf("world");
```

the program will print "`hello 123world`" on a single line.

It is possible to force new lines using a special character combination: "`\n`" (a backslash, then small letter `n`). So you can reproduce the behavior of `println` for example as follows:

```
System.out.printf("hello\n");
```

It is also possible to use "`\n`" in the middle of a format string, for example:

```
System.out.println("hello,\nworld!\n");
```

will print "hello," on a first line, then "world!" on a second line.

> **Tip**
> "`\n`" is one of multiple special combinations that can be found in strings. This will be handled in a later lecture on strings and character literals.

## Important concepts

- the difference between `print`, `println` and `printf`;
- how `printf` interprets its *format string*;

- the difference between *format conversion* (1 character `s`, `f`, etc.) and *format specifier* (the entire group from `%` to conversion);
- the "basic" *format conversions*: `s`, `f`, `d`, `c`, `%`;
- how to extend format specifiers for alignment and changing the number of decimals;
- using the special combination "`\n`".

## Further reading

- Introduction to Programming, section 2.4.1 (pp. 38-39)
- Absolute Java, section 2.1 (pp. 58-67), also exercises 1-7

---

## Copyright and licensing