Input and Output

InputStreams

- An **InputStream** is a class that receives data from an input device such as keyboard, mouse, or touchscreen.

- System.in is a predefined input stream object reference that is associated with a system's standard input, which is usually a keyboard.

- **Methods**: nextInt, next, nextLine, nextDouble.

OutputStreams

•An **OutputStream** is a class that supports output data to a screen, file, or elsewhere.

•System.out is a predefined OutputStream object reference that is associated with a system's standard output, usually a computer screen.

•**Methods**: print, println, printf.

•Non-string objects are converted to string by the print methods.

- primitive types
- reference types

Use this object to call functions. print, println,and printf

- print() – converts parameter to a string (if not already one) and prints it out
- println() – prints parameter, and also prints a newline after
- printf() –Formatted string, followed by parameters to ”fill in the blanks”

Example

```
System.out.print("Hello, World"); // no newline
```

```
Hello, World
```

```
System.out.println("Hello\nWorld"); // adds newline at end
```

```
Hello
World
```

```
int feet = 6, inches = 3;
System.out.printf("I am %d feet and %d inches tall\n", feet,
inches);
I am 6 feet and 3 inches tall
```

Concatenation

If the + operator is used with at least one string operand, then th operation is string concatenation.

Other types will be auto-converted to type string if needed

```
System.out.println("The number of states in the U.S.
                                              is " + 50);


int sides = 8;
System.out.println("Number of sides on a stop
                                    sign = " + sides);
```

Output Formatting

•A programmer can adjust the way that a program's output appears, a task known as **output formatting**.

•The first argument of the printf() method, the **format string**, specifies the format of the text to print along with any number of **format specifiers** for printing numeric values.

•A format specifier begins with the % character followed by another character that indicates the value type to be printed.

Formatting text with `printf`

System.out.printf("**format string**", **parameters**);

A format string can contain *placeholders* to insert parameters:

- %d integer
- %f real number
- %s string
  - these placeholders are used instead of + concatenation

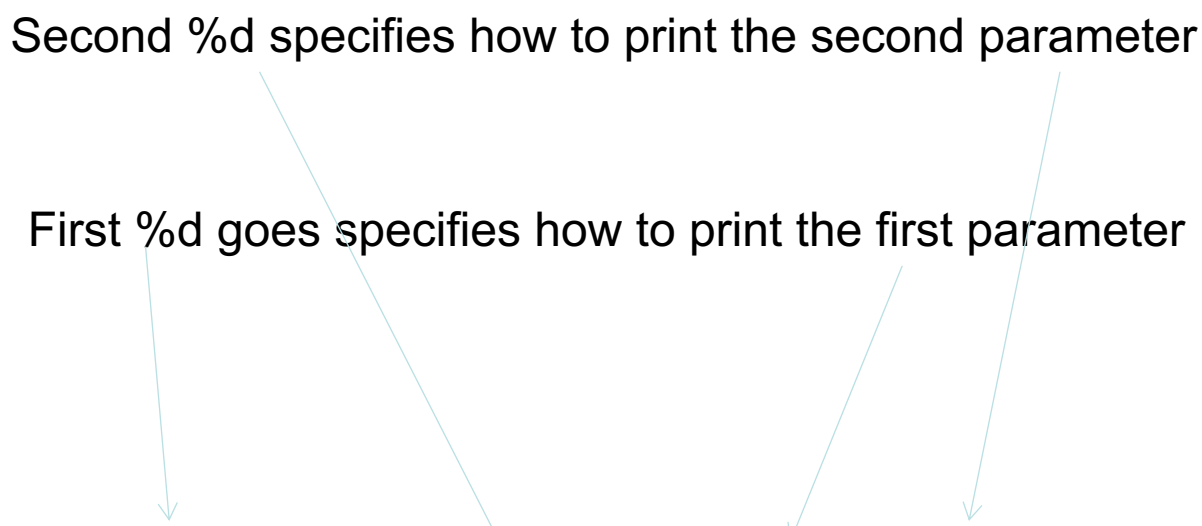
- Example:

```
int value1 = 3;
int value2 = -17;
System.out.printf("x is %d and y is %d!\n", value1, value2);
                  // x is 3 and y is -17!
```

  - printf does not drop to the next line unless you write \n

Formatting text with `printf`

System.out.printf("**format string**", **parameters**);




```
int value1 = 3;
int vaule2 = -17;
System.out.printf("x is %d and y is %d!\n", value1, vaule2);
                        // x is 3 and y is -17!
```

printf does not drop to the next line unless you write \n

9

Example:

```
String account = "Prime";
double total = 210.35;
int years = 5;
                    Prime                  210.35      5
System.out.printf("The %s account saved you $%f over %d years\n",
    account, total, years);
       %s      %f      %d
```

The Prime account saved you $210.350000 over 5 years.

Captions

1. The printf() format string contains 3 format specifiers: %s, %f, and %d.
2. The 3 arguments after the format string are matched with the format specifiers. %s = account, %f = total, %d = years
3. The string is printed with the 3 variables substituted into the format string.

Table 5.2.1: Format specifiers for the printf() and format() methods.

Format specifier	Data type(s)	Notes
%c	char	Prints a single Unicode character
%d	int, long, short	Prints a decimal integer value.
%o	int, long, short	Prints an octal integer value.
%h	int, char, long, short	Prints a hexadecimal integer value.
%f	float, double	Prints a floating-point value.
%e	float, double	Prints a floating-point value in scientific notation.
%s	String	Prints the characters in a String variable or literal.
%%		Prints the "%" character.
%n		Prints the platform-specific new-line character.

Example

```
int numStudents = 25;
char letterGrade = 'A';
double gpa = 3.95;

System.out.printf("There are %d students\n",
                                numStudents);
System.out.printf("Bobby's course grade was %c, and
                  his GPA is %f\n", letterGrade, gpa);

// The output from this example is:
//    There are 25 students
//    Bobby's course grade was A, and his GPA is 3.950000
```

Floating-point values

- Formatting floating-point output is commonly done using sub-specifiers. A ***sub-specifier*** provides formatting options for a format specifier and are included between the % and format specifier character.

- Ex: The .1 sub-specifier in printf("%.1f", myFloat);causes the floating-point variable myFloat to be output with only 1 digit after the decimal point; if myFloat is 12.34, the output would be 12.3. Format specifiers and sub-specifiers use the following form:

Construct 5.2.1: Format specifiers and sub-specifiers.

```
%(flags)(width)(.precision)specifier
```

The table below summarizes the floating-point sub-specifiers available. Assume myFloat has a value of 12.34. Recall that %f is used for floating-point values and %e is used to display floating-point values in scientific notation.

Table 5.2.2: Floating-point formatting.

Method calls to printf() apply to PrintStream objects like System.out.

Sub-specifier	Description	Example
width	Specifies the minimum number of characters to print. If the formatted value has more characters than the width, the value will not be truncated. If the formatted value has fewer characters than the width, the output will be padded with spaces (or 0's if the '0' flag is specified).	`printf("Value: %7.2f", myFloat);` `Value:   12.34`

.precision	Specifies the number of digits to print following the decimal point. If the precision is not specified, a default precision of 6 is used.	`printf("%.4f", myFloat);`
		`12.3400`
		
		`printf("%3.4e", myFloat);`
		`1.2340e+01`

flags	-: Left aligns the output given the specified width, padding the output with spaces.
	+: Prints a preceding + sign for positive values. Negative numbers are always printed with the - sign.
	0: Pads the output with 0's when the formatted value has fewer characters than the width.
	space: Prints a preceding space for positive value.
		`printf("%+f", myFloat);`
		`+12.340000`
		`printf("%08.2f", myFloat);`
		`00012.34`

`printf` precision

- To specify how many decimal places for the output of a floating point value, modify the '%f' symbol in this format:

```
%.Df        // where D is the number of decimal places
```

Example:

```
double gpa = 3.275;
double PI = 3.1415;
System.out.printf("gpa = %.2f", gpa);
System.out.printf("PI = %.3f", PI);
```

Output is:

```
gpa = 3.28
 PI = 3.142
```

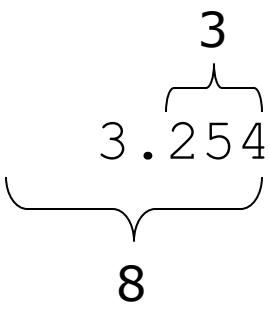
`printf` precision

- `%.Df` real number, rounded to **D** digits after decimal
- `%W.Df` real number, **W** chars wide, **D** digits after decimal
- `%-W.Df` real number, **W** wide (left-align), **D** after decimal

```
double gpa = 3.253764;
System.out.printf("your GPA is %.1f\n", gpa);
System.out.printf("more precisely: %8.3f\n", gpa);
```

Output:

```
your GPA is 3.3
more precisely:    3.254
```



`printf` width

- `%`**`W`**`f` real number, **W** characters wide, right-aligned

```
Example:
myFloat= 12.34f
printf("Value: %7.2f", myFloat);
printf("Value: %8.2f", myFloat);
```

Output:

```
Value:  12.34
Value:   12.34
```

printf flag

- `%-Wf` real number, W characters wide, *left*-aligned
- `%+Wf` real number, W characters wide, right-aligned

  Prints a preceding + sign for positive values. Negative numbers are always printed with the - sign.
- `%0Wf` real number, W characters wide, right-aligned

  Pads the output with 0's when the formatted value has fewer characters than the width.

Example:

```
myFloat= 12.34f

printf("Value: %-7.2f", myFloat);
printf("Value: %+7.2f", myFloat);
printf("Value: %07.2f", myFloat);
```

Output:

```
Value:12.34__
Value:_+12.34
Value:0012.34
```

Exercise

Print formatting - What is the output from the following print statements, assuming: float myFloat = 45.1342f;

System.out.printf("%09.3f", myFloat);
(answer: 00045.134)

System.out.printf("%.3e", myFloat);
(answer: 4.513e+01)

System.out.printf("%09.2f", myFloat);
(answer: 000045.13)

Integer values

Formatting of integer values is also done using sub-specifiers. The integer sub-specifiers are similar to the floating-point sub-specifiers except no .precision exists. For the table below, assume myInt is 301.

Method calls to printf() apply to PrintStream objects like System.out.

Sub-specifier	Description	Example
width	Specifies the minimum number of characters to print. If the formatted value has more characters than the width, the value will not be truncated. If the formatted value has fewer characters than the width, the output will be padded with spaces (or 0's if the '0' flag is specified).	`printf("Value: %7d", myInt);` `Value:     301`

flags	-: Left aligns the output given the specified width, padding the output with spaces.
	+: Print a preceding + sign for positive values. Negative numbers are always printed with the - sign.
	0: Pads the output with 0's when the formatted value has fewer characters than the width.
	space: Prints a preceding space for positive value.
		`printf("%+d", myInt);`
		`+301`
		
		`printf("%08d", myInt);`
		`00000301`
		
		`printf("%+08d", myInt);`
		`+0000301`

Print formatting - What is the output from the following print statements, assuming: int myInt= 301

printf("%+d", myInt);
(answer: +301

printf("%08d", myInt);
(answer: 00000301

printf("%+08d", myInt);
(answer: +0000301

- Print formatting - What is the output from the following print statements, assuming: int value of -713

printf("%+04d", myInt);
(answer: -713

printf("%05d", myInt);
(answer: -0713

printf("%+02d", myInt);
(answer: -713

Strings

Strings may be formatted using sub-specifiers. For the table below, assume the myString variable is "Formatting".

Table 5.2.4: String formatting.

Method calls to printf() apply to PrintStream objects like System.out.

Sub-specifier	Description	Example
width	Specifies the minimum number of characters to print. If the string has more characters than the width, the value will not be truncated. If the formatted value has fewer characters than the width, the output will be padded with spaces.	`printf("%20s String", myString);` `          Formatting String`
.precision	Specifies the maximum number of characters to print. If the string has more characters than the precision, the string will be truncated.	`printf("%.6s", myString);` `Format`
flags	-: Left aligns the output given the specified width, padding the output with spaces.	`printf("%-20s String", myString);` `Formatting           String`

- `%`**`w`**`s` string, Specifies the minimum number of characters to print.
- `%.`**`D`**`s` string, Specifies the maximum number of characters to print.
- `%-ws` string, w wide (left-align)

Print formatting - What is the output from the following print statements, assuming: String myString = "Testing";
(show all responses inside quotes " ")

printf("%4s", myString);
(answer: Testing

printf("%8s", myString);
(answer: Testing

printf("%.4s", myString);
(answer: Test

Examples of printf format specifiers:

Type	Specifier	Example	Output
int		%d	123
		%5d	123
		%-5d	128
double		%f	1.23
		%8.1f	1.2
		%8.4f	1.2300
		%-8.2f	1.23
String		%s	hello
		%10s	hello
		%-10s	hello

First value (5, 8 or 10 above) is the field width.
No value means as much space as required.
.value (e.g., .1, .4, .4) number of decimal places
- (minus) means left justified.

26

TABLE 4.5 Common Format Specifiers

Specifier	Result
`%d`	integer
`%8d`	integer, right-aligned, 8-space-wide field
`%-6d`	integer, left-aligned, 6-space-wide field
`%f`	floating-point number
`%12f`	floating-point number, right-aligned, 12-space-wide field
`%.2f`	floating-point number, rounded to nearest hundredth
`%16.3f`	floating-point number, rounded to nearest thousandth, 16-space-wide field
`%s`	string
`%8s`	string, right-aligned, 8-space-wide field
`%-9s`	string, left-aligned, 9-space-wide field

27