Input and Formatted Output (printf)

CSE 1310 – Introduction to Computers and Programming
University of Texas at Arlington

Book reference

- 2.3 Input and Output

User input with Scanner

```
import java.util.Scanner;
public class Example1 {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Please enter number of weeks: ");
    int weeks = in.nextInt();
    int days = weeks * 7;
    System.out.println("There are " + days + " days in " + weeks
+ " weeks");
  }
}
```

- There are several new things here:
  - the **import** line.
  - The **Scanner** object.
  - The **in.nextInt** method.
- The **Scanner** object allows us to obtain user input.
- To create a **Scanner** object, we need to:
  - Put the import statement at the top of the Java file.
  - Create a Scanner object, as shown in the first line of the main method:

```
Scanner in = new Scanner(System.in);
```

Input

- Import: java.util.Scanner
- Create a Scanner object:

Scanner kb = new Scanner(System.in);

(You can use any name instead of kb. E.g: Scanner in = new Scanner(System.in);)

- E.g. read in (from the user/keyboard):

Read data and store it as type:	Method call (on a Scanner object named kb)	Example instruction
int	kb.nextInt()	int n = kb.nextInt();
double	kb.nextDouble()	double salary = kb.nextDouble();
String	kb.nextLine()	String name = kb.nextLine(); (data until Enter, Enter dissapears)
String	kb.next()	String name = kb.next(); (data until space or Enter, Enter remains)

- Any of the methods above for reading data in, when executed, “halt the program” to allow the user to type data (the user has control). Only after the user hits Enter the program continues to execute (the control is given back to the program).

Entering more than one piece of data per line

```
int age=0;
double temp=0;
String nm = "";

System.out.println("Enter (separated by one space): age name number");
age = kb.nextInt();
nm = kb.next(); // NOT kb.nextLine()
temp = kb.nextDouble();

System.out.println("Hi " + nm + "!");
System.out.println("The outside temperature is " + temp);
System.out.println("Your age is " + age);
```

Sample run:

Enter (separated by one space): age name number
20 Sam 87.5
Hi Sam!
The outside temperature is 87.5
Your age is 20

next() vs nextLine()

Code	Output

System.out.print("What is your name? ");
 	
// nextLine reads the whole line (up to new line):
name = **kb.nextLine()**;
 	
System.out.println("Hello " + name);
	What is your name? Alex Stefan
	Hello **Alex Stefan**

System.out.print("What is your name? ");
 	
// next reads up to first separator (space, Enter, tab):
name = **kb.next()**;
 	
System.out.println("Hello " + name);
	What is your name? Alex Stefan
	Hello **Alex**

Why nextLine() seems to not work sometimes

- When the user presses Enter, the corresponding symbol (%n) is recorded in the input stream. So there will be a character there for the Enter (also called new-line character: %n or \n).
- nextInt(), nextDouble() and next() do NOT "eat" the Enter at the end of the line.
- nextLine() DOES consume the Enter from the end of the line.
- If nextLine() is used after a next()/nextInt()/nextDouble() it will just read the Enter left there from these methods. Solution: use an extra nextLine() just to consume that Enter.

Code	Output

System.out.print("What is your name? ");
name = **kb.next()**; // will NOT "eat" the Enter after Alex
System.out.println("Hello " + name);
System.out.print("What is your last name? ");
last = **kb.nextLine()**;
System.out.println("Hello " + last);
	What is your name? Alex
	Hello Alex
	What is your last name? Hello

System.out.print("What is your name? ");
name = **kb.next()**; // will NOT "eat" the Enter after Alex
System.out.println("Hello " + name);
System.out.print("What is your last name? ");
kb.nextLine(); // it will "eat" the Enter after Alex
last = **kb.nextLine()**;
System.out.println("Hello " + last);
	What is your name? Alex
	Hello Alex
	What is your last name? Stefan
	Hello Stefan

How the output screen “works”

- You always print:
  - Left to right and
  - From top down
- You can go back to the beginning of the line you are on with \r
- You can NEVER go up a line
  - So if you need to print something that looks like a table of numbers, you must print it row –by-row (according to the desired output). You cannot print it column-by-column.
- If you printed more than a screen length, it scrolls down so you always have the line you are on at the bottom. To see the earlier lines, scroll up.
- You can simulate clearing the screen by printing enough new lines to push the previous outputted text out of the visible range.
  - See this image.

System.out.printf()

Problem with output:

Code:

```
int hours = 100;
double days = 100/24.0; // used 24.0 to avoid integer division
System.out.print("days: ");
System.out.print(days);
```

Output:

```
days: 4.166666666666667
```

Solution:

Code:

```
int hours = 100;
double days = 100/24.0; // used 24.0 to avoid integer division
System.out.print("days: ");
System.out.printf("%.3f",days);
```

Output:

```
days: 4.167
```

System.out.printf()

- Used for “formatted printing”
- Uses *format specifiers*. They indicate the type. The format specifier must match the type of the value that will be printed.

Format specifier	Data type
%d	int (Integer)
%f	double, float, (numbers with decimal values)
%s	String
%b	Boolean
%c	char (a single symbol)

- Specify number of decimals to be displayed for real numbers: %**.3**f
- Specify a minimum width (number of spaces) when printing that value: %**10**.3f
- With the minimum width, text is aligned to the right by default. To align to the left use -: %**-**10.3f
- %n – moves a new line
- %% - prints the % symbol

Specifying Width

- After the % sign, you can put a number, specifying the minimum width to be used when printing that value. For example:
  - %5d means "allocate at least 5 spaces for that int".
  - %10s means "allocate at least 10 spaces for that string".
  - %7f means "allocate at least 7 spaces for that double".
  - %7.2f means "allocate at least 7 spaces for that double, but only two after the decimal point".
  - %.2f means "allocate as many spaces as needed for that double, but use only two of them after the decimal point".
- Use the width to print data aligned like in a table.

Examples

```
public class hello1 {
  public static void main(String[] args) {
        System.out.printf("%-10.2f%n", 18.0);      // left aligned: -
        System.out.printf("%10.2f%n", 20.0);       // right aligned
        System.out.printf("%10.3f", 10.2);         // no text
        System.out.printf("%n");                   // only %n
        System.out.printf("%10.2f%5d%n", 15.7,3);  // no text and %n
        System.out.printf("%10.2f%d%n", 15.7,3);   // no space before 3
        System.out.printf("%10.2f", 15.7);
        System.out.printf("%n%10.2f%n%5d%n", 11.3,8);
       }
}
```

Example Output:

```
18.00
      20.00
     10.200

      15.70     3
      15.703
      15.70
      11.30
     8
```

Notice the effect of **`%-10.2f`** for 18.00

Looks like one number, but 2 different numbers are printed here: 15.70 and 3

You can mix text and [multiple pieces of] data

Code:

```
System.out.printf("%s is %d years old","Jane", 23);
```

Output:

```
Jane is 23 years old
```

The format specifiers must match the arguments in the given order.

Here **%s** for **"Jane"** and **%d** for **23**

Escape sequences

```
public class hello1 {
  public static void main(String[] args) {
        System.out.printf("some\ttext%nmore text");
       System.out.printf("%nI\'m here\"\\");
        System.out.printf("%n");     }
}
```

Output:

some text
more text
I'm here"\

Escape sequences:

- %n - new line (note: use %n (platform appropriate), not \n)
- \t - tab
- \' - insert single quote in text
- \" - insert double quote in text
- \\ - insert backslash in text
- %% - insert % symbol in text : E.g. `System.out.printf("%%");`

Formatted Printing - References

- Java API: https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax
- Oracle Java tutorial (brief): https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

- FUN: You can also build a string (without printing it) using String.format(…). The format method works like the System.out.printf(…), but it returns a string.

```
String s;
s = String.format("There are %d days in %s%n", 31, "July");
System.out.println(s);
```

Print a table using printf

- Write a program that asks the user information for 3 cities as follows: 3 times it should ask for:
  - city name,
  - Area in kilometers (km),

  Next it should print a table with this information. The table will have 1 header and 3 rows (one row per city). Simulate horizontal and vertical lines in the table with |, - and + symbols.
- Problem solving:
  - Plan how you want the interface to look like
  - what type will you use for each piece of data
  - Print the information aligned
  - Add horizontal and vertical lines

System.out.printf – build the formatting string

- System.out.printf takes 1 or more arguments. The first must be a string and it is called *the formatting string*. It will include specifications for formatting the text such as the reversed width for displaying certain data.
- But the formatting string is just a string and it can be built before the method (printf) is called.
- Example: the user enters a width, w, (as int) and a String, s, and the program must print the string s on w reserved spaces.

```
System.out.print("Enter width: ");
int w = kb.nextInt();    // assume user enters 15
System.out.print("Enter text: ");
String s = kb.nextLine();  // Assume user enters CSE 1310
String fStr = "|%" + w + "s |";  //fStr is "|%10s|"
System.out.printf(fStr, s);
// Output: |      CSE 1310|
```