Lesson 2
Console Input and Output

Absolute Java

Walter Savitch

Formatting Output with printf()

- This method is used the same way as the method print but **allows you to add formatting instructions** that specify such things as the number of digits to include after a decimal point.
- **Syntax:**

  System.out.printf(format specifier, value/variable/expression)
- Example:

```
double price = 19.8;
System.out.printf("%6.2f", price);
//Displays " 19.80" (one blank followed by 19.80)
```

Formatting Output with printf()

- In this statement the first argument to printf is a string known as the format specifier, and the second argument is the number or other value to be output in that format.
- The format specifier %6.2f says to output a floating-point number in a field (number of spaces) of width 6 (room for six characters) and to show exactly two digits after the decimal point.
- Any extra blank space is added to the front (left-hand end) of the value output.
- The **f** means the output is a floating-point number, that is, a number with a decimal point.
- printf, like print, does not advance the output to the next line.

Formatting Output with printf()

- In this statement the first argument to printf is a string known as the format specifier, and the second argument is the number or other value to be output in that format.
- The character % signals the end of text to output and the start of the format specifier.
- The end of a format specifier is indicated by a conversion character (f in our example).
- The conversion character specifies the type of value that is output in the specified format.
- The first number specifies the total number of spaces used to output the value. If that number is larger than the number of spaces needed for the output, extra blanks are added to the beginning of the value output. If that number is smaller than the number of spaces needed for the output, enough extra space is added to allow the value to be output; no matter what field width is specified, printf uses enough space to fit in the entire value output.

Formatting Output with printf()

- Both of the numbers in a format specifier such as %6.2f are optional. You may omit either or both numbers, in which case Java chooses an appropriate default value or values (for example, %6f and %.2f).
- The dot goes with the second number.
- You can use a format specifier that is just a % followed by a conversion character, such as %f or %g, in which case Java decides on the format details for you. For example, the format specifier %f is equivalent to %.6f, meaning six spaces after the decimal point and no extra space around the output.

Format specifiers with printf()

Display 2.1 Format Specifiers for `System.out.printf`

CONVERSION CHARACTER	TYPE OF OUTPUT	EXAMPLES
d	Decimal (ordinary) integer	`%5d` `%d`
f	Fixed-point (everyday notation) floating point	`%6.2f` `%f`
e	E-notation floating point	`%8.3e` `%e`
g	General floating point (Java decides whether to use E-notation or not)	`%8.3g` `%g`
s	String	`%12s` `%s`
c	Character	`%2c` `%c`

Format specifiers with printf()

- The s and c formats, for strings and characters, may include one number that specifies the field width for outputting the value, such as %15s and %2c. If no number is given, the value is output with no leading or trailing blank space.
- **%n is used for new line**

Right justified and Left justified

- By default the values are displayed as right justified. When the value output does not fill the field width specified, blanks are added in front of the value output. The output is then said to be right justified.
- If you add a hyphen after the %, any extra blank space is placed after the value output, and the output is said to be left justified.

Format specifiers with printf()

Example

```
double value = 12.123;
System.out.printf("Start%8.2fEnd", value);
System.out.println();
System.out.printf("Start%-8.2fEnd", value);
System.out.println();
```

Output

The first line has three spaces before the 12.12, and the second has three spaces after the 12.12.

```
Start   12.12End
Start12.12   End
```

Formatting Output with printf()

- printf can output any number of values.
- The first argument always is a string known as the format string, which can be followed with any number of additional arguments, each of which is a value to output.
- The format string should include one format specifier, such as %6.2f or %s, for each value output, and they should be in the same order as the values to be output.
- **Example**

```
double price = 19.8;
String name = "magic apple";
System.out.printf("$%6.2f for each %s.", price, name);
System.out.println();
System.out.println("Wow");
```

- **Output**

```
$ 19.80 for each magic apple.
Wow
```

Scanner Class methods

Method for Scanner kbd;	Return Type	Description
`next()`	`String`	Returns the string value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.
`nextLine()`	`String`	Reads the rest of the current keyboard input line and returns the characters read as a value of type `String`. Note that the line terminator `'\n'` is read and discarded; it is not included in the string returned.
`nextInt()`	`int`	Returns the next keyboard input as a value of type `int`.
`nextDouble()`	`double`	Returns the next keyboard input as a value of type `double`.
`nextFloat()`	`float`	Returns the next keyboard input as a value of type `float`.
`nextLong()`	`long`	Returns the next keyboard input as a value of type `long`.
`nextByte()`	`byte`	Returns the next keyboard input as a value of type `byte`.
`nextShort()`	`short`	Returns the next keyboard input as a value of type `short`.
`nextBoolean()`	`boolean`	Returns the next keyboard input as a value of type `boolean`. The values of `true` and `false` are entered as the words true and false. Any combination of uppercase and lowercase letters is allowed in spelling true and false.
`useDelimiter` (*Delimiter_Word*)	`Scanner`	Makes the string *Delimiter_Word* the only delimiter used to separate input. Only the exact word will be a delimiter. In particular, blanks, line breaks, and other whitespace will no longer be delimiters unless they are a part of *Delimiter_Word*. This is a simple case of the use of the `useDelimiter` method. There are many ways to set the delimiters to various combinations of characters and words, but we will not go into them in this book.

Scanner Class methods cont.

- The method nextLine of the class Scanner reads the *remainder* of a line of text *starting wherever the last keyboard reading left off*.
- **Example:**

```
int n = keyboard.nextInt();
String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

Scanner Class methods cont.

- Now, assume that the input typed on the keyboard is the following:

  2
  heads are better than
  1 head.

- **You might expect**
  - the value of n to be set to 2,
  - the value of the variable s1 to "heads are better than", and
  - that of the variable s2 to "1 head".
- But that is not what happens. What actually happens is that
  - the value of the variable n is set to 2,
  - the variable s1 is set to the empty string,
  - the variable s2 to "heads are better than".
- **When combining different methods for reading from the keyboard, you sometimes have to include an extra invocation of nextLine to get rid of the end of a line (to get rid of a '\n').**

Scanner Class methods cont.

- **Other Input Delimiters:** When using the Scanner class for keyboard input, you can change the delimiters that separate keyboard input to almost any combination of characters.
- Example:

```
Scanner keyboard2 = new Scanner(System.in);
keyboard2.useDelimiter("##");
//Changes the delimiter for the object keyboard2 to "##"
String word1, word2;
word1 = keyboard2.next();
word2 = keyboard2.next();
```

- suppose the user enters the following keyboard input:

```
one two##three##
```

- The code would read the two strings "one two" and "three" and make them the values of the variables word1 and word2:

Importing Packages and Classes

- Libraries in Java are called **packages**. A package is simply a collection of classes that has been given a name and stored in such a way as to make it easily accessible to your Java programs. Java has a large number of standard packages that automatically come with Java. One such package is java.util.
- Scanner class is in java.util package.
- In order to use Scanner, you must import the class as follows:

import java.util.Scanner;