

# Chapter 2

## Console Input and Output

Prof. Choonhwa Lee

Dept. of Computer Science and Engineering  
Hanyang University

# `System.out.println` for console output

- `System.out` is an object that is part of the Java language
  - `println` is a method invoked by the `System.out` object that can be used for *console output*
    - The data to be output is given as an argument in parentheses
    - A plus sign is used to connect more than one item
    - Every invocation of `println` ends a line of output
- ```
System.out.println("The answer is " + 42);
```

# `println` Versus `print`

- Another method that can be invoked by the `System.out` object is `print`
- The `print` method is like `println`, except that it does not end a line
  - With `println`, the next output goes on a new line
  - With `print`, the next output goes on the same line

# Formatting Output with `printf`

- Starting with version 5.0, Java includes a method named `printf` that can be used to produce output in a specific format
- The Java method `printf` is similar to the `print` method
  - Like `print`, `printf` does not advance the output to the next line
- `System.out.printf` can have any number of arguments
  - The first argument is always a *format string* that contains one or more *format specifiers* for the remaining arguments
  - All the arguments except the first are values to be output to the screen

# printf Format Specifier

- The code

```
double price = 19.8;  
System.out.print("$");  
System.out.printf("%6.2f", price);  
System.out.println(" each");
```

will output the line

```
$ 19.80 each
```

- The format string "**%6.2f**" indicates the following:
  - End any text to be output and start the format specifier (**%**)
  - Display up to 6 right-justified characters, pad fewer than six characters on the left with blank spaces (i.e., *field width* is **6**)
  - Display exactly 2 digits after the decimal point (**.2**)
  - Display a floating point number, and end the format specifier (i.e., the *conversion character* is **f**)

# Right and Left Justification in `printf`

- The code

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

will output the following

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

- The format string `"Start%8.2fEnd"` produces output that is right justified with three blank spaces before the `12.12`
- The format string `"Start%-8.2fEnd"` produces output that is left justified with three blank spaces after the `12.12`

# Multiple arguments with `printf`

- The following code contains a `printf` statement having three arguments

- The code

```
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");
```

will output

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

- Note that the first argument is a format string containing two format specifiers (`%6.2f` and `%s`)
- These format specifiers match up with the two arguments that follow (`price` and `name`)

# Format Specifiers for `System.out.printf`

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

---

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



# The `printf` Method (Part 1 of 3)

## Display 2.2 The `printf` Method

---

```
1 public class PrintfDemo
2 {
3     public static void main(String[] args)
4     {
5         String aString = "abc";

6         System.out.println("String output:");
7         System.out.println("START1234567890");
8         System.out.printf("START%sEND %n", aString);
9         System.out.printf("START%4sEND %n", aString);
10        System.out.printf("START%2sEND %n", aString);
11        System.out.println();
```

(continued)

# The `printf` Method (Part 3 of 3)

## Display 2.2 The `printf` Method

### SAMPLE DIALOGUE

String output:


START1234567890

STARTabcEnd

START abcEnd

STARTabcEnd

*The value is always output. If the specified field width is too small, extra space is taken.*



Character output:

START1234567890

STARTZEND

START    ZEND

Floating-point output:

START1234567890

START12345.123457END

START12345.1235END

START12345.12END

START  12345.1235END

START1.234512e+04END

START 1.23451e+04END

*Note that the output is rounded, not truncated, when digits are discarded.*

# The `printf` Method (Part 2 of 3)

## Display 2.2 The `printf` Method

---

```
12      char oneCharacter = 'Z';

13      System.out.println("Character output:");
14      System.out.println("START1234567890");
15      System.out.printf("START%cEND %n", oneCharacter);
16      System.out.printf("START%4cEND %n", oneCharacter);
17      System.out.println();

18      double d = 12345.123456789;

19      System.out.println("Floating-point output:");
20      System.out.println("START1234567890");
21      System.out.printf("START%fEND %n", d);
22      System.out.printf("START%.4fEND %n", d);
23      System.out.printf("START%.2fEND %n", d);
24      System.out.printf("START%12.4fEND %n", d);
25      System.out.printf("START%eEND %n", d);
26      System.out.printf("START%12.5eEND %n", d);
27  }
28 }
```

(continued)

# The `printf` Method (Part 3 of 3)

## Display 2.2 The `printf` Method

### SAMPLE DIALOGUE

String output:


START1234567890

STARTabcEnd

START abcEnd

STARTabcEnd

*The value is always output. If the specified field width is too small, extra space is taken.*



Character output:

START1234567890

STARTZEND

START   ZEND

Floating-point output:

START1234567890

START12345.123457END

START12345.1235END

START12345.12END

START  12345.1235END

START1.234512e+04END

START 1.23451e+04END

*Note that the output is rounded, not truncated, when digits are discarded.*

# The `printf` Method (Part 2 of 3)

## Display 2.2 The `printf` Method

---

```
12      char oneCharacter = 'Z';

13      System.out.println("Character output:");
14      System.out.println("START1234567890");
15      System.out.printf("START%cEND %n", oneCharacter);
16      System.out.printf("START%4cEND %n", oneCharacter);
17      System.out.println();

18      double d = 12345.123456789;

19      System.out.println("Floating-point output:");
20      System.out.println("START1234567890");
21      System.out.printf("START%fEND %n", d);
22      System.out.printf("START%.4fEND %n", d);
23      System.out.printf("START%.2fEND %n", d);
24      System.out.printf("START%12.4fEND %n", d);
25      System.out.printf("START%eEND %n", d);
26      System.out.printf("START%12.5eEND %n", d);
27  }
28 }
```

(continued)

# The `printf` Method (Part 3 of 3)

## Display 2.2 The `printf` Method

### SAMPLE DIALOGUE

String output:


START1234567890

STARTabcEnd

START abcEnd

STARTabcEnd

*The value is always output. If the specified field width is too small, extra space is taken.*



Character output:

START1234567890

STARTZEND

START   ZEND

Floating-point output:

START1234567890

START12345.123457END

START12345.1235END

START12345.12END

START  12345.1235END

START1.234512e+04END

START 1.23451e+04END

*Note that the output is rounded, not truncated, when digits are discarded.*

# Importing Packages and Classes

- Libraries in Java are called *packages*
  - A package is a collection of classes that is stored in a manner that makes it easily accessible to any program
  - In order to use a class that belongs to a package, the class must be brought into a program using an *import* statement
  - Classes found in the package **java.lang** are imported automatically into every Java program

```
import java.text.NumberFormat;  
// import the NumberFormat class only  
import java.text.*;  
//import all the classes in package java.text
```

# Console Input Using the **Scanner** Class

- Starting with version 5.0, Java includes a class for doing simple keyboard input named the **Scanner** class
- In order to use the **Scanner** class, a program must include the following line near the start of the file:  
`import java.util.Scanner`
- This statement tells Java to
  - Make the **Scanner** class available to the program
  - Find the **Scanner** class in a library of classes (i.e., Java *package*) named `java.util`



# Console Input Using the **Scanner** Class

- The following line creates an object of the class **Scanner** and names the object **keyboard** :  
`Scanner keyboard = new Scanner(System.in) ;`
- Although a name like **keyboard** is often used, a **Scanner** object can be given any name
  - For example, in the following code the **Scanner** object is named **scannerObject**  
`Scanner scannerObject = new Scanner(System.in) ;`
- Once a **Scanner** object has been created, a program can then use that object to perform keyboard input using methods of the **Scanner** class

# Console Input Using the **Scanner** Class

- The method **nextInt** reads one **int** value typed in at the keyboard and assigns it to a variable:  
`int numberOfPods = keyboard.nextInt();`
- The method **nextDouble** reads one **double** value typed in at the keyboard and assigns it to a variable:  
`double d1 = keyboard.nextDouble();`
- Multiple inputs must be separated by *whitespace* and read by multiple invocations of the appropriate method
  - Whitespace is any string of characters, such as blank spaces, tabs, and line breaks that print out as white space

# Console Input Using the **Scanner** Class

- The method **next** reads one string of non-whitespace characters delimited by whitespace characters such as blanks or the beginning or end of a line
- Given the code

```
String word1 = keyboard.next();  
String word2 = keyboard.next();
```

and the input line

```
jelly beans
```

The value of **word1** would be **jelly**, and the value of **word2** would be **beans**

# Keyboard Input Demonstration (Part 1 of 2)

## Display 2.6 Keyboard Input Demonstration

---

```
1  import java.util.Scanner;
2  public class ScannerDemo
3  {
4      public static void main(String[] args)
5      {
6          Scanner keyboard = new Scanner(System.in);
7
8          System.out.println("Enter the number of pods followed by");
9          System.out.println("the number of peas in a pod:");
10         int numberOfPods = keyboard.nextInt();
11         int peasPerPod = keyboard.nextInt();
12
13         int totalNumberOfPeas = numberOfPods*peasPerPod;
14
15         System.out.print(numberOfPods + " pods and ");
16         System.out.println(peasPerPod + " peas per pod.");
17         System.out.println("The total number of peas = "
18                             + totalNumberOfPeas);
19     }
20 }
```

*Makes the Scanner class available to your program.*

*Creates an object of the class Scanner and names the object keyboard.*

*Each reads one int from the keyboard*

# Keyboard Input Demonstration

## (Part 2 of 2)

### Display 2.6 Keyboard Input Demonstration

#### SAMPLE DIALOGUE 1

Enter the number of pods followed by  
the number of peas in a pod:

22 10

22 pods and 10 peas per pod.

The total number of peas = 220

*The numbers that are  
input must be  
separated by  
whitespace, such as  
one or more blanks.*

#### SAMPLE DIALOGUE 2

Enter the number of pods followed by  
the number of peas in a pod:

22

10

22 pods and 10 peas per pod.

The total number of peas = 220

*A line break is also  
considered whitespace and  
can be used to separate the  
numbers typed in at the  
keyboard.*

# Console Input Using the **Scanner** Class

- The method **nextLine** reads an entire line of keyboard input
- The code,  

```
String line = keyboard.nextLine();
```

reads in an entire line and places the string that is read into the variable **line**
- The end of an input line is indicated by the escape sequence '**\n**'
  - This is the character input when the **Enter** key is pressed
  - On the screen it is indicated by the ending of one line and the beginning of the next line
- When **nextLine** reads a line of text, it reads the '**\n**' character, so the next reading of input begins on the next line
  - However, the '**\n**' does not become part of the string value returned (e.g., the string named by the variable **line** above does not end with the '**\n**' character)

# Another Keyboard Input Demonstration (Part 1 of 3)

## Display 2.7 Another Keyboard Input Demonstration

---

```
1  import java.util.Scanner;


2  public class ScannerDemo2
3  {
4      public static void main(String[] args)
5      {
6          int n1, n2;
7          Scanner scannerObject = new Scanner(System.in);

8          System.out.println("Enter two whole numbers");
9          System.out.println("seperated by one or more spaces:");


10         n1 = scannerObject.nextInt();
11         n2 = scannerObject.nextInt();
12         System.out.println("You entered " + n1 + " and " + n2);

13         System.out.println("Next enter two numbers.");
14         System.out.println("Decimal points are allowed.");
```

*Creates an object of the class **Scanner** and names the object **scannerObject**.*



*Reads one **int** from the keyboard.*



(continued)

# Another Keyboard Input Demonstration (Part 2 of 3)

## Display 2.7 Another Keyboard Input Demonstration

```
15     double d1, d2;
16     d1 = scannerObject.nextDouble();
17     d2 = scannerObject.nextDouble();
18     System.out.println("You entered " + d1 + " and " + d2);

19     System.out.println("Next enter two words:");

20     String word1 = scannerObject.next();
21     String word2 = scannerObject.next();
22     System.out.println("You entered \"" +
23         word1 + "\" and \"" + word2 + "\"");

24     String junk = scannerObject.nextLine(); //To get rid of '\n'

25     System.out.println("Next enter a line of text:");
26     String line = scannerObject.nextLine();
27     System.out.println("You entered: \"" + line + "\"");
28 }
29 }
```

*Reads one double from the keyboard.*

*Reads one word from the keyboard.*

*This line is explained in the Pitfall section "Dealing with the Line Terminator, '\n'".*

*Reads an entire line.*

(continued)



# Another Keyboard Input Demonstration (Part 3 of 3)

## Display 2.7 Another Keyboard Input Demonstration

---

### SAMPLE DIALOGUE

Enter two whole numbers  
separated by one or more spaces:

**42 43**

You entered 42 and 43

Next enter two numbers.

A decimal point is OK.

**9.99 57**

You entered 9.99 and 57.0

Next enter two words:

**jelly beans**

You entered "jelly" and "beans"

Next enter a line of text:

**Java flavored jelly beans are my favorite.**

You entered "Java flavored jelly beans are my favorite."

# Pitfall: Dealing with the Line Terminator, ' \n '

- The method `nextLine` of the class `Scanner` reads the remainder of a line of text starting wherever the last keyboard reading left off
- This can cause problems when combining it with different methods for reading from the keyboard such as `nextInt`
- Given the code,

```
Scanner keyboard = new Scanner(System.in);  
int n = keyboard.nextInt();  
String s1 = keyboard.nextLine();  
String s2 = keyboard.nextLine();
```

and the input,

2

Heads are better than

1 head.

what are the values of `n`, `s1`, and `s2`?

# Pitfall: Dealing with the Line Terminator, ' \n '

- The method `nextLine` of the class `Scanner` reads the remainder of a line of text starting wherever the last keyboard reading left off
- This can cause problems when combining it with different methods for reading from the keyboard such as `nextInt`
- Given the code,

```
Scanner keyboard = new Scanner(System.in);  
int n = keyboard.nextInt();  
String s1 = keyboard.nextLine();  
String s2 = keyboard.nextLine();
```

and the input,

2

Heads are better than

1 head.

what are the values of `n`, `s1`, and `s2`?

```
2\nHeads are better than\n1 head.\n
```

# Pitfall: Dealing with the Line Terminator, ' \n '

- Given the code and input on the previous slide  
    **n** will be equal to **"2"**,  
    **s1** will be equal to **" "**, and  
    **s2** will be equal to **"heads are better than"**
- If the following results were desired instead  
    **n** equal to **"2"**,  
    **s1** equal to **"heads are better than"**, and  
    **s2** equal to **"1 head"**  
then an extra invocation of **nextLine** would be  
needed to get rid of the end of line character ( **' \n '** )

```
2\nHeads are better than\n1 head.\n
```

# Methods in the Class **Scanner**

## (Part 1 of 3)

### Display 2.8    **Methods of the Scanner Class**

---

The `Scanner` class can be used to obtain input from files as well as from the keyboard. However, here we are assuming it is being used only for input from the keyboard.

To set things up for keyboard input, you need the following at the beginning of the file with the keyboard input code:

```
import java.util.Scanner;
```

You also need the following before the first keyboard input statement:

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

The *Scanner\_Object\_Name* can then be used with the following methods to read and return various types of data typed on the keyboard.

Values to be read should be separated by whitespace characters, such as blanks and/or new lines. When reading values, these whitespace characters are skipped. (It is possible to change the separators from whitespace to something else, but whitespace is the default and is what we will use.)

```
Scanner_Object_Name.nextInt()
```

Returns the next value of type `int` that is typed on the keyboard.

(continued)

# Methods in the Class **Scanner**

## (Part 2 of 3)

### Display 2.8    **Methods of the Scanner Class**

---

*Scanner\_Object\_Name.nextLong()*

Returns the next value of type `long` that is typed on the keyboard.

*Scanner\_Object\_Name.nextByte()*

Returns the next value of type `byte` that is typed on the keyboard.

*Scanner\_Object\_Name.nextShort()*

Returns the next value of type `short` that is typed on the keyboard.

*Scanner\_Object\_Name.nextDouble()*

Returns the next value of type `double` that is typed on the keyboard.

*Scanner\_Object\_Name.nextFloat()*

Returns the next value of type `float` that is typed on the keyboard.

(continued)

# Methods in the Class **Scanner**

## (Part 3 of 3)

### Display 2.8    **Methods of the Scanner Class**

---

*Scanner\_Object\_Name.next()*

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.

*Scanner\_Object\_Name.nextBoolean()*

Returns the next value of type `boolean` that is typed on the keyboard. The values of `true` and `false` are entered as the strings `"true"` and `"false"`. Any combination of upper- and/or lowercase letters is allowed in spelling `"true"` and `"false"`.

*Scanner\_Object\_Name.nextLine()*

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.

*Scanner\_Object\_Name.useDelimiter(New\_Delimiter);*

Changes the delimiter for keyboard input with *Scanner\_Object\_Name*. The *New\_Delimiter* is a value of type `String`. After this statement is executed, *New\_Delimiter* is the only delimiter that separates words or numbers. See the subsection "Other Input Delimiters" for details.