



CSS 142

Preview : take a look before Lab 2

Lecture 4b

Arkady Retik

aretik@uw.edu

TODAY'S CONTENT



1. L4 a recap

a. Class assignment, String

Slide deck 4a

2. Console Input and Output

3. println vs print vs printf

Slide deck 4b

4. Scanner: intro

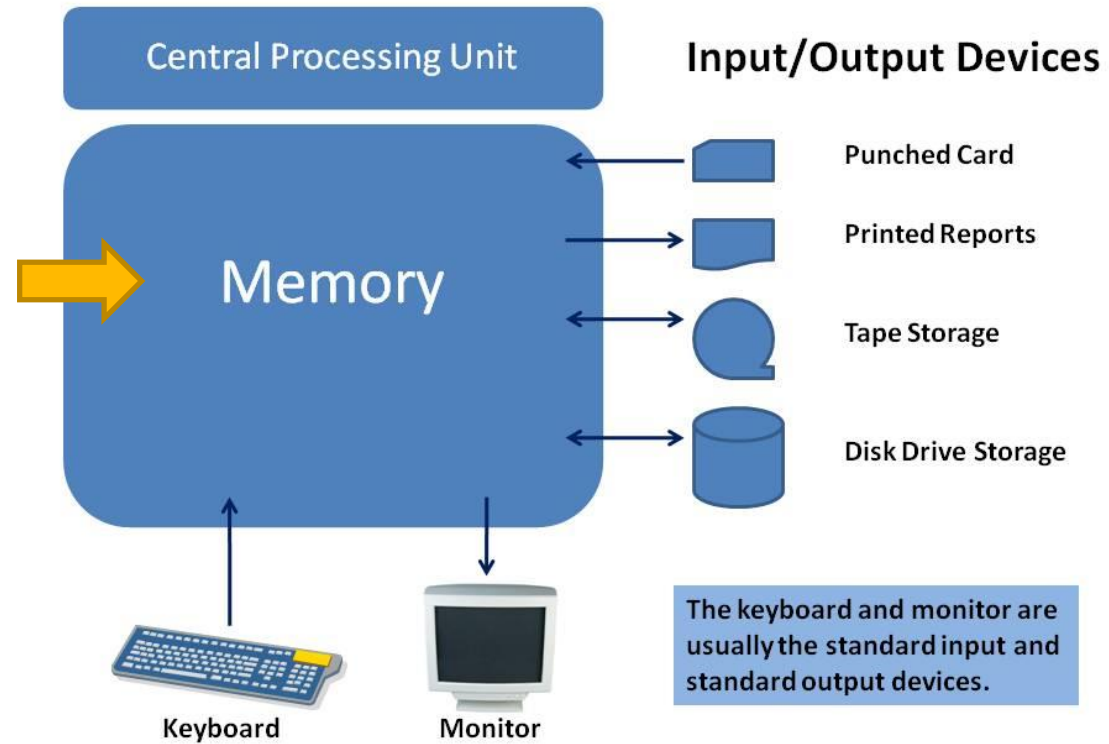
Lab 2:
Scanner

5. Next lecture (Mon)

❖ Scanner: cont

❖ Chapter 3 (3.1, 3.2) + Activity in class (on I/O; Scanner)

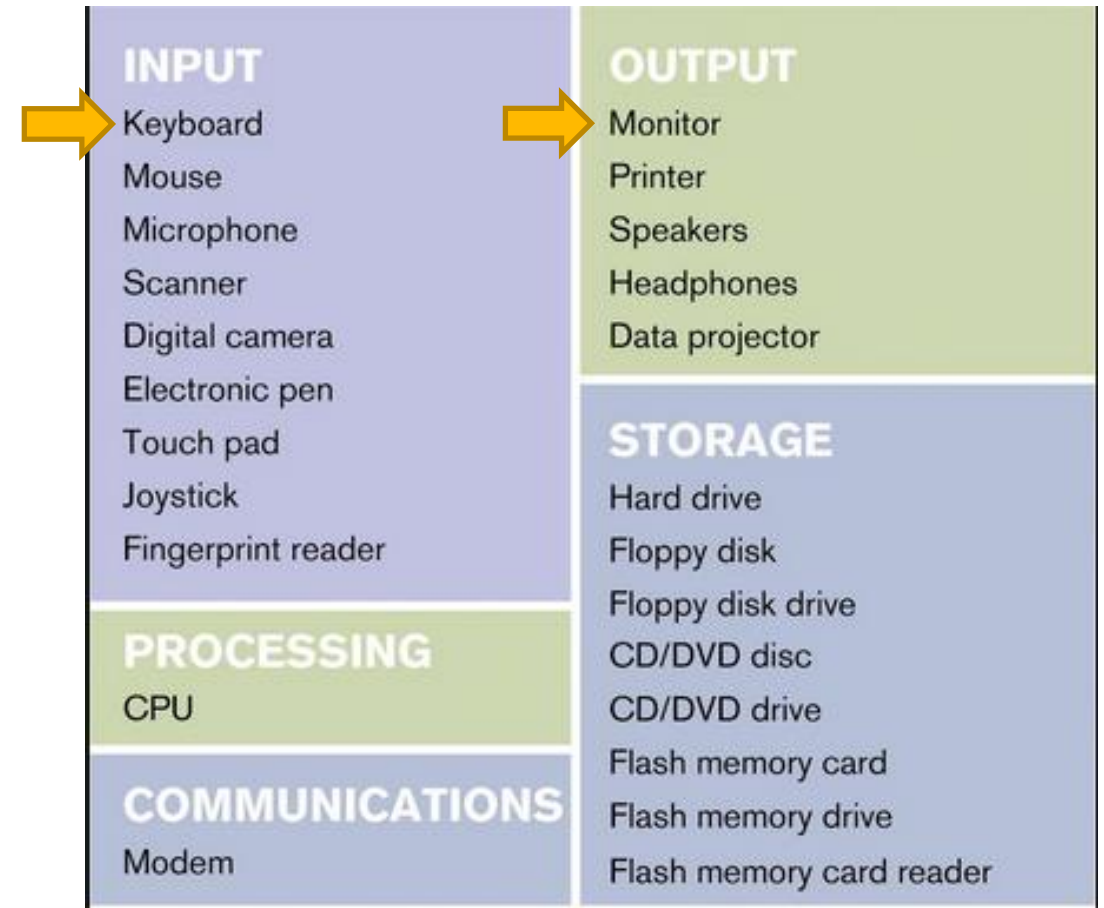
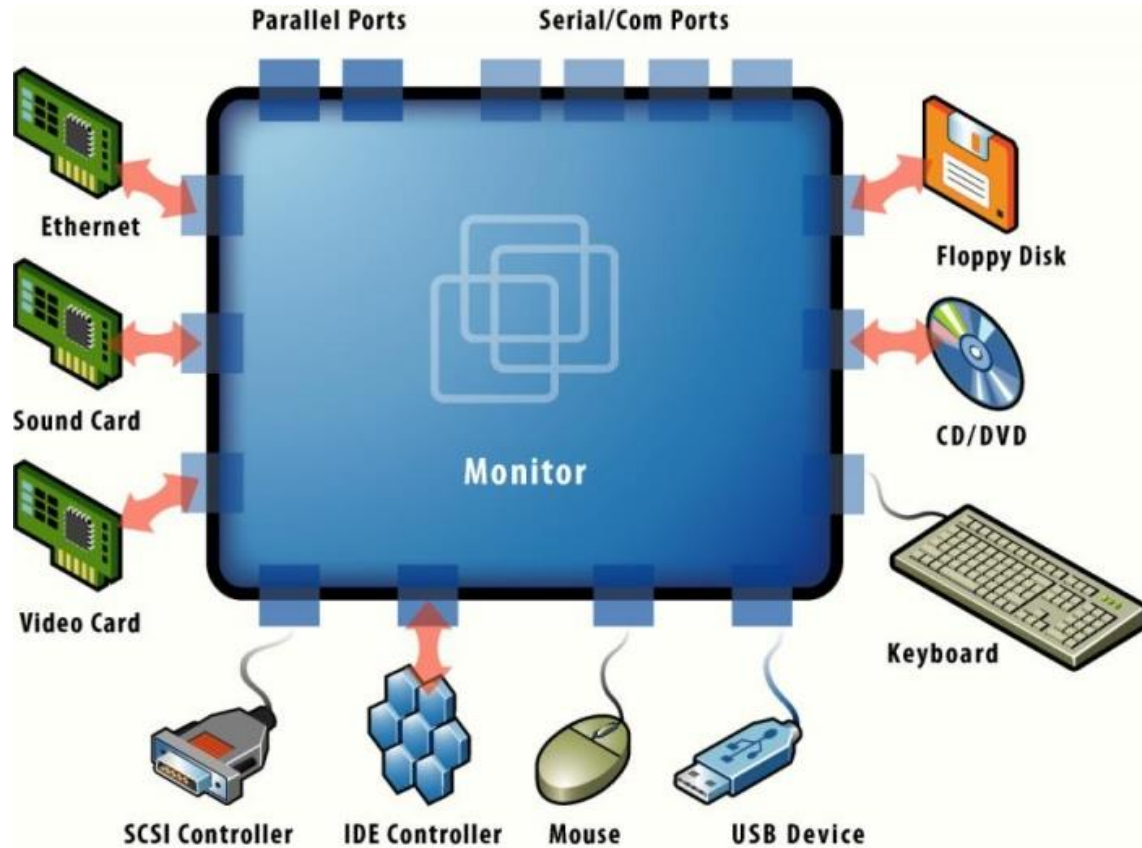
Console I/O



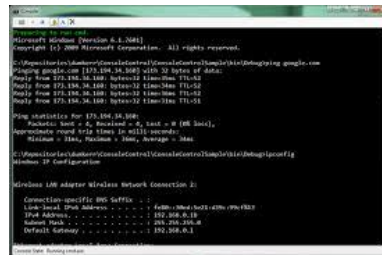
Don't confuse with Game Consoles



I/O Devices



Historically: Command line | text | one line



“Standard In” is Keyboard
“Standard Out” is Monitor
“Standard Err” is Error output to Standard Out

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

*Java™ Platform
Standard Ed. 6*

java.lang



Class System

[java.lang.Object](#)

└─ [java.lang.System](#)

```
public final class System
extends Object
```

The `System` class contains several useful class fields and methods. It cannot be instantiated.

Among the facilities provided by the `System` class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

Since:

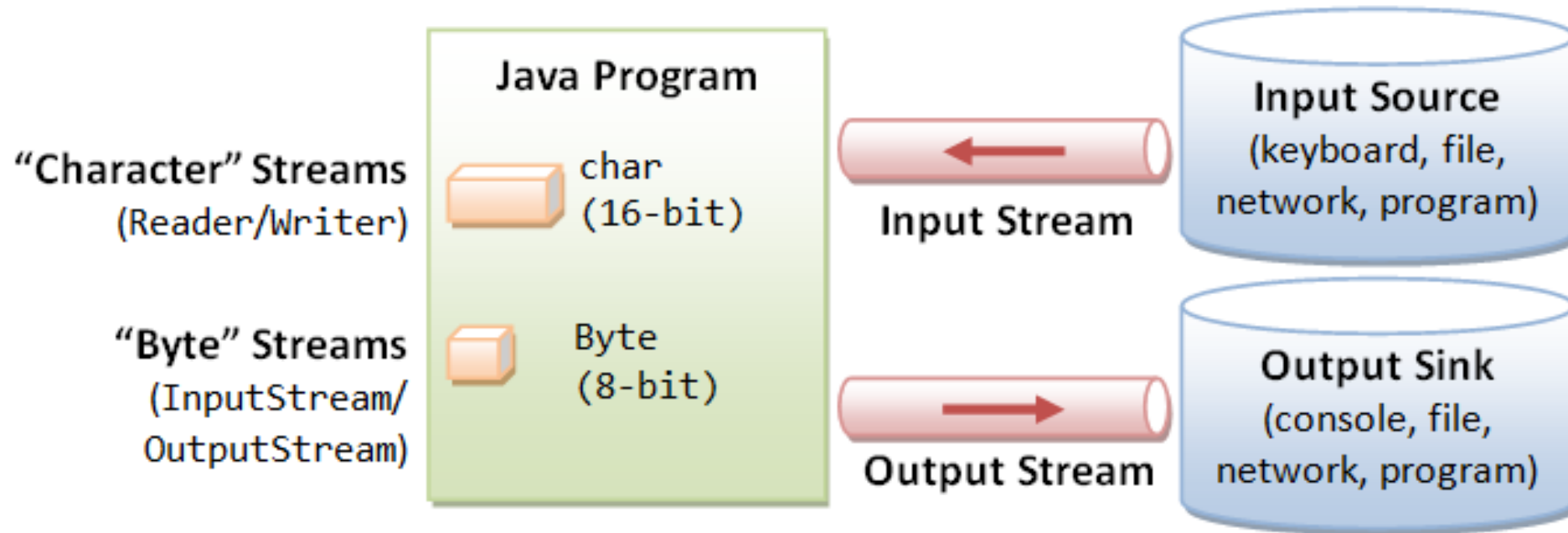
JDK1.0

Field Summary

static PrintStream	err	The "standard" error output stream.
static InputStream	in	The "standard" input stream.
static PrintStream	out	The "standard" output stream.

I/O Streams

TMF



Internal Data Formats:

- Text (char): UCS-2
- int, float, double, etc.

External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

System.out.println for console output

println Output

You can output one line to the screen using `System.out.println`. The items that are output can be quoted strings, variables, numbers, or almost any object you can define in Java. To output more than one item, place a plus sign between the items.

SYNTAX

```
System.out.println(Item_1 + Item_2 + ... + Last_Item) ;
```

EXAMPLE

```
System.out.println("Welcome to Java.");  
System.out.println("Elapsed time = " + time + " seconds");
```

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 SLA is " + 99.999) ;
```


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

- **example1:** What is the output produced by the following lines?

```
System.out.println(2 + " " + 2);
```

```
System.out.println(2 + 2);
```

- **example 2:** What is the output produced by the following lines?

```
System.out.print(2 + " " + 2);
```

```
System.out.print(2 + 2);
```



Write down your
answer

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 (.80)
 - 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`)

Line Breaks with `printf`

- Line breaks can be included in a format string using `%n`
- The code

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

will output

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



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 %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

The `printf` Method (Part 1 of 3)

Display 2.2 The `printf` Method (part 1 of 2)

```
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();
```



String output:

START1234567890

STARTabcEND

START abcEND

STARTabcEND

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

The printf Method (Part 2 of 3)

Display 2.2 The printf Method (part 2 of 2)

```
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 }
```



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.

Formatting Money Amounts with `printf`

- A good format specifier for outputting an amount of money stored as a double type is `%.2f`
- It says to include exactly two digits after the decimal point and to use the smallest field width that the value will fit into:

```
double price = 19.99;
```

```
System.out.printf("The price is $%.2f each.")
```

produces the output:

```
The price is $19.99 each.
```

Legacy Code

TMI

- Code that is "old fashioned" but too expensive to replace is called ***legacy code***
- Sometimes legacy code is translated into a more modern language
- The Java method `printf` is just like a C language function of the same name
- This was done intentionally to make it easier to translate C code into Java

Money Formats

- Using the `NumberFormat` class enables a program to output amounts of money using the appropriate format

- The `NumberFormat` class must first be *imported* in order to use it



```
import java.text.NumberFormat
```

- An object of `NumberFormat` must then be created using the `getCurrencyInstance()` method
 - The `format` method takes a floating-point number as an argument and returns a `String` value representation of the number in the local currency

Money Formats

```
import java.text.NumberFormat;

public class CurrencyFormatDemo
{
    public static void main(String[] args)
    {
        System.out.println("Default location:");
        NumberFormat moneyFormater =
            NumberFormat.getCurrencyInstance();

        System.out.println(moneyFormater.format(19.8));
        System.out.println(moneyFormater.format(19.81111));
        System.out.println(moneyFormater.format(19.89999));
        System.out.println(moneyFormater.format(19));
        System.out.println();
    }
}
```

Output

```
Default location:
$19.80
$19.81
$19.90
$19.00
```



Specifying Locale

- Invoking the `getCurrencyInstance()` method without any arguments produces an object that will format numbers according to the **default location**
- In contrast, the location can be explicitly specified by providing a location from the `Locale` class as an argument to the `getCurrencyInstance()` method
 - When doing so, the `Locale` class must first be imported

```
import java.util.Locale;
```

Specifying Locale

```
import java.text.NumberFormat;
import java.util.Locale;

public class CurrencyFormatDemo
{
    public static void main(String[] args)
    {
        System.out.println("US as location:");
        NumberFormat moneyFormater2 =
            NumberFormat.getCurrencyInstance(Locale.US);

        System.out.println(moneyFormater2.format(19.8));
        System.out.println(moneyFormater2.format(19.81111));
        System.out.println(moneyFormater2.format(19.89999));
        System.out.println(moneyFormater2.format(19));
    }
}
```

Output

US as location:

\$19.80

\$19.81

\$19.90

\$19.00



Locale Constants for Currencies of Different Countries

Display 2.4 **Locale Constants for Currencies of Different Countries**

<code>Locale.CANADA</code>	Canada (for currency, the format is the same as US)
<code>Locale.CHINA</code>	China
<code>Locale.FRANCE</code>	France
<code>Locale.GERMANY</code>	Germany
<code>Locale.ITALY</code>	Italy
<code>Locale.JAPAN</code>	Japan
<code>Locale.KOREA</code>	Korea
<code>Locale.TAIWAN</code>	Taiwan
<code>Locale.UK</code>	United Kingdom (English pound)
<code>Locale.US</code>	United States

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

asterisk

from Greek:
ἀστερίσκος, asteriskos, "little star"

The `DecimalFormat` Class

- Using the `DecimalFormat` class enables a program to format numbers in a variety of ways
 - The `DecimalFormat` class must first be *imported*
 - A `DecimalFormat` object is associated with a pattern when it is created using the `new` command
 - The object can then be used with the method `format` to create strings that satisfy the format
 - An object of the class `DecimalFormat` has a number of different methods that can be used to produce numeral strings in various formats

The DecimalFormat Class (Part 1 of 3)

Display 2.5 The DecimalFormat Class

```
1 import java.text.DecimalFormat;

2 public class DecimalFormatDemo
3 {
4     public static void main(String[] args)
5     {
6         DecimalFormat pattern00dot000 = new DecimalFormat("00.000");
7         DecimalFormat pattern0dot00 = new DecimalFormat("0.00");

8         double d = 12.3456789;
9         System.out.println("Pattern 00.000");
10        System.out.println(pattern00dot000.format(d));
11        System.out.println("Pattern 0.00");
12        System.out.println(pattern0dot00.format(d));

13        double money = 19.8;
14        System.out.println("Pattern 0.00");
15        System.out.println("$" + pattern0dot00.format(money));
16
17        DecimalFormat percent = new DecimalFormat("0.00%");
```

Pattern 00.000

12.346

Pattern 0.00

12.35

Pattern 0.00

\$19.80

The number is always given, even if this requires violating the format pattern.

(continued)

The DecimalFormat Class (Part 2 of 3)

Display 2.5 The DecimalFormat Class

```
18      System.out.println("Pattern 0.00%");
19      System.out.println(percent.format(0.308));

20      DecimalFormat eNotation1 =
21          new DecimalFormat("#0.###E0");//1 or 2 digits before point
22      DecimalFormat eNotation2 =
23          new DecimalFormat("00.###E0");//2 digits before point

24      System.out.println("Pattern #0.###E0");
25      System.out.println(eNotation1.format(123.456));
26      System.out.println("Pattern 00.###E0");
27      System.out.println(eNotation2.format(123.456));

28      double smallNumber = 0.0000123456;
29      System.out.println("Pattern #0.###E0");
30      System.out.println(eNotation1.format(smallNumber));
31      System.out.println("Pattern 00.###E0");
32      System.out.println(eNotation2.format(smallNumber));
33  }
34 }
```

```
Pattern 0.00%
30.80%

Pattern #0.###E0
1.2346E2

Pattern 00.###E0
12.346E1

Pattern #0.###E0
12.346E-6

Pattern 00.###E0
12.346E-6
```

The Class Scanner

Review at home; we will recap on Wedn the key points

Console Input Using the `Scanner` Class

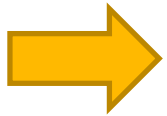
- Java includes a class for doing simple keyboard input named the `Scanner` class
- In order to use the `Scanner` class, a program must include the following line near the start of the file:

➡ `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

These are Delimiters

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 for the input line, what would be the value of `word1`

```
jelly beans
```

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



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)

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



Savitch p.77

(continued)

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.*

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.*



Why it says "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."

Always "echo" (i.e. output)
the user input for verification



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 '

- 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 '`)

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**



Scannner_Object_Name.nextLong()

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



Scannner_Object_Name.nextByte()

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



Scannner_Object_Name.nextShort()

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



Scannner_Object_Name.nextDouble()

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




Scannner_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.

Programming Tip: Prompt for Input

- ❖ A program should always prompt the user when he or she needs to input some data:

```
System.out.println(  
    "Enter the number of pods followed by");  
System.out.println(  
    "the number of peas in a pod:");
```

Programming Tip: Echo Input

- Always echo all input that a program receives from the keyboard
- In this way a user can check that he or she has entered the input correctly
 - Even though the input is automatically displayed as the user enters it, echoing the input may expose subtle errors (such as entering the letter "O" instead of a zero)

Self-Service Checkout Line (Part 1 of 2)

Display 2.9 Self-Service Check Out Line

```
1  import java.util.Scanner;

2  public class SelfService
3  {
4      public static void main(String[] args)
5      {
6          Scanner keyboard = new Scanner(System.in);

7          System.out.println("Enter number of items purchased");
8          System.out.println("followed by the cost of one item.");
9          System.out.println("Do not use a dollar sign.");

10         int count = keyboard.nextInt();
11         double price = keyboard.nextDouble();
12         double total = count*price;

13         System.out.printf("%d items at $%.2f each.%n", count, price);
14         System.out.printf("Total amount due $%.2f.%n", total);

15         System.out.println("Please take your merchandise.");
16         System.out.printf("Place $%.2f in an envelope %n", total);
17         System.out.println("and slide it under the office door.");
18         System.out.println("Thank you for using the self-service line.");
19     }
20 }
21
```



Write down the output,
assume input, i.e. count of 10
items, \$19.99 each
(you have 2 minutes)

*The dot after %.2f is a period in the
text, not part of the format specifier.*

(continued)

Self-Service Checkout Line (Part 2 of 2)

Display 2.9 Self-Service Check Out Line

SAMPLE DIALOGUE

Enter number of items purchased
followed by the cost of one item.
Do not use a dollar sign.

10 19.99

10 items at \$19.99 each.
Total amount due \$199.90.
Please take your merchandise.
Place \$199.90 in an envelope
and slide it under the office door.
Thank you for using the self-service line.

The Empty String

- A string can have any number of characters, including zero characters
 - `""` is the empty string
- When a program executes the `nextLine` method to read a line of text, and the user types nothing on the line but presses the **Enter** key, then the `nextLine` Method reads the empty string

Other Input Delimiters

- The delimiters that separate keyboard input can be changed when using the **Scanner** class
- For example, the following code could be used to create a **Scanner** object and change the delimiter from whitespace to "##"

```
Scanner keyboard2 = new Scanner(System.in);
```

```
Keyboard2.useDelimiter("##");
```

- After invocation of the **useDelimiter** method, "##" and not whitespace will be the only input delimiter for the input object **keyboard2**

Changing the Input Delimiter (Part 1 of 3)

Display 2.10 Changing the Input Delimiter

```
1  import java.util.Scanner;

2  public class DelimiterDemo
3  {
4      public static void main(String[] args)
5      {
6          Scanner keyboard1 = new Scanner(System.in);
7          Scanner keyboard2 = new Scanner(System.in);
8          keyboard2.useDelimiter("##");
9          //Delimiter for keyboard1 is whitespace.
10         //Delimiter for keyboard2 is ##.
```

(continued)

Changing the Input Delimiter (Part 2 of 3)

Display 2.10 Changing the Input Delimiter

```
11      String word1, word2;
12      System.out.println("Enter a line of text:");
13      word1 = keyboard1.next();
14      word2 = keyboard1.next();
15      System.out.println("For keyboard1 the two words read are:");
16      System.out.println(word1);
17      System.out.println(word2);
18      String junk = keyboard1.nextLine(); //To get rid of rest of line.
19
20      System.out.println("Reenter the same line of text:");
21      word1 = keyboard2.next();
22      word2 = keyboard2.next();
23      System.out.println("For keyboard2 the two words read are:");
24      System.out.println(word1);
25      System.out.println(word2);
26  }
27 }
```

(continued)

Changing the Input Delimiter (Part 3 of 3)

Display 2.10 Changing the Input Delimiter

SAMPLE DIALOGUE

Enter a line of text:

one two##three##

For keyboard1 the two words read are:

one

two##three##

Reenter the same line of text:

one two##three##

For keyboard2 the two words read are:

one two

three

Next Mon

- **Read:** Savitch Chapter 2 and Chapter 3 (3.1, 3.2)
 - We will also have Activity in class
- **Homework 1 is due : see Canvas**
 - Read it and ask questions