Input and Output

The system console
In the beginning …

- When computers were relatively expensive and rare, most users interacted with a terminal
  - CRT screen with keyboard
  - Remotely attached to a central, shared computer
- Although PCs have by and large put an end to this model, and windowing systems have largely put an end to the notion of a single screen, the model lives on in the form of the *system console*
Console I/O in Java

- Although Java is better-known for its deep support of GUI interfaces, a simple console interface is available as well.
- We will use this model for most of the semester, as it is the most like the model used by the majority of programming languages, and will make the file I/O model easier to follow.
Output methods: println

• We have already used the println method in several program examples

• The syntax is:

  System.out.println(arg);

  – Where “arg” is a String expression or an expression of one of the primitive types (int, double, char, etc.)

  – To output more than one value, include at least one String and use the + operator to concatenate additional values with the String
Output methods: print

- The print method is identical to the println method, except for the absence of “In” and one aspect of its action:
  - When the println method is called, it automatically places the cursor on the next output line when it has finished outputting its argument
  - The print method, on the other hand, leaves the cursor at the end of the current line when it has completed its output
Example

System.out.println(“Here is a brand new line:”);
System.out.print(“But despite the capital letter”);
System.out.print(“This is not a new line at all”);
Output:

Here is a brand new line
But despite the capital letter
This is not a new line at all
Formatting Output

- We call the space occupied by an output value the **field**. The number of characters allocated to a field is the **field width**. The diagram shows output right-justified in a field width of 6 (spaces are represented by dashes).

```
- - - - - 3 - - - - 3 4 - - - - 5 6 8 4 - - - - 9 8 - - - - 2 3 1
- - - - - 4 4 5 - - - - 3 3 9 - - - - 2 3 4 - - - - 4 5 3 - - - - 3 4 4 4
```

Each value occupies six spaces. If the value has three digits, we put three blank spaces in front. If the value has four digits, we put two blank spaces in front, and so forth.
The printf method

- The printf method uses the concept of field width to format console output
- printf takes a minimum of one argument, and may take several
  - The first argument is the **control string**, which specifies the format for the remaining arguments, if any
  - If the control string is the only argument, its contents are an ordinary string literal, and printf works exactly like print
  - If there are additional arguments, they follow the control string
Control Strings

• Integers
  \%<field width>d

• Example:
  System.out.printf("The result is: %5d\n", 100);
  Output:
  The result is: 100

• In the example above, the number is printed right-justified in a field of 5 spaces
Control strings

• Real Numbers
  \%<field width>.<decimal places>f

• Example:
  System.out.printf("You owe: $\%7.2f\n", 3.15679e2);
  Output:
  You owe: $ 315.68

• In the example, the specified field width was one space wider than required to print the number with 2 decimal places

• If you specify a field width that is too narrow for the output, the field width value is simply ignored
Control strings

• Strings
  \%s

• Example:
  System.out.printf("%10s%10s%10s\n", "Yours", "Mine", "Ours");

• Output:
  Yours      Mine      Ours
## Display 2.1 Format Specifiers for `System.out.printf`

<table>
<thead>
<tr>
<th>CONVERSION CHARACTER</th>
<th>TYPE OF OUTPUT</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Decimal (ordinary) integer</td>
<td>%5d %d</td>
</tr>
<tr>
<td>f</td>
<td>Fixed-point (everyday notation) floating point</td>
<td>%6.2f %f</td>
</tr>
<tr>
<td>e</td>
<td>E-notation floating point</td>
<td>%8.3e %e</td>
</tr>
<tr>
<td>g</td>
<td>General floating point (Java decides whether to use E-notation or not)</td>
<td>%8.3g %g</td>
</tr>
<tr>
<td>s</td>
<td>String</td>
<td>%12s %s</td>
</tr>
<tr>
<td>c</td>
<td>Character</td>
<td>%2c %c</td>
</tr>
</tbody>
</table>
Right and Left Justification in printf

- The code
  ```java
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
Which line produces the output below given the following: double v = 42.70145;

The value is: $42.70 // this is the output

A. System.out.println(“The value is: $” + v);
B. System.out.printf(“The value is: $” + v);
C. System.out.printf(“The value is: $.2f” + v);
D. System.out.printf(“The value is: $.2f", v);
Suppose you want to print the value of Math.PI, as follows:

- With 2 digits after the decimal point
- Left-justified
- In a field of 8 characters

With what should you replace the blank space below?

```java
System.out.printf("%__.f", Math.PI);
```
Standard input

• The System class has a member object named `in` which is analogous to member object `out`: we can use this object to read input from the keyboard.

• The `read` method of `System.in` reads data in the form of bytes (ASCII character data); in order to read even simple type data, we need to associate `System.in` with an object of a class defined in the `java.util.*` package, `Scanner`.

• We can read values of the simple numeric types from the keyboard by using the `next` method of a `Scanner` object.
import java.util.*;
...
Scanner kb;
double d;
int i;

kb = new Scanner(System.in);

System.out.print("Enter a real number: ");
d = kb.nextDouble();
System.out.print("Enter a whole number: ");
i = kb.nextInt();
System.out.println("You entered: "+d+" and "+i);
Output from example

Enter a real number: 4.35
Enter a whole number: -9
You entered 4.35 and -9
The prompt/read/echo pattern

• The previous example illustrates a pattern of program behavior that should be used when your program reads keyboard input:
  – Prompt: a message to the user requesting input; for example:
    
    ```java
    System.out.print("Enter a real number: ");
    ```
  – Read: a line of code that reads the requested data; example:
    
    ```java
    d = kb.nextDouble();
    ```
  – Echo: Write out the data read in so user can see what is actually stored in memory; example:
    
    ```java
    System.out.println("You entered: "+d+" and "+i);
    ```
What is a prompt?

A. An output statement
B. A message to the user
C. A request for input
D. All of the above
Place these statements in logical order

1. `int x = kb.nextInt();`
2. `System.out.println ("You entered " + x);`
3. `System.out.print ("Enter a number: ");`
4. `Scanner kb = new Scanner(System.in);`
Dealing with users

• When reading keyboard input, it is best to prompt for and read one data item at a time; while this can be tedious to program, it provides an unambiguous interface to the user – s/he knows exactly what the program expects and can proceed accordingly
Reading String data from the console

• To read String data, we can apply the next() method of the Scanner class; next() will read all of the text the user types until s/he hits the enter key or types another white space character, as in the example below:

```java
Scanner kb = new Scanner(System.in);
String myInput;
System.out.print(“Enter your name: ”);
myInput = kb.next();
```
Reading String data from the system console

• The code on the previous slide will only read the first word the user types; for instance, if I typed Cathleen M. Sheller, then only the word “Cathleen” would be stored as myInput.

• The `nextLine()` method can be used to read an entire line of text, as in the example below:

```
Scanner kb = new Scanner(System.in);
System.out.print ("Enter your full name: ");
String fullName = kb.nextLine();
```
Combining nextLine with other input methods

• One thing to keep in mind about nextLine is that it considers a newline character (\n) as the delimiter between one input String and another.

• Because nextInt and nextDouble read data up to the next delimiter, an extra newline character may be left to be read.

• This can be a problem if the next input statement is a nextLine.

• The solution is to use two calls to nextLine instead of one when a nextLine is preceded by at nextInt or nextDouble – see page 85 for details.
Dealing with users: coding defensively

• Even with prompts, our users can get confused and give us improper data
• And let’s face it, some of them are just EVIL
• For these reasons, it’s a good idea to do whatever we can to protect our programs from crashing in the face of incompetent or malevolent users
• One aspect of defensive programming is anticipating bad input
Defensive coding: reading all input as Strings

• Input tends to be Achilles heel of many programs
• The nextInt and nextDouble methods of the Scanner class don’t react well to wrong input
• On the other hand, nextLine, which reads input as a String, can handle just about any data you throw at it
• And, if nextLine is the only input method you use, you never have to worry about throwing in extra nextLine calls – each one disposes of its own terminating newline character
Wrapper classes

• We can read any input as a String, but most of the time we want numbers to be numbers
• We can convert Strings into numbers by using methods of the numeric wrapper classes
• Each simple type has a corresponding wrapper class, as shown on the next slide
Wrapper classes

• For simple type:
  – int
  – double
  – float

• Use wrapper class:
  – Integer
  – Double
  – Float
Converting Strings to numbers

• Each of the wrapper classes has an associated method called a parse method that takes a String argument
  – For the Integer class, the name of the method is parseInt
  – For Double, the name is parseDouble and for Float, it is parseFloat

• Each method returns a value of the underlying simple data type
Example

• The following code reads a String, then converts it to an int value:

```java
String input;
int n;
Scanner kb = new Scanner(System.in);
System.out.print("Enter a number: ");
input = kb.nextLine();
n = Integer.parseInt(input);
```
What good is this?

• At the moment, not much; the parse methods will react as badly to the wrong kind of input as the input methods will

• The advantage will become clearer when we start looking at selection structures and can test values before attempting the conversion
Which of the following statements converts a String named s to a double?

A. Double d = Double.parseDouble(s);
B. double d = Double.parse(s);
C. double d = double.parseDouble(s);
D. double d = Double.parseDouble(s);