BASIC INPUT/OUTPUT

standard input

command-line arguments

standard output

standard audio

standard drawing

Fundamentals of Computer Science
Outline: Basic Input/Output

- Screen Output
- Keyboard Input
Simple Screen Output

System.out.println("The count is " + count);

• Outputs the sting literal "the count is "
• Followed by the current value of the variable count.

• We've seen several examples of screen output already.
  • System.out is an object that is part of Java.
  • println() is one of the methods available to the System.out object.
Screen Output

- The concatenation operator (+) is useful when everything does not fit on one line.
  
  ```java
  System.out.println("Lucky number = " + 13 + "Secret number = " + number);
  ```

- Do not break the line except before or after the concatenation operator (+).
Screen Output

• Alternatively, use `print()`

```java
System.out.print("One, two,"即);
System.out.print("buckle my shoe.即");
System.out.println("Three, four,"即);
System.out.println("shut the door.即");
```

ending with a `println()`.
**FORMATTED PRINTING**

**Input:** `printf("Color %s, Number %d, Float %5.2f", "red", 123456, 3.14;)

**Output:** Color red, Number 123456, Float 3.14
Pretty Text Formatting

- **printf-style formatting**
  - Common way to nicely format output
  - Present in many programming languages
    - Java, C++, Perl, PHP, ...
  - Use a special format language:
    - Format string with special codes
    - One or more variables get filled in

- **In Java, used via:**
  - `System.out.printf()` – output to standard out
  - `String.format()` – returns a formatted String
Floating-Point Formatting

```java
double d = 0.123456789;
float f = 0.123456789f;

// %f code is used with double or float variables
// %f defaults to rounding to 6 decimal places
System.out.printf("d is about %f\n", d);
System.out.printf("f is about %f\n", f);

// Number of decimal places specified by .X
// Output is rounded to that number of places
System.out.printf("PI is about %.1f\n", Math.PI);
System.out.printf("PI is about %.2f\n", Math.PI);
System.out.printf("PI is about %.3f\n", Math.PI);
System.out.printf("PI is about %.4f\n", Math.PI);

// %e code outputs in scientific notation
// .X specifies number of significant figures
final double C = 299792458.0;
System.out.printf("speed of light = %e\n", C);
System.out.printf("speed of light = %.3e\n", C);
```

- d is about 0.123457
- f is about 0.123457
- PI is about 3.1
- PI is about 3.14
- PI is about 3.142
- PI is about 3.1416
- C = 2.99792458e+08
- C = 2.998e+08
// %d code is for integer values, int or long
// Multiple % codes can be used in a single printf()
long power = 1;
for (int i = 0; i < 8; i++)
{
    System.out.printf("%d = 2^%d\n", power, i);
    power = power * 2;
}

// A number after the % indicates the minimum width
// Good for making a nice looking tables of values
power = 1;
for (int i = 0; i < 8; i++)
{
    System.out.printf("%5d = 2^%d\n", power, i);
    power = power * 2;
}

You can have multiple % codes that are filled in by a list of parameters to printf()

Minimum width of this field in the output. Java will pad with whitespace to reach this width (but can exceed this width if necessary).
Flags

// Same table, but left justify the first field
power = 1;
for (int i = 0; i < 8; i++)
{
    System.out.printf("%-5d = 2^%d\n", power, i);
    power = power * 2;
}

// Use commas when displaying numbers
power = 1;
for (int i = 0; i < 17; i++)
{
    System.out.printf("%,5d = 2^%d\n", power, i);
    power = power * 2;
}
Text Formatting

// Characters can be output with %c, strings using %s
String name = "Bill";
char grade = 'B';
System.out.printf("%s got a %c in the class.\n", name, grade);

Bill got a B in the class.

// This prints the same thing without using printf
System.out.println(name + " got a " + grade + " in the class.");

An equivalent way to print the same thing out using good old println().
Creating Formatted Strings

- **Formatted String** creation
  - You don't always want to immediately print formatted text to standard output
  - Save in a String variable for later use

```java
// Formatted Strings can be created using format()
String lines = "";
for (int i = 0; i < 4; i++)
    lines += String.format("Random number %d = %.2f\n", i, Math.random());
System.out.print(lines);
```

Random number 0 = 0.54
Random number 1 = 0.50
Random number 2 = 0.39
Random number 3 = 0.64
The Format Specifier

% [flags][width][.precision]type

Special formatting options like inserting commas, making left justified, etc.

Sets the number of decimal places, don't forget the .

Type is the only required part of specifier. "d" for an integer, "f" for a floating-point number.

Minimum number of characters used, but if number is longer it won't get cut off.

System.out.printf("%,6.1f", 42.0);
Printf Gone Bad

- **Format string specifies:**
  - Number of variables to fill in
  - Type of those variables
- **With great power comes great responsibility**
  - Format must agree with #/types of arguments
  - Runtime error otherwise
  - Compiler / Eclipse won't protect you

```java
// Runtime error %f expects a floating-point argument
System.out.printf("crash %f\n", 1);  

// Runtime error, %d expects an integer argument
System.out.printf("crash %d\n", 1.23);  

// Runtime error, not enough arguments
System.out.printf("crash %d %d\n", 2);  
```
# printf Puzzler

<table>
<thead>
<tr>
<th>Code</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.out.printf(&quot;%f&quot;, 4242.00);</code></td>
<td>E</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%d&quot;, 4242);</code></td>
<td>A</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%.2f&quot;, 4242.0);</code></td>
<td>B</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%.3e&quot;, (double) 4242);</code></td>
<td>C</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%d&quot;, 4242);</code></td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4242</td>
</tr>
<tr>
<td>B</td>
<td>4242.00</td>
</tr>
<tr>
<td>C</td>
<td>4.242e+03</td>
</tr>
<tr>
<td>D</td>
<td>4,242</td>
</tr>
<tr>
<td>E</td>
<td>4242.00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>System.out.printf(&quot;%d%d&quot;, 42, 42);</code></td>
<td>2</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%d+%d&quot;, 42, 42);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%d %d&quot;, 42);</code></td>
<td>5</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%8d&quot;, 42);</code></td>
<td>3</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%-8d&quot;, 42);</code></td>
<td>4</td>
</tr>
<tr>
<td><code>System.out.printf(&quot;%d&quot;, 42.0);</code></td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42+42</td>
</tr>
<tr>
<td>2</td>
<td>4242</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>runtime error</td>
</tr>
</tbody>
</table>
Keyboard Input

- Java has reasonable facilities for handling keyboard input.
- These facilities are provided by the `Scanner` class in the `java.util` package.
  - A `package` is a library of classes.
Simple Input

- Sometimes the data needed for a computation are obtained from the user at run time.
- Keyboard input requires
  
  ```java
  import java.util.Scanner
  
  at the beginning of the file.
  ```
Simple Input

- Data can be entered from the keyboard using
  ```java
  Scanner keyboard = new Scanner(System.in);
  ```
  followed, for example, by
  ```java
  eggsPerBasket = keyboard.nextInt();
  ```
  which reads one `int` value from the keyboard and assigns it to `eggsPerBasket`. 
Using the Scanner Class

• Near the beginning of your program, insert
  
import java.util.Scanner;

• Create an object of the Scanner class
  
Scanner keyboard =
    new Scanner (System.in)

• Read data (an int or a double, for example)
  
int n1 = keyboard.nextInt();
  double d1 = keyboard.nextDouble();

• Close the Scanner
  
keyboard.close();
Some `Scanner` Class Methods

`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.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.nextInt()`
Returns the next keyboard input as a value of type `int`.

`Scanner_Object_Name.nextDouble()`
Returns the next keyboard input as a value of type `double`.

`Scanner_Object_Name.nextFloat()`
Returns the next keyboard input as a value of type `float`. 
Some **Scanner** Class Methods

- **Figure 2.7b**

```java
Scanner_Object_Name.nextLong()
Returns the next keyboard input as a value of type long.

Scanner_Object_Name.nextIntByte()
Returns the next keyboard input as a value of type byte.

Scanner_Object_Name.nextIntShort()
Returns the next keyboard input as a value of type short.

Scanner_Object_Name.nextIntBoolean()
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 low-
ercase letters is allowed in spelling true and false.

Scanner_Object_Name.useDelimiter(Delimiter_Word);
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.
```
**nextLine() Method Caution**

- The `nextLine()` method reads
  - The remainder of the current line,
  - Even if it is empty.

- **Example – given following declaration.**
  ```java
  int n;
  String s1, s2;
  n = keyboard.nextInt();
  s1 = keyboard.nextLine();
  s2 = keyboard.nextLine();
  ```

- **Assume input shown**
  
  `n` is set to 42
  but `s1` is set to the empty string.
Outline: Basic Input/Output

- Screen Output
- Keyboard Input