Representing real numbers

- Representing real numbers
  - First, convert into binary numbers
    - A little trickier than it first seems: to the right, each bit represents $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, etc.
      - $5.75 = ?$
  - Next, put into binary scientific notation: $a \times 2^b$
    - $101.11 \times 2^3$
  - Normalize so that first significant digit is immediately to the right of the binary point
    - $10111 \times 2^3$
  - Mantissa and exponent (and signs) then stored
  - What’s the ultimate result?

Administrivia

- HW#1 due next Tuesday
- Does everyone have both textbooks? (Bookstore is asking us)
Representing text

- Characters are mapped onto binary numbers
  - ASCII code set
    - 8 bits per character (byte); 256 character codes
  - UNICODE code set
    - 16 bits per character; 65,536 character codes
    - Much more complex, but better international support
  - Let's Google “ASCII”
- Text strings are sequences of characters in some encoding

Sound

- Multimedia data is sampled to store a digital form, with or without detectable differences
- Representing sound data
  - Sound data must be digitized for storage in a computer
  - Digitizing means periodic sampling (frequency) of amplitude values (levels)
  - From samples, original sound may be approximated
  - To improve the approximation:
    - Sample more frequently
    - Use more bits for each sample value
  - CD quality: 44kHz sampling, 16-bit levels, stereo = 176kbps
  - DVD audio/SACD different; effectively 100kHz

(a) Sampling the Original Signal

(b) Recreating the Signal from the Sampled Values
Images

- Representing image data
  - Images are sampled by reading color and intensity values at even intervals across the image
  - Each sampled point is a pixel
  - Image quality depends on number of bits at each pixel and the number of pixels in an image
  - 24bpp common for consumer-grade equipment today
  - “Megapixel”: what does it mean?

And now...

- We’re about to delve deep into Java syntax
- Don’t be afraid to write test programs to fully understand the concepts discussed here
- I’ll write a number of test programs today and make them downloadable
- **ASK QUESTIONS** if you don’t understand things
- Much easier now than later; we’re only going to become more complex

Character strings in Java

- A string of characters can be represented as a *string literal* by putting double quotes around the text
- For example...
- Every character string is an object in Java, defined by the `String` class
- Every string literal represents a `String` object
- Don’t worry about all the implications just yet
The println Method

- We used `println` method to print a character string.
- `System.out` is an object that represents a destination.

```
System.out.println("Whatever you are, be a good one.");
```

The print Method

- There are other methods in `System.out`.
- For example, the `print` method is similar to the `println` method, except that it does not advance to the next line.
- How do we find out more information about such methods? Java API documentation
  - `http://java.sun.com/j2se/1.5.0/docs/api/index.html`
  - A little overwhelming at first, but ultimately useful
  - We’ll spend more time on this later.

String Concatenation

- The string concatenation operator (+) is used to append one string literal to the end of another "Peanut butter " + "and jelly"
- It can also be used to append a number to a string
  "I am not " + 65 + " years old"
- We could just represent the number as a string in this case
- A single string literal cannot be broken across two lines in a program.
Escape Sequences

- What if we wanted to print a the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```java
System.out.println("I said "Hello" to you.");
```

Escape Sequences (II)

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```java
System.out.println("I said \"Hello\" to you.");
```

Escape Sequences (III)

- Some Java escape sequences:

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>
Example

- What does the following print? (Roses.java, p. 68)

```
System.out.println("Roses are red,\
Violets are blue,\
" +
"Sugar is sweet,\
But I have "commitment issues",\
So I'd rather just be friends\At this point in our " +
"relationship.\n";
```

Variables

- A variable represents a piece of information in memory. It's declared by specifying:
  - The type of information (e.g., `int`)
  - The name of the variable
- You can name more than one variable per statement

```
data type    variable name
int total;
int count, temp, result;
```

Variable initialization

- You can give an initial value in the declaration:

```
int sum = 0;
int base = 32, max = 149;
```
- When a variable is referenced in a program, its current value is used
- You change the value of the variable by assigning it a new value

```
sum = 55;
```
Assignment

- The assignment operator is the = sign
- The expression on the right is evaluated and the result is stored in the variable on the left
- Previous value in sum, if any, is overwritten
- You can only assign a value to a variable that is consistent with the variable's declared type

Constants

- Identifier that is similar to a variable except that it holds the same value during its entire existence, i.e., constant
  - Usually ALL_CAPITALS to avoid confusion
- The compiler will issue an error if you try to change the value of a constant
- In Java, we use the final modifier to declare a constant
  ```java
  final int NUM_DAYS_IN_YEAR = 365;
  ```
- Useful in making program easier to read, or to change some predefined concepts

Primitive Data

- There are eight primitive data types in Java
  - Integers: byte, short, int, long
  - Floating point numbers: float, double
  - Characters: char
  - Boolean values: boolean
    - Can be assigned true or false; one bit of info
- Note that they're all lowercase
- Strings are not primitive data types
  - They're a reference type, but can be used in a similar fashion (with some caveats)
  - String (note uppercase S)
Numeric Primitive Data

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt; -9 x 10^18</td>
<td>&gt; 9 x 10^18</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10^38 with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10^308 with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>

Characters

- A char variable stores a single character
- Character literals are delimited by single quotes: 'a' 'X' '7' '$' ',' '\n'
- Example declarations:
  ```java
char topGrade = 'A';
char terminator = ';', separator = ' ';```
- Note the distinction between a primitive character variable, which holds only one character, and a String object, which can hold multiple characters
- Java supports both ASCII and Unicode characters

Expressions

- An expression is a combination of one or more operators and operands
- Arithmetic expressions compute numeric results and make use of the arithmetic operators:
  ```
  Addition +
  Subtraction -
  Multiplication *
  Division /
  Remainder %
  ```
- If either or both operands used by an arithmetic operator are floating point, then the result is a floating point
Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded).

  \[
  14 \div 3 \quad \text{equals} \quad 4 \\
  8 \div 12 \quad \text{equals} \quad 0
  \]

- The remainder operator (\%) returns the remainder after dividing the second operand into the first.

  - Also called \texttt{mod} operator (modulus)

  \[
  14 \% 3 \quad \text{equals} \quad 2 \\
  8 \% 12 \quad \text{equals} \quad 8
  \]

Operator Precedence

- Given the following compound expression, in what order are the operands evaluated?

  \[
  \text{result} = \text{total} + \text{count} / \text{max} - \text{offset};
  \]

- Variation on PEMDAS (no exponents)

- Arithmetic operators with the same precedence are evaluated from left to right.

- The assignment operator itself has lower precedence, so this works.

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable.

  \[
  \text{First, one is added to the original value of count} \\
  \text{count} = \text{count} + 1; \\
  \]

  Then the result is stored back into count (overwriting the original value)
Assignment Operators

- This is such a common paradigm, Java provides assignment operators to simplify the process.
- For example, `count += 1;` is equivalent to `count = count + 1;`

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td><code>x += y</code></td>
<td><code>x = x + y</code></td>
</tr>
<tr>
<td><code>-=</code></td>
<td><code>x -= y</code></td>
<td><code>x = x - y</code></td>
</tr>
<tr>
<td><code>*=</code></td>
<td><code>x *= y</code></td>
<td><code>x = x * y</code></td>
</tr>
<tr>
<td><code>/=</code></td>
<td><code>x /= y</code></td>
<td><code>x = x / y</code></td>
</tr>
<tr>
<td><code>%=</code></td>
<td><code>x %= y</code></td>
<td><code>x = x % y</code></td>
</tr>
</tbody>
</table>

- What does `result /= (total-MIN) % num;` do?

Assignment Operators (II)

- The behavior of some assignment operators depends on the types of the operands.
- If the operands to the `+=` operator are strings, the assignment operator performs string concatenation.
- The behavior of an assignment operator (`+=`) is always consistent with the behavior of the corresponding operator (`+`).

Increment and Decrement

- Also turns out that adding or subtracting one is extremely common, so much so there are special one-operand operators for these tasks.
- The increment operator (`++`) adds 1 to its operand.
- The decrement operator (`--`) subtracts 1 from its operand.
- The statement `count++;` is functionally equivalent to `count = count + 1;`
Next time

- Finish chapter 2 of Lewis/Loftus