CS 4: Lecture 2
Monday, January 23, 2006

TYPES
=====
Last lecture, we saw how to declare an "int" variable that stores an integer. Java has several other _types_ of variables. One reason is because a computer does not have infinite memory, and there is a limit to how big a number you can store in a variable. For example, an "int" cannot be bigger than 2147483647, because Java doesn't allocate enough memory to an "int" to store a bigger one. So Java provides several integer types of different sizes.

Most computers don't store numbers as decimal digits. Instead, they store numbers as _bits_ (binary digits)--in other words, as base-2 numbers. Java allocates 32 bits for each int, so an int can represent one of 2^32 different numbers. Java offers four integer types.

```
  byte:  An 8-bit integer in the range -128...127.  (One bit is the sign.)
  short:  A 16-bit integer in the range -32768...32767.
  int:  A 32-bit integer in the range -2147483648...2147483647.
  long:  A 64-bit integer, range -9223372036854775808...9223372036854775807.
```

Just to be safe, you should use longs for most purposes, so you're less likely to accidentally try to store a big number in a small space. What happens if you do that? Sometimes, Java stores a nonsense value that fits.

```
  int atoms;
  atoms = 50000 * 50000;          /* Too big to fit! */
  System.out.println(atoms);      /* Java prints "-1794967296". */
```

Sometimes, Java won't let you do it. Java will let you assign a small integer to a big type, but not vice versa.

```
  byte little = 20;
  byte small = 500;              /* Compiler error--Java won't compile this. */
  int big = little;              /* "big" stores 20. */
  little = big;                  /* Compiler error--Java won't compile this. */
```

Java won't let you compile these lines because it's trying to protect you from accidentally creating a nonsense value and a hard-to-find bug. But the only reason to use any type shorter than long is when you need to store a huge number of numbers, and can't afford the memory to make them all long.

When you do division with Java integers, but they don't divide exactly, the division rounds toward zero.

```
  atoms = 5 / 3;     /* atoms is now 1 */
```

Besides the operations "+", "-", "*", "/", Java integers also offer a "%" operator, which tells you the remainder left over after doing division.

```
  atoms = 5 % 3;     /* 2 */
  atoms = -5 % 3;    /* -2 */
```

Java offers increment "++" and decrement "--" operators, which are useful shorthand for increasing or decreasing a variable by one.

```
  atoms++;           /* Same as "atoms = atoms + 1;" */
```

Floating-Point Numbers

Integer types can only store integers. Java has types that store real numbers, with decimal points. The one you should usually use is called "double".

```
  double quotient = 8.75 / 2.5;
  System.out.println(quotient);      /* Java prints "3.5". */
```

There are two floating-point types.

```
  double:  A 64-bit floating-point number like 18.355625430920409.
           _Double_precision_ numbers store approximately 17 decimal digits.
  float:  A 32-bit floating-point number.
          _Single_precision_ numbers store approximately 9 decimal digits.
```

The names are historical. Back when computers were a lot slower than they are today, computing with single precision numbers (floats) was faster than computing with doubles. Those days are gone, and today the only reason to use floats is if you need to store a huge number of them, and can't afford the memory to store doubles.

Floating-point numbers support the operations "+", "-", "*", "/", but not "%" or "++" or "--". The division operation does _not_ round to the nearest integer, but it usually does round off somewhere, because computers can't store an infinite number of decimal places.

When you write a number in Java, Java decides whether it's an integer or a floating-point number based on whether or not it has a decimal point. If you want Java to treat a number as a real number, you must use a decimal point, even if the number happens to be an integer.

```
  System.out.println(11 / 3);        /* Prints "3". */
  System.out.println(11. / 3.);      /* Prints "3.6666666666666665". */
```

Booleans

A boolean variable can have only two values: "true" or "false". What makes booleans interesting is that they are created by _comparison_operators_.

```
  boolean b1 = false;
  boolean b2 = 6 > 3;      /* true */
  boolean b3 = 2 + 3 < 4;  /* false */
  boolean b4 = 5 <= 5;     /* true      "<=" means "less than or equal to" */
                           /*           there's also a >= operator */
  boolean b5 = 8 == 1;     /* false     "==" means "is equal to" */
  boolean b6 = 8 != 1;     /* true      "!=" means "is not equal to" */
```

Observe that Java uses "==" for "is equal to", and not "=" as you might expect. The latter is reserved for assignment.

Booleans are used most often in _conditional_ statements...

CONDITIONALS
============
An "if" statement uses a boolean expression to decide whether to execute a set of statements. The form is

```
  if (boolValue) {
    statements;
  }
```

The statements are executed if and only if "boolValue" is "true". The parentheses around the boolean expression are required. Here's an example.

```
  boolean pass = score >= 170;
  if (pass) {
    System.out.println("You pass CS 4");
  }
```

The phrase with the quotes is called a _string_ (we'll learn more about strings in future classes). This code prints out the phrase "You pass CS 4" if the variable "score" is 170 or greater. If score is less than 170, it does nothing.

An "if" statement can also have an "else" clause that says what to do if the boolean value is false.

```
  if (savings >= ferrariprice) {
    ferraris++;
    savings = savings - ferrariprice;
  } else {
    /* This is what happens if savings < ferrariprice. */
    toyotas++;
    savings = savings - toyotaprice;
  }
```

if-else clauses can be _nested_: you can have one inside another.

```
  if (x > y) {
    if (x > z) {
      maximum = x;
    } else {
      maximum = z;
    }
  } else {
    if (y > z) {
      maximum = y;
    } else {
      maximum = z;
    }
  }
```

if-else clauses can also be daisy-chained together.

```
  if (month == 1) {
    days = 31;
  } else if (month == 2) {
    days = 28;
  }
```