CS 61B: Lecture 4
Wednesday, January 29, 2014

Today's reading: S&B pp. 10-14, 49-53, 75, 78-79, 86, 117, 286-287, 292, 660.

PRIMITIVE TYPES

Not all variables are references to objects. Some variables are primitive types, which store values like "3", "7.2", "h", and "false". They are:

- byte: A 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.
- double: A 64-bit floating-point number like 18.355625430920409.
- float: A 32-bit floating-point number; has fewer digits of precision.
- boolean: "true" or "false".
- char: A single character.

long values are written with an L on the end: long x = 43L;
This tells the compiler to internally write out "43" in a 64-bit format.
double and float values must have a decimal point: double y = 18.0;
float values are written with an f at the end: float f = 43.9f;

	Object types	Primitive types
Variable contains a	reference	value
How defined?	class definition	built into Java
How created?	"new"	"6", "3.4", "true"
How initialized?	constructor	default (usually zero)
How used?	methods	operators ("+", "*", etc.)

Operations on int, long, short, and byte types.

```
    -x               x * y
    x + y            x / y    <-- rounds toward zero (drops the remainder).
    x - y            x % y    <-- calculates the remainder of x / y.
```

Except for "%", these operations are also available for doubles and floats. Floating-point division ("/") doesn't round to an integer, but it does round off after a certain number of bits determined by the storage space.

The java.lang library has more operations in...

```
    - the Math class.
      x = Math.abs(y);  // Absolute value.  Also see Math.sqrt, Math.sin, etc.
    - the Integer class.
      int x = Integer.parseInt("1984");  // Convert a string to a number.
    - the Double class.
      double d = Double.parseDouble("3.14");
```

Converting types: integers can be assigned to variables of longer types.

```
    int i = 43;
    long l = 43;    // Okay, because longs are a superset of ints.
    l = i;          // Okay, because longs are a superset of ints.
    i = l;          // Compiler ERROR.
    i = (int) l;    // Okay.
```

The string "(int)" is called a cast, and it casts the long into an int. In the process, high bits will be lost if l does not fit in the range -2147483648... 2147483647. Java won't let you compile "i = l" because it's trying to protect you from accidentally creating a nonsense value and a hard-to-find bug. Java requires you to explicitly cast longs to ints to show your acknowledgment that you may be destroying information.

Similarly, "float f = 5.5f; double d = f;" is fine, but you need an explicit cast for "double d = 5.5; float f = (float) d;". Integers (even longs) can be directly assigned to floating-point variables (even floats) without a cast, but the reverse requires a cast because the number is truncated to an integer.

Boolean Values

A boolean value is either "true" or "false". Booleans have operations of their own, signified "&&" (and), "||" (or), and "!" (not).

a	b	a && b	a || b	!a
false	false	false	false	true
false	true	false	true	
true	false	false	true	false
true	true	true	true	

Boolean values can be specified directly ("true", "false") or be created by the comparison operators "==", "<", ">", "<=", ">=", "!=" (not equal to).

```
    boolean x = 3 == 5;       // x is now false.
    x = 4.5 >= 4.5;           // x is now true.
    x = 4 != 5 - 1;           // x is now false.
    x = false == (3 == 0);    // x is now true.
```

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 (for no good reason).

```
    boolean pass = score >= 75;
    if (pass) {
      output("You pass CS 61B");
    } else {
      // The following line executes if and only if score < 75.
      output("You are such an unbelievable loser");
    }
```

if-then-else clauses can be (1) nested and (2) daisy-chained. Nesting allows you to build decision trees. Daisy-chaining allows you to present more than two alternatives. For instance, suppose you want to find the maximum of three numbers.

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

Some long chains of if-then-else clauses can be simplified by using a "switch" statement. "switch" is appropriate only if every condition tests whether a variable x is equal to some constant.

```
    switch (month) {        |      if (month == 2) {
    case 2:                 |        days = 28;
      days = 28;            |      } else if ((month == 4) || (month == 6) ||
      break;                |                 (month == 9) || (month == 11)) {
    case 4:                 |        days = 30;
    case 6:                 |      } else {
    case 9:                 |        days = 31;
    case 11:                |      }
      days = 30;            |
      break;
    default:
      days = 31;
      break;
    }                   //  These two code fragments do exactly the same thing.
```

IMPORTANT: "break" jumps to the end of the "switch" statement. If you forget a break statement, the flow of execution will continue right through past the next "case" clause, which is why cases 4, 6, and 9 work right. If month == 12 in the following example, both Strings are printed.

```
    switch (month) {
    case 12:
      output("It's December.");
      // Just keep moving right on through.
    case 1:
    case 2:
    case 11:
      output("It's cold.");
    }
```

However, this is considered bad style, because it's hard to read and understand. If there's any chance that other people will need to read or modify your code (which is the norm when you program for a business), don't code it like this. Use break statements in the switch, and use subroutines to reuse code and clarify the control flow.

Observe that the last example doesn't have a "default:" case. If "month" is not 1 nor 2 nor 11 nor 12, Java jumps right to the end of the "switch" statement (just past the closing brace) and continues execution from there.

THE "return" KEYWORD
====================

Like conditionals, "return" affects the flow of control of a program. It causes a method to end immediately, so that control returns to the calling method.

Here's a recursive method that prints the numbers from 1 to x.

```
    public static void oneToX(int x) {
      if (x < 1) {
        return;
      }
      oneToX(x - 1);
      System.out.println(x);
    }
```

The return keyword serves a dual purpose: it is also the means by which a function returns a value. A _function_ is a method that is declared to return a non-void type. For instance, here's a function that returns an int.

```
    public int daysInMonth(int month) {
      switch (month) {
      case 2:
        return 28;
      case 4:
      case 6:
      case 9:
      case 11:
        return 30;
      default:
        return 31;
      }
    }
```

The "return" value can be an expression. Some examples:

```
    return x + y - z;

    return car.velocity(time);
```