Index of /~cs61b/lectures/04

Name	Last modified	Size	Description
Parent Directory	31-Aug-98 09:46	-	

```
                     CS61b: Lecture 4
                    Monday, August 31
ANNOUNCEMENTS:
-------------
1) Card keys are available in 390 Soda.
2) HW 2 is online (since 5pm Friday)

JAVA VS. SCHEME: WHY ARE THE LANGUAGES DIFFERENT?
-------------------------------------------------
 - "redundancy" in Java to help the programmer avoid mistakes
 - Desire to make programs run fast in C influenced Java
 - Desire to save programmer typing in C influenced Java (alas)

PRIMITIVE TYPES
---------------
See handout in the Reader on "Scheme to Java" and "Numeric Types."
In addition to the Object types we've seen, Java has some primitive types:
    boolean, char, int, double, float, byte, short, long
Of these, you're likely to use the first 4 or 5 in this class

                        Object Types            Primitive Types
----------------------------------------------------------------------
Type defined by         class definition        built-in to language
Value create by         new                     6, 3.4, true, etc.
Value initialized by    constructor             default, e.g., 0 for int
Variable contains       Reference               Primitive value
Operate using           Messages                Operators
Draw as                 Pointer to box          Value

For each primitive type, Java has a "wrapper" class that makes them into Objects:
  Boolean, Character, Integer, Double, Float, Long, Short, Byte

MODELING INTEGERS
-----------------
The "int" type is use for integer values.  It has the usual operations:
   int x, y;
   x = x + y; x = x - y, x = x * y; x = x / y; x = x % y; x = -y;
   x += y; x -= y; x++; // saves typing, the wrong thing do to

Note that x/y is integer division (the remainder is dropped);
         x % y is is the "modulo" or "mod" operation and returns the remainder

In addition, the java.lang library has some more operations:
 - The "Math" class has more basic operations on int, float, double, etc.
   x = Math.abs(y);  // absolute value
 - The Integer class, which "wraps up" an int into an Object has:
   x = Integer.parseInt("3");
```

Mixing types: in general, Java's strong typing prevents mixed assignments

```
  int i = 43;
  long l = 43L;   // a larger range integer that takes more space
  i = true;       // compilation error: redundancy of declaring x catches this
  l = i;          // OK, because longs are superset of ints
  i = l;          // error, but   i = (int) l;  is OK
```

MODELING REAL NUMBERS

float and double are the two types for modeling reals.
double has a larger range, but takes more space; rules similar to long and int.
double often use by default so programmers "don't have to worry"

```
  String s = "3.14";
  double d = Double.valueOf(s).doubleValue();
  d = d + 1.0;  d = d + 1; d += 1;   // What is d?
```

BOOLEAN

"boolean" is another primitive type; it has operations & (and), | (or), etc.
Usually see (b1 && b2) rather than (b1 & b2):

```
  evaluate b2, if it is false, don't evaluate b2; the whole expression is false
               if it is true, evaluate b2; the whole expression depends on b2
```

Also a || for "conditional or", which avoids evaluating b2 if b1 is true
Why use && and || rather than & and |?

CONDITIONALS: if and if/else

Simple if:

```
    if (grade >= 75) {        /* if alone */
       System.out.println("Pass");
    }
```

You may omit the { }'s in this case, but shouldn't.

If-then-else:

```
    if (grade >= 75) {   /* if with else */
       System.out.println("Pass");
    }
    else {
       System.out.println("Fail");
    }
```

Nested if-then-elses:

```
    if (grade >= 185) {
       System.out.println("A+");
       }
       else if (grade >= 175) {
             System.out.println("A");
       }
       else if (grade >= 165) {
             System.out.println("A-");
       }
       else {
             System.out.println("Not an A");
       }
    }
```

Always use the { }'s and let emacs indent for you.

SWITCH

For a long set of if/then/elses as above, the condition is a integer ==, use:

```
switch (month) {
  case 2:
    daysIn = 28;
    break;
  case 9: case 4: case 6: case 11:
    daysIn = 30;
    break;
  default:
    daysIn = 31;
    break;
}
```

As written, if month is 2 (February) daysin will be 28.
What if we omit the first "break"?
What if we omit the second "break"?
What if we omit both of the first two "break"s?
Why have the break in the last case?
Why does switch work this way?