Types and Expressions

Java Programming

Vermont Technical College

Peter Chapin

Variables

- Each data object must be *declared* (given a name and a type).
  - ```
    int   x;   // x holds integers.
    float y;   // y holds "floating point" numbers.
    char  z;   // z holds characters (letters).
    ```
- The names should actually be descriptive.
  - ```
    int   maxClassSize;
    float milesPerGallon;
    char  middleInitial;
    ```
- Yes, spell out words fully!
  - Capitalization called "camelCase." It's an industry standard.

Assignment Statements

- You give values to variables by assigning to them.
  - ```
    float base;
    float height;
    float triangleArea;

    base = 2.0;
    height = 3.5;
    triangleArea = base * height / 2;
    ```
- When first declared their values are “uninitialized.”
  - Don’t use uninitialized variables!
    ```

Initialized Declarations

- It is also possible (even recommended) to initialize when declaring.
  - ```
    float base = 2.0;
    float height = 3.5;
    float triangleArea = base * height / 2.0;
    ```
- Sometimes you won't have a value for a variable until later.
  - That's fine… just be sure you don't use it uninitialized!
    ```

Type?

- A very important concept…
  - Every variable (and every expression) has a *type* that defines a set of possible values that variable (or expression) can have.
  - `int x;`
    - Here `x` can only be an integer… not a number with a fractional part (3.14), not a letter (‘a’), not a string (“Hello”), etc.
  - You need to decide what type to make each variable… *requires thinking!*
    - Number of students in a class: `int` (we don’t have ½ a student in a class)
    - Your weight in pounds: `float`? `int`? (do you want to deal with fractional pounds?)
    - The user’s first name: `String` (a sequence of characters like “Peter”)
    - The area of a circle: `float` (most likely)

Kinds of Types

- Java divides types into two broad categories.
  - *Primitive types*: These are built into the language and represent simple concepts like numbers, letters, etc.
  - *Class types*: These you define yourself to represent more abstract concepts of your choosing like clocks, cats, printers, money, etc.
    - A lot of programming is about deciding what types are most suitable and then defining them properly.
    - *We will talk about this at length!*

Primitive Types

Type Name	Minimum Value	Maximum Value	Size (bytes)
byte	-128	127	1
short	-32768	32767	2
int	-2147483648	2147483647	4
long	-9223372036854775808	9223372036854775807	8
float	-3.403e+38 (approx)	3.403e+38 (approx)	4
double	-1.798e+308 (approx)	1.798e+308 (approx)	8
char	0	65535	2
boolean	false	true	

Basic Numeric Operators

Operator	Name
+	Addition
–	Subtraction (or negation)
*	Multiplication
/	Division
% (percent symbol)	Remainder (“modulus”)

Basic Expressions

- Use the operators in combination to do numeric calculations
  - ```
    area = width * height;
    cost = basePrice + extraItems * costPerItem;
    finalCost = cost + taxRate * cost;
    ```
  - (Quiz: what types would be reasonable for the variables above?)
- Use parentheses to organize complex computations
  - ```
    gamma = (load - source) / (load + source);
    root1 = (-b + Math.sqrt(b*b - 4*a*c)) / (2 * a);
    twisted = (a + (b/(c - d) % 5)) / 2;
    ```

Remainder Operator?

- Division of floating point numbers “works.”
  - 5.0/2.0 is 2.5 (approximately)
    5.0/4.0 is 1.25 (approximately)
- Division of integers throws away the fractional part.
  - 5/2 is 2 (exactly)
    5/4 is 1 (exactly)
- The remainder operator gives the remainder after integer division
  - 5 % 2 is 1 (2 goes into 5 twice with remainder of 1)
    5 % 4 is 1 (4 goes into 5 once with remainder of 1)
    5 % 3 is 2 (3 goes into 5 once with remainder of 2)

Precedence

- Question: What is 3 + 4 * 5?
  - Is that (3 + 4) * 5 which is 35 *or…*
    Is that 3 + (4 * 5) which is 23?
- The answer depends on the relative *precedence* of + and *
- Answer: multiplication has “higher” precedence; binds more tightly.
  - 3 + 4 * 5 is the same as 3 + (4*5)
  - If you want the other interpretation you *must* use parentheses.

Associativity

- Question: What is 3 * 4 / 3 * 5?
  - (3*4)/(3*5) which is 12/15 which is 0 (why?)
    3*(4/3)*5 which is 3*1*5 which is 15
- Multiplication and Division have the same precedence.
  - However, they have *left-to-right associativity*
  - (((3 * 4) / 3) * 5) which is ((12/3) * 5) which is (4*5) which is 20
- *If you are unsure use extra parentheses!*
- Consider:
  - `root1 = (-b + Math.sqrt(b*b - 4*a*c)) / (2*a)`
  - The parentheses around `2*a` are necessary. Why?