CS101

Java Programming

Fall 2012

Thursdays 10:00-12:00noon

RF Academy

Announcements

- Assignment #2 has been distributed
  - Must be completed with only material from Chapter 1 of the text
- Assignment #3 will be sent out in a few days
- Generic questions about the assignments should be posted to Piazza
  - Students are welcome to post answers if they know the answers
  - Please do not post your code to Piazza
- Questions??

Java's Primitive Types

- four integer types (`byte`, `short`, `int`, and `long`)
  - `int` is most common
- two floating-point types (`float` and `double`)
  - `double` is most common
- one character type (`char`)
- one boolean type (`boolean`)

- Why does Java distinguish integers vs. real numbers?
- Types that are not primitive are called *object types*. (seen later)

Expressions

- **expression**: A value or operation that computes a value.
  - Examples:

```
1 + 4 * 5
(7 + 2) * 6 / 3
42
```

- The simplest expression is a *literal value*.
  - Such as the value 42 above.

- A complex expression can use operators and parentheses.

Arithmetic operators

- **operators**: Combines multiple values or expressions.

  `+` addition
  `-` subtraction (or negation)
  `*` multiplication
  `/` division
  `%` modulus (a.k.a. remainder)

- As a program runs, its expressions are *evaluated*.

  `1 + 1` evaluates to `2`

  `System.out.println(3 * 4);` prints `12`

  - How would we print the text `3 * 4` ?

Integer division with /

- When we divide integers, the quotient is also an integer.

  `14/4` is `3`, not `3.5` (the fractional part is truncated)

```
      3                 4                 52
4 ) 14            10 ) 47           27 ) 1425
    12                40                135
     2                 7                   75
                                           54
                                           21
```

- More examples:
  - `32 / 5` is `6`
  - `84 / 10` is `8`
  - `156 / 100` is `1`

  - Dividing by 0 causes an error when your program runs.

Integer remainder with %

- The % operator computes the remainder from integer division.

```
14 % 4    is 2
218 % 5   is 3
```

```
        3                    43
    4 ) 14               5 ) 218
        12                   20
         2                    18
                              15
                               3
```

- Applications of % operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is even/odd: `42 % 2` is `0`, `7 % 2` is `1`
- What is `8 % 20`?

  `8` !

Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed

- examples:

```
(10 + 213) * 37
10 + (213 * 37)
```

- Without parentheses, an expression is evaluated according to the *rules of precedence.*

Precedence Rules

Highest Precedence

First: the unary operators: +, –, ++, – –, and !

Second: the binary arithmetic operators: *, /, and %

Third: the binary arithmetic operators: + and –

Lowest Precedence

Precedence Rules

- What is the difference between *unary* +/– operators and *binary* +/– operators?

- Unary +/– make a number positive or negative
- Binary +/– perform addition or subtraction
- Examples:

```
–5+7
–(5+7)
```

Precedence Rules

- When binary operators have equal precedence, they are evaluated left-to-right.

  `1+2-3+4` is the same as `((1+2)-3)+4`

- When unary operators have equal precedence, they are evaluated right-to-left.

Precedence Rules

- Even when parentheses are not needed, they can be used to make the code clearer.

- Spaces also make code clearer:

  `1 + 2*3`

  but spaces do not dictate precedence:

  `1+3 * 4-2` is `11`

Precedence examples

```
1 * 2 + 3 * 5 % 4
 \_/
  |
2   + 3 * 5 % 4
        \_/
         |
2   + 15   % 4
        \____/
           |
2   +      3
 \_____________/
        |
        5
```

- Equivalent fully parenthesized expression:

```
((1 * 2) + ((3 * 5) % 4))
```

Real numbers (type `double`)

- Examples: `6.022`, `-42.0`, `2.143e17`
  - Placing `.0` or `.` after an integer makes it a `double`.
- The operators `+-*/%()` all still work with `double`.

  `/` produces a `double` answer: `15.0 / 2.0` is `7.5`

  Precedence is the same: `()` before `*/%` before `+-`

Real number example

```
2.0 * 2.4 + 2.25 * 4.0 / 2.0
  \____/
    |
 4.8     + 2.25 * 4.0 / 2.0
                \____/
                  |
 4.8     +     9.0    / 2.0
                \_______/
                    |
 4.8     +         4.5
    \_________________/
              |
            9.3
```

Mixing types

- When `int` and `double` are mixed, the result is a `double`.

  `4.2 * 3` is `12.6`

- The conversion is per-operator, affecting only its operands.

```
– 7 / 3 * 1.2 + 3 / 2
–   \_/
     |
     2   * 1.2 + 3 / 2
–     \____/
       |
      2.4     + 3 / 2
–                 \_/
                   |
      2.4     +    1
–        \__________/
              |
             3.4
```

  – `3/2` is `1` above, not `1.5`.

String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hi" + " there" is "hi there"
"hello" + 42    is "hello42"
"abc" + 1 + 2   is "abc12"
1 + 2 + "abc"   is "3abc"
"abc" + 9 * 3   is "abc27"
"1" + 1         is "11"
4 - 1 + "abc"   is "3abc"
```

- Use + to print a string and an expression's value together.

```
System.out.println("Grade: " + ((95.1 + 71.9) / 2));
```

Output: `Grade: 83.5`

Receipt example

- What's bad about the following code?

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax & 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);

        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .08 +
                           (38 + 40 + 30) * .15);
    }
}
```

  - The subtotal expression `(38 + 40 + 30)` is repeated
  - So many `println` statements

Variables and Values

- *Variables* store data such as numbers and letters.
  - Think of them as places to store data.
  - They are implemented as memory locations.
- The data stored by a variable is called its *value.*
  - The value is stored in the memory location.
- Its value can be changed.

Naming and Declaring Variables

- Variables have two attributes: a *name* and a *type*
  - The name is an *identifier* and must obey Java's rules

- When you *declare* a variable, you provide its type and name.

```
int numberOfBaskets, eggsPerBasket;
```

- A variable's *type* determines what kinds of values it can hold (`int, double, char,` etc.).

- A variable must be declared before it is used.

- Choose names that are helpful such as `count` or `speed`, but not `c` or `s`.

Syntax and Examples

- syntax

  `type  variable_1, variable_2, …;`

  (`variable_1` is a generic variable called a syntactic variable)

- examples:

```
int styleChoice, numberOfChecks;
double balance, interestRate;
char jointOrIndividual;
```

Where to Declare Variables

- Declare a variable…
  - just before it is used for the first time, or
  - at the beginning of the section of your program that is enclosed in {}.

```
public static void main(String[] args)
{
  // declare variables here

}
```

Assignment Statements

- An assignment statement is used to assign a value to a variable.

```
answer = 42;
```

- The “equal sign” is called the *assignment operator.*

- We say, “The variable named `answer` is assigned a value of 42,” or more simply, “`answer` is assigned 42.”

Assignment Statements, cont.

- Syntax

  variable = expression;

  where ***expression*** can be another variable, a *literal* or *constant* (such as a number), or something more complicated which combines variables and literals using *operators* (such as + and -)

Assignment Examples

```
amount = 3.99;

firstInitial = 'W';

score = numberOfCards + handicap;
```

Assignment Evaluation

1. The expression on the right-hand side of the assignment operator (=) is evaluated first.

2. The result is then used to set the value of the variable on the left-hand side of the assignment operator.

```
score = numberOfCards + handicap;

eggsPerBasket = eggsPerBasket - 2;
```

Using variables

- Once given a value, a variable can be used in expressions:

```
int x;
x = 3;
System.out.println("x is " + x);      // x is 3

System.out.println(5 * x - 1);        // 14
```

- You can assign a value more than once:

```
int x;
x = 3;
System.out.println(x + " here");      // 3 here

x = 4 + 7;
System.out.println("now x is " + x); // now x is 11
```

Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators (including `+`, `-`, `*`, `/`, and `%`).

```
amount = amount + 5;
```

can be written as

```
amount += 5;
```

yielding the same results.

Shorthand	Equivalent longer version
variable += expr;	**variable = variable + (expr);**
variable -= expr;	**variable = variable - (expr);**
variable *= expr;	**variable = variable * (expr);**
variable /= expr;	**variable = variable / (expr);**
variable %= expr;	**variable = variable % (expr);**

Declaration/initialization

- A variable can be declared & initialized in one statement.

- Syntax:

  type name = **value**;

```
double myGPA = 3.95;
int x = (11 % 3) + 12;
```

Assignment Compatibilities

- Java is said to be *strongly typed.*
  - You can't, for example, assign a floating point value to a variable declared to store an integer.

```
int myNumber = 7.5; // Error: Compiler will not allow
```

- Sometimes conversions between numbers are possible.

  `double myVariable = 7;`

  is possible even if `myVariable` is of type `double`.

  In this case, the compiler will automatically convert the integer 7 into a floating point `7.0`.

- This automatic conversion is called a *coercion*.

Assignment Compatibilities

- A value of one type can be assigned to a variable of any type further to the right

  `byte --> short --> int --> long --> float --> double`

  but not to a variable of any type further to the left.

- E.g., you can assign a value of type `char` to a variable of type `int`, or a value of type `int` to a variable of type `double`, but you cannot assign a value of type `double` to a variable of type `int`.

Type Casting

- A *type cast* temporarily changes the *value* of a variable from the declared type to some other type. It does **not** change the variable.

- For example,

```
double distance;
distance = 9.0;
int points;
points = (int)distance;
```

  the above is illegal without the `(int)`

- Uses:
  - To promote an `int` into a `double` to get real-number division from the `/` operator
  - To truncate a `double` from a real number to an integer

Type Casting, cont.

- The value of `(int)distance` is `9`, but the value of `distance`, both before and after the cast, is `9.0`.

- Any nonzero value to the right of the decimal point is *truncated* rather than *rounded.*
  - Thus if the value of `distance` was `9.7`, the value of `(int)distance` would still be `9`
  - Again, the value of `distance` is not changed and would still be `9.7`.

Examples:

```
double result = (double) 19 / 5;      // 3.8
int result2 = (int) result;           // 3
```

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2;  // 1.0`
  - `double y = 1 + (double) 1 / 2;  // 1.5`
- You can use parentheses to force evaluation order.
  - `double average = (double) (a + b + c) / 3;`
- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`
  - `double average = (a + b + c) / 3.0;`