CS 106
Introduction to Computer Science I

09 / 18 / 2015

Instructor: Michael Eckmann
Today’s Topics

• Questions / Comments?
• User input / output using JOptionPane
• Pseudocode
• Larger exercise
• User input using Scanner
• How java determines type
• Increment and decrement operators
import statement

• The import statement is of the form
  – import package.class;

• `javax.swing` is a package and `JOptionPane` is the class we want to use in it.

• `import javax.swing.*` would allow us to use any class in the `javax.swing` package.
import statement

• We didn’t need to import anything to use System.out in the HelloWorld program because System is a class that is part of the java.lang package and java.lang gets implicitly imported in all Java programs.

• Java API Specification is on the web and is where you’ll find all the classes available to you in the Java API. There is a link to it on our course web page.
javax.swing.JOptionPane

• Contains methods to show dialog boxes.
• JOptionPane.showMessageDialog displays messages in a dialog box.
• JOptionPane.showInputDialog allows for user input in a dialog box.
• Nicer to use than reading from / printing to the console screen.
JOptionPane class for I/O

• `showMessageDialog` takes two parameters, the first of which should always be null for now.
• The second is a String that should be outputted.

• `showInputDialog` takes one parameter which is a String that displays --- it should tell the user what to enter. The user's input is returned as a String.
JOptionPane class for I/O

• Example usage:

    import javax.swing.JOptionPane; // need this line above your class

    String input_string; // variable to store the user's input

    JOptionPane.showMessageDialog(null, "Hey");
    input_string = JOptionPane.showInputDialog("Enter something.");
javax.swing.JOptionPane

• Let’s write Hello World using a message dialog box instead of System.out.println.
java.util.Scanner

• Scanner input_scan = new Scanner(System.in);
• methods that you can call on your scanner object include:
  – nextInt() --- reads an int from keyboard
  – nextDouble() --- reads a double from keyboard
  – nextLine() --- reads a line (as a String) from keyboard
  – next() --- reads a “word” from the keyboard --- which is a string of nonwhitespace chars delimited by whitespace. whitespace is \n, blank space character, \t, \r
java.util.Scanner

• `nextLine()` --- reads a line (as a String) from keyboard
  – this method “consumes” the `\n` but does not make it part of the String that is returned.

    String s1, s2;
    Scanner my_scan = new Scanner(System.in);
    s1 = my_scan.nextLine();
    s2 = my_scan.nextLine();
  – if input is:
    CS106
    Introduction to Computer Science I
  – there is a `\n` after CS106 in the input (e.g. user hit enter key), but s1 will be “CS106” and s2 will be “Introduction to Computer Science I” neither will have `\n` as a character in its String.
There's an unfortunate problem with Scanner when using one to get both numeric and String input.

If you need to get both numeric and String input from user input from the keyboard I recommend creating two Scanners, one that only gets numeric input (nextInt, nextDouble) and one that only gets the String input (nextLine, next).
Pseudocode

• pseudocode is an informal use of English to describe what a program is to do and in what order
• pseudocode is not an actual computer programming language
• it is used prior to writing actual code to help the programmer in the planning stages of a program
Example Application Exercise

• write a program to compute the number of projected home runs a baseball player will hit for the season based on how many homers he’s hit so far.

• Output should look like:

  *player’s name* is projected to hit *number* home runs in 162 games.

• Any ideas?
Pseudocode for our example

get player’s name from the user
get the number of homeruns so far
get the number of games played so far

compute the number of projected homeruns for this player based on a season of 162 games by using the following calculation

\[
\frac{\text{projected homers}}{162} = \frac{\text{homers so far}}{\text{games played so far}}
\]
from this equation,

\[
\frac{\text{projected homers}}{162} = \frac{\text{homers so far}}{\text{games played so far}}
\]

we can multiply both sides of the equation by 162 and get

\[
\text{projected homers} = \frac{\text{homers so far} \times 162}{\text{games played so far}}
\]
Pseudocode for our example
(continued)

Print out the following with actual values for *player’s name* and *number*

*player’s name* is projected to hit *number* home runs in 162 games.

Pseudocode could be more fleshed out than what we have done here --- use as much or as little detail in pseudocode as you prefer.
Pseudocode for our example (continued)

• Now we can write the program based on our pseudocode.
How Java determines type

• Each operator in a complex expression is evaluated one at a time. The order they are done is according to the precedence rules we learned last time.

• What Java does (and we need to do as well if we are to understand how it works) is it evaluates one operator at a time and uses information about the type of operand(s) being worked on to determine the type of the result.

• example expressions:

  14 * some_int     // assume that the type of some_int is int

  3.14159 * diameter

    // assume that the type of diameter is double

• Java figures out the resulting types according to the types of the operands. This is easy if the operands are the same type.
How Java determines type

• If the operands are of different types, Java promotes/converts one of the operands to the type of the other thereby temporarily making that value the same type and makes that the resulting type as well. This promotion is sometimes called type coercion.

• We saw a kind of type coercion when we concatenate Strings with non-strings. Java temporarily converts the non-string to a String and concatenates them.

• In expressions among operands of different types, Java does the following. It temporarily converts the smaller type into the larger type, then does the operation and the result is the larger type.

• The order of the primitive types from “smaller” to “larger” is:
  – byte, short, int, long, float, double
  – this list excludes char and boolean
How Java determines types

- Examples
  45 + 34.5     // an int is being added to a double

  19 / 9        // an int is being divided by an int
How Java determines types

• Examples:
  
  int a = 5, b = 6, c;
  double d1 = 4.3, d2 = 88.4, d3;
  
  d3 = d1 * a;
  /* here a's value (not the a variable) is temporarily converted to be used as a double (just for this calculation) and then the multiplication is done and the result is a double. This result is then assigned to the variable d3 */
  
  // problem if try to assign a double to an int, e.g.
  c = d2 * b; // what will happen? Try it and see.
another promotion example

• E.g.

    int count = 3;
    double sum=6.7, result;
    result = sum / count;
    /* count's value is promoted to be used as a double automatically so the division can take place */
**Assignment conversion**

- The name given to Java’s type conversion that is done when a value of one type is assigned to a variable of another type. This is allowed as long as we are assigning a “smaller” type to a “larger” type. What does that mean?

- E.g.
  ```java
  int count = 3;
  float result;
  result = count;
  /* count is converted to be used as a float automatically and its value is assigned to result 3.0 */
  ```
Casting conversion

• The name given to the programmer’s ability to force type conversion.

• E.g.

```c
int number_of_students, number_of_faculty;
double students_per_faculty;
students_per_faculty =
    (double) number_of_students / number_of_faculty;
```

/* force number_of_students to be used as a double and then result of division is double… also contains use of promotion --- where is promotion done? */
Casting conversion

• Another example.

```c
int homers_so_far, games_so_far;
double projected_homers;
// assume the ints get values from somewhere then ...
projected_homers =
    (double)162 * homers_so_far / games_so_far;

// I'll explain exactly what is going on here with the types
```
Increment & decrement (sec. 2.2)

• `++` and `- -`
• `note: there must be no space btwn the minuses or the plusses`
• `++` adds 1 to the variable
• `- -` subtracts 1 from the variable

```
int count = 55;
count++; // same as count = count + 1;
```
Increment & decrement (sec. 2.2)

```java
count++; // adds 1 to count
++count; // also adds 1 to count

count--; // subtracts 1 from count
--count; // also subtracts 1 from count
```

- Which side the `++` or `- -` is on only matters when it's used within an expression.
Increment & decrement (sec. 2.2)

• Note:
  ++ before a variable or expression is a preincrement,
  ++ after a variable or expression is postincrement,
  -- before a variable or expression is a prededecrement,
  -- after a variable or expression is a postdecrement.
Increment & decrement (sec. 2.2)

• Which side the ++ or -- is on only matters when it's used within an expression.

• e.g.
  total = count++;  

  acts differently than 

  total = ++count;
Increment & decrement (sec. 2.2)

total = count++;  
/* the above line assigns the value of count to total and then adds 1 to count (hence the name post-increment) */

// so what's the difference?

total = ++count;  
/* the above line adds 1 to the value of count and then assigns this new value of count to total (pre-increment) */
Increment & decrement (sec. 2.2)

Also similar behavior in conditions of if's

e.g.

```cpp
if (index++ >= 5)
    // execute something if true
else
    // execute something if false
```

/* the above first compares index and 5 to determine if index is >=5, then 1 is added to index, then depending on result of the compare, the if or else line will execute */
Increment & decrement (sec. 2.2)

Also similar behavior in conditions of if's

e.g.

    if (++index >= 5)
        // execute something if true
    else
        // execute something if false

    /* the above first adds 1 to index and then compares
       this new value of index to 5 to determine if index is
       >=5, then depending on result of the compare, the if
       or else line will execute */