CS 106
Introduction to Computer Science I

09 / 23 / 2015

Instructor: Michael Eckmann
Today’s Topics

• Comments and/or Questions?
• Increment / Decrement operators
• More assignment operators
• Wrapper classes
• method terminology
• comparing data
• other useful String class methods
Increment & decrement (sec. 2.2)

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

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

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 predecrement,
  -- 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.

```
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 */
More assignment operators besides =

- +=, -=, *=, /=, %=  

Here's what they mean. Suppose we have:
```
int a=7;
    // is equivalent to: a += 6;
```

These assignment operators are used only if the LHS (left hand side of the assignment) is also the first variable on the RHS.
More assignment operators besides =

- +=, -=, *=, /=, %= 

More examples:

```c
degrees -= 5;
/* subtract 5 from degrees and store this new value in degrees. */
halve_me /= 2;
/* divides the value in halve_me by 2 and stores this new value in itself */
fine *= 3; // value in fine is tripled
```
“Wrapper classes”

• Double is not the same as double. double is a primitive type, Double (with a capital D) is a class. There are what are called “wrapper” classes for all the primitive types. They are:

  Integer    int
  Double     double
  Character  char
  Float      float
  Boolean    boolean
  Byte       byte
  Long       long
  Short      short
using wrapper classes for conversion

- We used a method in a few of the wrapper classes to convert a String into the wrapper class' associated primitive type.
- How did we do that? Does anyone remember?
using wrapper classes for conversion

- We used a method in a few of the wrapper classes to convert a String into the wrapper class' associated primitive type.
- How did we do that? Does anyone remember?

```java
int homers_int;
String homers_str = "49";
homers_int = Integer.parseInt(homers_str);
```

```java
double weight;
String weight_str = "182.5";
weight = Double.parseDouble(weight_str);
```
method terminology

- **void** – another “type” in Java which is used when no value is needed.
method terminology

• **Parameter** – methods have 0 or more parameters. These specify what types of values are used when a method call is made.

• **Calling a method** – invoking a method by giving its name in a statement in your program:
  
  – e.g.
  
  – System.out.println("Hey");  // method call for println method
  
  – height = Integer.parseInt(height_str);  // method call for parseInt method

• **Note**: A String is being passed in as a parameter for println. Same for parseInt.
method terminology

• **Return type of a method** – This is what type the result of a method call gives.
  
  e.g.
  – `System.out.println("Hey"); // nothing is returned (void)`
  – `height = Integer.parseInt(height_str); // an int is returned`

• The return type and number of parameters and types are all specified in the definition of a method. For the Java API methods, we can look this stuff up online.
Comparing data

• Comparing primitive type variable values is different than comparing variables of non-primitive types.
• Recall the primitive types: char, byte, short, int, long, float, double, boolean.
• String, Integer, Double, and many others which we have yet to see are classes which can be types for variables. These kinds of variables are not of the primitive types. These kids of variables are also called objects or object references.
Comparing data

• Example chars
  
  ```
  char some_char = 'a', first_letter = 'E', a_digit = '7';
  // note: a char literal is always in single quotes.
  ```

• Examples of how to compare chars for equality
  
  ```
  (some_char == 'a')
  (some_char == 'b')
  (some_char == a_digit)
  ```

• Also could compare less than, greater than, etc. for chars

• What would that mean?
Comparing data

- Example ints (same for byte, long, short)
  ```java
  int height = 65, length = 4, width = 6;
  ```

- Examples of how to compare ints
  ```java
  (height < 70)
  (width == length)
  ```
Comparing data

- Example booleans
  
  ```java
  boolean done_yet = false, old_enough = true;
  ```

- Examples of how to compare booleans
  
  ```java
  (done_yet == true)
  (done_yet)
  (old_enough)
  (!old_enough)
  (done_yet == old_enough)
  ```
Comparing data

• Example doubles (same for floats)
  double weight = 5.6;
  // note: a decimal number literal is assumed by Java to be
  // a double (not a float)  5.6f  is how to create a float literal

• Examples of how to compare doubles
  (weight > 5.1)
  (weight == 5.6) // problem because doubles (and floats)
  // are not stored exactly (depending on the value)
  // so it is dangerous to check for equality – don't do it.
Comparing data

• Example Strings
  String name = “Mike”;
  // note: a String literal is enclosed in double quotes

• Example of how not to compare Strings
  (name == “Mike”) /* sometimes works sometimes
  doesn't because since name is an object (not a variable
  of a primitive type), when compared using the
  comparison operators, Java compares the memory
  locations of name and “Mike”, not the values. */
Comparing data

• Example Strings
  String name = “Mike”;

• Example of how to compare Strings (use method(s) in the String class for this purpose)
  (name.equals(“Mike”)) // or
  (name.equalsIgnoreCase(“mike”))

• Here we are calling methods in the String class. If you had to guess, what is the type of the value that is returned by these methods?
Comparing data

• Another way to compare Strings is to use the compareTo method of the String class.
• The calling String (the one to the left of the . ) is compared lexicographically to the String that is passed in as a parameter. If the calling String is less than the parameter String the result is a negative number. If they are equal, the result is 0. Otherwise the result is a positive number.
• What is the type of the value that is returned by this method?
other String class methods

- boolean equals(String s)
- boolean equalsIgnoreCase(String s)
- char charAt(int index)
- int length()
- String toUpperCase()
- String toLowerCase()

All of these methods are called by using a String object followed by the . (dot operator) then the name of the method (and any necessary parameters within the parens).
calling methods in String

- Notice how we call methods in the String class:
  - first we need something of type String (an object/variable OR a String literal)
  - then we use the . (dot operator)
  - then we specify the name of the method
  - then in parentheses we put whatever parameters are required
  - we also pay attention to what type is returned by the method

- examples:

  ```java
  String name = "Eckmann";
  int len, len2;
  len = name.length();
  len2 = "CS106".length();
  ```
other String class methods

- Examples (let's put some of this code in a program):
  - String name = “Joe”, name2, lastname;
  - char a_char;
  - if (name.length() > 5)
    - a_char = name.charAt(2);
  - name2 = name.toUpperCase();
if (lastname.compareTo(“Jones”) < 0)
    System.out.println(lastname + “ comes before Jones”);
else
  if (lastname.compareTo(“Jones”) > 0)
    System.out.println(“Jones comes before ” + lastname);
  else
    System.out.println(“Jones is the same as ” + lastname);