Chris Piech
CS 106A

Handout #17
January 30, 2017

Variables, variables, everywhere…

Based on a handout by Patrick Young.

Local Variables

Local variables are created local to the method (or the block—see "Block Scope" section below) in which they are defined. They are destroyed when execution of the method has been completed. Local variables can only be accessed from within the method in which they are declared. Because of their transient nature, local variables cannot store persistent information about an object between method calls.

Local Variables Example

Consider, for example, the following snippet of code.

```
public class AnExample extends ConsoleProgram {

   public void methodOne {
       int a = readInt("Enter a: ");
       println("a = " + a);
   }

   public void methodTwo {
       int b = a;  // BUG!: cannot refer to variable a in methodOne
       println("b = " + b);
   }
}
```

The variables **a** and **b** are local variables declared within different methods in the class **AnExample**. Because these variables are local variables, **a** can only be referred to within **methodOne** and variable **b** can only be accessed within **methodTwo**. Our attempt to initialize **b** using the value of **a** is illegal, as code in **methodTwo** cannot access local variables from **methodOne** or any other method.

Because local variable values do not persist after their containing method has completed, the variable **a** will be destroyed when **methodOne** has completed execution. The next time **methodOne** is called, a new variable **a** will be created.

Block Scope

While we typically think of local variables as local to a particular method, in Java local variables are actually local to a *block* of code. While a method defines a block of code (since the opening and closing braces of the method define a block), **for** and **while** loops, **if**-statements, and other constructs are also considered blocks of code. If we declare a local variable inside one of these constructs, the local variable will be created when it is declared in the block and destroyed when the block ends execution.

Consider the following example:

```
public class AnExample extends ConsoleProgram {

   public void methodThree {
      int a = 4;

      for (int i = 0; i < 5; i++) {
         int b = a; // this is okay

         if (a > b) { // okay to access a and b here
            int c = 3;

            println(i); // okay to access i here
            println(b); // okay to access b here
            println(c); // okay to access c here
         }

         println(c); // illegal: c is no longer in scope here
      }

      println(a); // okay to access a here
      println(b); // illegal: b is only in scope in body of for loop
      println(i); // illegal: i is only in scope in body of for loop
   }
}
```

The variable **a** is local to **methodThree**, so accessing **a** within the **for** loop, or in the **println** is fine. The variables **i** and **b** are only in scope (i.e., "alive") within the confines of the **for** loop in which they are declared. They are no longer available by the time the last two **println**s try to access them. The variable **c** is only in scope within the confines of the **if** statement body in which it is declared. Note that a new copy of **c** will get created each time the **if** statement body is executed, therefore it cannot be used to store values across iterations of the **for** loop.

Instance Variables

Instance variables (also known as "I-vars") are defined as part of a class, but not within any particular method of the class. Each object of the class will have its own independent copy of all the instance variables defined in a class. Instance variables are created when an object is created. They are destroyed when their corresponding object is destroyed. Instance variables can be accessed from any method defined as part of the class in which the instance variable is defined. Access to instance variables from *other* classes is controlled by the variable's visibility specifier (e.g., **public** or **private**). Instance variables that are **public** are accessible from methods in other classes while those that are **private** (which is by far the more common case) are not.

Instance Variables Example

```
public class Thing {
   public Thing() {
      x = 0;
   }

   /* Public instance variable */
   public int x;
}

public class MyProgram extends ConsoleProgram {

   public void run() {
      Thing th1 = new Thing();
      Thing th2 = new Thing();
      Thing th3 = new Thing();
      th1.x = 8;        // Can access x here because it is public
      th2.x = 17;
      th3.x = 18;
      println(th1.x);
      println(th2.x);
      println(th3.x);
   }
}
```

In this case we have a class **`Thing`** which has an instance variable **`x`**. Each instance of the class has its own *independent* copy of the variable. Our main program creates three instances (objects) of the **`Thing`** class. It then sets the value of each instance's **`x`** variable. Because there are three different **`Thing`** objects, there are three different **`x`** variables—one **`x`** variable per instance. We can change the variable values independently from one another. When we print out the results we'll see:

```
8
17
18
```

Constants

As we discussed in class, we will sometimes find it useful to define a "variable" whose value does not change during the entire program. Such a variable is called a **constant**. In this case, the constant does not fulfill the traditional purpose of a variable (i.e., storing a temporary program value). To declare a variable as a constant, use the keyword **`final`** when declaring the variable. As mentioned above, virtually all constants are *class* variables. However, it is possible to also define constant instance variables or constant local variables.