1.00 Lecture 6

Methods and Scope

Reading for next time: Big Java: sections 2.6-2.10, 3.1-3.8

Java Methods

- Methods are the interface or communications between program components
  - They provide a way to invoke the same operation from many places in your program, avoiding code repetition
  - They hide implementation details from the component using the method
  - Variables defined within a method are not visible to users of the method; they have local scope within the method
  - The method cannot see variables in the component that calls it either. There is logical separation between the two, which avoids conflicts in variable names

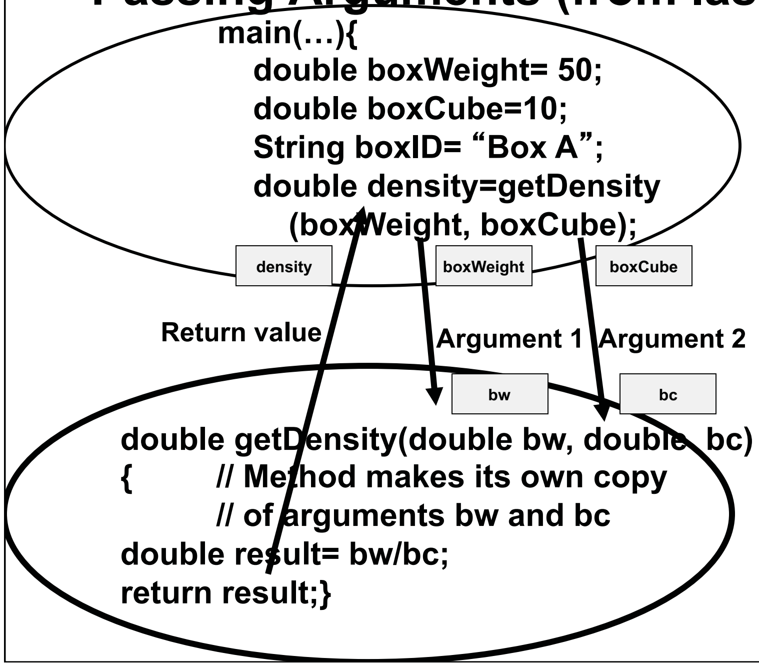
From last time

Method example

```
public class MethodExample {
   public static void main(String[] args) {
       double boxWeight= 50;
       double boxCube= 10;
       String boxID= “Box A”;
       double density= getDensity(boxWeight, boxCube);
       System.out.println("Density: "+ density);
       printBox(boxWeight, boxCube); // Prints density 2nd time
   }
   public static double getDensity(double bw, double bc) {
        double result= bw/bc; // 'result' could be 'density'
        return result;
   }
   public static void printBox(double w, double c) {
        System.out.println("Box weight: "+w+" cube: "+c);
        System.out.println(" Density: "+getDensity(w,c));
//      System.out.println(“ ID: “+boxID);  // No access to ID
   }                                      // Won’t compile!
}
```

From last time

Passing Arguments (from last time)



```
main(…){
   double boxWeight= 50;
   double boxCube=10;
   String boxID= “Box A”;
   double density=getDensity
      (boxWeight, boxCube);
```

```
double getDensity(double bw, double  bc)
{        // Method makes its own copy
         // of arguments bw and bc
double result= bw/bc;
return result;}
```

Communi-cation only via arg list, return value

Arguments matched by position

Data type, meaning must match

From last time

Method exercise

- Write a class MethodExercise
  - main() method:
    - Declares String name, int age, double height
    - Sets variables to your name, age, height
    - Calls isOldEnough() method
    - Prints out whether old enough (true or false)
    - Calls printInfo() method
  - Method isOldEnough() returns true if age >= 21, false otherwise
  - Method printInfo() prints name, age, height
  - Choose appropriate arguments, return values

Method exercise continued

```
public class MethodExercise {
   public static void main(String[] args) {
       …
   }

   public static boolean isOldEnough(int a) {
       …
   }

   public static void printInfo(String n, int a, double h) {
       …
   }
}

// Exercise, continued:
// Compute BMI= 703 * weight / (height)² in its own method
// (Weight in pounds, height in inches)
// Declare and initialize weight variable in main()
// Output BMI in printInfo(). Change printInfo arguments.
```

AvgTest Exercise: step 1

- In Eclipse, create a new class AvgTest
  - Have Eclipse write the main() signature but leave it empty for now
- After the main method, write methods to:
  - Return the average of three doubles x1, x2, x3
  - Return the maximum of three doubles x1, x2, x3
  - The methods will be:

    ```
    public static double average3(…) { … }
    public static double maximum3(…) { … }
    ```
    - In maximum3(), compare pairs of values
    - There s an easy way and a hard way

AvgTest Exercise: step 2

- In the main method, which is currently empty:
  - Define two sets of variables (doubles):
    - 10, 17, 55 (r1, r2, r3)
    - 59, -3, 85 (r4, r5, r6)
  - Call average3() and maximum3() on the first 3 doubles (10, 17, 55)
  - Output (System.out.println) the results
  - Call average3() and maximum3() on the next 3 doubles (59, -3, 85)
  - Output (System.out.println) the results

Pass by copy

- In Java, arguments are passed from one method to another by copy (also called by value):
  - The called method makes a copy of the arguments. Even if it changes their values, they do not change in the calling method.
  - What is the output (4 values) of the following program?

```
public class TripleTest {
  public static void main(String[] args) {
     double z=5.0;
     System.out.println("z main 1: "+z);
     triple(z);
     System.out.println("z main 2: "+z);
  }
  public static void triple(double z) {
     System.out.println("z 1: "+z);
     z *= 3;
     System.out.println("z 2: "+z);
  } }
```

Scope

- You' ve already seen that methods have different scope:
  - A variable of the same name in two methods is two separate variables
- Scope of local variables, the only kind we' ve seen so far, is defined by additional rules
- And, there are other kinds of variables, with their own scope rules
- We' ll revisit all this later, but for now, we focus on local variable scope

Local Variable Scope

- Local variables (in a method or block)
  - Exist from point of definition to end of block
    - Blocks are defined by curly braces{ }
    - Blocks are most often used to define:
      - Method body
      - Multiple statements in if-else and loop operations
  - Local variables are very restricted:
    - Other methods cannot see local variables even in the same class.
    - Variables of the same name in different methods are different variables
    - More generally, variables of the same name in different blocks are different variables
  - Arguments to a method are local variables:
    - The method copies them upon receipt and they live until the ending curly brace of the method
  - Variables defined in for, while and do-while statements exist in the loop body

Exercise

- Mark where variables d, e, i, j exist (i is given as example)

```
 public class ScopeTest0 {
    public static void main(String[] args) {
i        int i= 1;
|        double d= 0.0;
|        for (int j= 0; j < 5; j++) {
|               double e= j;
|               d += i;
|               e += j;
|               System.out.println("d: "+d+" e: "+e);
|        }
|        if (d > 0) {
|               int j= 2;
|               double e= 4.0;
|               System.out.println("If line d: "+d+" e: "+e);
|        }
|        double e= 0.0;
|        e += d + i;
|        System.out.println("Last line d: "+ d+" e: "+e);
↓   }
 }
```

Scope exercise

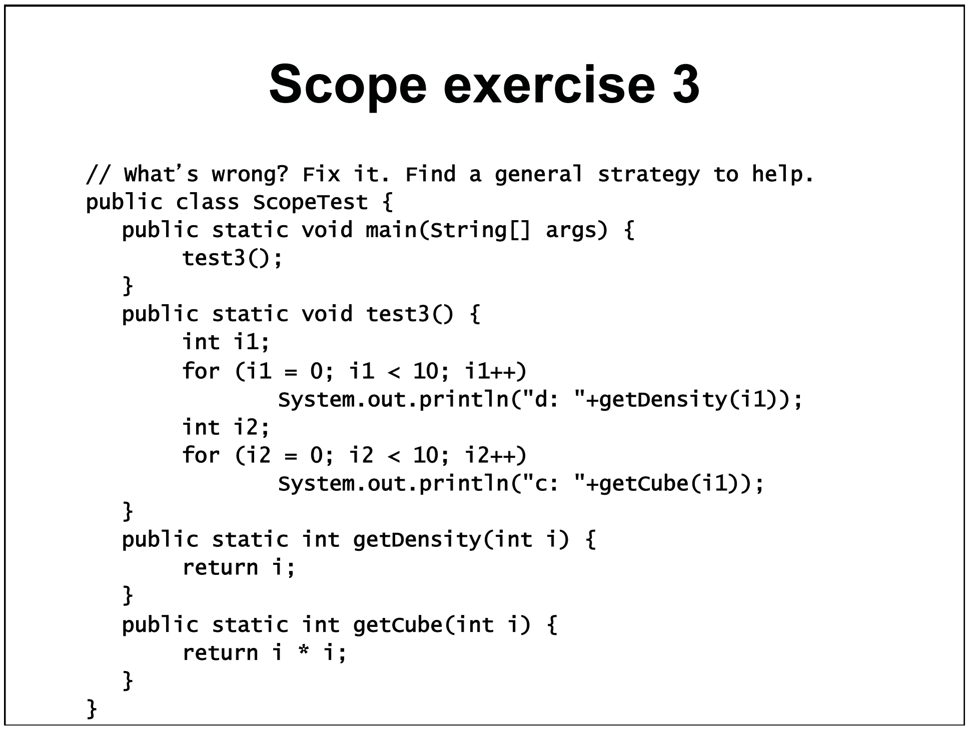
- **The following code doesn't work. Fix it.**

```
public static int test1() {
    for (int i=0; i < 10; i++) {
          if (Math.sqrt(i) > 2.5)
                break;
    }
    return i;
}
```

Scope exercise 2

- **The following code doesn't work. Fix it.**

```
public static void test2() {
    int i= 4;
    if (i*i > 6) {
          int i6= i;
    }
    int i7= i6 + 2;
}
```



Scope exercise 3

```
// What's wrong? Fix it. Find a general strategy to help.
public class ScopeTest {
   public static void main(String[] args) {
        test3();
   }
   public static void test3() {
        int i1;
        for (i1 = 0; i1 < 10; i1++)
                System.out.println("d: "+getDensity(i1));
        int i2;
        for (i2 = 0; i2 < 10; i2++)
                System.out.println("c: "+getCube(i1));
   }
   public static int getDensity(int i) {
        return i;
   }
   public static int getCube(int i) {
        return i * i;
   }
}
```

MIT OpenCourseWare
http://ocw.mit.edu

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.