DECLARING METHODS, SCOPE, OVERLOADING

Robert A. Fulkerson
University of Nebraska at Omaha
College of Information Science and Technology

Defining A Method

- Think about creating methods in terms of three things:
  - Parameters
    - Values to send to method
    - Ask yourself, "What does the method need so that it can do its job?"
  - Task
    - What does the method do?
  - Return/Results
    - What does the method produce as a result?
    - What is its final answer?
    - Does it even need to return anything?
- In other words, if you think you need a method, try to identify these three things for each method before you write the code.

Basic Structure of a Method

accessmodifiers specify where a method can be accessed. For now, public and static.

```
returntype methodName (parameterlist)
{
  declarations and statements
}
```

- `accessmodifiers` returntype methodName (parameterlist) This entire line is known as the "method header"
- `accessmodifiers` specify where a method can be accessed. For now, public and static
- `returntype` is the data type of the result returned from the method to the caller.
- `methodName (parameterlist)` is any valid identifier
- `declarations and statements` form the "body" of the method and do all the work
- `parameterlist` is a comma-separated list of the parameters received by the method when it is called. If a method does not receive any values, the parameter list is left empty. A type must be listed explicitly for each parameter in the parameter list.
Square a number with a method

```java
1. import java.util.Scanner;
2. public class SquareExample {
3.   public static void main ( String args[] ) {
4.     for ( int x = 1; x <= 10; x++ )
5.       System.out.printf ( "%2d squared is %3d
", x, square(x) );
6.   } // end of main() method
7. }
8. public static int square ( int temp ) {
9.   int result = (int) (Math.pow ( temp, 2 ) );
10.  return result;
11. } // end of square() method
12. } // end of class
```

Finding the largest of three numbers

```java
1. import java.util.Scanner;
2. public class MaximumFinder {
3.   public static void main ( String args[] ) {
4.     Scanner in = new Scanner ( System.in );
5.     System.out.print("Enter three values:");
6.     double n1 = in.nextDouble();
7.     double n2 = in.nextDouble();
8.     double n3 = in.nextDouble();
9.     // method call is below
10.    double result = maximum ( n1, n2, n3 );
11.    System.out.println ( "Maximum is: " + result );
12. } // end of main() method
```

```java
13. public static double maximum
14.   ( double x, double y, double z ) {
15.    double maximumValue = x;
16.    if ( y > maximumValue )
17.       maximumValue = y;
18.    if ( z > maximumValue )
19.       maximumValue = z;
20.    return maximumValue;
21. } // end of maximum() method
22. } // end of class
```

Part 2 of 2

Finding the largest of three numbers

```java
13. public static double maximum
14.   ( double x, double y, double z ) {
15.    double maximumValue = x;
16.    if ( y > maximumValue )
17.       maximumValue = y;
18.    if ( z > maximumValue )
19.       maximumValue = z;
20.    return maximumValue;
21. } // end of maximum() method
22. } // end of class
```

// Notice the use of static in line 13. This is done
// so that the static main method can call this method.
Scope

- Scope refers to the portion of a program that a variable, method, or class can be used in.

  - Block scope:
    - The scope of a parameter variable is the body/block in which the declaration appears.
    - The scope of a local variable inside of a method is from the point of declaration to the end of that block, usually a closing curly brace: }.
    - The scope of a local variable in the initialization of a for loop is the rest of the for header and the body of the for loop itself.

  - Class scope:
    - The scope of a method or field is the entire body of the class, meaning all methods can access all other methods and fields.

ScopeDemo example

1. public class ScopeDemo {
2.     private static int x = 1; // class scope
3.     public static void main ( String args[] ) {
4.         int x = 5; // local x for method, block scope
5.         System.out.println("local x in main is " + x);
6.         useLocalVariable();
7.         useField();
8.         useLocalVariable();
9.         useField();
10.        System.out.println("local x in main is " + x);
11.    }
12. }

13. public static void useLocalVariable () {
14.     int x = 25; // local x, block scope
15.     System.out.println("local x entering useLV is " + x);
16.     ++ x; // modifies uLV's x only
17.     System.out.println("local x before leaving useLV is " + x);
18. }
19. public static void useField () {
20.     System.out.println("field x on entering useF is " + x);
21.     x *= 10; // modifies field x of class
22.     System.out.println("field x before leaving useF is " + x);
23. }

ScopeDemo example

12. public static void useLocalVariable () {
13.     int x = 25; // local x, block scope
14.     System.out.println("local x entering useLV is " + x);
15.     ++ x; // modifies uLV's x only
16.     System.out.println("local x before leaving useLV is " + x);
17. }
18. public static void useField () {
19.     System.out.println("field x on entering useF is " + x);
20.     x *= 10; // modifies field x of class
21.     System.out.println("field x before leaving useF is " + x);
22. }
23. }
Method Overloading

- Java allows several methods of the same name to be defined ...
- ... as long as these methods have unique parameter lists.
- This is known as method overloading.
- When an overloaded method is called, the compiler picks the appropriate method to invoke based upon the number, types and order of the arguments in the call.

Method Overloading

- Usually used to provide different methods (of the same name) that provide similar tasks on different data types.
- For instance, there are four versions of methods `abs`, `min` and `max` in class Math:
  - One that handles two `double` parameters
  - One that handles two `float` parameters
  - One that handles two `int` parameters
  - One that handles two `long` parameters

MethodOverloadOne example

```java
public class MethodOverloadOne {
    public static void main ( String args[] ) {
        System.out.println("Square of 7 is "+ square(7));
        System.out.println("Square of 7.5 is "+ square(7.5));
    }

    public static int square ( int iValue ) {
        System.out.println("Called int with "+ iValue);
        return iValue * iValue;
    }

    public static double square ( double dValue ) {
        System.out.println("Called double with "+ dValue);
        return Math.pow(dValue, 2);
    }
}
```
public class MethodOverloadTwo {
    public static void main ( String args[] ) {
        int a = 7, b = -2, c = 88;
        int two = max ( a, b );
        int three = max ( a, b, c );
        System.out.printf ( "\nMax of two: %d\n", two );
        System.out.printf ( "\nMax of three: %d\n", three );
    }
}

public static int max ( int x, int y ) {
    int maxvalue;
    if ( x > y ) maxvalue = x;
    else maxvalue = y;
    return maxvalue;
}

public static int max ( int x, int y, int z ) {
    int maxvalue = x;
    if ( y > maxvalue ) maxvalue = y;
    if ( z > maxvalue ) maxvalue = z;
    return maxvalue;
}