More on Java Methods

Precondition & Postcondition
Overloading Methods
Scope and Visibility

Dan McElroy

Definitions

Precondition - a condition that must always be true just prior to the execution of some section of code. For example: the factorial is only defined for integers greater than or equal to zero.

Postcondition - a condition that must always be true just after the execution of some section of code.

Wikipedia

Overloading

Overloading occurs when two or more methods have the same name but the parameter list is different. Here are examples from the Math class

static double	Math.max(double a, double b)	Returns the greater of two double values.
static float	Math.max(float a, float b)	Returns the greater of two float values.
static int	Math.max(int a, int b)	Returns the greater of two int values.
static long	Math.max(long a, long b)	Returns the greater of two long values.

In each case, the name of the method is the same, **max**. However, the the parameter list has different data types. The compiler chooses the closest match when deciding which version of **max** to use. Example,

double Math.max(14.2, 7); // has a double 14.2 and integer 7

The 7 is promoted to a double so Math.max(double, double) is used.

More Examples of Overloading

```
public static void Employee (String name);
public static void Employee (String name, double salary);
public static void Employee (String name, double payRate);
```

Examples 2 and 3 are not legal in the same class because they both have the same 'signature' for their parameter list: String and double.

```
public static int avgScore (int s1, int s2, int s3);
public static double avgScore (int s1, int s2, int s3);
```

These two examples are not legal in the same class because the method name and parameter list are exactly the same. Only the name of the function and the parameter list make up the 'signature', not return type.

Scope

The scope of a variable identifies where it can be used.

```
1   package javascopedemo;
2
3   public class JavaScopeDemo {
4
5       private static final double RATE = 12.25;  // percent interest
6
7       public static void main(String[] args) {
8           double savingsAccount;
9
10          savingsAccount = computeInterest (5000.00, RATE, 5);
11          System.out.printf ("Ending balance is $%.2f\n\n", savings/  ount);
12      } // end of main
13
14      private static double computeInterest (double balance, doubl  ate, double years) {
15          double newBalance = balance;
16          for (int i=0; i<years; i++) {
17              newBalance += newBalance * rate/100.0;  // add interes  each year
18          }
19          return newBalance;
20      } // end of computeSavings
21
22  } // end of JavaScopeDemo class
```

RATE can be seen anywhere in the class

5

Scope

The scope of a variable identifies where it can be used.

```
1   package javascopedemo;
2
3   public class JavaScopeDemo {
4
5       private static final double RATE = 12.25;  // percent interest
6
7       public static void main(String[] args) {
8           double savingsAccount;
9
10          savingsAccount = computeInterest (5000.00, RATE, 5);
11          System.out.printf ("Ending balance is $%.2f\n\n", savings/  ount);
12      } // end of main
13
14      private static double computeInterest (double balance, double rate, double years) {
15          double newBalance = balance;
16          for (int i=0; i<years; i++) {
17              newBalance += newBalance * rate/100.0;  // add interest each year
18          }
19          return newBalance;
20      } // end of computeSavings
21
22  } // end of JavaScopeDemo class
```

savingsAccount is a local variable seen only within main( )

6

Scope

The scope of a variable identifies where it can be used.

```
1   package javascopedemo;
2
3   public class JavaScopeDemo {
4
5       private static final double RATE = 12.25;  // percent interest
6
7       public static void main(String[] args) {
8           double savingsAccount;
9
10          savingsAccount = computeInterest (5000.00, RATE, 5);
11          System.out.printf ("Ending balance is $%.2f\n\n", savingsAccount);
12      } // end of main
13
14      private static double computeInterest (double balance, double rate, double years) {
15          double newBalance = balance;
16          for (int i=0; i<years; i++) {
17              newBalance += newBalance * rate/100.0;  // add interest each year
18          }
19          return newBalance;
20      } // end of computeSavings
21
22  } // end of JavaScopeDemo class
```

balance, rate and **years** are local variables and seen only within computeInterest( )

7

Scope

The scope of a variable identifies where it can be used.

```
1   package javascopedemo;
2
3   public class JavaScopeDemo {
4
5       private static final double RATE = 12.25;  // percent interest
6
7       public static void main(String[] args) {
8           double savingsAccount;
9
10          savingsAccount = computeInterest (5000.00, RATE, 5);
11          System.out.printf ("Ending balance is $%.2f\n\n", savingsAccount);
12      } // end of main
13
14      private static double computeInterest (double balance, double rate
15          double newBalance = balance;
16          for (int i=0; i<years; i++) {
17              newBalance += newBalance * rate/100.0;  // add interes
18          }
19          return newBalance;
20      } // end of computeSavings
21
22  } // end of JavaScopeDemo class
```

newBalance is a local variable and seen only within computeInterest( )

8

Scope

The scope of a variable identifies where it can be used.

```
1   package javascopedemo;
2
3   public class JavaScopeDemo {
4
5       private static final double RATE = 12.25;  // percent interest
6
7       public static void main(String[] args) {
8           double savingsAccount;
9
10          savingsAccount = computeInterest (5000.00, RATE, 5);
11          System.out.printf ("Ending balance is $%.2f\n\n", savingsAccount);
12      } // end of main
13
14      private static double computeInterest (double balance, double rate, double years) {
15          double newBalance = balance;
16          for (int i=0; i<years; i++) {
17              newBalance += newBalance * rate/100.0;  // add interest each year
18          }
19          return newBalance;
20      } // end of computeSavings
21
22  } // end of JavaScopeDemo class
```

i can be seen only within the for loop

9

Access Modifier Rules

	Class	Nested class	Method, or Member variable	Interface	Interface method signature
public	visible from anywhere	same as its class	same as its class	**visible from anywhere**	**visible from anywhere**
protected	N/A	its class and its subclass	its class and its subclass, and from its package	N/A	N/A
package	**only from its package**	**only from its package**	**only from its package**	N/A	N/A
private	N/A	only from its class	only from its class	N/A	N/A

*The cases in **bold** are the default.*

en.wikibooks.org/wiki/Java_Programming/Scope

10