CS 106
Introduction to Computer Science I

10 / 23 / 2015

Instructor: Michael Eckmann
Today’s Topics

• Comments and/or Questions?
• Programmer defined methods
example method that returns nothing and has no parameters

```java
public static void printErrorMsg()
{
    System.out.println("Invalid entry.");
    System.out.println("You must reenter.");
}
```

• how to call this method: note that there are no arguments passed in nor is there a variable to which to assign the returned value (because it doesn’t return a value.)

printErrorMsg();
example method that returns nothing but has one parameter

```java
public static void printMsg(String theMsg)
{
    System.out.println("The message is: " + theMsg);
    // note: no return statement since nothing to return
    // The return type of this method is void.
}

• how to call this method:
    printMsg("Good afternoon gentlemen. I am a HAL 9000 computer.");
```
example method that returns a String but has no parameters

```java
public static String randomString()
{
    int num = (int) (Math.random() * 3);
    if (num == 0)
        return "Hello";
    if (num == 1)
        return "Goodbye";
    // if (num == 2)
    return "$#@&!*";
}
```

• how to call this method:

```java
System.out.println("the random string is: " + randomString());
```
reasons to create and use methods

- separation and modularization of ideas is helpful when writing large programs because smaller parts of a problem are easier to solve
- the individual methods can be tested and confirmed correct to reduce debugging woes
- methods lend themselves to software reuse
- less code repetition
- more readable, better designed code
scope

- the scope of an identifier (a variable, reference or method name) is defined as the portion of the program that can access that identifier
- the two scopes for identifiers are class scope and block scope
- class scope starts at the opening left curly brace of a class definition and ends at its corresponding right curly brace
- block scope refers to identifiers that are accessible from when they are declared until the next right curly brace.
we've seen block scope when we declare a variable inside the main method.
we've seen block scope when we declare a variable in the first expression of a for loop.
we've seen block scope when we declare a variable within the curly braces of a while loop.
etc.
What is the effect of block scope?
variables in methods

• variables declared in methods are said to be local variables --- they are local to that method
• parameters of the method are also considered local variables to that method.
• they are not usable, nor are they even known outside that method --- they have block scope
• they are created (memory is allocated for them) on entry into the method during execution
• they are destroyed (memory for them is marked for deallocation) on exit from the method
some method terminology

• the name of the method is `salaryAfterRaise`, and the body of the method is the three lines of code between the curly braces.

```java
public static double salaryAfterRaise(double currSal, double raisePct)
{
    double newSal;
    newSal = currSal * (1 + raisePct / 100);
    return newSal;
}
```
example of local variables

• in this method, `newSal` is a local variable, and the parameters of the method: `currSal` and `raisePct` are also local variables.

```java
public static double salaryAfterRaise(double currSal, double raisePct )
{
    double newSal;
    newSal = currSal * ( 1 + raisePct / 100 );
    return newSal;
}
```

• they are only known to this method
• what does that mean “only known to this method”
return type and argument types

• the *return type* of the method is double
• the two *parameters* are of type double

```java
public static double salaryAfterRaise(double currSal, double raisePct )
{
    double newSal;
    newSal = currSal * ( 1 + raisePct / 100 );
    return newSal;
}
```

• note that the return statement returns the double and this is where the execution of the method ends
method calls

• in the call of the method, two values are passed in to the method (we call them the arguments) and the value returned by the method call is stored in newSalary

double newSalary;
double oldSalary = 300;
double raise = 4.5;

// our call to the salaryAfterRaise method
newSalary = salaryAfterRaise( oldSalary, raise );
variables passed in as arguments when calling a method

• the values of variables (of primitive types) that are passed into a method as arguments remain unchanged after the method ends its execution

• even if the corresponding parameter in the method changes its value, this change is only local to the method, and the value is not “passed back out” of the method.

• we’ll see this in the factorial method on the next slide.
public static int computeFactorial(int num)
{
    int tempFactorial = 1;
    while (num > 0)
    {
        tempFactorial = tempFactorial * num;
        num--;
    }
    return tempFactorial;
}
computeFactorial method

• this method is a programmer-defined method (that is, we make up the name and the purpose of the method.)
• the name of the method is: computeFactorial
• the return type of the method is: int
• there is one parameter to the method, which is: num (of type int)
• note that the value of num inside the method changes (it gets decremented in the loop) but this is only a local change
• the other local variable (tempFactorial) is used to compute the factorial and its value is the one that is returned to the caller.
computeFactorial method

• note that the value of num inside the method changes (it gets decremented in the loop) but this is only a *local* change the variable sent in as an argument does NOT change its value

• Let’s write a quick program to show how the variable that is “passed in” to the method as an argument doesn’t actually get changed.

• We’ll print the argument’s value before and after the call to the method.