Today’s Topics

• Comments and/or Questions?
• Programmer defined methods
what is a method

• A method is small piece of a program designed to achieve some specific task and usually returns some piece of information.
• A method is invoked by a method call.
• To call a method, you provide its name and the correct arguments that are necessary for the method to execute.
Here’s a good analogy of a worker and boss to describe methods and their callers.

A boss (the caller) asks a worker (the method that is called) to perform a task (the code in the method) and report (return results) back when the task is done.
methods

• All methods are defined within some class.
• we have seen the **main** method defined in the class of every application so far.
other methods we’ve used

• We have used other methods available in the Java API, methods like `parseInt` and `parseFloat` and `println` are available to us to use in their respective classes `Integer`, `Float` and `System.out`

• The methods in the Math class are other examples. These methods were defined for us to do a specific task. We call them when we need them.

• e.g. in the case of `Integer.parseInt` --- it’s task is to convert a String into an int. It takes a String as an argument (parameter) and returns a value of type int.
other methods we’ve used

• e.g. in the case of Integer.parseInt --- it’s task is to convert a String into an int. It takes a String as an argument (parameter) and returns a value of type int.

• example of a call to this method:

        someInt = Integer.parseInt( someStr );

• In this example, someStr is the argument that is being passed in to the parseInt method and someInt is the variable that will get set to the value returned by the method.
methods

• We can create our own methods that we can call to perform specific tasks.

• Let’s say we’re writing a program to handle employee’s salaries. We might need to compute a salary after a raise.

• This example method would need to have access to the current salary and the raise percentage.

• It would then calculate the salary after the raise and return this value.
example of a programmer-defined method

• we might name this method `salaryAfterRaise`
• we need to take in two values, one for the current salary and one for the raise percentage.
• What primitive type might these be?
• we also need to return the salary that is computed.
• What primitive type might this returned value be?
example of a programmer-defined method

• so, this method could look like:

```java
public static double salaryAfterRaise(double currSal, double raisePct )
{
    double newSal;

    newSal = currSal * ( 1 + raisePct / 100 );

    return newSal;
}
```
example of a programmer-defined method

• to call this method from some other method within the same class do the following:

```java
// example variable declarations...
double newSalary;
double oldSal = 35000;
double raise = 4.5;

// our call to the salaryAfterRaise method
newSalary = salaryAfterRaise( oldSal, raise );
```
example of a programmer-defined method

• Let’s look at a complete program that contains this method and calls it several times.
example of a programmer-defined method

• When certain code can be used in several places in a program, you may want to create a method containing that code. Then, wherever that code would have been, a simple method call appears.

• Programmer defined methods also aid in the ability to understand a program (readability) and make changes (maintainability).
when does a method end its execution?

- when it hits a `return;` statement
- when it hits a `return someExpression;` statement
- or, when it hits the right curly brace of the method
- whichever comes first.

- when the method ends its execution, the program execution continues back at the method call (from where the method was called)
more about methods

- methods that do not return a value are of type `void` (like the main method.)
- Methods can have 0 or more parameters separated by commas. Those with 0 parameters are defined with nothing between the parentheses
- to call a method with no parameters, you must still use the parentheses but with nothing between them
- Note that the return type of a method can be different than any or all of the types of parameters that get passed in.
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:
  ```java
  printMsg(“Good afternoon gentlemen. I am a HAL 9000 computer.”);
  ```
example method that returns a String but has no parameters

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:
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
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.

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