Methods
(may also be known as procedures or functions)

Introduction

One of the keys to successfully writing programs is to break the task or application down into smaller pieces, and a primary way to do that in many languages is to write a smaller piece of code as a procedure or function. (In Java, the other main way to subdivide the task is also to model the data as classes with methods.) Sometimes this process is called modularization.

A procedure is a separate piece of the program that is named and can be executed by calling it from other parts of the program. An advantage of writing a procedure is that code to do a particular part of the task can be written in one place and used in many other places, so that the programming task is streamlined.

In Java, procedures and functions are called **methods.**

Java Methods

A **method definition** has the form:

```
<scope-keywords> <return-type> methodName ( <parameter1>, <parameter2>, …)
        {
                <some declarations and statements>
        }
```

The first line of the method definition is the method header.

- The scope keywords define where the method can be used. We'll start with the keywords "public static" or "public" for methods that we use in the current program.
- The return type is the type, such as double, int, String, or boolean, of the value that the method returns. If no value is returned, the type is the keyword "void".
- The methodName follows the same rules as other variable names in Java. Traditionally, method names also start with a lower case letter.
- The parameters are declarations of the formal parameter variables used for input to the method. Even if there are no parameters, the parentheses must be included.
-

Inside the { and } are the declarations and statements that make up the body of the method. These statements are executed whenever the method is called.

- A return statement is also allowed inside the body of the method.

Example:

```
// This method receives an integer parameter, multiplies by 2, and prints the result
public static void twice ( int y) {
    int ydoubled;
    ydoubled = y * 2;
    System.out.println ("The number " + y + " doubled is " + ydoubled);
}
```

The variable "y" is the formal parameter of this method. The variable "ydoubled" is a **local variable**, which means that it can only be used within this method and not in any other part of the program. Since no value is returned, the return type is "void".

Example:
Although Java already has a built-in method to find the maximum of two numbers, we write a simple maximum method here. It takes two formal parameters of type double and returns a result of type double. The "return" statement is used to return the result.

```
public double max ( double y, double z) {
    // local variable for the result
    double maxvalue;

    if (y > z) {
        maxvalue = y;
    }
    else {
        maxvalue = z;
    }
    return maxvalue;
}
```

This method compares the value of the two formal parameters "y" and "z" and returns the value of the greater value of the two.

Example:
If the method takes no parameters, then the method definition has empty parentheses.

```
public static void printHello ( ) {
    System.out.println ("Hello!");
}
```

A **method call** has the form:

```
methodName( paramvalue1, paramvalue1, …).
```

The values paramvalue1, paramvalue2, etc. are called the **actual parameters**.

- The number and types of the actual parameters must match the number and types of the formal parameters in the method definition.

In the case that the return type of the method is “void”, then the method call can be used as a statement:

```
twice ( 3 );
```

In the case that the return type of the method is any type, then the result of the method call should be used in an expression or assigned to a variable of that type.

```
double amount;
amount = max ( 4.5, 4.6 );
```

Note: When you are looking at a program, how can you tell a method name from other variable names? The method name will always have parentheses after it. Even if there are no parameters, the method name will be followed by empty parentheses. (See printHello ( ) below .)

As the program executes the method call, it jumps to the method code and copies the values of the actual parameters in the method call to the formal parameter variables in the method definition. The method code is executed and then the flow of control jumps back to where it was called, and any return value can be used in the calling program.

The same method can be called any number of times with different values for the parameters.

If the method has no parameters, then empty parentheses are used in the method call.

```
printHello ( );
```

The actual parameters can be any constant, variable or expression of the right type.

```
int number = 2;

twice ( 3 );
twice ( number );
twice ( (number * 7) + 10 );
```

In any event, the actual parameter expression is evaluated and that value is passed to the method.