Lab 5
Methods and Parameters

Overview

At this point in the course, you should have a set of skills which allow you to create functionality at the level of using control structures like “if” statements and “while” loops. Determining how to nest these structures (if within if, loop within loop, if within loop, etc.) requires a lot of practice. In addition, as the size of our programs grow, the program complexity grows at an alarming rate and can quickly get out of our control.

Your experience is starting to tell you that creating programs that are extremely long will get so complicated that it will be nearly impossible to manage the complexity. There are a number of tools that we will begin to investigate in order to develop skills used to keep the complexity under control and to reuse our efforts in other programs.

Methods will allow us to package our efforts for easier reuse and parameters will further enhance the method to broaden utility of the code.

It’s also appropriate at this time to begin considering what abstraction means and how to use abstraction in the design processes. One can compare abstraction to generalization. If you consider 4-wheeled vehicles, there are a number of ways in which the vehicles are similar and ways in which they are different. You might first think of cars, and although cars are 4-wheeled, ATVs can be made as 4-wheeled vehicles and some of you would view trucks as being different from cars but still 4-wheeled vehicles. When we try to create a 4-wheeled vehicle as an abstraction, it quickly becomes clear that certain features vary within the general category.

As we consider how they are like rather than different, it also becomes clear that we interface (drive and ride in this case) with these devices in ways where we expect the interface to be similar even though the specific vehicles are different. We look for steering, ignition, seats, mirrors, gauges, etc. so that the process of driving a different type of vehicle allows us to reuse our experiences.
So it is with programming and developing code. As we create solutions to problems, a good programmer is always thinking of how to take this solution and generalize it for additional use. Whether it is because we are lazy and want to avoid the work of recreating the solution, or we see this design process almost as an art, the result is the same. It greatly increases the usefulness of the code and the satisfaction derived from creating it.

Let’s start with some of our previous work and examine how to package the code as a method and then reuse it in another program. As you go through this process, don’t lose sight of what you are trying to accomplish, design and abstraction, beginning with the concept of interfaces.

Also keep in mind that you are wearing many hats:

- the designer,
- the programmer,
- the user,

and the interface depends on the hat you are wearing. Here we are not talking about the interface of your program with the user. Instead we are talking about how another programmer will interface with your code; what another programmer will need to see/know in order to use your code in their program.

Consider a quote from Albert Einstein: “Everything should be made as simple as possible, but not one bit simpler.” As you design the interface try to consider making your code as easy as possible for your user (the other programmer) to use, but not to eliminate important functionality. Back to our example of the 4-wheeled vehicle, it is not important to us whether the motor uses gas or batteries, but the interface we use must have an ignition to turn it on/off, a pedal for applying power, a brake for stopping or slowing down and a mechanism for steering.

Learning to think this way will help you to more readily incorporate the efforts of other programmers’ work into your solution by integrating their libraries into your program. In the long run, the complexity of learning to design with and use their methods will allow you to simplify your programming activities and to create a more reliable result.
In a previous lab or lecture, you probably have seen something like the factorial calculation. Your code probably looked something like this:

```java
int fact = 1;
for (i=1; i<=n; i++)
{  fact = fact * i; }
```

We would agree that this takes a little time and thought to digest, but that every step of this calculation is necessary in order to solve the problem. The question here is how to package this for future use so that I can avoid this complexity in the future. Once the algorithm has been developed, I don’t want to have to do this again. Better yet, if someone else has solved the problem, I’ll use that result in solving my problem.

Consider a black box named “factorial” and consider what would go in and the result produced. Consideration of “what goes in and what comes out” defines the interface.

![Diagram of black box](image)

The previous code now only needs to be re-packaged/re-interfaced to look as follows:

```java
public static int factorial (int n)
{  int fact = 1; int i;
    for (i=1; i<=n; i++)
    {   fact = fact * i; }
    return fact;
}  
```
The important observation to make in this example is not the code on the inside of the method, but the definition of the interface. To extend our analogy of driving a vehicle, the interface given to the driver does not include everything required for the vehicle to operate. As a driver of your own car, you can agree that there are lots of operational aspects of your car that you don’t understand. So where is the “interface” in this problem?...

**public static int factorial (int n)**

*The first line of the method.* It shows what is coming in and what is going out.

In the above example, “n” is going in and the result calculated by the method (stored in “fact”) is returned. Consider the example calculating \(4! = 24\).

The value in the parentheses (int n) *represents* the input to the method and the type at the beginning of the line (int) is an indication of the type of the result that will be returned. What it says here is that you must pass an integer into the method and factorial will return an integer result back.

The following code shows the method defined in a class called MathMethods.

```java
public class MathMethods{
    public static int factorial (int n)
    {
        int fact = 1; int i;
        for (i=1; i<=n; i++){
            fact = fact * i;
        }
        return fact;
    }
    public static void main(String args[])
    {
        System.out.print("The factorial of 12 is ");
        System.out.print(MathMethods.factorial(12));
    }
}
```

[Diagram showing the method definition and execution flow]
} } 

Note how the method is called from within the main method using the statement:

```java
System.out.print(MathMethods.factorial(12));
```

Hopefully it is clear how much simpler this statement is compared to actually including the loop in the main routine. That’s the point. Soon, a large amount of the code you write will be utilizing the methods as `factorial` is used in the main routine of this example, not writing the methods itself. Obviously there will be instances when you write, use or write and use the methods. Using only the interface instead of the details of the process reduces complexity. In addition it allows us to reuse code that we have already written, saving time to produce the working program and allowing us to use code that has already been thoroughly tested. Once we know how to create methods, the resulting code that we create will also be simpler and easier to manage.

Also note that the main method does not prompt the user for input; instead the main routine is being used as a testing “driver” to test the method. In this example the main routine is not developing a classical application that will use the methods, but to simply provide a means of testing the methods to be sure they are functioning properly. Your main routine for this lab should do the same. Do not prompt the user for the input data. As illustrated in the example above, simply call the method with the test value as a parameter (in factorial above 12 was passed to the method for the test value).

---

**Programming Activities**

In the previous labs, you wrote code to implement the following:

- `absoluteValue`
- `minOf3Numbers`
- `leapYear` *(be careful about the type of result this should return)*
- `factorial`
- `GCD`
In your Lab05 folder, create a new java class called MathMethods. Convert each of these 5 to a method within the MathMethods class (you have the solution for factorial already). Also include a main method, though you should leave the body of this method blank for now. Do NOT try to actually execute this code at this point. The next part of the lab will investigate how to organize your program to call/use these methods.

Show your instructor when you have written the methods to be sure you are on the right track.
In the process of creating these labs, a philosophy has been adopted to present the features of java in an order that results in the clearest and best ordered sequence to optimize your learning of the material. One of the areas which is cumbersome in the early stages of learning java is how to actually incorporate methods into your program.

As you just experienced, the concept of a method is not that complicated, but the implementation of methods in your program introduces terms that are really beyond that which is appropriate at this stage.

In order to make this as simple as possible, the following convention for program organization has been adopted. Understand that a convention is just that: something we have agreed to do, not something that has to be. In fact, many of the programs that you will read in the text do not use this convention, but you should still be able to read and understand the text examples at a reasonable level once you understand the fundamentals principles of methods as explained in the lab.
The program organization suggested is as follows:

1. Create a class which is named the same as your file.

   ```java
   public class ClassName {
       public static void main(String args[]) {
           ClassName.method1();
           ClassName.method2();
           ...
       }
   }
   
   public static void method1() {
       //declare variables here if desired
       //code here
   }
   
   public static void main(String args[]) {
       // declare variables here if desired.
       ClassName.method1();
       ClassName.method1();
       ClassName.method2();
       ...
   }
   }
   
2. You can declare variables either inside methods or inside the main.

3. Call the methods by using the “ClassName” followed by a “.” followed by the “method name and parameters”.

So let’s start with a simple “hello world” program that shows how this goes together:

```java
public class HelloWorld {
    public static void sayHello()
    {System.out.println("Hello World");
    }
    public static void main(String args[])
    {
        HelloWorld.sayHello();
    }
}
```

Now modify the MathMethods class to test all 5 methods you created in the first part of the lab, invoking each method with ALL of the test cases required in the original labs. The factorial and leapyear examples do not provide test cases, so use the following:

- factorial of 5 and 6

In this class, Do NOT prompt the user for input; simply invoke the methods and print out the result as indicated in this example below for factorial:

```
System.out.println("The factorial of 5 is “ + MathMethods.factorial(5));
System.out.println("The factorial of 6 is “ + MathMethods.factorial(6));
```

Submit the same MathMethods file to WebCAT to grade each method separately. Name the methods as indicated with parameters.

- absoluteValue(double)
- minOf3Numbers(double, double, double)
- leapYear(int) *(Check the type of result this should return with instructor)*
- factorial(int)
• GCD(int,int) *(Instead of the error message, return -1 if the input has a negative value)*

Now let’s see how to use the factorial method to create a simple application and to use the program organization described in the previous step.

**Combinations**

One of the calculations used frequently in mathematics is to determine the number of ways of selecting subgroups of items from a larger group. Consider a deck of cards. How many different ways are there to choose 5 cards from a deck of 52 cards? If you choose “r” items from a group of “n” possible values, the number of different possibilities is shown on

http://www.mathwords.com/c/combination_formula.htm

Add one more method to the MathMethods class to incorporate the method

`combinations(int n, int r);`

where n (the number of items in the group) is first and r (the number of items being chosen) is second.

You will use your factorial method to create a “combinations” method. Combinations uses factorial three times to calculate the result so implement the combinations method with 3 calls to factorial.

Test the program by using combinations (6,3) to see how many ways there are to choose 3 items from a group of six. Do all of your work with integers.

Submit the MathMethods class again to WebCAT with the new combinations method.

**JUST A CHALLENGE... NOT FOR GRADING!**

Last try to use the “combinations” method to calculate the number of 5-card hands possible from a deck of 52 cards. You will get a surprise! See if you can write a combinations2 method that solves the problem. If you can, show your instructor.