**CSC 101 – Intro to Computer Programming**

**LAB #7**

**Creation of Static Methods**

**NAME:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Objectives**

1. Add additional methods to class (other than main).
2. Call these methods from main method.

3) Examine parameter passing to and from methods.

4) Understanding why these new methods are static methods.

**Getting Started**

The simple programs that we have created so far have had only one method inside one class.

class DoSomething {  
 public static void main( String [ ] args) {

// interesting code here

}

}

The **main** method is required to be in our class in order to define a starting point for code processing. We are not creating our own classes that will be used in other programs at this time, and instead creating simple tasks of your instructor’s whim.

Since this **main** method has nothing particularly to do with the class itself, other than be created inside it, it is a standalone method and needs the label **static**. Our class is not storing any data and is not expected to have any objects created out of it. We will do this at a later time.

There are not any restrictions with creating additional methods and storing them in the class. So we are going to explore that here.

What are methods and why do we want to create them?

A method (or function in other languages) is a standalone block of code that is used to do one calculation or perform one task. Such as computing the square root of a number, reading in data to process, or broadcasting a menu to the screen. Such methods, once written, can be seen as very useful to use in more than one program. Or even can be used multiple times in the same program. If we write them correctly the first time, we can store them and enter them in any future program that does the same thing. If instead we simply embedded them into our main function, reusing the code would mean pulling our lines of code from inside working programs and then time consuming effort to get them to work within our new usage.

Here is an example:

class DoCalculation {  
 public static void main(String[] args) {  
   
 double temp\_florida = 75;  
 double temp\_alaska = 35;  
   
 System.out.println("In Celsius the temperatures are:");  
 System.out.println("Florida="+ convertFtoC(temp\_florida));  
 System.out.println("Alaska=" + convertFtoC(temp\_alaska));   
   
 }   
}

As you can see the above code needed to do a conversion between temperature units twice. How awkward and wasteful of space would it be to rewrite the same equation twice, or potentially more times later? Instead we replaced the calculation with a simple function call and sent in the different values to evaluate. This is similar to function calls that we previously used when doing stuff from the Math library. Below is the function written out.

public static double convertFtoC(double temp) {  
 double answer;  
 answer = (temp - 32.0)\*(5.0/9.0);  
 return answer;  
 }

This method would be placed into the class called DoCalculation in order for it to work. Most likely right after the main. If the method was to be used inside another class, it would be called with the class name it was stored in included. (see below) *DoCalculation*.*convertFtoC*(value)

General format for a method is as follows:

<modifiers (if any)> <return type> <function name> <parameter list in parathesis>

* modifiers are java keywords such as public, private, or static
  + public means can be seen outside of class
  + private means only other methods in class can use it
  + static means not linked to any objects
* return type is the data type of what is being returned from the function – this can be **void** if nothing is being returned
* parameter list is all the values that will be sent into the function to do its job.

**Parameter Passing**

Parameters are passed by value. This is the most important statement to understand. The line convertFtoC(temp\_florida) is passing the current value of the temp\_florida variable into the function. Inside the function this value is received and stored into the “local” variable temp. It doesn’t matter what we do to the variable in the function, the original one will not change. The term “local” variable is used to mean that this variable only exists inside this code block and as soon as we return out of the function it no longer exists. Returned variables are also passed back by their value. And are placed at the location in the code where the method was originally called.

Variables inside methods can have any names, including names already used in other parts of the program. Since these are local to the blocks that they were created in, they are treated as two separate variables in the program.

See this example:

class DoLittle {  
 public static void main(String[] args) {  
 int cat = 12;  
 int dog = 17;  
   
 System.out.println(what(dog));  
 System.out.println(dog);  
 }   
 public static int what(int cat) {  
 int dog;  
 dog = cat + 8;  
 return dog;  
 }  
}

Can you tell me what the output would be?

Don’t get lost in the usage of repeating dogs and cats, the answer is:

25

17

Remember the function call in the first output statement sends in the value of dog at that point in the program (17) into the method *what*. The value of 17 is then placed in the local variable of cat and then the local variable dog in the method is assigned 17 + 8. So 25 is then returned to the called location which is printed out. The final output is simple reporting the value of the variable dog. Inside the main that has not changed. The dog variable in the method existed only in the method and is now gone.

Now you understand the principle of passing by value.

Things get a bit more complicated if we are passing an object as a parameter. Since an object is referenced to by a memory location, if sent in as a parameter, all we are sending is the address in memory. If we then make changes to the object itself, we are actually updating what is in memory and in fact are changing the original object.

Confused. Don’t worry, we will have time to figure this out later.

**Lab Tasks – complete these and show instructor before leaving**

Use the information provided here and from class discussion to do the following exercises before you leave.

1. Create a small program that asks the user for two integers and then outputs the larger absolute value of the two numbers. Your program should have a static method that is sent two integers as parameters and returns an integer that has the larger magnitude. The method should do only this, the rest is done in the main. If you are not creating a method, you will not get any credit for this problem. Inside the main you should also have a loop that allows you to repeat the query until 0 0 are given as the inputs.
2. You are creating a new way to do math that uses estimates instead of actual values. You are going to create a class called **EstimateMath**, and inside this class will be only two static methods **estimateAdd** and **estimateSubtract**. Both methods will take in two integers as parameters. Each number will be rounded to the nearest 10 and then the arithmetic will be done. Such as this method call:

**EstimateMath.estimateAdd(14,21)** would return 30. Write a main method in a *separate* class file that tests each of these methods by asking the user to input 2 integers and gives the results. Repeat this in a loop until zeros are input.

One more on the next page……

1. Analyze the following code. Determine what the output would be if this program is run. Make your prediction and then type it in and run it. Do not cheat – predict first and then try it. Problems like this will be on your next exam.

/\*  
 Use of a static function to test if you can follow  
 the passing of paramaters  
\*/  
class ConfuseStudents {  
 public static void main(String args[] ) {  
 int cherry = 12;  
 int orange = 9;  
 int grape = 5;  
 int banana = 8;

// static method can be called without class name first  
 // if method is defined in same class   
 // i.e. ConfuseStudents.swapit(grape,orange); is shortened to

swapit(grape,orange);

System.out.println(swapit(orange,cherry));  
 System.out.println(swapit(cherry,grape));  
 System.out.println(banana);  
   
 }// end main method  
   
 public static int swapit(int apple, int cherry) {  
   
 int banana = apple + cherry;  
 cherry = cherry + 2;  
 System.out.println(cherry);  
   
 return banana;  
 } // end swapit method  
   
} // end class

Make sure to show these working programs to your instructors before you leave lab to get full credit. If you can’t not finish all problems assigned during lab time and you were present and working the full time, you will have a chance to show me what you completed at the beginning of the next lab. You cannot just leave lab early and get full credit later.

public static int swapit(int x, int y) {  
   
 int banana = x + y;  
 y = y + 2;  
 System.out.println(y);  
   
 return banana;  
 } // end swapit method

Result:

swapit(valueone, valuetwo)

(valuetwo + 2)

(valueone + valuetwo)

int cherry = 12;  
 int orange = 9;  
 int grape = 5;  
 int banana = 8;

System.out.println(swapit(orange,cherry));  
 System.out.println(swapit(cherry,grape));  
 System.out.println(banana);

Final results:

11

14

21

7

17

8