CSc 110 Assignment 3
Static Methods with: Parameter Passing & Return Values

The Carbon Calculator

Due:

Friday October 3, 8pm.

Learning Outcomes:

When you have completed this assignment, you should understand:

- That passing parameters is equivalent to assignment.
- How to design and test static methods according to a given specification.
- How to use a `Scanner` to implement a program that collects input from the console (aka keyboard) using prompts.
- How to identify repetitive portions of code and replace them with a parameterized method that improves clarity and reduces the total size of your code.
- Limitations of the int and double types.
- How to build-up a complex program from simple methods (incremental development).

Problem Description:

This assignment requires you to write a program named **CarbonCalc.java** to calculate a user's carbon footprint over 1 year. It will prompt the user for a series of keyboard inputs, and use the entered values to calculate C02 emissions for transportation, electricity, and food. It will then print out a report summarizing the carbon footprint in metric tons per year.

Carbon Calculations:

We will use the following formulae to estimate the carbon footprint:

1. Transportation:
Regular gasoline in North America produces 2.3 kg CO2 per litre.

```
kgCO2 = 2.3 x litresUsedPerYear
litresUsedPerYear = 365 x (kmPerDay ÷ fuelEfficiency)
```

where fuelEfficiency of the car is in km/litre.

2. Electricity
The average CO2 emission for electricity produced in Canada is 0.257 kg/kWh

```
kgCO2 = (kWhPerMonth * 12 * 0.257) ÷ numPeopleInHome
```

Note: kilowatt hours per month can be determined from your electric bill.

3. Food
A rough estimate can be calculated based on the percentages of meat/fish, dairy, fruit/veggies, and carbohydrates in the user's diet:

```
Yearly kgC02 for meat = (% meat and fish eaten) x 53.1
Yearly kgCO2 for dairy = (% dairy eaten) x 13.8
Yearly kgC02 for fruit&veg = (% fruit and veg eaten) x 7.6
Yearly kgCO2 for Carbs = (% carbs eaten) x 3.1

Total yearly kgC02 for food is the sum of the above 4 values.
```

4. Total Carbon Footprint = (sum of footprints for transportation, electricity, and food) ÷ 1000 (This total is given in metric tons per year).

Program Requirements:

Create a program named CarbonCalc. It will calculate the carbon footprint using the formulae above.

Your program must include and use methods with the following signatures:

```
public static double determineTransportationEmission(Scanner input)
public static double determineElectricityEmission(Scanner input)
public static double determineFoodEmission(Scanner input)
public static double calculateTotalEmission(double trans, double elec, double food)
public static void printReport(<you decide what to pass in as parameters>)
```

You will need to create a Scanner in main and pass it to the first 3 methods to prompt for the user's values. The user will input any values that are not constants in the equations above. Be sure to prompt for the values you need, with enough detail in the prompt that the user knows what to enter.

For consistency, **you must ask the user for the values in the following order(s)**:
determineTransportationEmission:

1. The number of kilometres / day
2. The fuel efficiency

determineElectricityEmission:

1. The monthly kilowatt usage.
2. The number of people in the home.

determineFoodEmission:

1. The percentage of meat & fish.
2. The percentage of dairy.
3. The percentage of fruits & vegetables.
4. The percentage of carbohydrates.

To test your program for correctness, first compile your program, CarbonCalc. Then compile and run the given **Tester.java** file in the same directory as your program.

Your program must print a report matching the format of the sample output below. (Note that actual values will differ depending on what the user inputs.)

Sample output of the program:

```
You produce an annual total of 7.14432 metric tons of CO2 per year.
The breakdown is as follows:
       Car          32.90166%
       Electricity  0.43167156%
       Food         66.66667%
```

Tip: Use iterative development:

Start by creating a skeleton program that includes the methods above but has them do nothing. When necessary, return a dummy value like 1.0. Then work on one method at a time, testing to ensure it does what you expect before moving on to the next method. (Note: you can comment out lines in the main() method of Tester in order to test one method at a time.)

HAND IN: Submit CarbonCalc.java using the 'Assignments' link of the course connex site.

Bonus

For up to 10% bonus points, create a second program, ImprovedCalc.java. The carbon footprint estimate above is very rough. Make some improvement to the quality of the estimate and improve the report accordingly. For example, you could take additional input values and do something with them to produce a more accurate calculation. For additional details on calculating a carbon footprint, see: http://web.ncf.ca/bf250/carbonbudget.html.

You must receive at least 9/10 on the regular assignment to be considered for the bonus.

Hand in ImprovedCalc.java plus any other files used. (Be sure to also hand in the basic program CarbonCalc.java from the main assignment above.)

Grading – The marker will look for:

- Documentation: Documentation in the implemented code must include the author's identification and the purpose of the program. Each group of instructions must be preceded with a comment describing their common purpose. Each method must be preceded with a comment describing its purpose as well as an indication of the input to and the output from the method.
- *White Space* and *Indentation* in the code and adherence to Java naming conventions
- Your methods should accept input parameter values and return an output value ***exactly*** as described above. Your program should use the methods wherever it is appropriate. Other suitable methods can be created, of course, or no other methods, if that is appropriate.
- Compiles and produces correct output: The code constitutes a valid Java program (i.e. compiles *and* runs without error on the ECS 250 lab machines). It must accept input and print output EXACTLY as described above.
- Appropriate prompts are given to the user, with appropriate details such as units for the values that must be entered.

More to Discover

The process of creating and testing a small part of the entire program, then adding on another small part and then another is called incremental development. It allows the programmer to simplify a large problem working through only a few issues at a time: http://www.upedu.org/references/bestprac/im_bp1.htm or http://en.wikipedia.org/wiki/Iterative_and_incremental_development.