CSc 110 Assignment 2
Static Methods, Parameter Passing & Return Values

Due:

- Assignment 2 Part 1: Prior to your scheduled lab class during the week of October 4-8, 2010
- Assignment 2 Part 2: Prior to your scheduled lab class during the week of October 11-15, 2010

  (Monday labs: Hand in prior to 10 am on Tuesday, October 12)

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 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.
- How to build-up a complex program from simple methods.
- Limitations of the int and double types.

Overview

Assignment 2 introduces the concept of passing parameters to and returning values from methods. It also requires that methods be used over and over for different purposes.

Assignment 2 Part 1: Backwards Integers:

This program requires that a collection of integers be output, both frontwards and backwards. The backwards output prints out the last digit first, followed by the second last digit, followed by the third last digit, and so on, until the first digit it output. Using, for example, the integer 4289, the program would need to be able output:

```
The integer: 4289
Backwards: 9824
```

1. Begin will at least 3 example calculations: write down (on paper) 3 multiple digit numbers and then write them down backwards.
2. Examine your examples (in step 1 above) and consider how integer division and modulus (using 10 as the divisor) could be used to automate the process. In particular, try to write steps that can be done repeatedly, even if you do not know the original number.

3. Write a static method that performs the task of determining and printing the digits of the original number backwards. The static method should accept one integer parameter, the number. Although the static method does not return a value, it produces printed output, the number backwards.
4. Call the static method 5 times from your main method, each time passing a different parameter to the method. In particular, use with the numbers: 4289, 0, 13, 452876 and 1144112.

Sample output of this program:

```
The integer: 4289
Backwards: 9824

The integer: 1
Backwards: 1

The integer: 13
Backwards: 31

The integer: 452876
Backwards: 678254

The integer: 1144112
Backwards: 2114411
```

5. (Optional) Use the `Scanner` class to read data in from the keyboard, allow the program to input integers from the keyboard, pass those integers to the method and have it print them backwards.

PART 1 HAND IN: Submit your code for step 4 or 5 (above) using the 'Assignments' link of the course web page. Your code must be submitted before the beginning of your scheduled lab class in the week of October 4-8, 2010.

Assignment 2 Part 2: Vancouver Island Ferry System:

The *Vancouver Island Ferry System* is a community based cooperative that runs two small vessels between various points on Vancouver Island. The first vessel runs between Odgen Point in Victoria and Parsons Point in Sooke and costs $485.00 per hour to operate. The second vessel runs between Odgen Point in Victoria and Roberts Point in Sydney and costs $595.00 per hour to operate. Your task will be to write a program that calculates the total daily operational hours and the gross daily operational cost for the cooperative. The input to your

program will be numbers of runs and durations extracted from the daily ferry schedule. Although the ports are the same every day, the number of runs and their expected duration vary due to the Pacific Westerlies (the wind!). Your contact in the cooperative states that each run requires an additional half hour of operational time for loading and unloading.

Notes:

a) Times and trip durations should be input as 2 integer values, one representing the hours (on a 24 hour clock) and one representing the minutes. You can assume that all times are on the same day.
b) Your program should make use of static methods wherever possible and should not contain any redundant code.
c) One method that must be used has the following method signature:

```
public static double
convertHoursMinutesToDouble(int hours, int
minutes)
```

This method will expect that the input parameter `hours` contains an integer value that represents a number of hours and the input parameter `minutes` contains an integer value that represents a number of minutes. The method will convert these two integers into one real value that represents the time in hours. Note that the fractional part of this result will no longer represent minutes: it will be a fraction of an hour. This real value will be returned by the method.

Use this method at every possible opportunity. In particular, it must be used for all departure times and trip durations.

Your program could contain the following statements (or similar) to input time values:

```
System.out.print("Input hour (24 hour clock)==> ");
int hour = stdin.nextInt();
System.out.print("Input minutes ==>");
int minute = stdin.nextInt();
double time = convertHoursMinutesToDouble(hours, minutes);
```

d) One method that must be used has the following method signature:

```
public static double
RouteTotalOperationalTime(Scanner input)
```

This method uses the Scanner class to read data in from the keyboard. That data includes the number of runs for the route, and the duration of each

run. The method needs to calculate the sum of all the run durations and include that extra half hour per run. The method returns this total sum.

e) Your program should be able to handle any schedule that is similar to the example, that is, one which runs ferries between the same departure and arrival points, but could have a varying number of runs and varying trip durations.

1. First, choose a sample daily schedule (one is provided on the course web site) and hand calculate what the output should be. If you are not sure if your calculation is correct, do not try to write code(!!), instead find someone (a member of the teaching team, a colleague. .) with whom to discuss the problem.
2. Implement and test only small portions of your design at a single time. For example, you might start by writing the `convertHoursMintuesToDouble()` method and thoroughly testing that it is correct before you go on. Then write the `RouteTotalOperationalTime()` method and test it thoroughly. Run each method with a spectrum of different parameters. Analyze the results of testing and fixing any logic errors that are uncovered. Then choose another small part, etc.
3. When you believe the program is complete, use the correct answers that you calculated in step 1 (above) to test your complete implementation. Then try inputting various other values to your interactive program and see what happens!
4. Make sure that the `print` and `println` statements you use are meaningful and self explanatory. It is better to write short but descriptive phrases than complete sentences.
5. Examine your code. If you find that you have written methods that are almost the same, but each has changed words, variables or values, then they should be combined into one method that *accepts different parameters*. (If you are not sure about this step ASK! It is a vital learning objective of this assignment.)

PART 2 HAND IN: Submit your code for step 5 (above) using the 'Assignments' link of the course web page. Your code must be submitted before the beginning of your scheduled lab class in the week of October 11-15, 2010. Lab sections B0? And B0? (whose labs this week fall on a statutory holiday) must submit their code before 10 am on Tuesday, October 12.

Sample output of this program:

```
V A N C O U V E R   I S L A N D   F E R R Y   S Y S T E M
    Gross Operational Cost Calculation

Victoria to Sooke - Number of runs ==>7
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 45
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 47
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 50
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 51
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 51
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 50
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 50
Sooke to Victoria - Number of runs ==>7
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 35
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 35
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 34
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 32
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 30
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 30
  Duration - Number Hours (24 hour clock) ==>0
  Duration Minute (0-59) ==> 30
Victoria to Sydney - Number of runs ==>4
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 15
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 20
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 20
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 20
Sydney to Victoria - Number of runs ==>4
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 5
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 8
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 5
  Duration - Number Hours (24 hour clock) ==>1
  Duration Minute (0-59) ==> 10

RESULTS:
  Total daily operational hours: 30.216666666666665
  Gross daily operational cost:  16163.916666666666
```

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

- Methods

  The main method plus the `convertHoursMintuesToDouble` and `RouteTotalOperationalTime` methods must be called repeatedly, i.e., as much as possible passing different value in each call. (i.e., There should NOT be multiple variations of these methods for different runs.)

  Your methods should accept input parameter values and return an output value ***exactly*** as described above.

  Other suitable methods (or no other methods, if 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 (time values) and print output (cost info) ***exactly*** as described above.

More to Discover

- Consider the complexity of a program that would calculate the daily operational time of the fleet that is operated by the BC Ferry Corporation. Your program could easily be expanded to do this calculation. There are, however, more efficient methods of inputting the times, making execution less tedious.
- The problem of creating a schedule is actually quite a complex problem. In the case of a ferry corporation, there are likely to be constraints on the available times on a dock, the availability of staff, coordination with other transportation services (like busses), tide tables, etc. Many transportation industries, such as airlines, trucking companies, railways, need conflict free schedules. But so do building project managers, manufacturing business and universities. Imagine the challenge of scheduling all the

courses at this university! There are student groups, instructors, and classrooms that must all be free of conflicts.

- In general, the scheduling problem has been studied a lot and found to be very difficult (or perhaps impossible) to solve. That is, no efficient solutions have yet been found, even though many computer scientists has worked on the problem.