Given: Friday, September 11, 2015
Due: Friday, September 18, 2015 **before the start of the lecture**
Weight: **3%** of your course grade

The assignment is to be completed individually, in accordance with the policies and expectations in the course outline.

Objectives:

- Review of chapters 1 through 5 in the textbook, especially chapters 4 and 5
- Demonstrate your ability to design and build simple custom classes
- Demonstrate your ability to work with the ArrayList object
- Demonstrate your ability to program methods to:
  - ask for and receive user input from the console
  - read data from a text file using an iterator
  - traverse an ArrayList using while loops
  - write formatted output to the console
- Practice thinking about objects as having both *state* (via properties) and *behavior* (via public operations).
- Gain experience working with multiple objects ("instances") of a class, each with its own distinct identity and state.
- Gain experience with constructor methods.
- Demonstrate proper use of the "public static void main()" method to execute the program
- Demonstrate your ability to write proper documentation using JavaDoc

Associated textbook sections: before starting, students should be familiar with most of sections 4.1-4.5

This assignment serves as a short review of some of the concepts you should have learned in comp 1501. You will once again have to break the problem down into different methods and make sure each works properly. This time you will be using the special ***public static void main()*** method as the starting point that will call the important methods for your program to execute. **The main method should contain minimal code, with the majority of the functionality being provided by method calls to instantiated objects.**

What the program must do

You will be building a basic program that will allow the user to read a text file containing temperature measures for a weather station for every hour in a month and write a report profiling that information. The program do the following:

- Ask the user for the name of the text file that holds temperature data for a month
- Load the contents of the text file into an appropriate data structure in the program (using ArrayList)
- Scan the loaded data to calculate some simple statistics
- Write a formatted report describing the temperature profile of the month

User Interface

The user must start the application from blueJ. The program will perform the following Use Case which assumes the name of the main class in the program is **TemperatureProfiler**.

User Action	Program Behaviour
1. User right-clicks the TemperatureProfiler class and selects main	2. Program displays any welcoming messages and asks the user for the name of the file to load. The prompt should **not** go to a new line but instead wait for the user's input.
3. User types in the name of the file and presses enter	4. The program displays the information the user just typed and then outputs the report.

Note the following. In step 2 you should tell the user whether to type in the filenames suffix (.txt) or not.

Required functionality

The program must do the following:

1. Execute the user interface as described above
2. Segment the data into the following 4 daily zones:
    a. "Night" which is from 0:00 (i.e. midnight) to 5:00
    b. "Morning" which is from 6:00 to 11:00
    c. "Afternoon" which is from 12:00 to 17:00
    d. "Evening" which is from 18:00 to 23:00
3. Calculate the average temperature for each zone for all the days of the month (i.e. sum all the datapoints in the zone for all the days in the month and divide by the number of datapoints)
4. Write the required report in the proper format

The Input File

```
...
1     21    19.8
1     22    20.2
1     23    18.6
2     0     16.6
2     1     16.1
2     2     14.2
...
```

The input file will be a text file with each line holding one day's information. The first token in each line is the day of the month. The second token is the hour of the day. The third token is the temperature in Celsius degrees. Note that the temperature can be negative.

The tokens are separated by white space so use the **nextInt()** method to read the tokens in the correct sequence.

The file is well formed meaning that you can assume that the data is complete and in the correct sequence. Therefore your program does not need to check for error conditions in the data.

The Report

The program must be displayed on the console to look like the sample below. Note the following:

1. You must print out the name of the file that you processed
2. You must state how many days are In the month before you list the days
3. You must name the zone (in this case the Night zone)
4. You must report the average temperature for the zone before you list the days
5. Time of day must be in nn:00 format
6. The detail information must be centered under the column header

7. If the temperature is within 1 degree of the mean (plus or minus) there must be an asterisks on each side of the temperature
8. Notice that in this sample there is a space between the asterisk and the beginning of the temperature, that is because it is possible there could be a minus sign.

```
You requested: Calgary July 2015.txt
This month has 31 days

Zone: Night Average Temp =   13.9
Day\Time:    0:00      1:00      2:00      3:00      4:00      5:00
--------- --------- --------- --------- --------- --------- ---------
    1        16.4      16.9      17.3    * 14.4*   * 13.5*   * 13.8*
    2        16.6      16.1    * 14.2*   * 13.6*     12.6      12.7
    3        19.5      20.0      17.8      17.1      15.8    * 14.6*
    4        16.9      15.5      15.2      15.8      15.8      15.5
    5      * 13.2*     12.3      10.4      10.7      11.2      10.0
    6        11.4      11.7      11.1      10.9       9.0       8.9
```

The "Assign 1 Sample Report.txt" file shows a complete report.

Special Things to Think about

If you approach the problem in a methodical manner you will easily complete this assignment by the deadline. Here are some significant problems that you may encounter with strong advice as to how to avoid them.

Problem	Solution
Making sure the right numbers are read in the right order	Use next(), nextInt() or nextDouble () to read your tokens directly from the file in the right order. **Do not read a line at a time!**
The program can open the input file but find nothing there.	There is a bug in some versions of the Java SDK that will cause Scanner to not be able to see the contents of text files opened with the ***File*** object. Use the ***FileInputStream*** object instead.
Formatting the hour as nn:00	Do not use any date/time objects to format the hour. Try this: `hourStr=String.format("%2d",hourNum)`
Formatting the temperature	You can spend a lot of time exploring different ways to do this. Here is the easiest: `tempStr=String.format("%5.1f",hourTemp)`

Coding Style and Documentation Standards

- choose self-documenting identifiers (e.g. for variables, methods, etc.)
- explicitly initialize variables (where appropriate)
- consistently and correctly use of white space, including:
    - proper indentation
    - other use of white space to highlight the logical structure of an algorithm
- mark all methods as either public or private, as appropriate
- mark all instance variables as private
- make variables local, unless they are for genuine per-object properties
- avoid overly long/complex methods in favour of delegating key subtasks to other methods (this is a major part of avoiding code duplication)
- use inline comments, sparingly, to clarity especially tricky or less-obvious segments of code

- avoid hard-coded magic literals in favour of named constants

Submission Instructions

Before attempting the steps below, ensure your solution compiles without errors or warnings and ensure your program works as expected. Correct any problems before continuing. **Code that does not compile and can't be tested will be heavily penalized.**

Important reminder: this assignment is to be completed *individually*. Students are strictly cautioned against submitting work they didn't do themselves.

Please follow these submission instructions **Exactly**.

1. Rename your BlueJ project folder (if you haven't already) as:

   <LastName>_<FirstName>_***Asg1***

2. Submit the entire BlueJ project folder (with all its contents) to the "submit" folder, which appears under the "I:" drive each time you log in to a university PC. If you are submitting from a non-university computer (e.g. a home PC), you can access the submit folder via secure.mtroyal.ca, into which you can upload a compressed (e.g. ".zip") version of your main folder.

Note: ensure the datafiles your program is working with is in the project folder. The evaluation of the program will be done with a different file which will be placed in your project file.

Javadoc Templates

Replace all highlighted text with actual values

File header

```
/**
 *  <include description of the class here>
 * @author <your name>
 * @version 1.0
 * Last Modified: <date> - <action> <who made the change>
 */
```

Method with no parameters and no return type

```
/**
 *  <a description of what the method does>
 */
```

Method with no parameters and has a return type

```
/**
 *  <a description of what the method does>
 *  @return <a description of what is returned, including if errors are returned>
 */
```

Method with parameters and no return type

```
/**
 *  <a description of what the method does>
 *  @param <parame1name> <a one line description of the parameter>
```

```
 *  @param <parame2name> <a one line description of the parameter>
 *  < include all parameters in a similar way as above>
 */
```

Method with parameters and has return type

```
/**
 *  <a description of what the method does>
 *  @param <parame1name> <a one line description of the parameter>
 *  @param <parame2name> <a one line description of the parameter>
 *  < include all parameters in a similar way as above>
 * @return <a description of what is returned, including if errors are returned>
 */
```