CS149: Programming Fundamentals
James Madison University, Fall 2017 Semester

PA4: WindChillMapper

1. Due Dates and Submission Details
See below for all due dates and submission details. Note there are multiple submission parts.

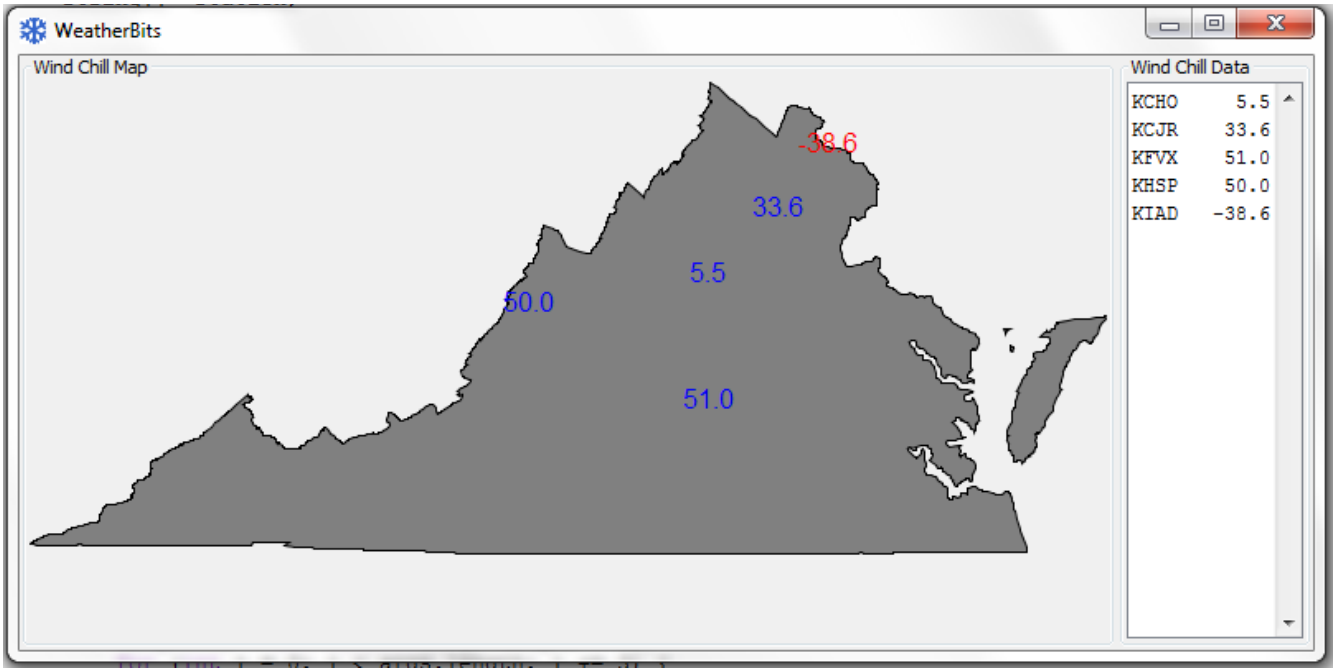
2. Honor Code
This assignment should be completed individually to maximize learning. It is important that you adhere to the Course Policies explained in detail at https://w3.cs.jmu.edu/cs149/coursepolicies/ listed under the section on Programming Assignments. Also relevant is the JMU Honor Code.

3. Learning Objectives
This programming assignment is designed to help you learn about (and assess whether you have learned about):

- Writing moderately complicated algorithms.
- Using loops of various kinds.
- Constructing arrays.
- Using arrays.

It is also intended to reinforce what you have already learned about integer arithmetic and the nesting of `if` statements. Finally, it is intended to demonstrate the value of reusable methods and classes (i.e., to make you appreciate the fact that methods/classes can, if designed and implemented properly, be used for multiple projects).

4. Background
WeatherBits is a (fictitious) company that develops weather-related software of various kinds. They were happy with the work you did on the `WindChillCalculator` and have asked you to develop an application that can create maps of wind chill values (for stations in Virginia) like the following.



5. Existing Resources

This assignment makes use of several existing resources, some of which were developed for the `WindChillCalculator` and some of which were developed specifically for the `WindChillMapper`.

If you were unable to complete the `WindChillCalculator`, you may use the solutions that were provided to you. However, you would be better-served by completing them yourself.

5.1 Resources for Performing Calculations

This assignment uses the `Convert` and `Weather` classes that were developed for the `WindChillCalculator` to perform calculations of various kinds. Since they do not need to be modified, you **may** copy the `.class` files into the directory/folder that contains the source code for this assignment. However, since that will complicate the submission process, you **should** copy the `.java` files instead.

5.2 Resources for Obtaining Input from the User

This assignment uses the `TerminalUI` class that was developed for the `WindChillCalculator` to obtain input from the user. However it will need to be modified as discussed below. Hence, you **must** copy the `.java` file into the directory/folder that contains the source code for this assignment (so that you don't confuse the two versions).

5.3 Resources for Mapping

This assignment uses the `Map` class (the byte code for which is in the file `Map.class`) that was developed by someone at *WeatherBits*. The only method you need to use from this class has the signature:

```
public static void showWindChills(String[]  icao4,
                                  double[]  windChill,
                                  boolean[] dangerous)
```

where `icao4` contains the 4-character ICAO codes for the stations, `windChill` contains the wind chill values for the stations, and `dangerous` contains `true` if the corresponding wind chill for the station is considered dangerous and `false` otherwise. Note that these are conformal arrays (i.e., they use the same indexing scheme so that element `i` of the three arrays all refer to the same station). This method **must** be called only once and **must** be passed arrays containing all of the information that is needed to map the wind chill values for all of the stations. Each time this method is called it "erases" the existing map.

This class makes use of the data files `boundary-va.txt` (which contains the map of Virginia) and `stations-va.txt` (which contains the longitude/latitude of the weather stations)

All three of these files **must** be in the directory/folder that contains the source code for this assignment. They can be downloaded (by right-clicking in most browsers) using the following links:

Map.class
boundary-va.txt
stations-va.txt

5.4 Resources for Converting Strings

The `Double.parseDouble(String s)` method (which is in the package `java.lang`, so need not be imported) can be used to convert the `String` parameter named `s` into a `double` value. It returns the corresponding `double` value.

5.5 Resources for Obtaining the Length of an Array

The `Array.getLength(String[] a)` method (which is in the package `java.lang.reflect`, so **must be** imported) can be used to determine the length of the `String[]` it is passed. It return the length as an `int` value.

If you know about array attributes, you can also use the `length` attribute (which is also an `int`) for the same purpose.

6. Classes for this Assignment

You must add a method to the `TerminalUI` class and write the main class (named `WindChillMapper`).

6.1 `TerminalIO`

If you use the `JMUConsole` class for input and output then you **must** add the following method to the `TerminalIO` class,

```
public static int promptForValidInt(String message, int min, int max)
```

whereas if you use a `Scanner` object for input and `System.out` for output then you **must** add the following method to the `TerminalIO` class.

```
public static int promptForValidInt(Scanner keyboard, String message,
                                     int min, int max)
```

This method **must** repeatedly print the given `message` until the user responds with an `int` that is in the closed interval [`min`, `max`], and it **must** then return that `int`.

6.2 `WindChillMapper`

`WindChillMapper` must be able to receive input from two possible sources, the command-line and the terminal/console. Specifically:

- If it is passed command-line arguments it must use them (perhaps prompting the user for additional input).
- If it is not passed command-line arguments, it must prompt the user to provide all of the necessary input.
- If it is passed command-line arguments that do not represent complete groups, it must prompt for the omitted information.

If they exist, the command-line arguments will be organized in groups/records of three elements as follows:

Element `i`	**Element `i+1`**	**Element `i+2`**
Temperature (F)	Wind Velocity (Mi/Hr)	Station (ICAO)

and there can be an arbitrary number of groups/records. So, for example, the map above was generated by executing the program from the command-line as follows:

```
java WindChillMapper 20.2 17.4 KCHO  40.0 10.0 KCJR  51.0 20.0 KFVX  50.0  3.0 KHSP -15.0 15.0 KIAD
```

which contains 5 groups/records.

If there are no command-line arguments or if the command-line arguments don't represent a collection of complete groups/records, the program must prompt the user for input as in the `WindChillCalculator`, except that, since this application only works for Virginia, it must not prompt for the country but must instead prompt for the number of data points. So, for example, the following execution also generates the map above.

```
java WindChillMapper
Number of Observations: 5
Temperature (degrees F): 20.2
Wind Velocity (mi/hr): 17.4
Location: KCHO
Temperature (degrees F): 40.0
Wind Velocity (mi/hr): 10.0
Location: KCJR
Temperature (degrees F): 51.0
Wind Velocity (mi/hr): 20.0
Location: KFVX
Temperature (degrees F): 50.0
```

```
Wind Velocity (mi/hr): 3.0
Location: KHSP
Temperature (degrees F): -15.0
Wind Velocity (mi/hr): 15.0
Location: KIAD
```

If there are some command-line arguments, but some of the elements of a group/record have been omitted, the program must prompt for the missing elements. So, for example, the map above was also generated by executing the program from the command-line as follows:

```
java WindChillMapper 20.2 17.4 KCHO  40.0 10.0 KCJR  51.0 20.0 KFVX  50.0  3.0 KHSP -15.0
```

with the following additional prompts and responses:

```
Wind Velocity (Mi/Hr): 15.0
Location: KIAD
```

In this example, 4 complete groups and 1 partial group are entered as command-line arguments and the user is prompted to complete the partial group.

The program **should** assume that all command-line arguments that are provided and all responses to prompts are of the appropriate type. However, when running interactively (i.e., when there are no command-line arguments), the program **must** validate that the number of observations entered by the user is in the closed interval [1, 100]. The following example illustrates a case in which the user entered two invalid responses to this prompt (before entering a valid value).

```
java WindChillMapper
Number of Observations: -1
Number of Observations: 200
Number of Observations: 5
Temperature (degrees F): 20.2
Wind Velocity (mi/hr): 17.4
Location: KCHO
Temperature (degrees F): 40.0
Wind Velocity (mi/hr): 10.0
Location: KCJR
Temperature (degrees F): 51.0
Wind Velocity (mi/hr): 20.0
Location: KFVX
Temperature (degrees F): 50.0
Wind Velocity (mi/hr): 3.0
Location: KHSP
Temperature (degrees F): -15.0
Wind Velocity (mi/hr): 15.0
Location: KIAD
```

8. A Recommended Process

This assignment is complicated enough that you are unlikely to complete it if you do not use a good process, especially because, unlike in the last assignment, it has not been divided into modules for you. What follows is one such process.

1. Make the necessary changes to the `TerminalUI` class.
2. Write a small driver for testing the modified `TerminalUI` class. (Hint: You probably will not want to use JUnit or the `Test` class. Instead, just write a small main class that calls the `promptForValidInt()` method.)
3. Write version 1 of the `WindChillMapper` class that ignores the command-line arguments and prompts the user for all inputs.
4. Test version 1 of the `WindChillMapper` class for several different numbers of stations.
5. Write version 2 of the `WindChillMapper` class that uses the command-line arguments if they are present, but assumes that all groups/records are complete.
6. Test version 2 of the `WindChillMapper` class for several different numbers of stations when they are provided from the command-line.
7. Test version 2 of the `WindChillMapper` class for several different numbers of stations when they are provided as responses to tests. In other words, re-run the tests you created for step 4 to ensure that you didn't break anything. (This is known as *regression testing*.)
8. Write version 3 of the `WindChillMapper` class that handles incomplete command-line arguments.
9. Test version 3 of the `WindChillMapper` class for several different numbers of stations. Some of you tests should have 1 missing command-line argument and some should have 2 missing command-line arguments.
10. Perform regression testing on version 3.

9. Submission

The submission for this assignment is divided into two parts that should be completed in order.

PA4-A: Understanding the Problem: due Friday, 10/27 11:00 pm

To complete Part A you should first carefully read the assignment specification. Once you feel confident that you have a good grasp of the assignment requirements, log into Canvas and complete Part A. **YOU MUST ANSWER ALL QUESTIONS CORRECTLY (earn 100%) TO GET ANY CREDIT FOR PART A**. You may take the quiz as many times as necessary but if you do not complete it on time, you will receive zero credit.

PA4-B: Java Implementation: due Friday, 11/3 11:00 pm

-25% before Saturday, 11/4 11:00 pm
-50% before Sunday, 11/5 11:00 pm
Not accepted after Sunday, 11:00 pm

PA4-B is a Web-CAT submission. You must submit a `.zip` file containing the source code for all of the classes necessary to compile and run `WindChillMapper` **except the `Map` class** (and the `JMUConsole` class if you use it). Your `.zip` file must not contain any source files you created for teasting purposes, any `.class` files, and/or the data files used by the `Map` class.

Note, since it is difficult to write automated system tests for `WindChillMapper`, **there is no limit on the number of Web-CAT submissions**. However, that does not mean that you shouldn't try to test your code "locally" before submitting it.

10. Grading

Your submissions will be graded according to the following criteria. Note that though there is no explicit category for Instructor grading based on style and code quality, individual points may be deducted in each category by the instructor for these types of issues. **Note .java files that do not compile will result in a grade of zero for Part B.**

Part A:	10%
Part B:	
Checkstyle	10%
Prompting for the Number of Observations	20%
Correctness without Command-Line Arguments	20%
Correctness when Command-Line Arguments are Complete	20%
Correctness when Command-Line Arguments are Incomplete	20%