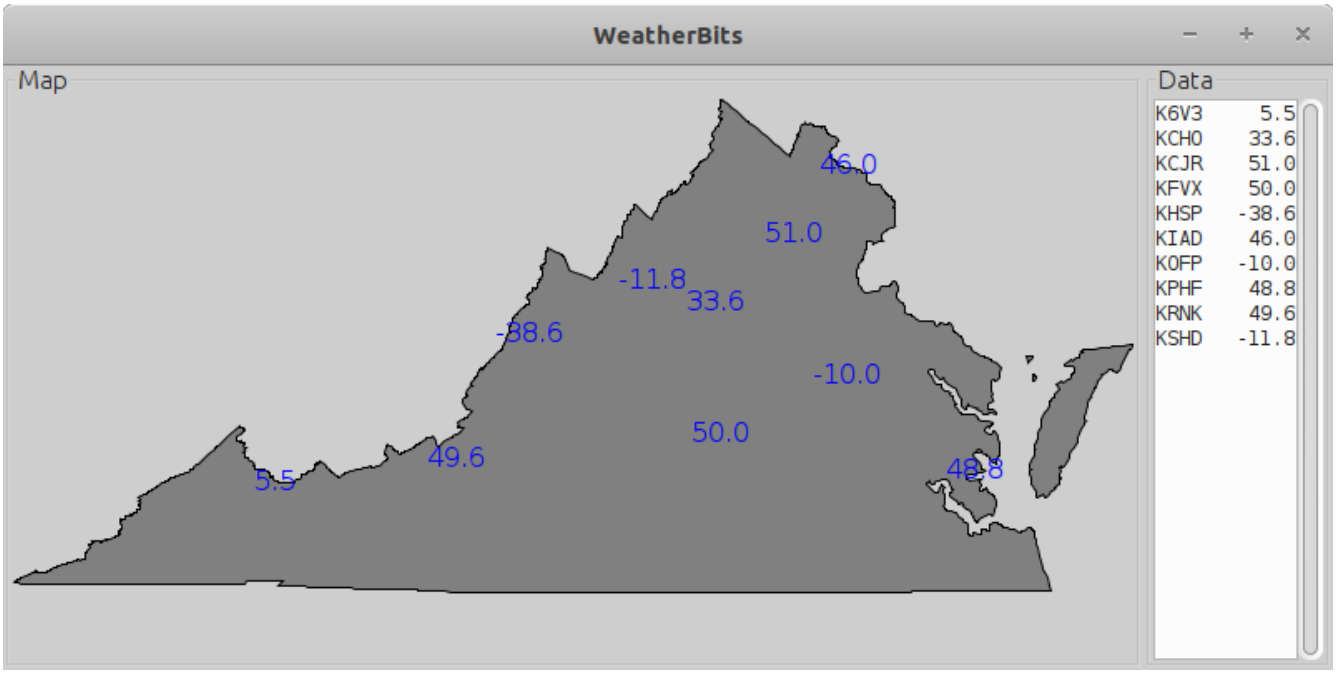
Programming Assignment 7

WeatherBits

`ChillMapper`

Overview

As you know from your previous work with them, *WeatherBits* is a company that develops weather-related software of various kinds. They have asked you to develop a main class (named `ChillMapper`) that can be used to display wind chill values (calculated using your `Weather` class for locations in North America) on a map like the following:



Specifications

The `ChillMapper` class must comply with the following specifications:

1. It must have a method with the following signature:

   ```
   public static void main(String[] args)
   ```

2. If there are no command-line arguments, the system must display a map containing wind chill values calculated from the weather data in the file named `"today.txt"`.

3. If there is one command-line argument, the system must display a map containing wind chill values calculated from the weather data in the file with the name contained in that argument (e.g., `"forecast.txt"`).

4. If there are 3, 6, 9, etc... command line arguments (i.e., if the number of command line arguments is a multiple of 3) then the system must display a map containing the wind chill values calculated from the weather data contained in the command-line arguments. Each record in this segmented/packed array contains three fields (described below).

5. In all other cases, the system must not display a map.

6. If either the temperature or the wind speed can't be converted into a valid double value for a particular station, that station must be ignored (but others must be processed).

7. Regardless of the source of the weather data (i.e., whether it is in a file or the command-line arguments), the system must use the first field as a `String` representation of the ICAO code, the second field as a `String` representation of the temperature (in degrees Fahrenheit), and the third field as a `String` representation of the wind speed (in mi/hr).

8. It must use the `windChillNA()` method in the `Weather` class to calculate the wind chill values.

9. It must use the `setTemperature()` method in the `Map` class to display the temperatures on the map.

10. The system must calculate and display the wind chill values for every station in the file or every station in the command-line arguments (depending on how the application is started). In the case of the former, there will be as many stations as there are lines in the file. In the case of the latter, there will be as many stations as there are command-line arguments divided by three.

Existing Components

The `Map` Class

Other programmers at *WeatherBits* have implemented and tested a `Map` class that you must use to draw the map. This class contains the following method:

> `void setTemperature(String icao, double temp)`
> Displays the given temperature on the map at the location of the weather station with the given ICAO code. Though this method can be passed any temperature, the `ChillMapper` application will only pass it wind chill values.

The `Text` Class (Revisited)

The `Text` class that you have used in the past has several methods for working with files. You must use (some or all of) these methods to read the data.

> `String readNextLineFrom(String fileName)`
> Reads the next "line" (from the file with the given `fileName`) and returns it. The first time this method is called it returns the first line in the file.

> `boolean thereAreMoreLinesIn(String fileName)`
> Returns `true` if there are more "lines" that can be read in the file with the given `fileName` and returns `false` otherwise.

> `String[] splitAtSpaces(String record)`
> Splits the given space-delimited `record` into fields, and returns the fields in an array. For example, if it is passed a `String` containing `"KSHD 5.0 13.0"` it will return a `String[]` in which element 0 is `"KSHD"`, element 1 is `"5.0"`, and element 2 is `"13.0"`.

> `double toDouble(String s)`
> Returns the `double` value of the `String` representation (named `s`) of a `double`. Note that, unlike `toNonnegativeDouble()`, this method will convert negative numbers. In addition, unlike `toNonnegativeDouble()`, this method returns the constant named `Double.NaN` if the `String` does not represent a number (i.e., is **N**ot **a N**umber, hence NaN). You can test to see whether a particular `double` value equals `Double.NaN` using the `Double.isNaN(double value)` method (which returns `true` if the `double` value is not a number and returns `false` otherwise). **You cannot use the `==` operator for this purpose.**

Existing Data Files

WeatherBits has provided you with several data files that you must use when testing your application.

`stations-va.txt`

This file contains information about the weather stations in Virginia. This file includes the 4-character International Civil Aviation Organization (ICAO) code for the weather station and its geographic coordinates. This file is used by the `Map` class. **You do not need to use this file directly and must not modify it in any way**.

`va.txt`

This file contains geographic coordinates for the boundary of the Commonwealth of Virginia. This file is used by the `Map` class. **You do not need to use this file directly and must not modify it in any way**.

`today.txt` and `forecast.txt`

These files contain weather information for different weather stations in the Commonwealth of Virginia. Each line is space-delimited and contains exactly three fields. The first field contains the 4-character ICAO code for the weather station, the second contains the temperature at that station (in degrees Fahrenheit), and the third contains the wind speed at that station (in mi/hr). `today.txt` contains measured information and `forecast.txt` contains forecasted information. Your `ChillMapper` class will need to use methods in the `Text` class to read these files. You must not modify these files in any way.

Recommended Process

1. Read and understand the entire assignment.
2. Create a directory/folder (e.g., named `pa7`) that will hold all of the files for this assignment.
3. Copy `Weather.java` into the directory you created for this assignment. You may use either your implementation or the solution that was provided to you.
4. Copy `Text.class` into the directory you created for this assignment.
5. Download the .zip file containing `Map.class` and the data files, and **unzip them into the directory you created for this assignment**.
6. Consider the case in which there are *exactly three* command-line arguments.
    - 6.1 Implement the code that handles this case.
    - 6.2 Test the code for this case with valid (i.e., numeric) arguments.
    - 6.3 Test the code for this case with a mix of valid and invalid arguments.

7 Consider the case in which there are *a multiple of three* command-line arguments.

   7.1 Implement the code that handles this case. (Hint: This should involve a loop.)

   7.2 Test the code for this case with valid arguments.

   7.3 Test the code for this case with a mix of valid and invalid arguments.

8 Delete the code for handling exactly three command-line arguments. In other words, that case is just a special version of the case in which there are a multiple of three command-line arguments.

9 Consider the case in which there are *zero* command-line arguments.

   9.1 Implement the code that handles this case.

   9.2 Test the code that handles this case.

   9.3 If you have any code duplication, eliminate it by adding a method to the `ChillMapper` class and then invoking that method multiple times. (Hint: This should involve another loop.)

   9.4 Test all of the cases again to make sure that you haven't introduced any defects.

10 Consider the case in which there is one command-line argument. If your code properly handles no command-line arguments, this case should be straightforward. However, again make sure that you do not have duplicate code.

11 Consider the other cases (i.e., two or more command-line arguments but not a multiple of three).

12 Submit both `Weather.java` and `ChillMapper.java` in a file named `pa7.zip` using Autolab. Do not include any other files in the `.zip` file.

Some Advice

Since the requirements do not specify what methods must be in the `ChillMapper` class (other than `main()`), you will have to both design and implement this class. That means that you will need to spend a considerable amount of time thinking about what methods you want to include.

The `ChillMapper` class can be implemented in a variety of different ways, some of which are elegant and some of which are inelegant. You should not focus exclusively on getting your code working, you should also focus on making your code elegant. (Indeed, your grade will be based on the elegance of your solution. So, even if your code works, you might receive a very low grade if your solution is not well-designed.)

A Note About Testing

Recall that we have made a distinction between unit testing (in which one module is tested at a time) and integration testing (in which multiple modules are tested together). You conducted unit testing for the previous assignment because you were only testing the the `windChillNA()` method in the `Weather` class. For this assignment, you will need to conduct integration testing, because you need to test how the `ChillMapper` class integrates with the `Weather` class. Hence, you will not need to make use of the `Test` class for this assignment, instead you will need to execute the `ChillMapper` providing it with a variety of different command-line arguments.

Grading

Your code will first be graded by Autolab and then by the Professor. The grade you receive from Autolab is the maximum grade that you can receive on the assignment.

Autolab Grading

Your code must compile (in Autolab, this will be indicated in the section on “Does your code compile?”), and all class names and method signatures comply with the specifications (in Autolab, this will be indicated in the section on “Do your class names, method signatures, etc. comply with the specifications?”) for you to receive any points on this assignment.

Autolab will then grade your submission as follows:

Conformance to the Course Style Guide:	**20 points** (All or Nothing)
Correctness:	**80 points** (Partial Credit Possible)
Handling of 0 Arguments:	16 points
Handling of 1 Argument:	16 points
Handling of 3, 6, 9, ... Arguments:	24 points
Other Cases:	24 points

Manual Grading

After the due date, the Professor may manually review your code. At this time, points may be deducted for inelegant code, inappropriate variable names, bad comments, etc.