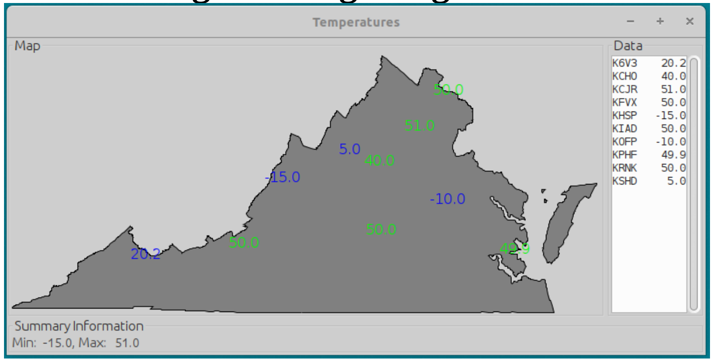
Fall 2022
CS159 - Advanced Programming

Programming Assignment 2



```
K6V3    20.2
KCHO    40.0
KCJR    51.0
KFVX    50.0
KHSP   -15.0
KIAD    50.0
KOFP   -10.0
KPHF    49.9
KRNK    50.0
KSHD     5.0
```

Min: -15.0, Max: 51.0

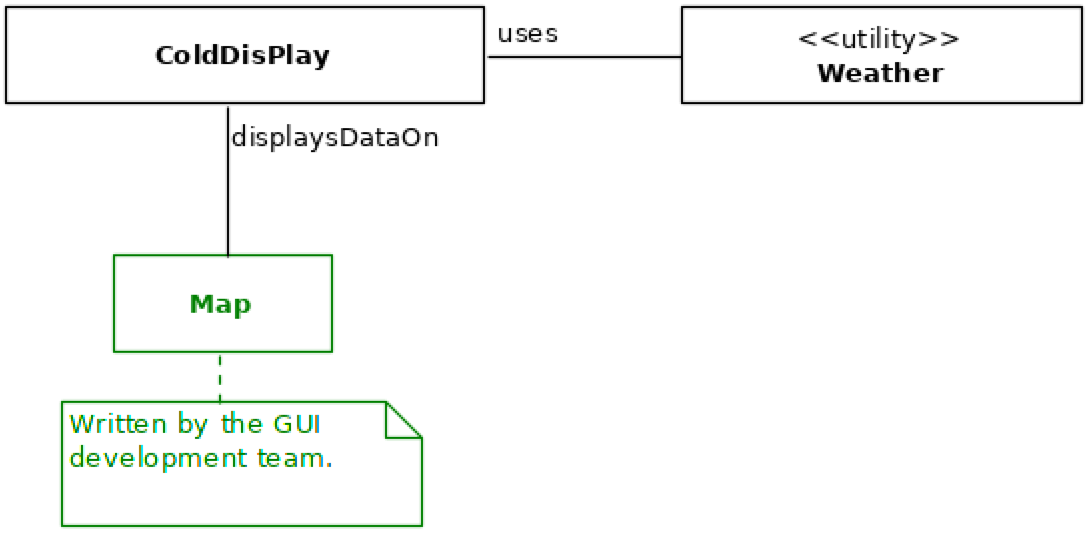
ColdDisPlay

1. Learning Objectives

This assignment is designed to help you learn two things. First, it will help you learn about conditions and loops. Second, it will help you learn several programming patterns (including accumulators, indicator methods, and segmented arrays,).

2. Overview

WeatherBits is a (fictitious) company that develops weather-related software of various kinds. They have asked you to create the first version of a mapping programming called *ColdDisPlay* that can display temperatures or wind chill factors. They have implemented the `Map` class which encapsulates the graphical user interface (GUI), and want you to implement a utility class (named `Weather`), and a main class (named `ColdDisPlay`). The relationships between these classes is illustrated in the following abstract UML class diagram:



Classes in green were written by the GUI development team, and classes in black are to be written by you for this assignment.

When executed, the `ColdDisPlay` class will be passed either the `String` `"Temperatures"` or `"WindChills"` as command-line argument 0. The subsequent command-line arguments will represent one or more records, each of which has three fields containing the station identifier, the temperature at that station, and the wind velocity at that station. It must then display the appropriate information on a `Map` object.

3. Background Information

Before you can start working on the classes, you need to learn a little bit about weather and weather collection.

3.1. Acronyms

ICAO	International Civil Aviation Organization
MPH	Miles Per Hour

3.2. Definitions

ICAO4 Codes. Four-letter codes for weather stations.

3.3. Calculation of Wind Chill Factors

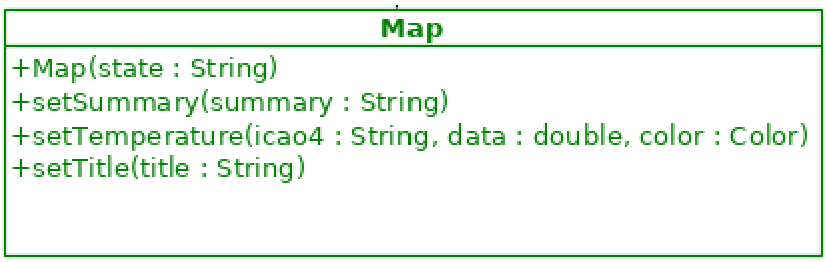
Wind chill values are calculated differently in different parts of the world (and have been calculated differently in the past). For the purposes of this project, you must use the following formula that was developed for North American **air temperatures at or below 50°F and wind speeds above 3mph**:

$$w = 35.74 + 0.6215 \cdot t - 35.75 \cdot v^{0.16} + 0.4275 \cdot t \cdot v^{0.16}$$

where *w* denotes the wind chill value in degrees Fahrenheit, *t* denotes the air temperature in degrees Fahrenheit, and *v* denotes the velocity in miles per hour. F**or other temperatures and wind speeds, the wind chill value must be the air temperature**.

4. The Existing Class

Other programmers at *WeatherBits* have created the GUI, which is encapsulated in the `Map` class. It can be summarized in a UML class diagram as follows:


Map
+Map(state : String)
+setSummary(summary : String)
+setTemperature(icao4 : String, data : double, color : Color)
+setTitle(title : String)

These methods are described below.

`Map()`
Constructs a `Map` object for the given state. (Note: You have only been provided with data for Virginia, so the actual parameter must be `"va"`.)

`setSummary()`
Sets the summary information to display below the geographic information.

`setTemperature()`
Sets the data to display, and the color to display it in, for a particular ICAO4 code.

`setTitle()`
Sets the title of the map.

It is available for download at:

```
https://w3.cs.jmu.edu/cs159/f22/pa2/Map.class
```

The data files for Virginia are available at:

```
https://w3.cs.jmu.edu/cs159/f22/pa2/va.txt
https://w3.cs.jmu.edu/cs159/f22/pa2/stations-va.txt
```

5. The Classes to be Written

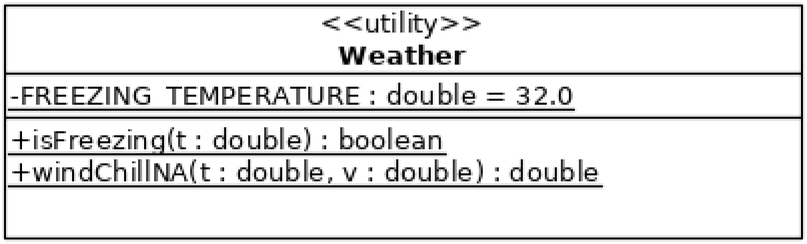
You must write two classes, both of which are described below.

5.1. The `Weather` Class

The `Weather` class is used to perform a variety of calculations related to weather measurement and forecasting.

5.1.1 UML Class Diagram

The following UML class diagram provides an overview of the attributes and methods in this class (which must be in the default package).



<<utility>>
Weather
-FREEZING_TEMPERATURE : double = 32.0
+isFreezing(t : double) : boolean
+windChillNA(t : double, v : double) : double

5.1.2 Detailed Design Specifications

In addition to the specifications contained in the UML class diagram, this class must conform to the following specifications.

1. All of the "constants" must be declared to be `final.`
2. `isFreezing()` must return `true` when the value of `t` is less than or equal to the `FREEZING_TEMPERATURE`.
3. `windChillNA()` must calculate the wind chill factor (for North America) from the given temperature, `t`, and wind velocity, `v`, using the approach described above.

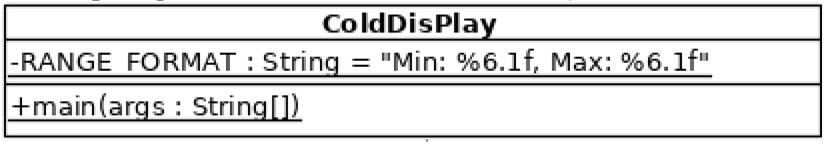
   3.1. It must not round or truncate the answer.

5.2. The `ColdDisPlay` Class

The `ColdDisPlay` class is the main class for the application.

5.2.1 UML Class Diagram

The following UML class diagram provides an overview of this class (which must be in the default package).



5.2.2 Detailed Design Specifications

In addition to the specifications contained in the UML class diagram, this class must conform to the following specifications.

1. Command-Line Arguments
   1.1. `args[0]` will contain a `String` describing the type of map to display (either `"Temperatures"` or `"WindChills"`).
   1.2. For `i` equal to 1, 4, 7, 10, etc., `args[i+0]`, `args[i+1]`, and `args[i+2]` will contain `String` representations of the station identifier (a `String` ICAO4 code), the temperature at that station (a `double`), and the wind velocity at that station (a `double`).
   1.3. There may be one or more such records.
2. The `main()` Method
   2.1. If the number of command-line arguments is invalid it must display a `Map` with the title of `"Data Error"`.
      2.1.1. It is invalid for `args.length` to be `1`.
      2.1.2. It is invalid for `args.length - 1` to be a multiple of anything other than `3`.
   2.2. If the number of command-line arguments is valid it must display a `Map` containing either temperatures or wind chill factors for every station based on the value of `args[0]` (which you may assume is always valid).
      2.2.1. The title of the `Map` must be identical to `args[0]`.
      2.2.2. If wind chills are being displayed, they must be displayed in `Color.BLACK`.
      2.2.3. If temperatures are being displayed, they must be displayed in either `Color.BLUE` or `Color.GREEN`.
         2.2.3.1. If a temperature is at or below freezing then it must be displayed in `Color.BLUE`.
         2.2.3.2. Otherwise it must be displayed in `Color.GREEN`.
   2.3. Must display the range (of temperatures or wind chill factors) as summary information.
      2.3.1. The summary information must be formatted using the attribute named `RANGE_FORMAT` which must contain the `String` literal `"Min: %6.1f, Max: %6.1f"`. Hint: You can use the `static` method named `format()` in the `String` class for this purpose.

Note that the `Color` class is in the package `java.awt` (not `java.lang)` so will need to be imported. In other words, the first statement in the class must be either:

```
import java.awt.*;
```

or

```
import java.awt.Color;
```

6. Submission

You must submit (using Gradescope):

1. Your implementation of the two classes you wrote. Do not include the `Map` class or the data files.

There is no limit on the number of submissions and no penalty for excessive submissions.

Note that your submission will not be graded if it does not comply with the specifications. So, your submission should include a stubbed-out version of all of the classes. (This will allow you to get credit for the classes/methods that you do implement correctly.)

7. Grading

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

7.1. Gradescope Grading

Your code must compile (in Gradescope, this will be indicated in the section on “Does your code compile?”) and all class names and method signatures must comply with the specifications (in Gradescope, 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. Gradescope will then grade your submission as follows:

Conformance to the Style Guide:	20 points	(Partial Credit Possible; Success Required)
Correctness:	80 points	(Partial Credit Possible)

Note that "Conformance to the Course Style Guide" is described as being "Success Required". This means that Gradescope will not assess subsequent criteria unless this criterion is satisfied (and you will receive a grade of 0 for the criteria that aren't assessed). So, in this case, if your code does not conform to the style guide then it will not be assessed for correctness (and you will receive a grade of 0 for correctness).

As before, Gradescope will provide you with hints, but may not completely identify the defects.

7.2. 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.

8. Recommended Process

Since nobody will be looking over your shoulder, you can use any process that you would like to use. However, it is strongly recommended that you use the process described here.

8.1. Get Started

1. Read and understand the entire assignment.
2. Create a project for this assignment named `pa2`. (Remember, **do not create a module** and **create separate folders for source and class files**).
3. Activate Checkstyle for this assignment.
4. Add the two data files to the project by dragging them from a file explorer into Eclipse. Specifically, drag them into the `pa2` project (**not** a directory/folder insider of the `pa2` project)
5. Add `Map.class` to the project. See the Department Wiki at:

   `https://wiki.cs.jmu.edu/student/eclipse/help#using_third-party_classjar_files`

   for help.

8.2. Understand the Calculations and Develop Test Cases

1. By hand (i.e., using paper and pencil), calculate the wind chill factors for the following temperatures and wind velocities.

Temperature	Wind Speed
20.2	17.4
40.0	10.0
-15.0	15.0
50.0	10.0
49.9	4.0
50.0	3.1
5.0	13.0
51.0	20.0
59.0	3.1
95.0	10.0
20.0	3.0
-10.0	3.0
49.9	2.9
53.0	3.0
93.0	2.0

Of course, you may use a calculator, but you should make sure you understand the differences that may arise when using a calculator and when using Java.

Note that the different test cases are color-coded. Before you start writing any code, make sure that you understand the different colors. (Hint: They are related to the algorithm for calculating wind-chill factors described above.)

8.3. Stub-Out the All of the Classes

1. Create a version of the `Weather` class that contains all of the methods (with appropriate signatures), each of which should return `0.0` or `false` as appropriate.
2. Add the "javadoc" comments to the `Weather` class and the methods in it.
3. Check the style of the `Weather` class and make any necessary corrections.
4. Create a version of the `ColdDisPlay` class that has a `main()` method (with appropriate signature) that does nothing.
5. Add the "javadoc" comments to the `ColdDisPlay` class and the methods in it.
6. Check the style of the `ColdDisPlay` class and make any necessary corrections.

8.4. Implement and Test the `Weather` Class

1. Implement the `isFreezing()` method.
2. Test the `isFreezing()` method using the examples from above and debug it if necessary. (Note: You will need to develop a main class to test your code.)
3. Implement the `windChillNA()` method.
4. Test the `windChillNA()` method using the examples from above and debug it if necessary.

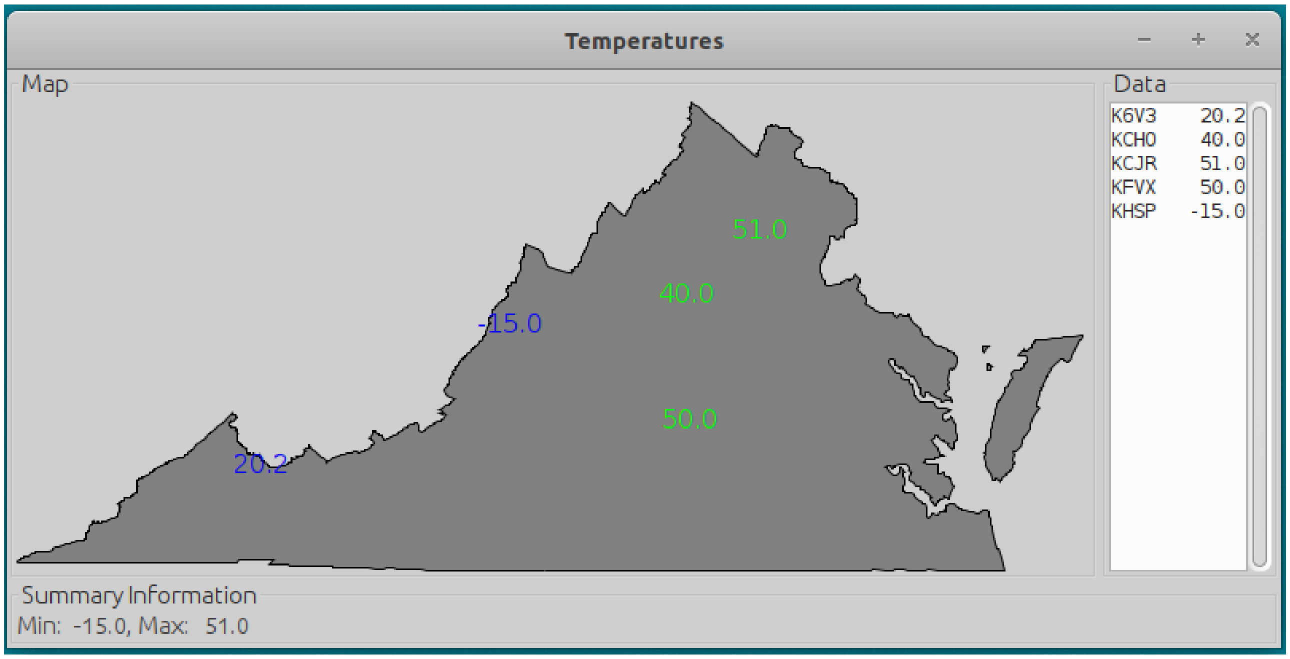
8.5. Implement and Test the `ColdDisPlay` Class

1. Implement and test the part of the code that determines if the correct number of command-line arguments has been given and sets the title to `"Data Error"` when appropriate.
2. Implement and test the part of the code that invokes the `setTitle()` method, passing it the appropriate `String`.
3. Implement and test the part of the code that processes the remaining records and invokes the `setTemperature()` method for each record.
4. Implement and test the part of the code that determines the minimum and maximum data value and invokes the `setSummary()` method.

The following command-line arguments:

```
Temperatures K6V3  20.2 17.4 KCHO  40.0 10.0 KCJR  51.0 20.0 KFVX  50.0  3.0 KHSP -15.0 15.0
```

should generate the following `Map`:



```
K6V3    20.2
KCHO    40.0
KCJR    51.0
KFVX    50.0
KHSP   -15.0
```

Min: -15.0, Max: 51.0

9. Help

The `Math` class has a `static` method with the signature `pow(double value, double exponent)` that you can use to calculate `value` raised to the `exponent` power.

10. Questions to Think About

You don't have to submit your answers to these questions, but you should try to answer them because they will help you determine whether or not you understand some of the important concepts covered in this assignment.

1. Why shouldn't the `isFreezing()` method use an `if` statement, `switch` statement, or ternary operator? (In other words, why will points be deducted if your method does?)
2. Why would it be even worse if the `isFreezing()` method used a loop? (In other words, why will even more points be deducted if your method uses a loop?)
3. How can you eliminate duplicate code within a single class?
4. How can you eliminate duplicate code across classes?

11. Looking Back - The Big Picture

By the time you complete this assignment you should have learned many things, including but not limited to the following “big picture” issues.

- The same kinds of problems need to be solved in a wider variety of applications. Hence, studying patterns, and being able to identify them, is incredibly important.
- Intermediate variables can be used to simplify code in which different actual parameters must be passed to a method depending on the value of a Boolean expression.
- One sometimes needs to iterate over an array (or other collection) in steps greater than one (e.g., in steps of 3 for this assignment).
- It is easy to mistakenly use multiple loops in a row when one will suffice (and be clearer).