# CS 2334
# Project 3: Java Collections Framework

April 3, 2018

**Due: 1 1:59 pm on Wednesday, April 18, 2018**

## Introduction

You have learned how to parse two different types of text files while working on your last two projects. In the second project, you have applied knowledge how to throw exceptions while parsing one of these data files. You have also assumed that data files needed are there and that parameters that you need are also there.

We had a discussion in our last class, and you have made many valuable observations. We will put some of them to use. You have noticed that we can simplify some of the method calls that need a particular parameter name. You can replace it with one method call that takes String with the name of a parameter. Let us call this parameter. You have also noticed that collected data could be stored in HashMap, or EnumMap, with String parameterId being the key. You will look for an opportunity to use data structures that would simplify your code. First, you will change:

```
/** Minimum tair across day. */
private StatMeasurement tairMin;
/** Maximum tair across day. */
private StatMeasurement tairMax;
/** Average tair across the days. */
private StatMeasurement tairAverage;
```

into

```
private HashMap<String, EnumMap<StatType, StatMeasurement> paramStats;
```

In addition, you can find at least one more place where member variables can be collected into some form of HashMap.

(***) Another observation was made. In order to find the minimum and maximum values, you can apply **java.util.Collections.min()** or **...max()** following this example:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ExtremeTest
{
    public static void main(String[] args)
    {
    List<Integer> list = new ArrayList<Integer>();

    // populate the list
    list.add(-18);
    list.add(40);
    list.add(-45);
    list.add(12);

    System.out.println("Minimum: " + Collections.min(list));
    }
}
```

Specifically, your final product will:

1. Reuse the Driver.java from project 2
2. Report the minimum, maximum, average, and total of a given statistics for specific stations
3. Find statistics the same way we did in project 2
4. EXTRA: Think about finding statistics using (***), but think about all possible cases with data being valid or not valid at different times. You don't have to submit this part, but if you have time please try to do it. Can you find an average, or total using Collections?

# Learning Objectives

By the end of this project, you should be able to:

1. Make use of **HashMaps**, **EnumMaps** to flexibly store data in a structure that is efficient to access

2. Simplify your code by reducing the number of methods used
3. Compute statistics over the stored data in a manner that does not rely on *a priori* knowledge of the specifics of the data
4. Continue to exercise good coding practices for Javadoc and for unit testing

## Proper Academic Conduct

This project can be done individually, or you can find a partner. If you work with a partner, indicate your partner in headers of your code, but keep it individual for WebCAT submissions.Since you will each turn in a copy of your solution, you will be individually responsible for your code review. You are to design the data structures and solution, and to implement and test this design.

## Strategies for Success

- The UML specification from the Project 2 constitutes good start for your refactoring.
- When you are implementing a class or a method, focus on just what that class/method should be doing. Try your best to put the larger problem out of your mind.
- Try to keep your old implementation working, gradually adding simplified implementation.
- We encourage you to work closely with your other team member, meeting in person when possible.
- Start this project early. In most cases, it cannot be completed in a day or two.
- Implement and test your project components incrementally. Don't wait until your entire implementation is done to start the testing process.
- Write your documentation as you go. Don't wait until the end of the implementation process to add documentation. It is often a good strategy to write your documentation **before** you begin your implementation.

## Preparation

Copy your project 2 implementation into a new *project3* project.

## Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59 pm on Wednesday, April 18, 2018
- Submit your project to Web-Cat using one of the two procedures documented in the Lab 2 specification.

## Grading: Code Review

The procedure is as follows:

- Submit your project for grading to the Web-Cat server.
- Any time following the submission, you may do the code review with the instructor or one of the TAs.
    1. Code review will be done during the assigned Lab time
- During the code review, we will discuss all aspects of the rubric, including:
    1. The results of the tests that we have executed against your code
    2. The documentation that has been provided (all three levels of documentation will be examined)
    3. The implementation.

## References

- The Java API: https://docs.oracle.com/javase/8/docs/api/
- The Oklahoma Mesonet: http://www.mesonet.org
- The API of the *Assert* class can be found at: http://junit.sourceforge.net/javadoc/org/junit/Assert.html

- JUnit tutorial in Eclipse:
  https://dzone.com/articles/junit-tutorial-beginners

# Rubric

The project will be graded out of 100 points. The distribution is as follows:

**Correctness/Testing: 45 points**

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against our set of tests. These unit tests will not be visible to you, but the Web-Cat server will inform you as to how many tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests).

**Style/Coding: 20 points**

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

**Design/Readability: 35 points**

This element will be assessed by a grader during the code review. Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation
- Missing or inappropriate inline documentation
- Inappropriate choice of variable or method names
- Inefficient implementation of an algorithm
- Incorrect implementation of an algorithm
- Incomplete coverage of your Unit Tests. We expect that your unit tests will test all lines of your code

If you do not submit compiled Javadoc for your project, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions(where points may be deducted).

**Bonus: up to 5 points**

You will earn one bonus point for every twelve hours that your assignment is submitted early.

**Penalties: up to 50 points**

You will lose ten points for every twenty four hours that your assignment is submitted late, up to fifty points.