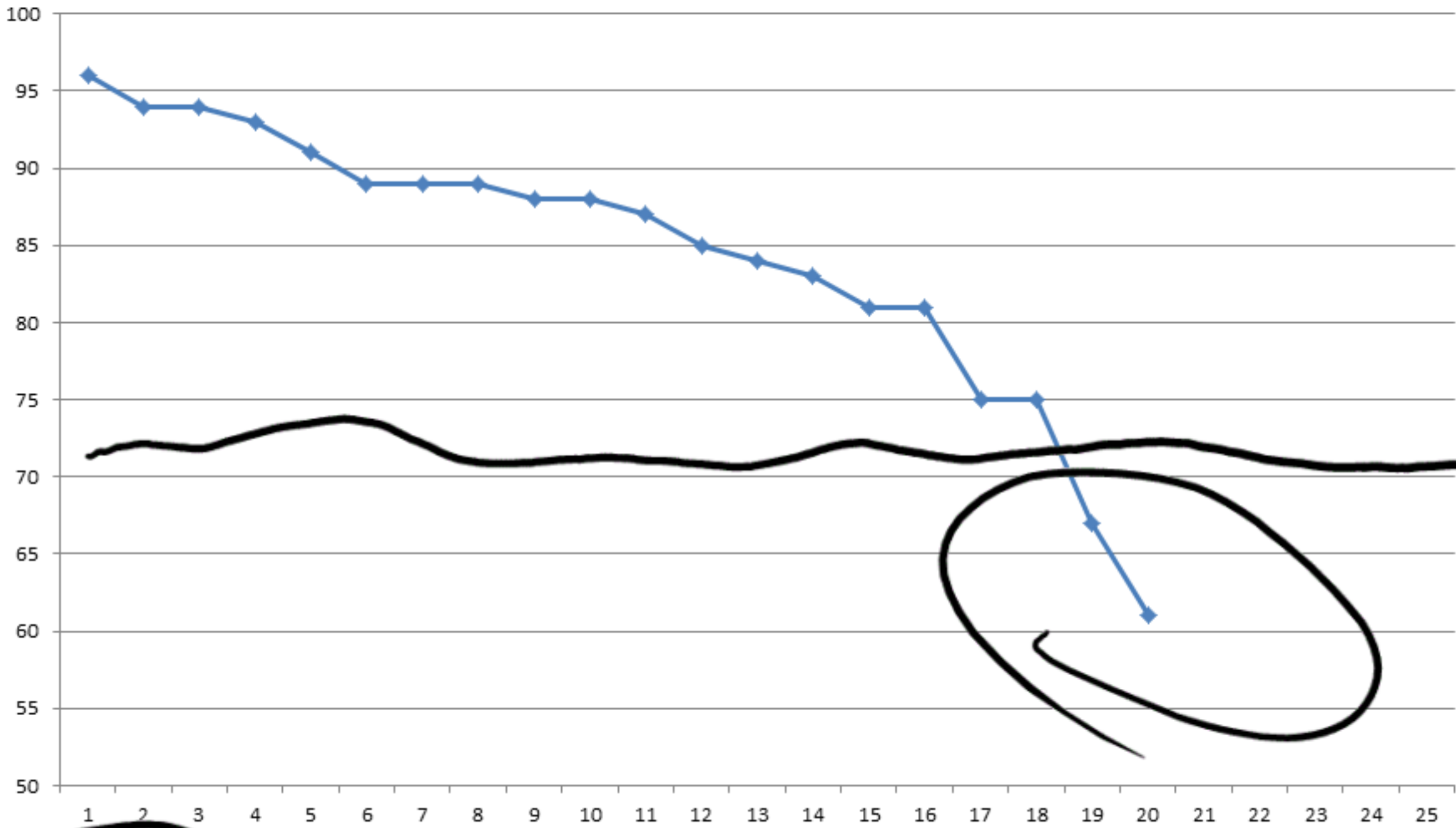


85%	Average	87%	87%	84%	81%	95%	86%	70%
88%	Median	88%	83%	100%	83%	100%	100%	73%
9%	STD	11%	9%	28%	16%	22%	27%	15%
96.00%	Max	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	86.67%
61.00%	Min	65.00%	66.67%	0.00%	33.33%	0.00%	13.33%	40.00%

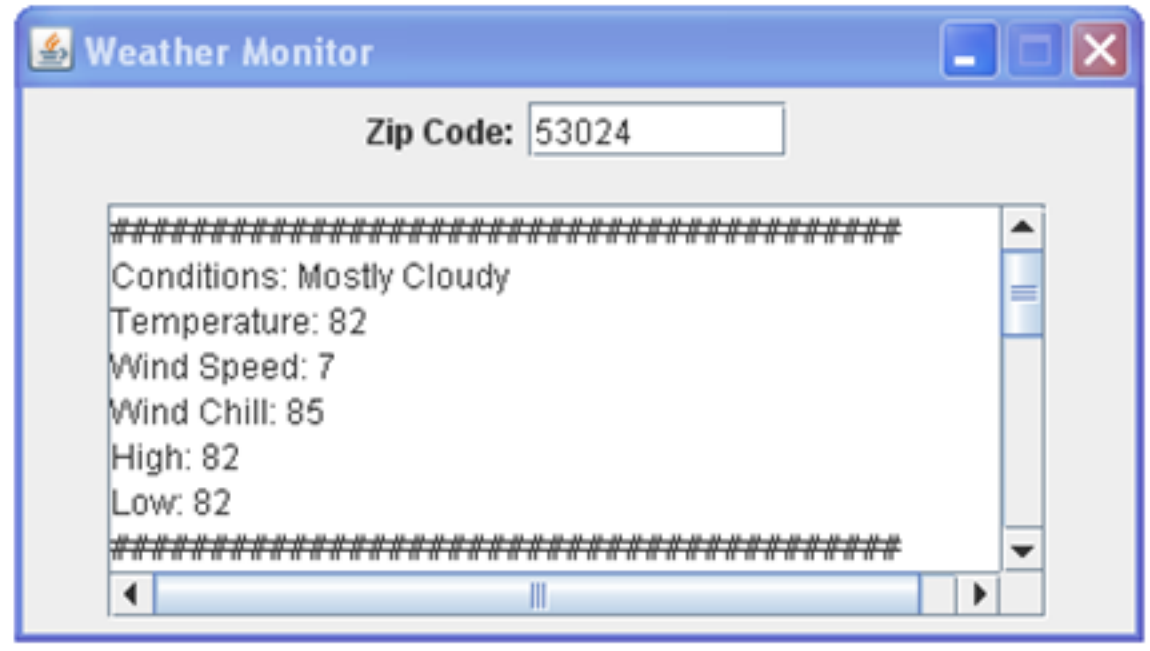
Testing with Jmock and Mock Objects

Lecture Objectives:

1) Explain the structure of a test created with Mock objects
2) Explain the purpose for the
   - Mockery context
   - Context mocks
   - Expectations
3) List the jar files necessary to test with JMock
4) Construct a mock object for a rudimentary system and use this object for the purpose of constructing unit tests.

Weather Bug

- The weather bug is a Java program which monitors the current conditions for a given zip code.
  - Current Temperature
  - conditions
  - Windspeed
  - Wind chill



- Will display an alert message if the wind chill is dangerous (below 32)

Weather Bug Design

class WeatherAnalyzer

WeatherAnalyzer

- currentWeather: WeatherConditionsInterface
- high: int = Integer.MIN_VALUE
- low: int = Integer.MAX_VALUE
- ui: WeatherMonitorInterface
- weatherSource: WeatherGrabberInterface
- windChill: double = 0.0
- WINDCHILLWARNINGTHRESHOLD: double = 32.0 {readOnly}

+ calculateWindChill() : double
+ refresh() : void
+ resetHighAndLowReadings() : void
+ WeatherAnalyzer(ui :WeatherMonitorInterface, weatherSource :WeatherGrabberInterface)
- writeLogEntry() : void

-ui

-currentWeather

-weatherSource

«interface»
WeatherMonitorInterface

+ *appendConditions(textToAppend :String) : void*
+ *displayWarning(message :String) : void*
+ *getZipCode() : String*

«interface»
WeatherConditionsInterface

+ *getConditions() : String*
+ *getTemperature() : int*
+ *getWindspeed() : int*

«interface»
WeatherGrabberInterface

+ *getConditions() : WeatherConditionsInterface*
+ *requestWeather() : void*
+ *setZip(zipCode :String) : void*

JFrame

gui::WeatherMonitorGUI

+ HEIGHT: int = 200 {readOnly}
- res: DisplayPanel
- serialVersionUID: long = 1L {readOnly}
+ WIDTH: int = 400 {readOnly}
- zp: ZipPanel

+ appendConditions(textToAppend :String) : void
- createContents() : void
+ displayWarning(message :String) : void
+ getZipCode() : String
+ main(args :String[]) : void
+ WeatherMonitorGUI()

WeatherState

- conditions: String
- temperature: int
- windspeed: int

+ getConditions() : String
+ getTemperature() : int
+ getWindspeed() : int
+ WeatherState(conditions:String, temperature :int, windspeed :int)

YahooWeatherGrabber

- temperature: int = 0
- weather: String = ""
- windspeed: int = 0
- zip: String

+ getConditions() : WeatherConditionsInterface
- giveMeTextBetween(s:String, before :String, after :String) : String
+ requestWeather() : void
+ setZip(tempZip :String) : void
+ YahooWeatherGrabber(tempZip :String)

mock

sut

mock

sd WeatherAnalyzer

Operating System

gui::WeatherMonitorGUI

main(String[])

WeatherMonitorGUI()

:WeatherMonitorGUI

YahooWeatherGrabber(String)

:YahooWeatherGrabber

WeatherAnalyzer(WeatherMonitorInterface, WeatherGrabberInterface)

:WeatherAnalyzer

refresh()

getZipCode() :String

setZip(String)

requestWeather()

getConditions() :WeatherConditionsInterface

«Interface»
:WeatherConditionsInterface

getConditions() :String

getTemperature() :Int

getTemperature() :Int

calculateWindChill() :double

getTemperature() :Int

getWindspeed() :Int

writeLogEntry()

appendConditions(String)

[windchill < Wind Chill Warning Threshold]:displayWarning(String)

jMock

- JMock is a library that supports test-driven development of Java code with mock objects.
- Mock objects help you design and test the interactions between the objects in your programs.
- The jMock library:
  - makes it quick and easy to define mock objects, so you don't break the rhythm of programming.
  - lets you precisely specify the interactions between your objects, reducing the brittleness of your tests.
  - works well with the auto completion and refactoring features of your IDE
  - plugs into your favorite test framework
  - is easy to extend.

www.jmock.org

Structuring our Test

- What is the class under test?
- Which classes need to be mocked?

Basic source code

```
import junit.framework.TestCase;

import org.jmock.Mockery;
import org.jmock.Expectations;

import edu.msoe.se2831.examples.WeatherAnalyzer.WeatherAnalyzer;
import edu.msoe.se2831.examples.WeatherAnalyzer.WeatherGrabberInterface;
import edu.msoe.se2831.examples.WeatherAnalyzer.WeatherMonitorInterface;
import edu.msoe.se2831.examples.WeatherAnalyzer.WeatherState;

public class WeatherBugAnalyzerTester extends TestCase {
    private WeatherAnalyzer classUnderTest;
    private WeatherGrabberInterface mockWeatherSource;
    private WeatherMonitorInterface mockUserInterface;
    private Mockery context;
```

Defining Mock Objects

- Mock objects revolve around execution scenarios
  - Derived from sequence diagrams

- Testing the high and low
  - How can we test the getHigh and getLow methods?

Lets start simple

Testing high and low

1. Read the high of an unread system?
    1. What should it be?
2. Read the low of an unread system
    1. What should the low be?
3. Simulate a temperature of 72F.
    1. High = 72, low = 72
4. Simulate a temperature of 73F
    1. High = 73, low = 72
5. Simulate a temperature of 80F
    1. High 80, low 72
6. Simulate a temperature of 65
    1. High 80, low 65

How will we test this system?

- Plan of Attack
  - Create a mock object which implements the interface for the two classes
  - Instantiate an instance of the class to be tested interfacing with the mock object.
  - Write the Junit tests for this

Lets build the project...