[illegible]

Mock Objects

Laugh at Make fun of...

Lecture Objectives:

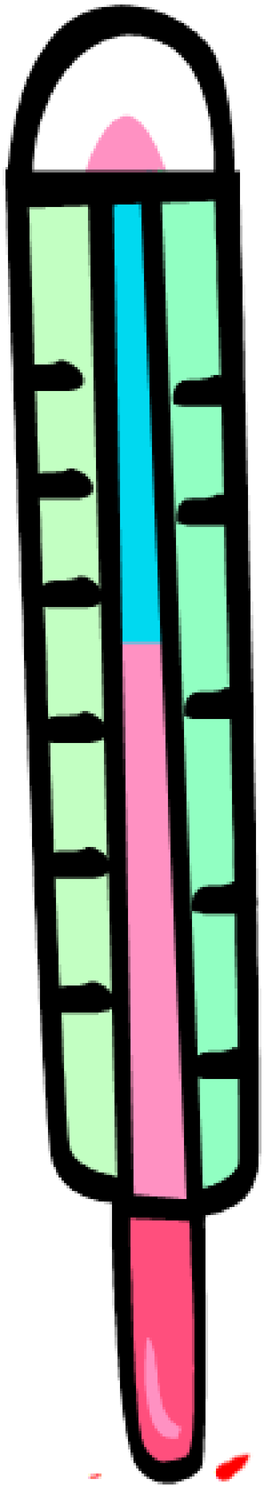
1) Explain the concept of testing stubs.
2) Define the term test double
3) Explain the relationship between Dummy Objects, Fake Objects, Stubs, and Mocks
4) Explain the concept of the Mock Object testing pattern.
5) List the three steps necessary for testing a system with mock objects.
6) Construct a mock object for a rudimentary system and use this object for the purpose of constructing unit tests.

"I have bad news....

The exams are !graded. They may be done for Wednesday (hopefully for Wed......)

The average grade for completed exams is 0%...

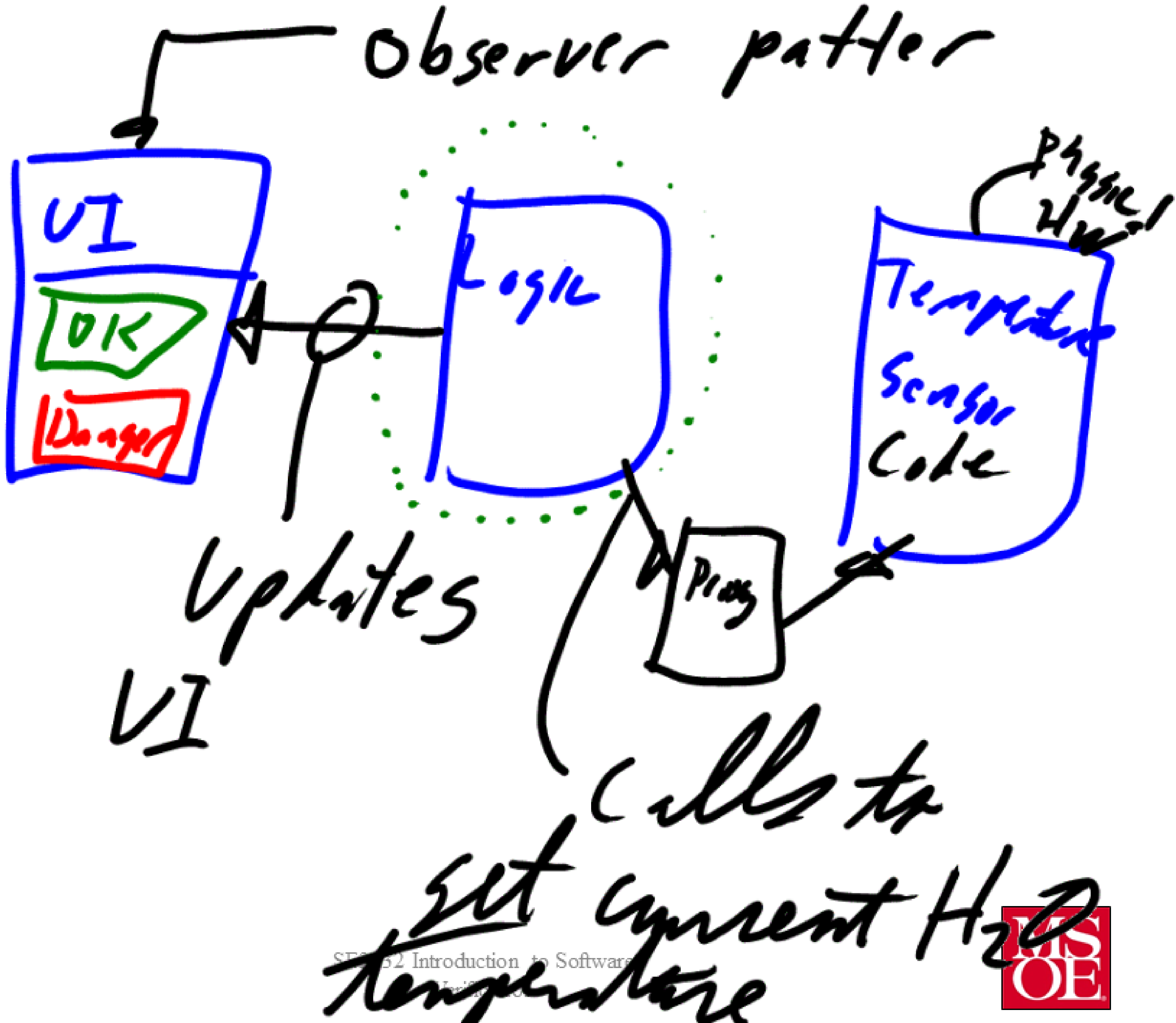
An Ice Warning System



< 32° ⇒
ok Ice
still frozen

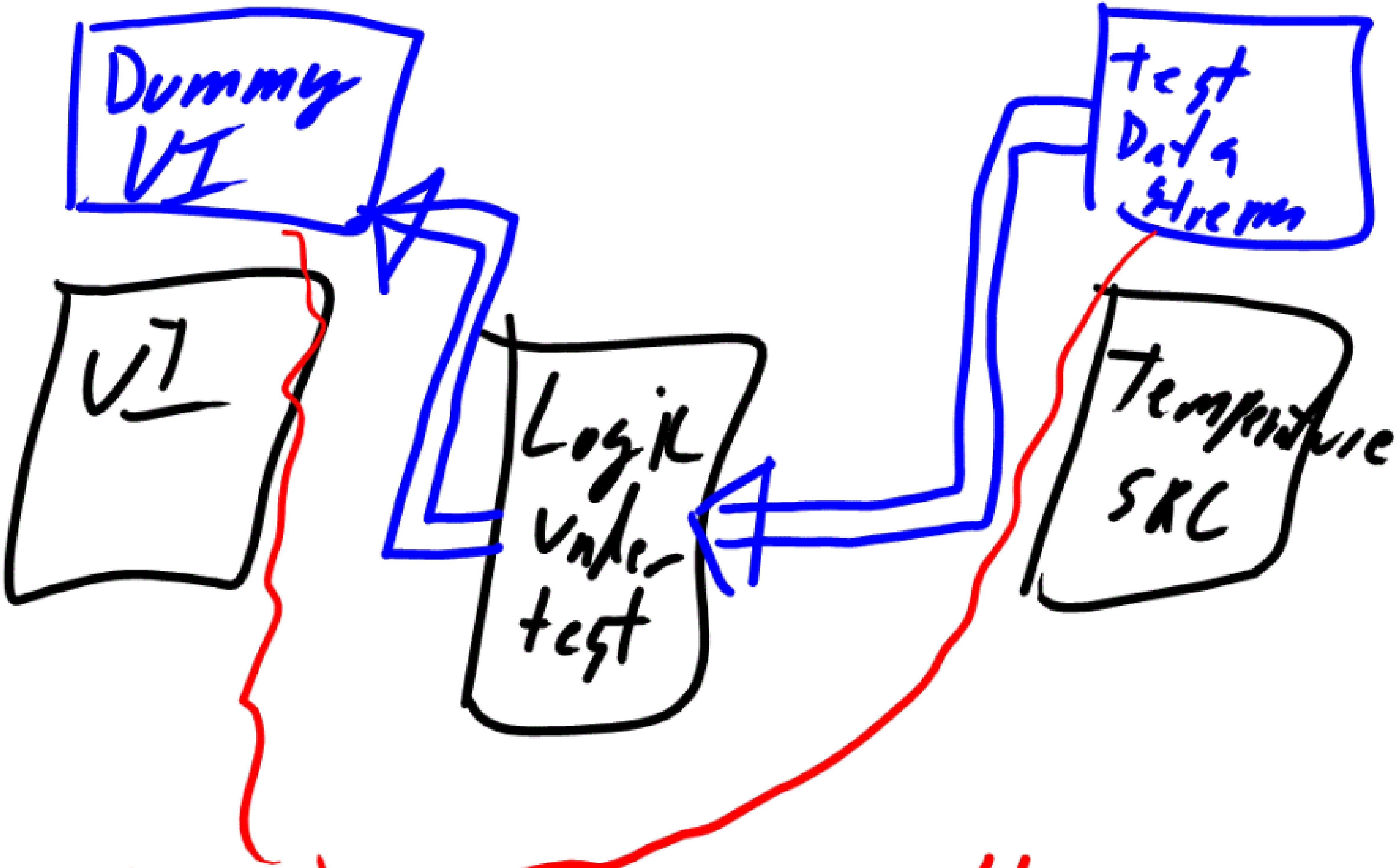
Temperature
above 32
Get off the
the ice. . . .

Basic System Design



$\Rightarrow$ Fake temperature stream

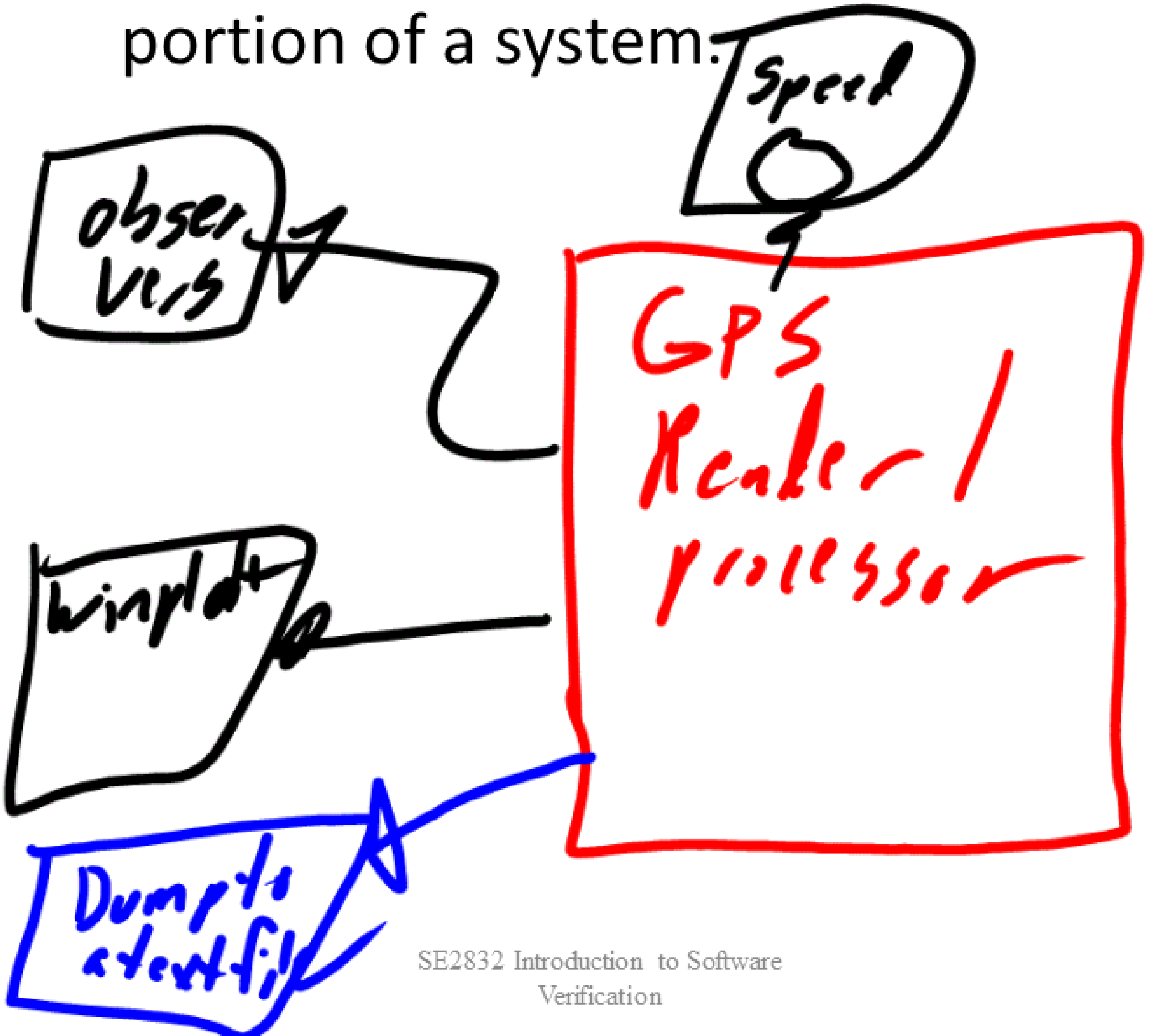
$\Rightarrow$ Record observation / Update calls to a "fake" VI.


Dummy VI
test Data Stream
VI
Logic under test
Temperature SRC

"Mock object"

What is a testing stub?

- A Stub is a dummy procedure, module or unit that stands in for an unfinished portion of a system.



Test double

"Spaceballs"

Test double

Stunt Doubles

©Gerard Meszaros

Test double:

generic term for anything that stands in for another object.

Test Double Types

- **Dummy objects**
  - Objects that are passed around but never actually used. Usually they are just used to fill parameter lists. — Need data but never do anything with it.
- **Fake objects**
  - Objects which actually have working implementations, but usually take some shortcut which makes them not suitable for production. $\Rightarrow$ "Simple"
- **Stubs**
  - provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.
- **Mocks Objects**
  - Objects pre-programmed with $\overbrace{\text{expectations}}^{\text{what should happen.}}$ which form a specification of the calls they are expected to receive.

Mock Objects

- Any dummy object that stands in for a real object that is not available, or is difficult to use in a test case
- (More rigid): A mock object must have the ability to set up expectations and provide a self-validation mechanism

More Formal

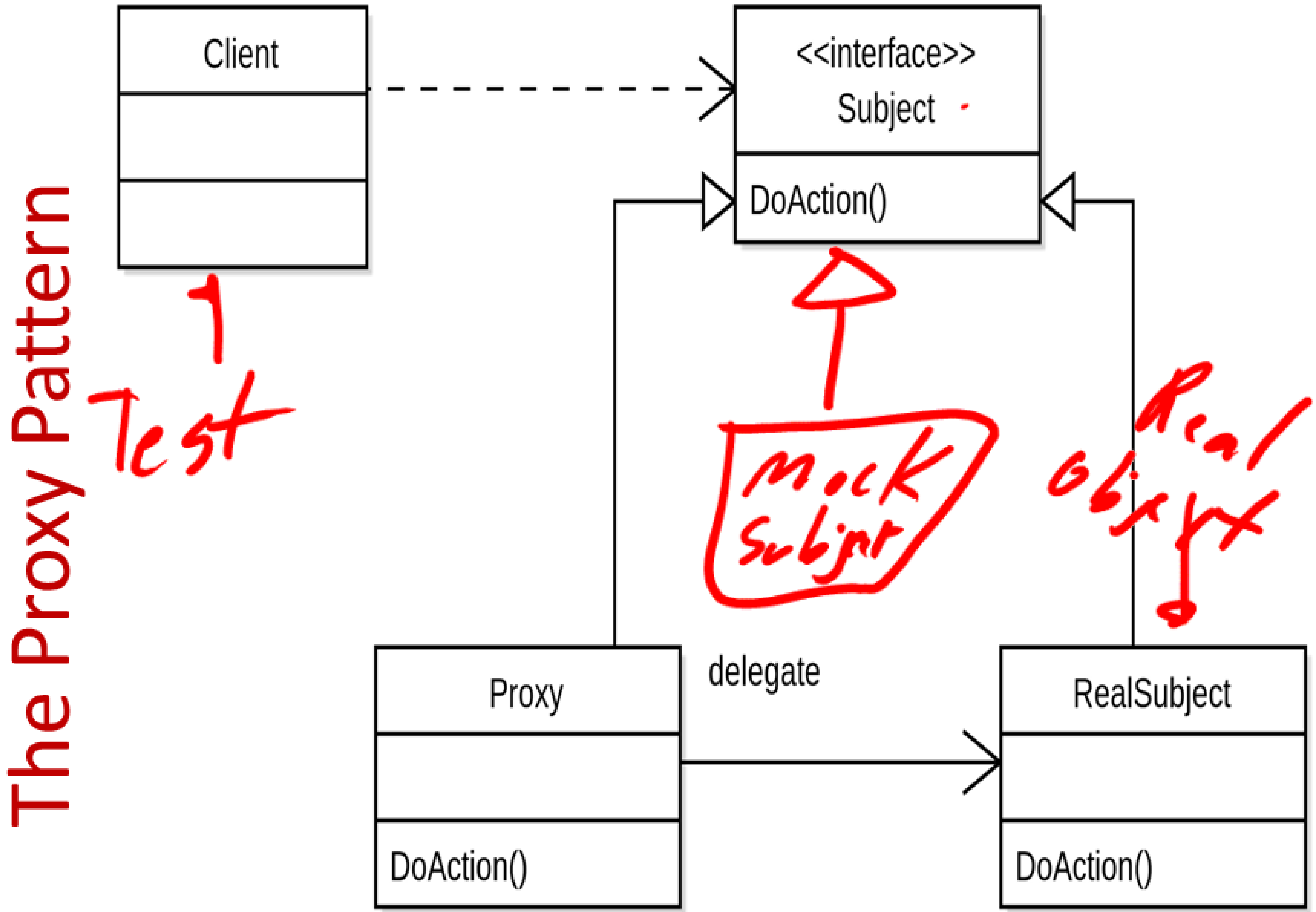
→ what should happen

what actually did happen.

How can Mock Objects Help Us?

- The Real object has nondeterministic behavior (it produces unpredictable results, like a stock market feed)
- The real object is difficult to set up
- The real object has behavior that is hard to trigger (for example, a network error)
- The real object is slow
- The real object has (or is) a user interface
- The test needs to ask the real object about how it was used (for example, confirm that a callback function actually was called)
- The real object does not yet exist (a common problem when interfacing with other teams or new hardware systems)

The Proxy Pattern



How to Use Mock Objects

- Create them by hand from scratch -
- Create them by hand, using the MockObjects library - tools
- Use MockMaker to generate the mock object prior to execution
- Use EasyMock to generate the mock object at runtime
- Use another Mock Object Tool

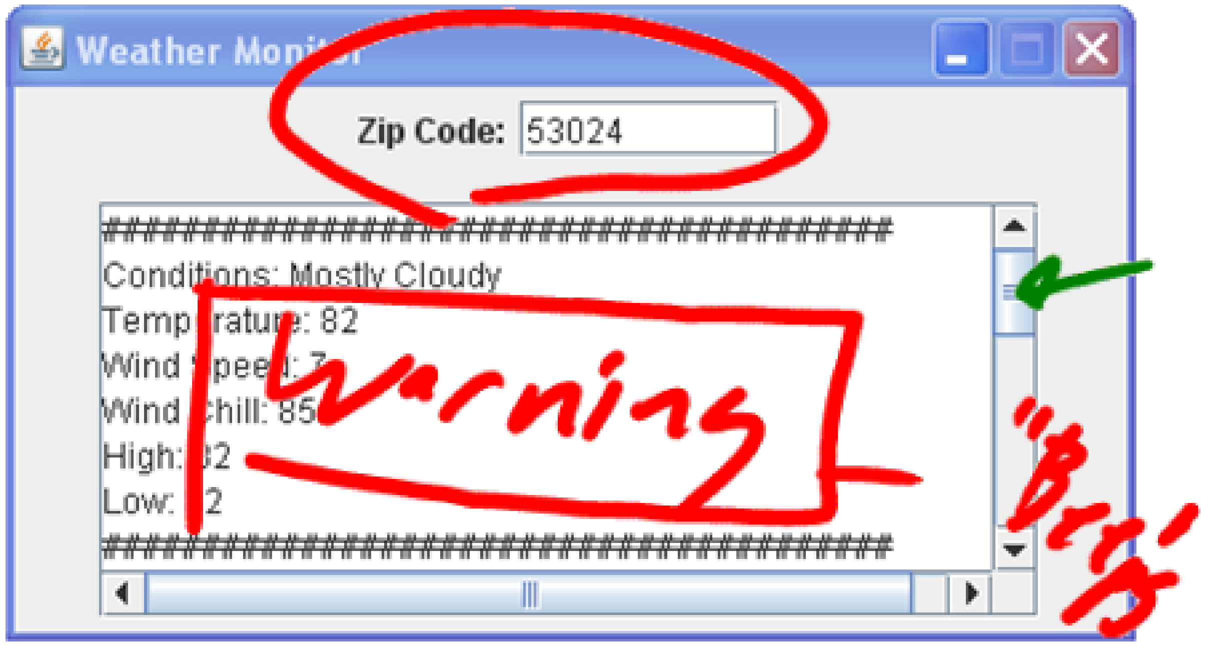
Tools

Steps necessary to test with Mock Objects

- Describe an object through the usage of an interface $\Rightarrow$ Allows us to create multiple implementing
- Implement the interface for production code $\Rightarrow$ Code we will test....
- Implement the interface in a mock object for testing $\Rightarrow$ Create a test version.

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

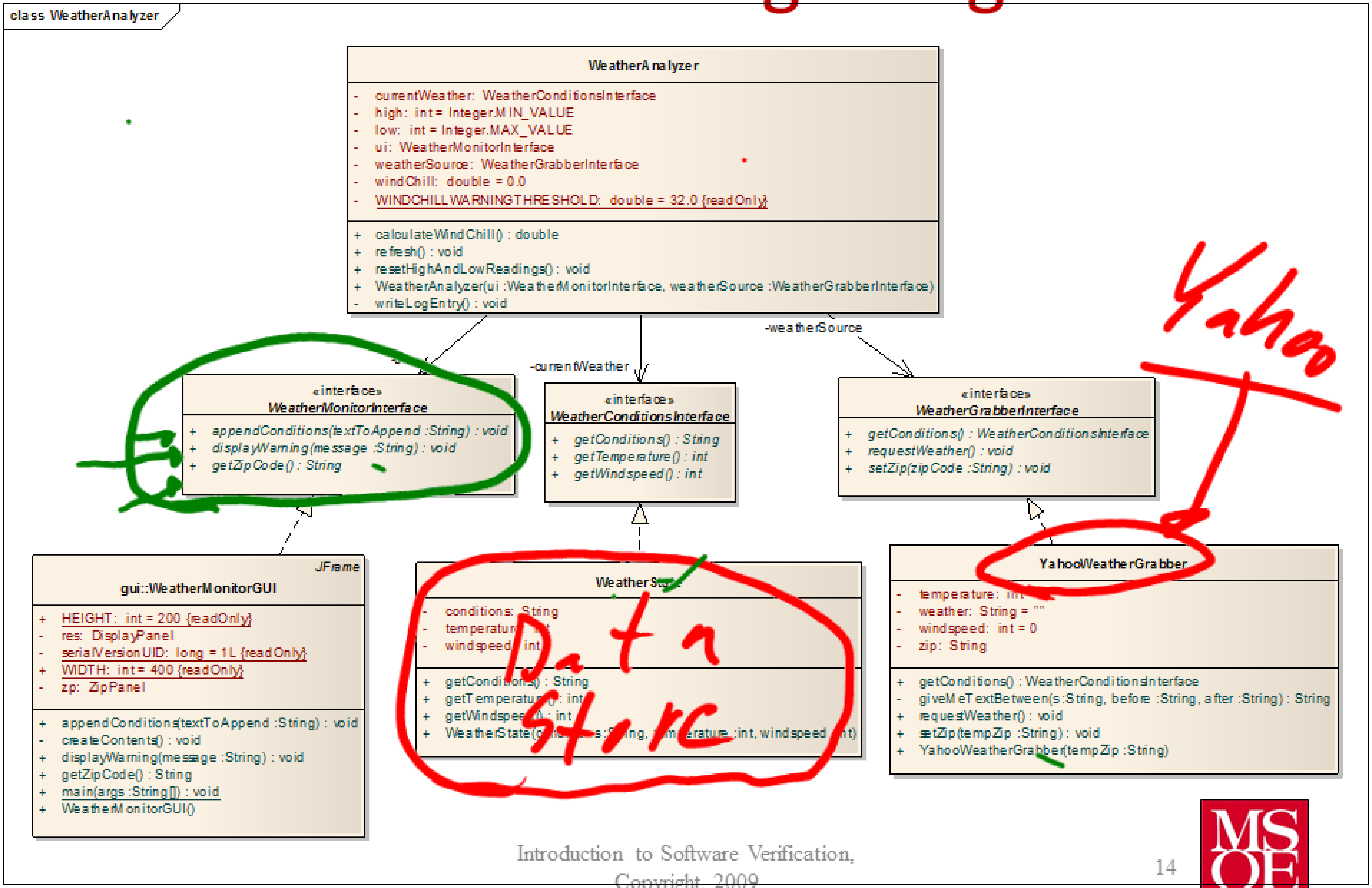


Zip Code: 53024

```
##########################################
Conditions: Mostly Cloudy
Temperature: 82
Wind Speed: 7
Wind Chill: 85
High: [illegible]2
Low: [illegible]2
##########################################
```

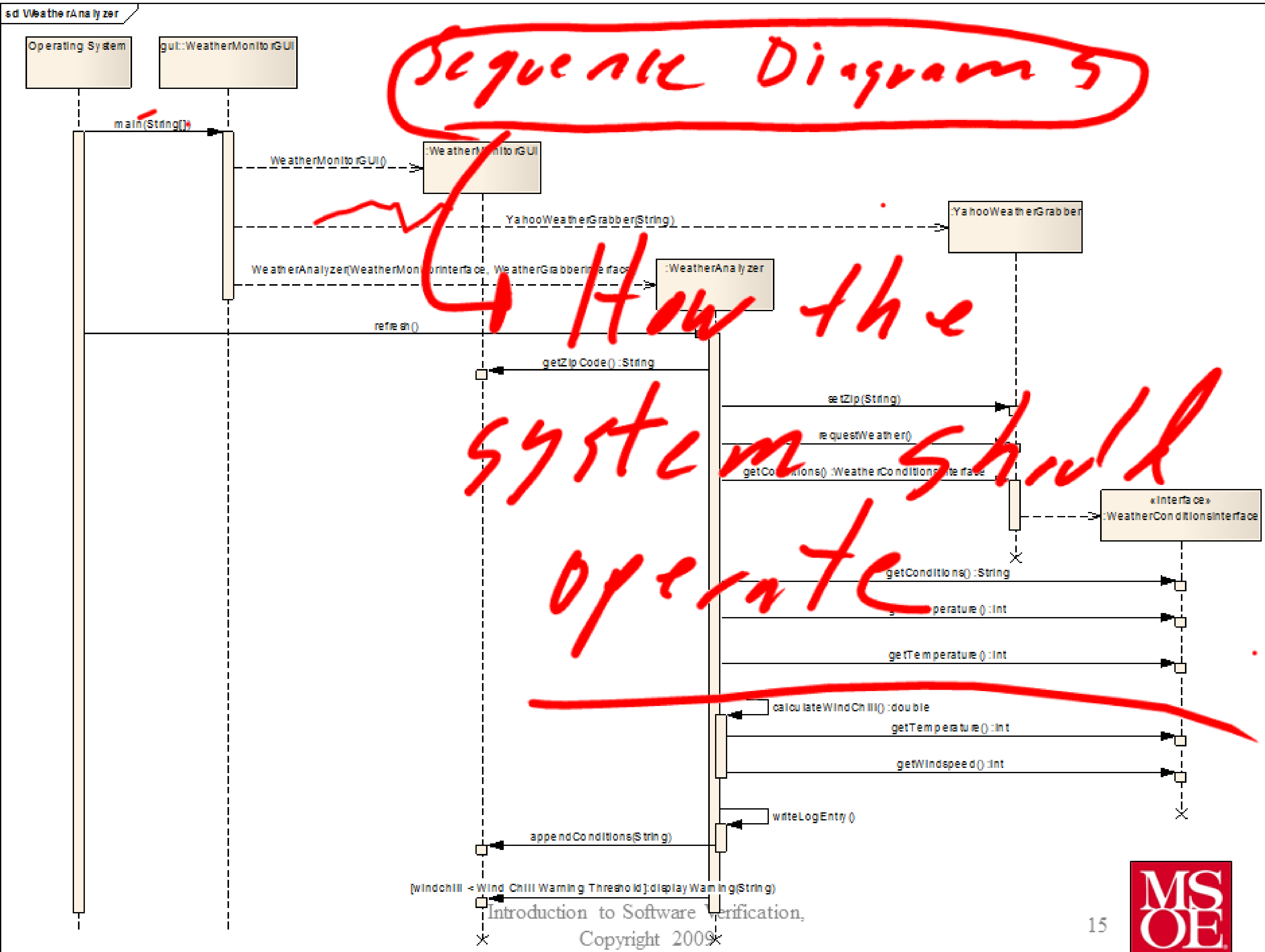
- Will display an alert message if the wind chill is dangerous (below 32)

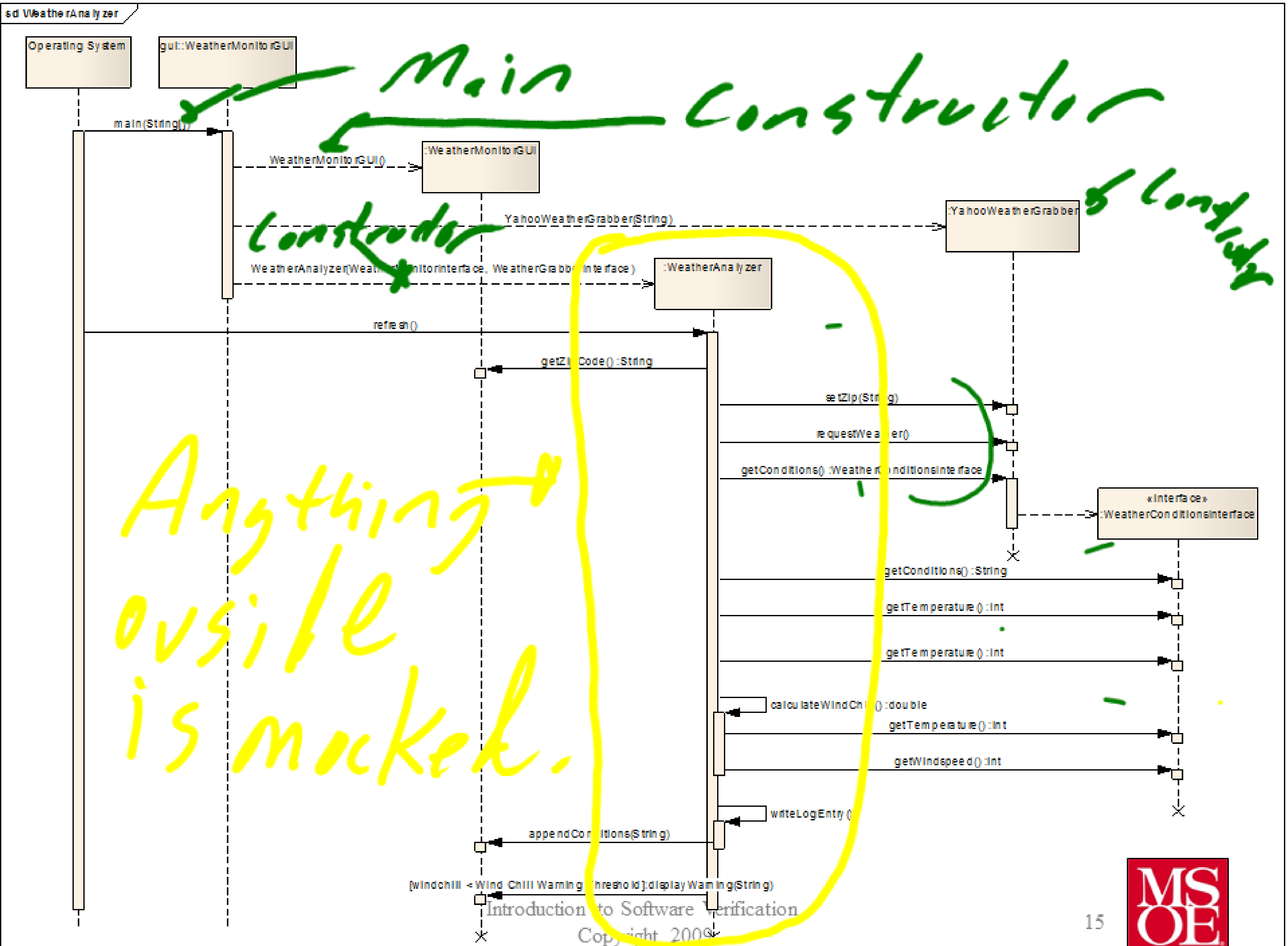
Weather Bug Design


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
-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
- 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)
Yahoo
Data store

Sequence Diagrams

How the system should operate




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)
Main
Constructor
Constructor
& Constructor
Anything ousile is mocked.

JMock

Structuring our Test

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

Basic source code

```
import junit.framework.TestCase;


import org.easymock.EasyMock;

public class StockQuoteAnalyzerTester extends TestCase
   {

   private WeatherAnalyzer classUnderTest;
   private WeatherGrabberInterface mockWeatherSource;
   private WeatherMonitorInterface mockUI;
```

Defining Mock Objects

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

Lets start simple

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

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