Project #2: Airplane Monitoring System

In this project you will simulate an airplane monitoring system.

Principals of Operation

The purpose of the airplane monitoring system is to track and log changes to various phenomena present in an airplane. Examples of such phenomena are wing vibration, engine temperature, and cabin pressure. Normally this log would be written to a shielded data recorder within the airplane (a “black box”); we will be writing the log to an instance of std::ostream.

Each phenomenon will be modeled as a voltage that varies with time. The voltage at a point in time represents the level of the phenomenon occurring at that moment. We will call this changing voltage over time a signal. Normally signals would vary unpredictably based on a complex set of contributing factors. For simulation purposes we will model three types of signals: constant, saw tooth, and sine wave. These signals will be hard coded to be predictable; a specific signal will always generate the same voltage for a given point in time.

The monitoring system tracks changes to phenomena by attaching a sensor to each phenomenon’s signal. A sensor applies a meaning to the voltage generated by the signal. For example, a sensor attached to the signal generated by vibrations in the wings of an airplane would interpret a voltage level of 42 to mean the wings are vibrating at 42 Hertz (cycles per second). To recap, a signal represents a changing voltage over time (42 volts at time = 3 seconds) while a sensor interprets that voltage to have a specific meaning by applying a name (wing vibration) and unit of measurement (Hertz) to the voltage.

Any number of sensors can be connected to the airplane monitoring system. At any point in time the monitoring system can be instructed to log the current voltage being observed by each sensor. This is called recording a data point.

Details

The following class diagram shows all classes that will be used to model the airplane monitoring system.

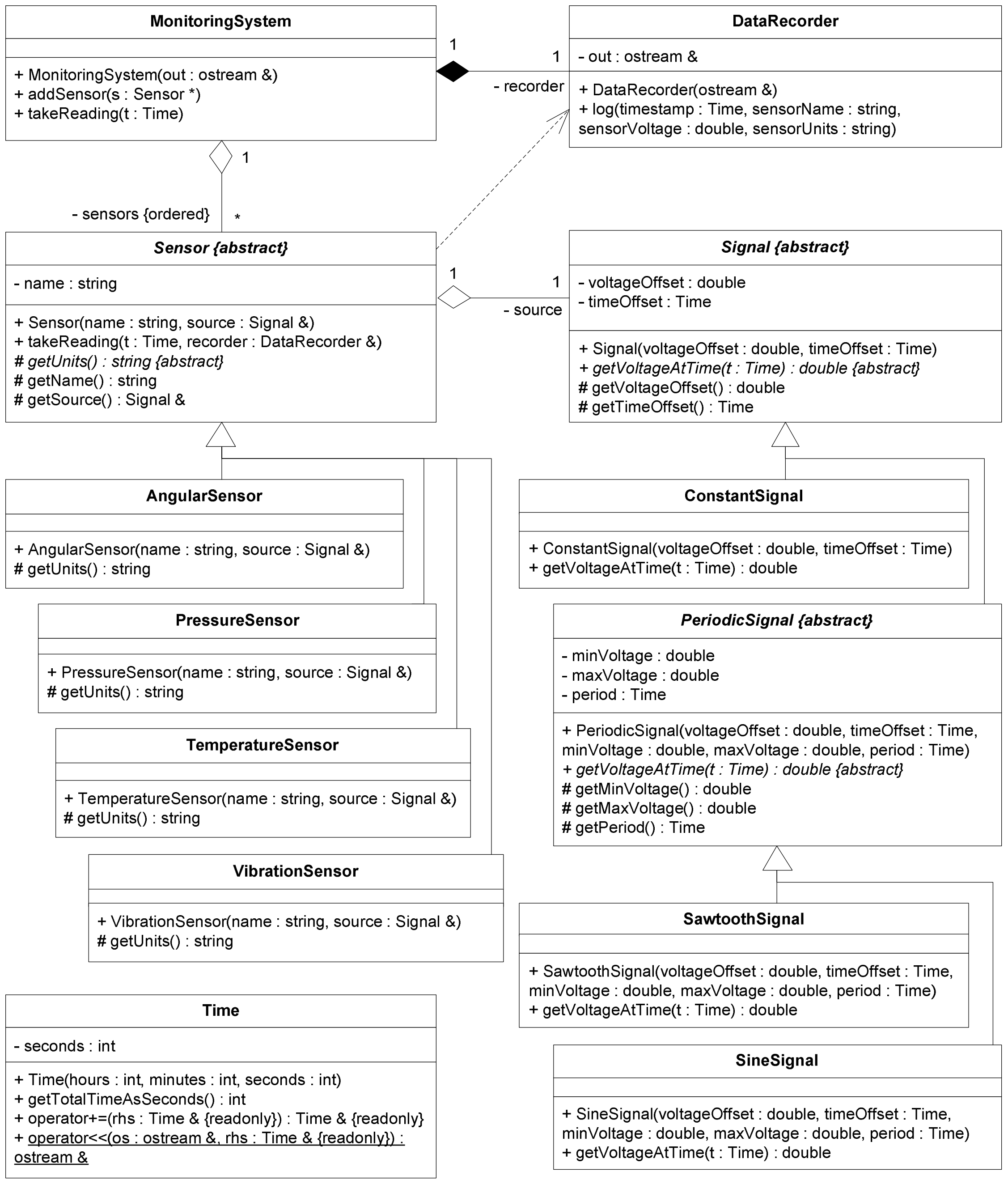
MonitoringSystem
+ MonitoringSystem(out : ostream &)
+ addSensor(s : Sensor *)
+ takeReading(t : Time)
1
1
- recorder
DataRecorder
- out : ostream &
+ DataRecorder(ostream &)
+ log(timestamp : Time, sensorName : string, sensorVoltage : double, sensorUnits : string)
1
- sensors {ordered}
*
Sensor {abstract}
- name : string
+ Sensor(name : string, source : Signal &)
+ takeReading(t : Time, recorder : DataRecorder &)
# getUnits() : string {abstract}
# getName() : string
# getSource() : Signal &
1
1
- source
Signal {abstract}
- voltageOffset : double
- timeOffset : Time
+ Signal(voltageOffset : double, timeOffset : Time)
+ getVoltageAtTime(t : Time) : double {abstract}
# getVoltageOffset() : double
# getTimeOffset() : Time
AngularSensor
+ AngularSensor(name : string, source : Signal &)
# getUnits() : string
PressureSensor
+ PressureSensor(name : string, source : Signal &)
# getUnits() : string
TemperatureSensor
+ TemperatureSensor(name : string, source : Signal &)
# getUnits() : string
VibrationSensor
+ VibrationSensor(name : string, source : Signal &)
# getUnits() : string
ConstantSignal
+ ConstantSignal(voltageOffset : double, timeOffset : Time)
+ getVoltageAtTime(t : Time) : double
PeriodicSignal {abstract}
- minVoltage : double
- maxVoltage : double
- period : Time
+ PeriodicSignal(voltageOffset : double, timeOffset : Time, minVoltage : double, maxVoltage : double, period : Time)
+ getVoltageAtTime(t : Time) : double {abstract}
# getMinVoltage() : double
# getMaxVoltage() : double
# getPeriod() : Time
SawtoothSignal
+ SawtoothSignal(voltageOffset : double, timeOffset : Time, minVoltage : double, maxVoltage : double, period : Time)
+ getVoltageAtTime(t : Time) : double
SineSignal
+ SineSignal(voltageOffset : double, timeOffset : Time, minVoltage : double, maxVoltage : double, period : Time)
+ getVoltageAtTime(t : Time) : double
Time
- seconds : int
+ Time(hours : int, minutes : int, seconds : int)
+ getTotalTimeAsSeconds() : int
+ operator+=(rhs : Time & {readonly}) : Time & {readonly}
+ operator<<(os : ostream &, rhs : Time & {readonly}) : ostream &

Figure 1. UML class diagram

Signals

A Signal represents a voltage that varies with time. Each Signal is of a specific type. The Signal's type determines the equation that will be used to produce a Voltage at a given time. Each type of signal is detailed below.

ConstantSignal

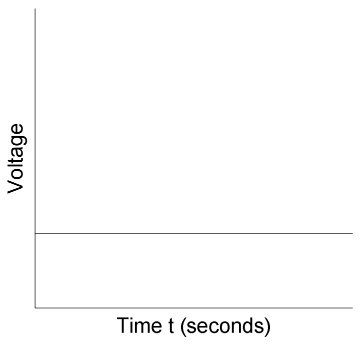

Figure 2. ConstantSignal voltage vs. time

A ConstantSignal is a Signal that never changes. The equation for the voltage of a ConstantSignal is:

```
V(t) = voltageOffset
```

SawtoothSignal

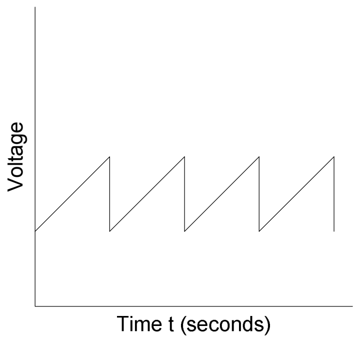

Figure 3. SawtoothSignal voltage vs. time

A SawtoothSignal is a Signal that varies between a minimum and maximum voltage, climbing from the minimum to the maximum voltage at a constant rate then falling immediately back the minimum voltage upon reaching the maximum voltage. The equation for the voltage of a SawtoothSignal is:

```
V(t) = voltageOffset + min
       + ((t + timeOffset) * (max - min) / period) % (max - min)
```

Note that floating point modulus (%) must be used (use the `fmod` c-function).

SineSignal

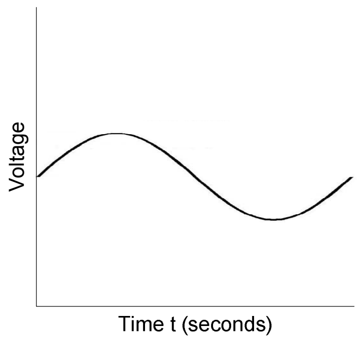

Figure 4. SineSignal voltage vs. time

A SineSignal is a Signal that varies between a minimum and maximum voltage according to the sine function. The equation for the voltage of a SineSignal is (letting 1 second = 2 pi radians):

```
V(t) = voltageOffset + min + (max - min) / 2
       + sine((t + timeOffset) * 2 * PI / period) * (max - min) / 2
```

Assume that PI has the following value:

```
PI = 3.14159265358979323846
```

Sensors

A Sensor interprets the raw voltage produced by a Signal as a named quantity of a specific unit of measure. Each type of Sensor is detailed below.

AngularSensor

An AngularSensor interprets Signal voltages as the following unit of measure:

```
“Radians”
```

PressureSensor

A PressureSensor interprets Signal voltages as the following unit of measure:

```
“Pounds per square inch (PSI)”
```

TemperatureSensor

A TemperatureSensor interprets Signal voltages as the following unit of measure:

```
“Degrees Celsius”
```

VibrationSensor

A VibrationSensor interprets Signal voltages as the following unit of measure:

```
“Hertz (Hz)”
```

Recording a Data Point

The following sequence diagram illustrates the objects and operations involved when the airplane monitoring system records a data point.

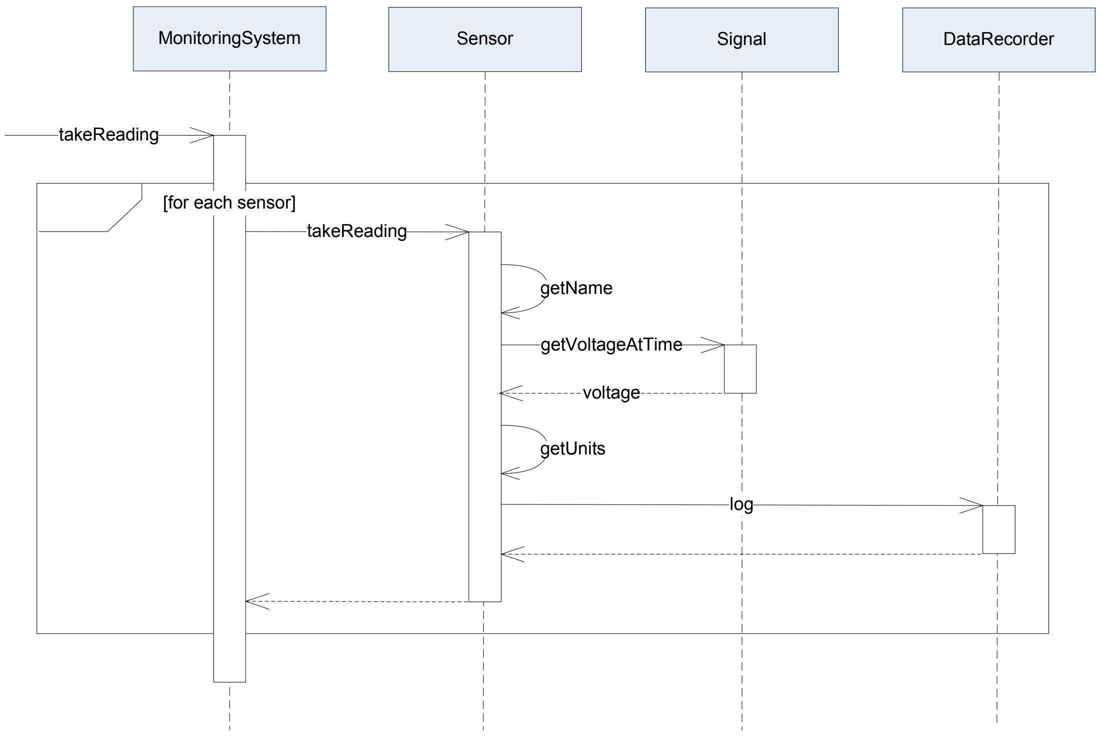

Figure 5. UML Sequence diagram for recording a data point

Project Files

Files you must implement

You must create and complete the implementation of the following files:

- AngularSensor.h
- ConstantSignal.h
- DataRecorder.h
- MonitoringSystem.h
- PeriodicSignal.h
- PressureSensor.h
- SawtoothSignal.h
- Sensor.h
- Signal.h
- SineSignal.h
- TemperatureSensor.h
- Time.h
- VibrationSensor.h

- AngularSensor.cpp
- ConstantSignal.cpp
- DataRecorder.cpp
- MonitoringSystem.cpp
- PeriodicSignal.cpp
- PressureSensor.cpp
- SawtoothSignal.cpp
- Sensor.cpp
- Signal.cpp
- SineSignal.cpp
- TemperatureSensor.cpp
- Time.cpp
- VibrationSensor.cpp

Files you must not change

These following files have already been implemented and must not be changed. I will use my own original version of these files when grading. **These files must not be modified:**

- Test Framework – *Generic test framework for running unit tests*
    - TestFramework.cpp
    - TestFramework.h
- Unit Tests – *These tests test the logic of the classes and functions you will be implementing*
    - main.cpp
    - UnitTest.cpp
    - UnitTest.h

Work Required

You will need to implement of all of the files listed under the “Files you must implement” section.

Grading

You will be graded based on how your solution performs against the unit tests and how few memory leaks your program exhibits.

You can earn a number of points on this project equal to the total number of unit tests present in UnitTest.cpp. You will lose 1 point for each unit test your program fails. You will lose 1 point for each unique memory leak your program exhibits. **If your program fails to compile you will receive 0 points.**

Turning In the Project

Submit a single archive file (e.g. .zip, .tar) containing only the files listed under the “Files you must implement” section. *I will not use any other files you send; I will be using my original versions of all other files.*

Note: You may submit your project early if you would like. I will let you know what grade your project would receive. You may then make corrections and resubmit. You may repeat this as many times as desired up until the due date.