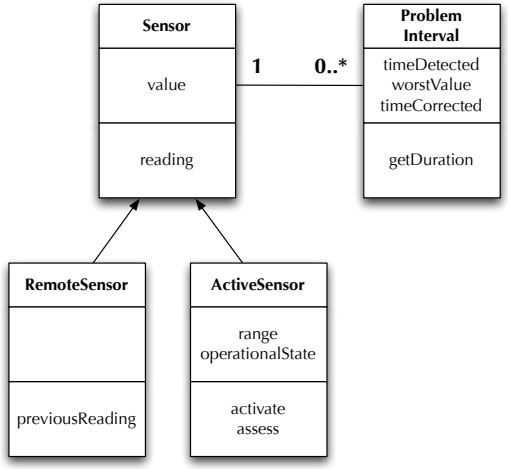
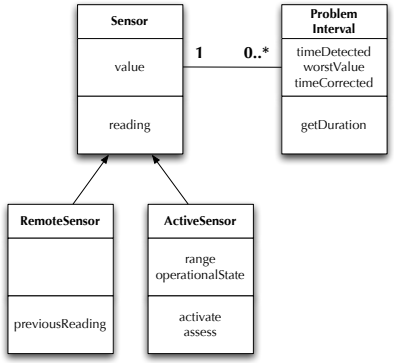
last week...



sensors with problem intervals, some clean-up



Now we would like to consider areas that contain groups of sensors as a unit. We will monitor a "zone" just as we monitor and assess sensors.

add zones to the class diagram

sensors with problem intervals, some clean-up

a common design pattern:

composite

move the relationship
up to the interface or
abstract class at the
root of the system

Dependency Inversion Principle

bad design

rigidity
fragility
immobility

1. **rigidity** -- It is hard to change a component, because each change affects (too many) other parts of the system.

2. **fragility** -- When you change a component, unexpected parts of the system break.

3. **immobility** -- It is hard to reuse a component in another application because it cannot be disentangled from the current application.

three basic principles

Single Responsibility Principle

Liskov's Substitution Principle

Interface Segregation Principle

simple truths about design and programming (not just OO)

Single Responsibility Principle

A class has a single responsibility:
it does it all, does it well, and does it only.
— Bertrand Meyer

Each responsibility should be a separate class, because each responsibility is an axis of change.

A class should have one, and only one, reason to change.

If a change to the business rules causes a class to change,
then a change to the database schema, GUI, report format,
or any other segment of the system
should not force that class to change.

Substitution Principle

Whenever possible, we should design a system in terms of **substitutable objects**.

Often see reference to Liskov's Substitution Principle (LSP).
Liskov's original statement was about subtypes in language.
It has been generalized into a design principle that is too broad.
I have added a qualifier.

Barbara Liskov .. PL researcher .. abstraction, encapsulation .. CLU .. Turing Award .. OOPSLA keynote
http://www.oopsla.org/oopsla2009/program/invited-speakers/215-oopsla-keynote-speaker-turing-award-lecture-reprise

Substitutable Objects

An object X is *substitutable* for an object Y if

- we can use X any place we use Y

and

- *the client code not know the difference*.

Example: An interface or an abstract class with implementing classes. Variables typed to the abstraction.

Interface Segregation Principle

No object should implement
an interface it doesn't need.

so...

Separate interfaces into
the smallest possible units.

... and design the system so that objects do not to rely on interfaces they don't need.

Example: TimedDoor

two trickier principles

Open-Closed Principle

Dependency Inversion Principle

These make sense but change how we tend to think about design.

They are more peculiar to software than other design disciplines, following from the abstract nature of our raw material.

Open-Closed Principle

Design modules that **never change**.

A change to one class should **not** change another class.

Instead:
- make it possible to create useful subclasses
- build relationships with abstractions and extensible classes

Examples:
- client/server -> abstract server
- GUI with shapes

violations of the OCP

public instance variables

global variables

RTTI
(or switching on an object's class)

... or **protected** instance variables, either!

"No module that depends upon a global variable [or a public IV] can be closed against any other module that might write to that variable. Any module that uses the variable in a way that the other modules don't expect, will break those other modules. It is too risky to have many modules be subject to the whim of one badly behaved one."
— Robert Martin (Uncle Bob)

Dependency Inversion Principle

High-level modules should **not** depend
upon low-level modules.

Abstractions should **not** depend
upon details.

a structural implication of the OCP and the LSP

... Both should depend upon abstractions.
... Details should depend upon abstractions.

frameworks

next week

10/30/09

11/03/09

11/05/09

Friday: project designs are due (or: iteration 2 is due)

Tuesday: discuss designs in class ... informal presentations

Thursday: midterm exam