CPSC 427a: Object-Oriented Programming

Michael J. Fischer

Lecture 12
October 19, 2010

Interacting Classes and UML (cont.)

Design Exercise: Family Datebook

Model-Viewer-Controller Paradigm

Demo: Stopwatch

Interacting Classes and UML (cont.)

Accessing `B` in `A`'s methods

Access patterns:

- parameter, local variable, or return has type `B`/`B&`/`B*`
- a method in `A` accesses `B`'s data members: `B::var` or `b.var`
- a method in `A` invokes `B`'s methods: `B::func()` or `b.func()`
- indirect: `c.b.func()`

If `A` knows `B` only through parameter or local variables, we also say that `A` **uses** `B`. The **use** relationship is generally considered to be a weak relationship.

"Law" of Consistency/Encapsulation

Relation of `B::var` or `b.var` in `A` is typically not recommended because it violates encapsulation and may lead to inconsistent state.

Why is the design below not desirable?

```
class SpeedDataCollection{
  ...
  // add a new data value
  public void addValue(int speed);

  // return average speed
  public double averageSoFar;
  ...
};
```

"Law" of Demeter

Chaining such as `c.b.func()` is typically not recommended as it increases coupling.

For example, assume class `A` has a data member `Dog* dog`. One way to ask the `dog` object to move is `dog->leg()->walk()`. But this is less desirable than calling `dog->walk()`.

In OO design, this is called the "Law" of Demeter, also called "Law" of Least Knowledge:

- "the method of a class should not depend on any way on the structure of any class, except the immediate (top-level) structure of its own class."

This principle has other names such as **Delegation** and **Do not talk to Strangers**.

"Law" of Demeter

Formally, the "Law" of Demeter for functions requires that a method `M` of an object `A` may only invoke the methods of the following kinds of objects:

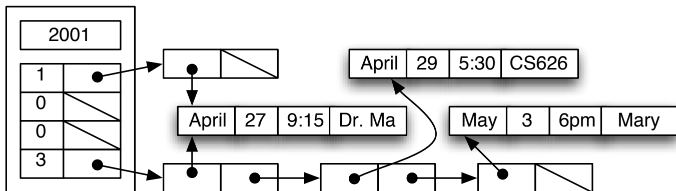
- `A` itself
- `M`'s parameters
- any objects created/instantiated within `M`
- `A`'s direct component objects
- a global variable, accessible by `A`, in the scope of `M`

One can consider layered architecture of many systems (e.g., the layered network architecture) as following this design guideline.

Design Exercise: Family Datebook

Design Exercise: FamilyDatebook

Requirement: design a datebook that can be shared by a family, where each family member can have a list of appointments, and an appointment may involve multiple family members.



What classes do we design and their relationships?

Model-View-Controller Paradigm

Model-Viewer-Controller design paradigm

Interactive systems evolve over time.

Model-Viewer-Controller is a design paradigm for addressing such systems.

- The model keeps the state of the evolving system.
- The viewer allows the state to be examined.
- The controller responds to external inputs and causes state changes.

Example: Airline reservation system

- Database keeps the state of reservations.
- Viewer show the empty seats.
- Controller is the ticket agent.

Demo: Stopwatch

Realtime measurements

`StopWatch` is a class for measuring realtime performance of code.

It emulates a stopwatch with 3 buttons: `reset`, `start`, and `stop`.

At any time, the watch displays the cumulative time that the stopwatch has been running.

(See demo.)