CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2001
Doc 4 Design Pattern Intro
Contents

References

Patterns for Classroom Education, Dana Anthony, pp. 391-406, ***Pattern Languages of Program Design 2***, Addison Wesley, 1996

Demo Prep: A Pattern Language for the Preparation of Software Demonstrations, Todd Coram, pp. 407-416, ***Pattern Languages of Program Design 2***, Addison Wesley, 1996

A Pattern Language, Christopher Alexander, 1977

Software Patterns, James Coplien, 1996, 2000, http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995

Reading

Design Patterns chapter 1.

Design Patterns

What is a Pattern?

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution"

Christopher Alexander on architecture patterns

"Patterns are not a complete design method; they capture important practices of existing methods and practices uncodified by conventional methods"

James Coplien

Examples of Patterns
A Place To Wait[1]

The process of waiting has inherent conflicts in it.

Waiting for doctor, airplane etc. requires spending time hanging around doing nothing

Can not enjoy the time since you do not know when you must leave

Classic "waiting room"
- Dreary little room
- People staring at each other
- Reading a few old magazines
- Offers no solution

Fundamental problem
- How to spend time "wholeheartedly" and
- Still be on hand when doctor, airplane etc arrive

Fuse the waiting with other activity that keeps them in earshot
- Playground beside Pediatrics Clinic
- Horseshoe pit next to terrace where people waited

Allow the person to become still meditative
- A window seat that looks down on a street
- A protected seat in a garden
- A dark place and a glass of beer
- A private seat by a fish tank

[1] Alexander 1977, pp. 707-711

A Place To Wait

Therefore:

"In places where people end up waiting create a situation which makes the waiting positive. Fuse the waiting with some other activity - newspaper, coffee, pool tables, horseshoes; something which draws people in who are not simple waiting. And also the opposite: make a place which can draw a person waiting into a reverie; quiet; a positive silence"

Chicken And Egg[2]

Problem

Two concepts are each a prerequisite of the other

To understand A one must understand B

To understand B one must understand A

A "chicken and egg" situation

Constraints and Forces

First explain A then B

Everyone would be confused by the end

Simplify each concept to the point of incorrectness to explain the other one

People don't like being lied to

Solution

Explain A & B correctly by superficially

Iterate your explanations with more detail each iteration

[2] Anthony 1996

A Pattern Language for the Preparation of Software Demonstrations[3]

The patterns:

- Element Identification
- Catalytic Scenarios
- Mutable Code
- Prototyping Languages
- Lightweight User Interfaces
- Judicious Fireworks
- Archive Scenarios

[3] Coram, 1996

Element Identification Pattern
Problem

Selecting the right features to demo is a critical part of keeping the customer's confidence

Context

Have requirements

Working on demo to easy customers doubts about committing to or continuing with the software project

Forces

Need to demonstrate your ability to deliver "things that work"

Need to show some level of functionality

Customer wants to see the product's face - the GUI

If customer is not happy with the demo, they are not likely to like the end product

Demos build confidence and create anticipation

Element Identification Pattern Solution

Identify key areas that concern the customer

- Talk to the customer
- Listen carefully

Stay away from excessive animations or other visual embellishments

- Unless the product is a game, the product is to help the customer get some work done not to entertain people

The product's face can be shown through Lightweight User Interface (pattern 5)

Functionality can be addressed by Prototyping Languages (pattern 4)

Catalytic Scenarios
Problem

The customer has specified what they think they want

You don't want to build the wrong thing

Context

Starting a project to develop software based on requirements and specification that have already been agreed on

Forces

Customer may not really know what they want

Requirements may not accurately reflect customer's requirements

Requirements may be ambiguous

Customer expects to be given vision of the finished product

Demos consume developer's resources

Solution

Use demonstrable scenarios as a catalyst to open a dialogue between you and the customer

If the specs are ambiguous develop alternative scenarios

Do not demonstrate capabilities that will be hard to incorporated into your design

If you do not want to change the spec make sure the demo scenarios follow the spec

Keep demo scenarios simple and short

Mutable Code
Problem

How much code should you write for the demo?

Context

You have identified your Catalytic Scenarios and are evaluating the amount of effort required to develop them

Forces

Some demo code
- Can not be used in the end product
- Should not be used in the end product

Development time for demo impacts product development

Customer does not like to pay for developing something twice

Solution

Build modifiable code

Use tools that support a high level of abstraction
- GUI builders
- Scripting languages

Write as little code as possible for the demo
- Use as much real code as you can

If you build screens then use Lightweight User Interfaces

Prototyping Languages (pattern 4) discusses integrating demo code into end product

Benefits of Software Patterns

By providing domain expertise patterns

- Reduce time to find solutions
- Avoid problems from inexpert design decisions

Patterns reduce time to design applications

- Patterns are design chunks larger than objects

Patterns reduce the time needed to understand a design

Common Forms For Writing Design Patterns

Alexander

- Originated pattern literature

GOF (Gang of Four)

- Style used in Design Patterns text

Portland Form

- Form used in on-line Portland Pattern Repository
- http://c2.com/cgi/wiki?PortlandPatternRepository

Coplien

Design Principle 1

Program to an interface, not an implementation

Use abstract classes (and/or interfaces in Java) to define common interfaces for a set of classes

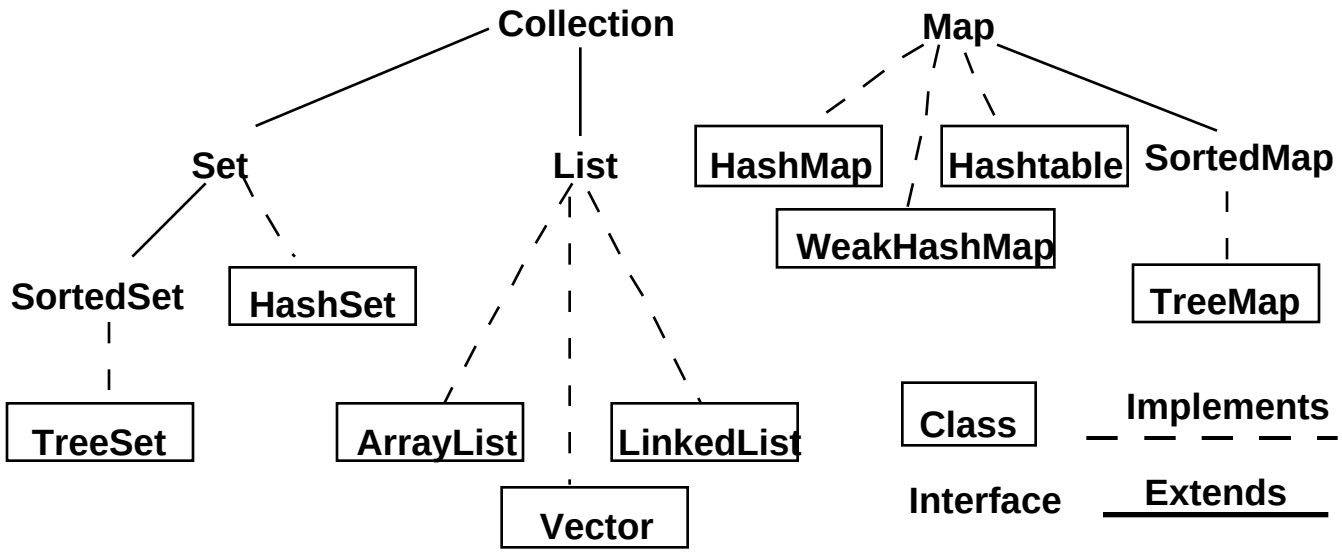
Declare variables to be instances of the abstract class not instances of particular classes

Benefits of programming to an interface

Client classes/objects remain unaware of the classes of objects they use, as long as the objects adhere to the interface the client expects

Client classes/objects remain unaware of the classes that implement these objects. Clients only know about the abstract classes (or interfaces) that define the interface.

Programming to an Interface
Java Collections



Collection students = new XXX;
students.add(aStudent);

students can be any collection type

We can change our mind on what type to use

Design Principle 2

Favor object composition over class inheritance

Composition

- Allows behavior changes at run time
- Helps keep classes encapsulated and focused on one task
- Reduce implementation dependencies

Inheritance

```
class A {
   Foo x
   public int complexOperation() { blah }
}

class B extends A {
   public void bar() { blah}
}
```

Composition

```
class B {
   A myA;
   public int complexOperation() {
      return myA.complexOperation()
   }

   public void bar() { blah}
}
```

Parameterized Types

Generics in Ada or Eiffel
Templates in C++

Allows you to make a type as a parameter to a method or class

```
template <class TypeX>
TypeX min(  TypeX a, Type b )
   {
   return a < b ? a  :  b;
   }
```

Parameterized types give a third way to compose behavior in an object-oriented system

Designing for Change

Some common design problems that GoF patterns that address

- Creating an object by specifying a class explicitly

  Abstract factory, Factory Method, Prototype

- Dependence on specific operations

  Chain of Responsibility, Command

- Dependence on hardware and software platforms

  Abstract factory, Bridge

- Dependence on object representations or implementations

  Abstract factory, Bridge, Memento, Proxy

- Algorithmic dependencies

  Builder, Iterator, Strategy, Template Method, Visitor

- Tight Coupling

  Abstract factory, Bridge, Chain of Responsibility, Command, Facade, Mediator, Observer

- Extending functionality by subclassing

  Bridge, Chain of Responsibility, Composite, Decorator, Observer, Strategy

- Inability to alter classes conveniently

  Adapter, Decorator, Visitor

Exercises

1. Select one of your old projects to study this semester. Start by looking for examples of coupling and cohesion in the project. Can you find examples of each type we studied in doc 2 & 3? Later we will use the project to look for patterns and places to apply patterns.

2. Students at SDSU get to wait a lot. In particular students get to wait for professors during office hours. In about a year the Computer Science department will move to a different building. With A Place to Wait pattern in mind, what you like to see done in the new Computer Science building to make waiting for professors less annoying.

3. Design patterns are used to record domain expert knowledge. Everyone is an expert at something. As students you may be an expert at studying, running study groups, crashing courses, finding parking spaces on campus, getting around university rules, finding an open terminal on campus, managing time between work-school-family, keeping INS happy, managing teams for class projects, hiding problems in class projects from professors, etc. Many of you also work, so there is another source of expertise. Select one area you are good at and write a pattern about it. One learns a lot about patterns when you write one.