

Lecture 12 Patterns

Lecturer: Tommy Yuan

Fall 2008 - Year 2

Object-Oriented Methods

Outline

- ◆ Patterns and Frameworks
- ◆ Pattern Template
- ◆ Patter Catalogue
 - Singleton
 - Façade
 - Observer
 - Composite
 - State
- ◆ Antipatterns

2



The idea of patterns is not new: architects, engineers, artists, musicians, ... have been using them for years...

3

Patterns in building architecture (Christopher Alexander)

- ◆ 'Entrance transition'
 - You never walk directly into a living room or bedroom from the outside
- ◆ 'Intimacy gradient' - degree of privateness
 - Bedrooms are to the rear and are the last rooms to enter
- ◆ 'Light on two sides of every room'
 - Rooms with no windows don't ever seem right
- ◆ There are even patterns to describe seating layout in a room.

4

What is a pattern?

- ◆ Successful designs do exist which exhibit recurring class/object structures which can be described using patterns.
- ◆ An object-oriented pattern is an abstraction of a small grouping of classes that is likely to be helpful over and over again in object-oriented software development.
- ◆ Applied to software design since early 90's. Now used in analysis, design and coding

5

What a pattern is not...

- ◆ Patterns are not actual designs
 - They must be instantiated.
- ◆ Patterns are not frameworks
 - A framework partially codifies a design for solving a family of problems in a specific domain.
 - The user typically customises a framework by defining subclasses.
 - Frameworks are larger and more concrete than patterns.
- ◆ Perhaps the best frameworks incorporate the best patterns.



The AWT is a GUI framework.

6

Levels of reuse

Application architecture level

Framework

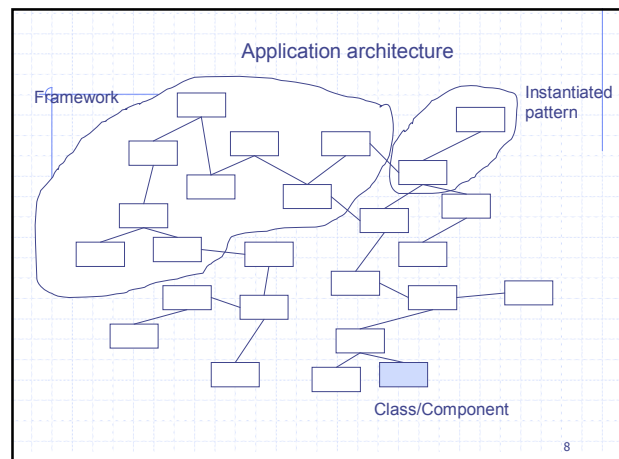
- frameworks are at a macro-architecture level

Pattern

- instantiated patterns are at a micro-architecture level

Class

7



8

Why use patterns?

- ◆ Saves analysis and design time
- ◆ Solution is thought to be good
- ◆ Easier to communicate with higher level of abstraction
 - a design language beyond technology
- ◆ But it is not always clear when a pattern is applicable
 - requires expertise in domain and patterns

9

Documenting with pattern templates

Recommended text book Ch 15.3.1 and 15.3.2

Name

- A pattern should be given a meaningful name that reflects the knowledge contained in the pattern. e.g. Singleton

Problem

- The problem the pattern is addressing should be described. e.g. "How can a class be constructed that has only one instance and can be accessed globally within an application?"

Context

- The circumstances or preconditions under which the pattern can occur.

Forces

- The forces embodied in the pattern are the constraints and issues that must be addressed by the solution.

10

Documenting...

Solution

- A description of the static and dynamic relationships between elements of the pattern.

Consequences

- good and bad implications of using the solution

Java API usage

- if there is an example in the Java API, it's given

Code example

- sample implementation

Related patterns

- list of related patterns

11

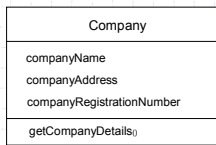
Pattern Categories

- ◆ Catalogue of 23 design patterns presented by Gamma et al. (1995), known as Gang of Four (GOF) Patterns
 - *Creational design patterns*
 - ◆ describe techniques to instantiate objects
 - *Structural design patterns*
 - ◆ describe ways of organising classes and objects into larger structures
 - *Behavioural design patterns*
 - ◆ assign responsibilities to objects
- ◆ Typically address issues concerning **changeability**

12

Creational Patterns: Singleton

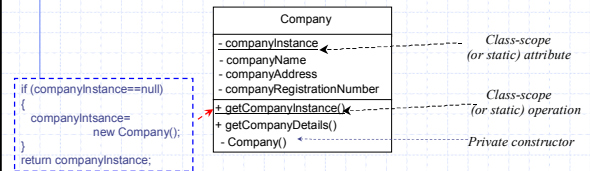
- ◆ How does one ensure that only one instance of the company class is created?



13

Singleton Pattern

- ◆ Solution – restrict access to the constructor!

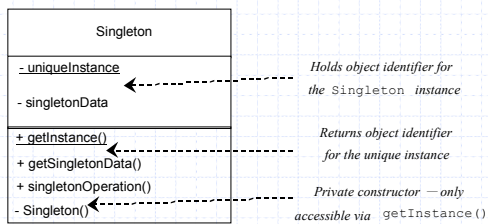


The use of class-scope operations allows global access

14

Singleton Pattern

General form of Singleton pattern



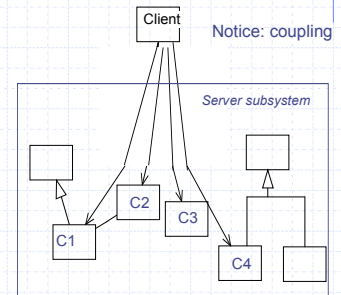
15

Structural Patterns - Façade

Recommended textbook Ch 20.5

Context

- ◆ Client's relationship with sub-system is too complex
- ◆ Client does not wish to deal with all the intricacies of the subsystem



16

Façade Pattern

“The Façade pattern simplifies access to a related set of objects by providing one object that all objects outside the set use to communicate with the set.”

Mark Grand.

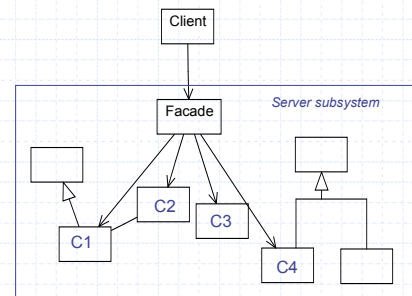
Façade is a word usually used to describe the front face of a building.



17

Façade Pattern

- ◆ Solution



- ◆ a pattern, not a design

18

Façade Pattern

Consequences

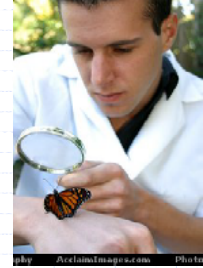
- ◆ easier to use sub-system
- ◆ clients don't need to know about classes behind the façade
 - reduced coupling
- ◆ can change the implementation of the classes behind the façade
- ◆ clients not prevented from using subsystem classes if they really need to
 - but best avoided to minimise coupling

19

Behavioural Patterns -Observer

Recommended textbook Ch 12.5.3

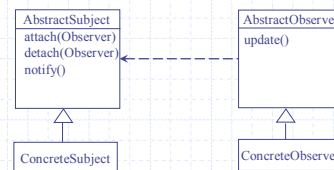
- ◆ Typically used in the Model-View-Controller (MVC) Framework.
- ◆ **Intent:** define a one-to-many dependency between objects so that when one object (subject) change state all its dependents (observers) are notified and updated.



Observer Pattern

- ◆ Observers
 - Defines an updating interface for objects that should be notified of changes in a subject
 - 'knows' about the subject, so they can register with the subject, and query the Subject to determine what has changed.
- ◆ Subject
 - One subject may be observed by many observers
 - Knows very little about the observers, other than how and when to notify them.
 - When the subject's state changes, it broadcasts no info. - just a notification (update message) to each of the Observers.

Observer Pattern



- ◆ Java package provide *Observable* class and the *Observer* interface which make up the pattern

Observer Pattern

GOF 1995 book pp.293-299

Consequences

- ◆ Reduce coupling between Subject and Observer
- ◆ Support for broadcast communication
- ◆ Needs to be used carefully - a simple change in the subject could trigger an enormous amount of messaging traffic between that subject and a list of registered observers.
- ◆ Observers need to work hard to find out the changes

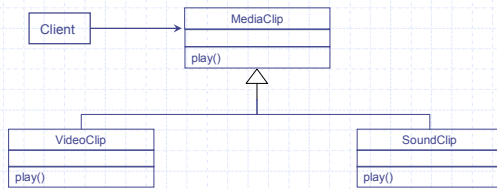
Structural Patterns: Composite

- ◆ Offer effective ways of using object-oriented constructs such as inheritance, aggregation and composition to satisfy particular requirements
- ◆ **Context:**
 - ◆ You want to represent part-whole hierarchies of objects
 - ◆ You want client to be able to ignore the differences between compositions of objects and individual objects.

24

Composite Pattern

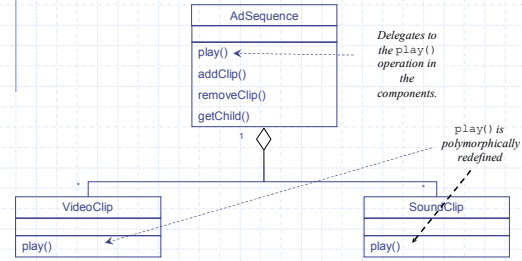
- ◆ How can we present the same interface for a media clip whether it is composite or not?



25

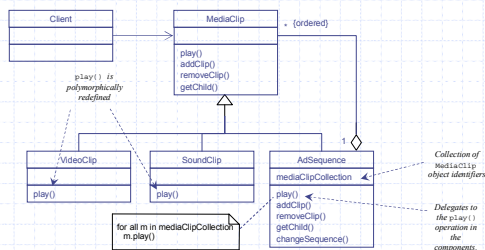
Composite Pattern

How can we incorporate composite structures?



26

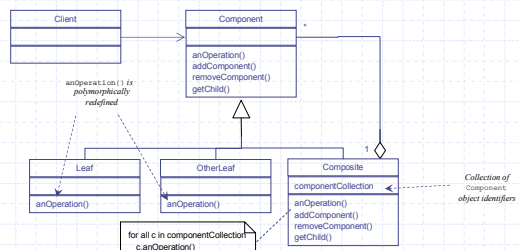
Composite Pattern



27

Composite Pattern

General Form



28

Composite Pattern

- ◆ Consequences
 - Makes the client simple
 - Makes it easier to add new kinds of components
 - But harder to restrict the components of a composite

29

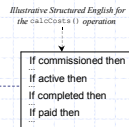
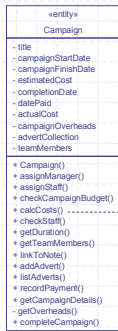
Behavioural Patterns: State

- ◆ Consider the class Campaign
- ◆ It has four states – Commissioned, Active, Completed and Paid
- ◆ A Campaign object has different behaviour depending upon which state it occupies
- ◆ Operations have case statements giving this alternative behaviour
- ◆ The class factored into separate components – one for each of its states

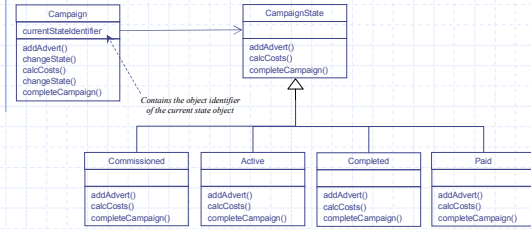
30

State Pattern

Campaign class: could have state pattern applied

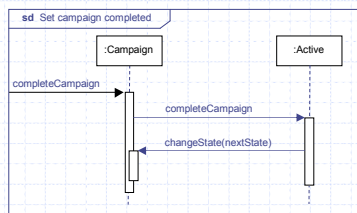


Behavioural Patterns: State

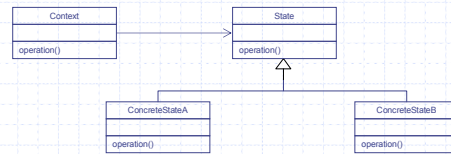


State pattern applied to the class Campaign

State Pattern: Sequence Diagram



General form of State Pattern



General form of State Pattern

Consequences

- Localise state-specific behaviour and partitions behaviour for different states
 - ◆ Remove large procedural and conditional statements
- State objects makes state transit explicit
- Introduce more classes

Development Antipattern: the Blob

www.antipatterns.com



-- "Negative solutions"

Symptoms

- ◆ An object with too many attributes and too many operations and other objects holding only data.

Solution

- ◆ refactor
 - reallocate behaviour away from the Blob

Development antipattern: Cut and Paste Programming

www.antipatterns.com

Symptoms

- ◆ Maintenance problems caused by copying source statements to reuse code.

Solution

- ◆ Black box re-use.

37

Reading

- ◆ Text Chapter 15
- ◆ Text Chapter 20.5 for Façade

38