UML & Strategy Pattern & Adapter Pattern & Facade Pattern

CS 247

University of Waterloo

cs247@student.cs.uwaterloo.ca

June 13, 2016
Overview

1. Unified Modelling Language
   - Composition vs Aggregation
   - Creating UML Class Diagrams

2. Strategy Pattern

3. Adapter Pattern

4. Facade Pattern
UML Tools

Microsoft Visio
UMLet
Violet UML
Draw.io
OmniGraffle (Mac)
Gliffy
UML Basics

- private
- public
- protected
- inheritance
- static
- pure virtual method and abstract class
However, composition and aggregation are two ways of modelling a “whole-part” relationship between two objects and it’s not always easy to tell which relationship to use.
Composition (denoted by a *black*, or filled in diamond), is a stronger relationship than Aggregation.
Composition implies that the container is *responsible* for the contained objects - they belong to a single container, and cannot exist without one. Also known as "owns-a" relationship
Composition

Typically, A owns a B if

- B has no existence outside A
- If A is copied, B is copied (deep)
- If A is destroyed, B is destroyed
Aggregation (denoted by a white, or not-filled-in diamond), is a weaker relationship than Composition. Aggregation implies that the container is not responsible for the contained objects - they can belong to multiple containers, and can exist on their own (as independent objects). Also known as ”has-a” relationship.
Aggregation

Typically, A has a B if

- B has an existence independent of A
- If A is copied, B is not (shallow)
- If A is destroyed, B is not
Create a UML class diagram representing a university. Things to include:

- A University has one (or many) Faculties.
- A Faculty offers several different courses.
- Students can be enrolled in courses.
- Some Students are Graduate Students.
- Some Graduate Students are TAs for some courses.
- Courses are taught by a Course Team.
- a Course Team consists of some number of Instructors, ISAs, TAs, and an ISC.
Goal:

- Strategy pattern gives several algorithms that can be used to perform particular operation or task.
- The strategy pattern allows programmers to change the implementation of something used at runtime.
- Strategy lets the algorithm vary independently from the clients that use it.
Strategy Pattern

How:

- Define a family of algorithms, encapsulate each one, and make them interchangeable
- Capture the abstraction in an interface, bury implementation details in derived classes

Example: strategy.cc
Goal:

- An adapter pattern converts the interface of a class into another interface the clients expect.
- Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.
Components:

- **Target**: This is the interface with which the client interacts.
- **Adaptee**: This is the interface the client wants to interact with, but can't interact without the help of the Adapter.
- **Adapter**: This is derived from Target and contains the object of Adaptee.

Example: adapter.cc
Facade Pattern

Goal:

- Facade Pattern provides a unified - simplified interface to a complex subsystem or set of interfaces.
- Facade Pattern provides a higher level interface simultaneously decoupling the client from the complex subsystem.
Facade Pattern

How:

- Identify a simpler, unified interface for the subsystem or component.
- Design a ‘wrapper’ class that encapsulates the subsystem.
- The client uses (is coupled to) the Facade object or pointer.

Example: facade.cc
End