WELCOME TO TUTORIAL 5

CS247
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
COMPOSITION VS AGGREGATION

• 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

• 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

• Aggregation (denoted by a \textit{white}, or not-filled-in diamond), is a weaker relationship than Composition.
• Aggregation implies that the container is \textit{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
UML CREATION

• 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.
OBSERVER DESIGN PATTERN  
A.K.A (DEPENDANTS, PUBLISH-SUBSCRIBE)

Intent

Define a one-to-many dependence between objects so that when one object changes state, all its dependents are notified and updated automatically.
MOTIVATION

Views

Model
Use the observer pattern in any of the following situations:

• When an abstraction has two aspects, one dependent on the other. Encapsulating these aspects in separate object lets you vary and reuse them independently.

• When a change to one object requires changing others, and you don’t know how many objects need to be changed.

• When an object should be able to notify other objects without making assumptions about who these objects are. In other words, you don’t want these objects tightly coupled.
STRUCTURE

**Subject**
- Attach(in Observer)
- Detach(in Observer)
- Notify()

**Observer**
- Update()

**ConcreteSubject**
- subjectState
- GetState()

**ConcreteObserver**
- observerState
+ Update()

foreach o in observers
  o.Update()

subject

return subjectState

observerState = subject.GetState()
PARTICIPANTS

• Subject
  • Knows its observers. Any number of observer objects may observe a subject
  • Provides an interface for attaching and detaching Observer objects.

• Observer
  • Defines an updating interface for objects that should be notified of changes in a subject

• ConcreteSubject
  • Stores state of interest to ConcreteObserver objects.
  • Sends a notification to its observers when its state changes.

• ConcreteObserver
  • Maintains a reference to a ConcreteSubject object.
  • Stores state that should stay consistent with the subject’s
  • Implements the Observer updating interface to keep its state consistent with the subject’s
PUSH MODEL VS PULL MODEL

• The subjects sends observers detailed information about the change, whether they want it or not.

• The decision to push or pull update information depends on the frequency of operations and the complexity of the data.

The subjects sends nothing but the most minimal notification, and observers ask for details explicitly thereafter.

Class Exercise - Try to convert the push model into pull model in given code.