CS-AD 209 - Spring 2013
Software Engineering
Lecture 4

Professor Michael Paik
mpaik@cs.nyu.edu

Attendance

Agenda

- Design Patterns Overview
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

Agenda

- Design Patterns Overview
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

What is a Design Pattern?

Design Pattern

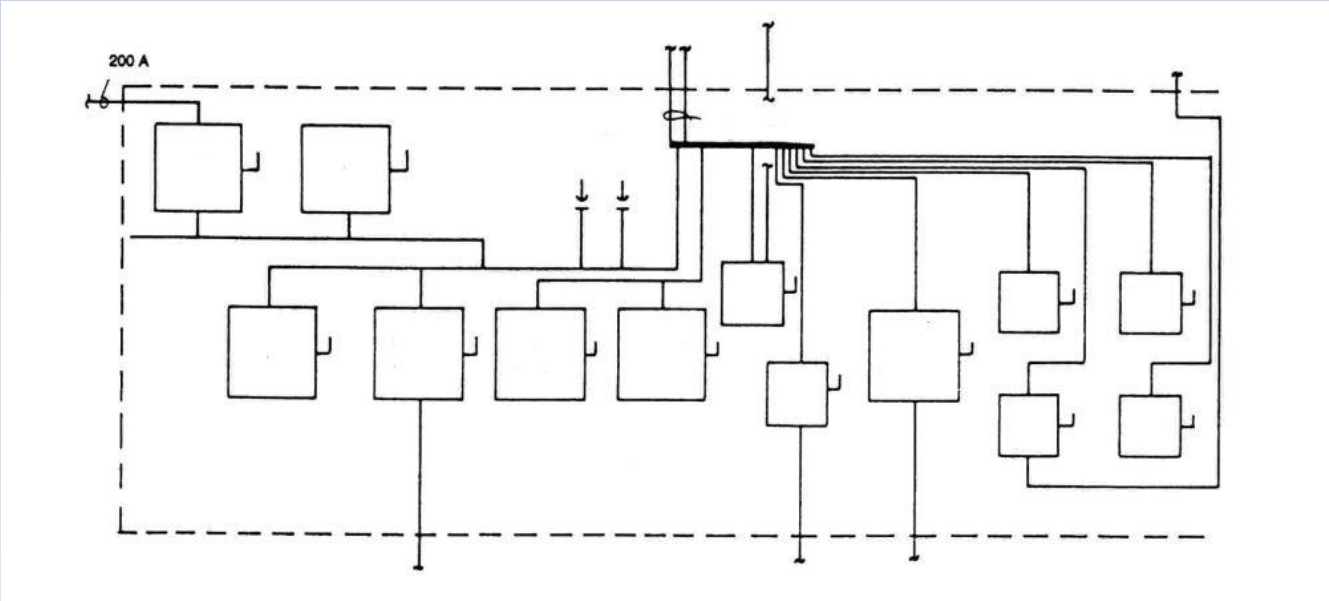
A design pattern is a *reusable*, best-practice way of solving a *commonplace* problem that arises from complex object oriented design

Design Pattern

A design pattern is *not* a template of code that you flesh out with your behaviors (like an abstract class or interface) but a way of wiring abstract and concrete classes and interfaces together

Design Pattern

A design pattern is, therefore, much like a wiring schematic where you don't know what the loads are



What is a Design Pattern *for*?

Design Pattern

With a few exceptions, design patterns are meant to allow you to decouple instance calls from concrete implementations through the use of abstraction, injection, and indirection

Design Pattern

This, in turn, allows you to program to an abstract API (Applicaton Programming Interface) and not worry about the implementation details of concrete classes, nor have to rewrite if you have to take into account new concrete classes

What makes a *good* Design Pattern?

Design Pattern

A good design pattern is comprehensible, covers a broad range of possible common use cases, and tends towards making code simpler

Design Patterns, according to Gamma et al., fall into three main categories:

Agenda

- Design Patterns Overview
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

Creational patterns deal with scenarios surrounding the instantiation of objects

Abstract Factory

The Abstract Factory defines a common interface to create a family of related concrete classes without specifying, *a priori*, what they are

Example

GoF p.87

Abstract Factory

Why can't you just have an abstract superclass for each type of concrete object the factory can produce?

Factory Method

The Factory Method or Virtual Constructor pattern makes an initialization method that would normally create a new instance of a concrete class *abstract* and by doing so, forces subclasses to decide for themselves what concrete implementation to use

Example

GoF p.107

Factory Method

For those of you familiar with Inversion-of-Control frameworks like Spring, IoC, where concrete implementations are 'injected' into classes by the framework, is a kissing cousin of the Factory Method pattern (solves the same problem with different semantics)

Singleton

The Singleton pattern is used in instances where you want to guarantee that there is only a single, common instance of a class, and that instance needs to be easily reachable

Example

Ideas?

Agenda

- Design Patterns Overview
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

Structural patterns deal with scenarios surrounding the presentation of objects to callers

Adapter

The adapter pattern is one of the simplest – it just translates from one interface to another

Usually this is because a vendor-supplied or other third-party module doesn't present the interface you want it to

Example

`psycopg2`
http://initd.org/psycopg/docs/usage.html

vs

`mysql-python`
http://mysql-python.sourceforge.net/MySQLdb.html

Adapter

If you have an existing implementation of something you want to replace, an Adapter can be the way to go

If you have an abstract class, but the concrete classes don't match the method calls or other behavior, you can use an Adapter to translate

Composition

Not to be confused with object composition, the Composition pattern is used when objects naturally nest within one another in a tree-like fashion

Examples

path elements

GUI elements

XML and other structured text

Composition

The important characteristic of the Composition pattern is that every sub-component of your root component should be substitutable with zero code change

Decorator

The Decorator pattern is used to augment functionality of a class without creating a separate subclass – it is an example of *object* composition, and therefore is often called the *Wrapper* pattern

Example

GoF p.175

Decorator

The Decorator pattern is often found in GUI frameworks, or otherwise linked with the Composition pattern seen earlier

However, many other uses exist, e.g. decorating any class with logging functionality

Proxy

The Proxy pattern is used to support lazy initialization/instantiation

This is useful when loading objects is a resource-heavy operation (e.g. large images, video files), and only some subset of objects is used each time

Examples

PDF viewers

Datastores

Proxy

The Proxy pattern's internal mechanics are like the Singleton's usual implementation: maintain an internal private variable with the actual content and instantiate on first use

On subsequent calls, simply return the content rather than reloading

Agenda

- Design Patterns Overview
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

Behavioral patterns deal with abstracting execution and notification specifics

Iterator

Most of you will be familiar with the Iterator pattern through the use of e.g. the `Iterable` interface in Java and the `iter()` builtin in Python

The iterator pattern simply presents a standard way of iterating through items in a collection without exposing any implementation details of how that collection works

Observer

The Observer pattern is the precursor to middleware, and is otherwise known as Pub/Sub (Publish/Subscribe)

In this pattern, objects register themselves as *observers* of some event occurring in some other class

The observed class, when the event occurs, notifies all objects that have subscribed as observers

Examples

Thread libraries

Data tickers

IFTTT

Observer

The Observer pattern is used within a *single* process – that is, when the objects have direct reference to one another

Middleware abstracts this pub/sub relationship over the network, allowing decoupling of publication and subscription

Strategy

The Strategy pattern is a way of encapsulating an algorithm so that a caller can provide relevant operands to an abstract strategy and swap various concrete strategies in and out

Examples

Search / sort

Task scheduling

Web search

Strategy

The Strategy pattern is most useful when different algorithms work more or less effectively on different workloads, and it is possible to use a heuristic or other predictor to intelligently select the best one

Design Patterns in Our Books

- Gamma et al. (Gang of Four / GoF)
  - Still the reference standard, but has lots of patterns that aren't used much in application programming (e.g. Interpreter)
- Freeman et al. (Head First)
  - Teaches more by example – less academic and more concrete
  - I can't forgive them for the stupid cover though
- Code Complete
  - Probably the most practical – covers the essential common patterns and better integrates them overall into concepts of software construction

Homework

We've covered 10 design patterns at a high level, but it's important to get closer to the metal

In groups of 4, find at least one example of each of these in an open source project on Github, Sourceforge, or Google Code and

1) Draw a MDD of the classes and interfaces immediately related to the pattern in practice, and
2) Write a 2-3 sentence blurb describing why this pattern is used in this case

(Google is your friend, and Java coders in particular love class names like AbstractBlahblahFactory)

Reading for Next Class

Chapters 2-3 of
The Little Book of Semaphores
(PDF on newclasses)