

Façade Pattern

- Actually, it's "facade"
 - (according to GoF book)
- References
 - section 23.8 of textbook
 - *Design Patterns*, Gamma et al. (in library)

GoF Facade Pattern

- It has similarities to the Adapter Pattern
 - Recall: various 3rd-party sales tax modules with different interfaces
 - Adapter pattern says
 - Make a single common interface object
 - Only its designer need worry about the 3rd-party APIs

Facade and Adapter: Differences

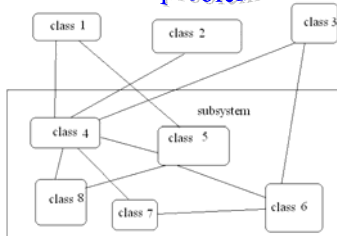
- Adapter pattern:
 - hide differences among interfaces
 - (of similarly-functioning modules)
- Facade pattern:
 - Hide different classes (and thus their interfaces)
 - By "interfaces" do we mean UIs or APIs?

Facade Pattern Background

- The facade pattern is a way to hide interfaces of parts of a "subsystem"
- What exactly is a "subsystem"?
 - A class encapsulates operations and states
 - (Operations are implemented in OO as what?)
 - (State is implemented as what?)
 - A subsystem encapsulates classes
 - In Java, for example, subsystems are defined using "package" syntax
 - So the modularity concept is very applicable here
 - Classes/objects are the OO innovation in modular software structure
 - Packages are not specific to OO
 - (Their need in Java shows that OO itself is not complete and perfect)

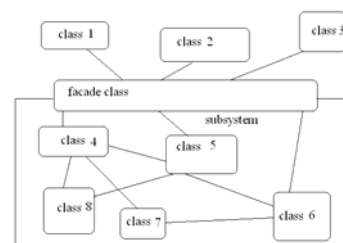
Facade Pattern Diagram

This is a problem!



Facade Pattern Diagram II

This is a solution!



Facade Pattern Comments

- Notice the hiding of the subsystem structure
- Interface to subsystem is now more controlled
 - ...and simpler in some ways
- Accessing classes in the subsystem:
 - how does the interface know what class to access?
- This is also an example of what other pattern(s)?
Pick one, two, or three:
low coupling, indirection, singleton
- The coupling to the subsystem is now
(pick one: lower, higher)

Facade and Adapter – a Recomparison

- Both use the Indirection Pattern
- Both hide complexity by providing a single interface to a “bunch of stuff”
- Both reduce coupling
- Adapter hides differences among interfaces to similar things
- Facade hides identities of classes in a subsystem
 - Do the classes have to be variations of each other?

Facade Pattern: Taking Things One Step More

- Layered systems can communicate using facades
- Let’s draw a diagram on the board to illustrate...

Facade Pattern: Taking Things Another Step

- Define a facade as a class
- Define subclasses, each for a different implementation situation
 - Registration example: a prerequisite checker for undergrads and one for grads
 - Different rules apply
 - Depending on the student, a *factory* creates either an object of class
 - PrereqCheckUndergrad, or PrereqCheckGrad
 - ...and returns it attached to a facade reference

Taking Things Another Step II

- Define a facade as a class
 - Optional: abstract class (e.g. in Java)
- Define subclasses, each for a different implementation situation
 - Registration example: a prerequisite checker for undergrads and one for grads
 - Different rules apply
- Depending on the student, a factory creates either
 - A prereqCheckUndergrad or a prereqCheckGrad object
 - ...and returns it attached to a facade reference
- What is meant by **object** and **reference** here?
- The prereqCheckUndergrad is *also* a facade
 - It interfaces to a prereqChecking subsystem for undergrads
- Similarly for prereqCheckGrad
- The top-level facade is hiding which “sub”-facade

Taking Things Another Step III

- Depending on the situation, a factory can create either
 - A facade for situation 1
 - or
 - A facade for situation 2 (...n)
- Access to the proper facade is via a top-level facade
- This way the “implementation version” is hidden
- This is supportable by the OO superclass/subclass idea
- Let’s see a couple more examples



Taking Things Another Step IV

- Depending on the situation, a factory creates either
 - A facade for situation 1
 - or
 - A facade for situation 2 (...n)
- Access to the proper facade is via a top-level façade
- Example 2:
 - System runs on either of 2 machine types
 - One handles high precision arithmetic in hardware
 - One has smaller byte size – doesn't – requires algorithmic sol'n
 - Top-level facade is linked to arithmetic subsystem for current machine
 - Why is this good?



Taking Things Another Step V

- Access to the proper facade is via a top-level façade:
- Example 3:
 - Different implementation versions (generations) of the subsystem exist (v1.1, v1.2, etc.)
 - For backward compatibility, a top-level facade is "connected" to facade objects specific to subsystem v1.1, v1.2, etc.
 - The overall system (and its programmers) need not worry about
 - Which version of the library or other subsystem it has on hand
 - The fact that there are multiple versions around
 - The factory does have to worry about this
 - What is meant by "connected"?