

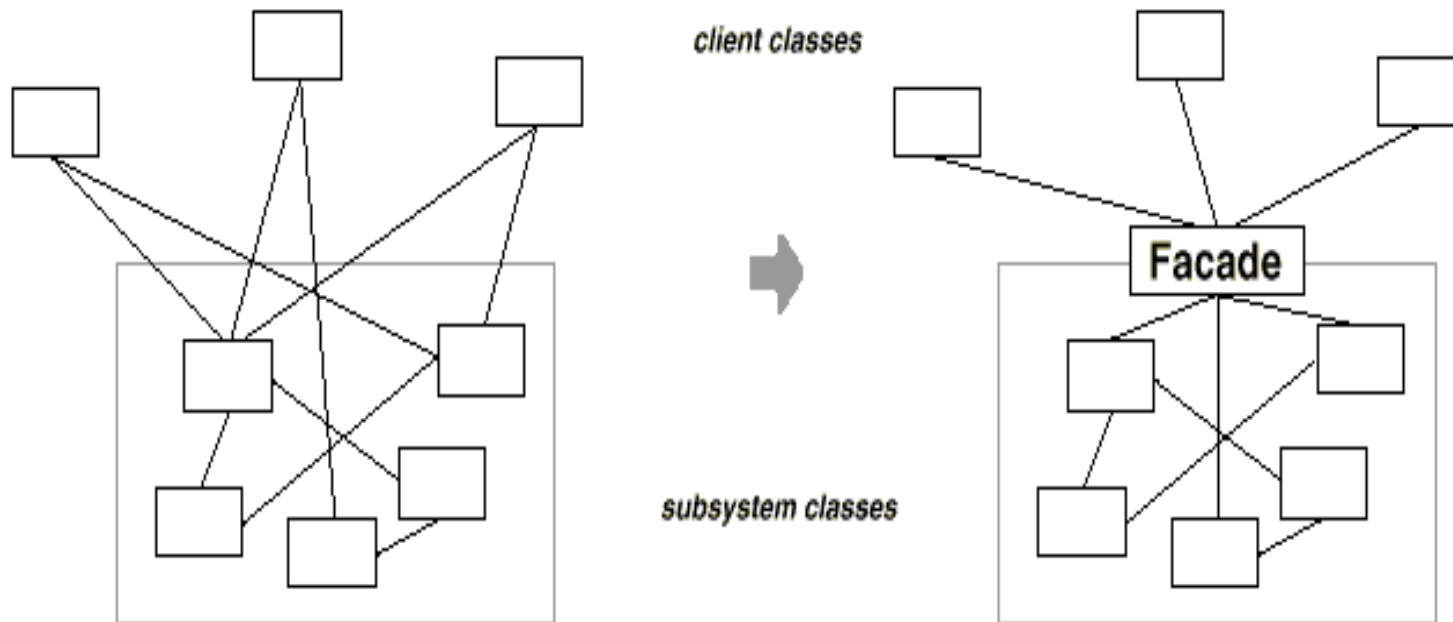
# The Facade Pattern

# The Facade Pattern

- Intent
  - ⇒ Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
- Motivation
  - ⇒ Structuring a system into subsystems helps reduce complexity
  - ⇒ Subsystems are groups of classes, or groups of classes and other subsystems
  - ⇒ The interface exposed by the classes in a subsystem or set of subsystems can become quite complex
  - ⇒ One way to reduce this complexity is to introduce a facade object that provides a single, simplified interface to the more general facilities of a subsystem

# The Facade Pattern

- Motivation



# The Facade Pattern

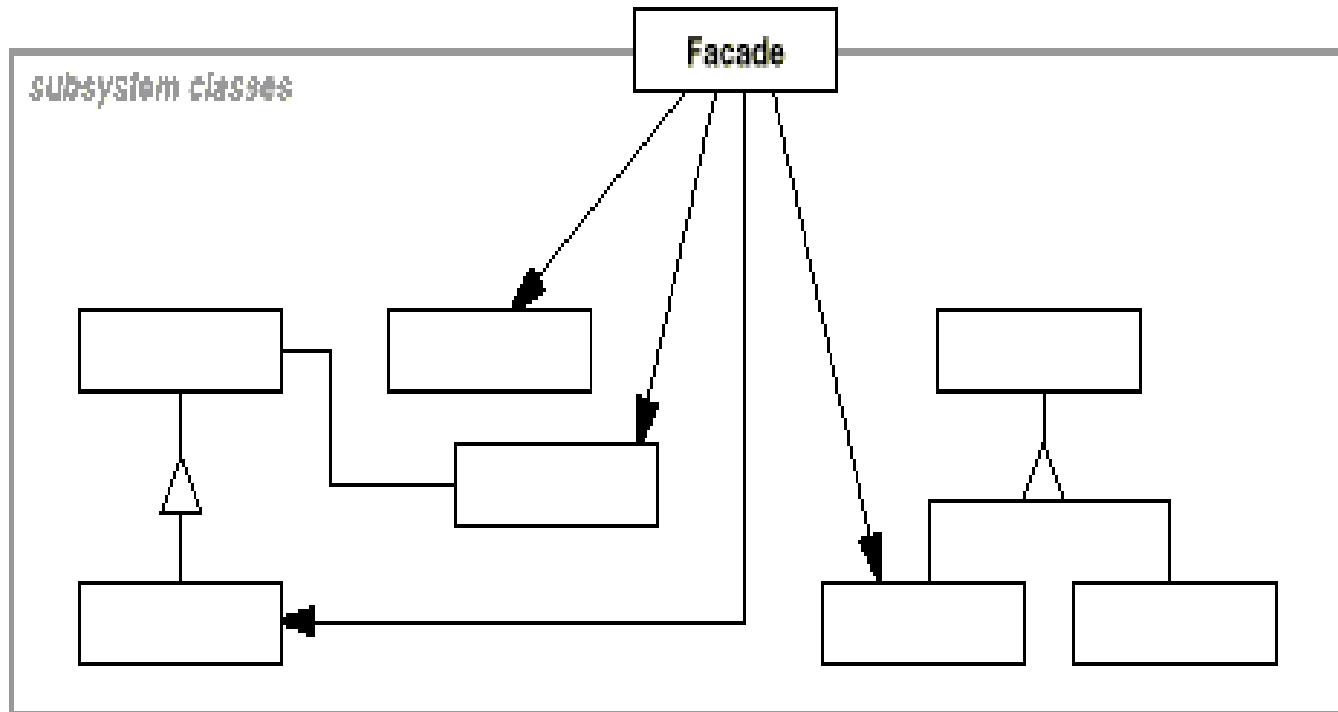
- Applicability

Use the Facade pattern:

- ⇒ To provide a simple interface to a complex subsystem. This interface is good enough for most clients; more sophisticated clients can look beyond the facade.
- ⇒ To decouple the classes of the subsystem from its clients and other subsystems, thereby promoting subsystem independence and portability

# The Facade Pattern

- Structure



# The Facade Pattern

- Consequences

- ⇒ Benefits

- It hides the implementation of the subsystem from clients, making the subsystem easier to use
    - It promotes weak coupling between the subsystem and its clients. This allows you to change the classes the comprise the subsystem without affecting the clients.
    - It reduces compilation dependencies in large software systems
    - It simplifies porting systems to other platforms, because it's less likely that building one subsystem requires building all others
    - It does not prevent sophisticated clients from accessing the underlying classes
    - Note that Facade does not add any functionality, it just simplifies interfaces

- ⇒ Liabilities

- It does not prevent clients from accessing the underlying classes!

# Facade Pattern Example

- A compiler

