Design Issue #7

Supporting Multiple Window Systems

- What about the Windowing System itself?
- The APIs differ… not just the visual elements
  - Can we use Abstract Factory?
    - Not easily… vendors already define class hierarchies
    - Need to align vendor-specific libraries to our ‘product’ abstractions
  - How do we make classes from different hierarchies comply to the same abstract type?

Design Issue #7

Encapsulating Implementation Dependencies

Responsibility	Operations
window management	virtual void Redraw() virtual void Raise() virtual void Lower() virtual void Maximize() virtual void Minimize()
graphics	virtual void DrawLine() virutal void DrawRect() virtual void DrawPolygon() Virtual void DrawText()

What functionalities will the abstract Window support?

1. Intersection of functionality – what is common to all

2. Union of functionality – capabilities of all systems

Design Issue #7

Encapsulating Implementation Dependencies

Responsibility	Operations
window management	virtual void Redraw() virtual void Raise() virtual void Lower() virtual void Maximize() virtual void Minimize()
graphics	virtual void DrawLine() virutal void DrawRect() virtual void DrawPolygon() Virtual void DrawText()

What functionalities will the abstract Window support?

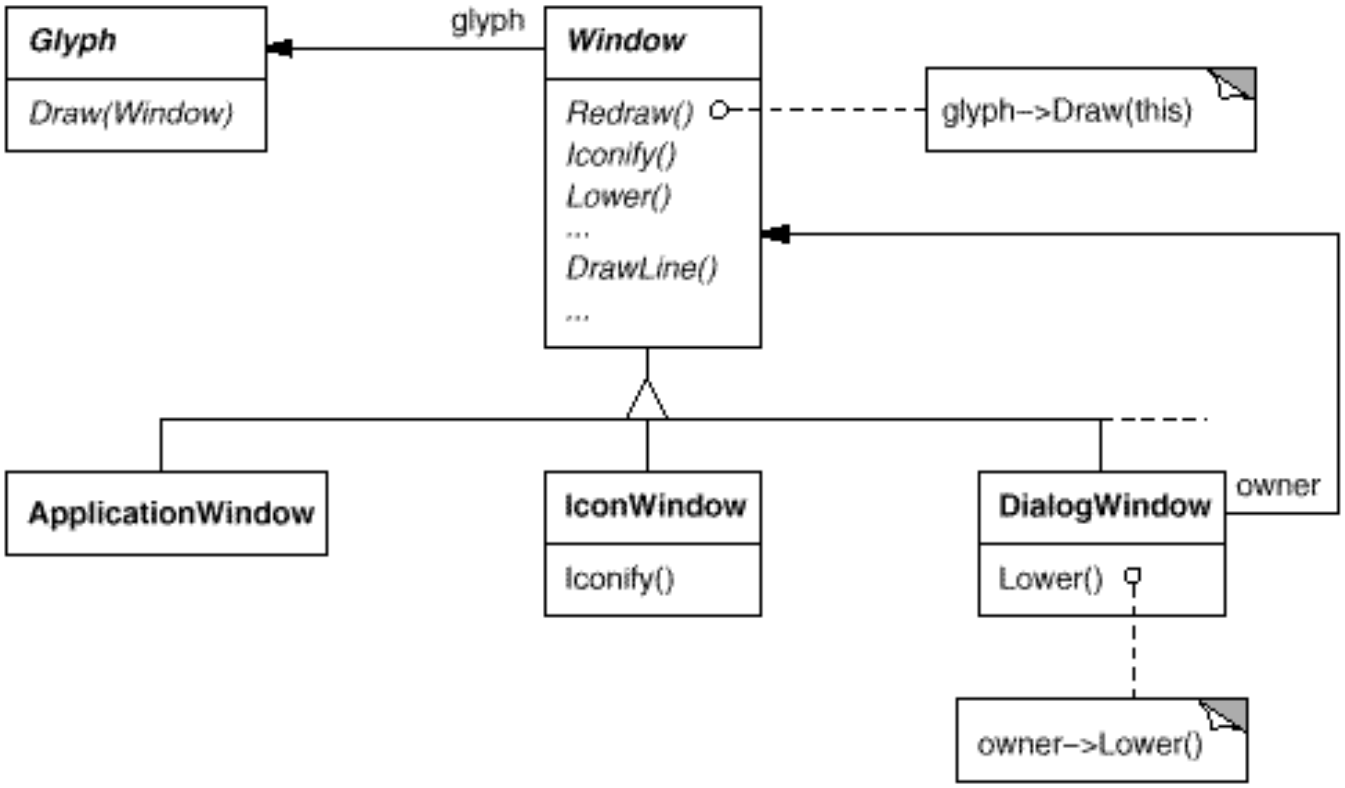
1. Intersection of functionality – what is common to all

something in between

2. Union of functionality – capabilities of all systems

Design Issue #7

Encapsulating Implementation Dependencies



Design Issue #7

Encapsulating Implementation Dependencies

Implement Window hierarchy for different windowing platforms, KDE / OSX / Windows.

Design Issue #7

Encapsulating Implementation Dependencies

Shortcomings

- Maintenance cost due to class explosion
- Not possible to change the Window System after compiling

- What can we do?

Design Issue #7

Encapsulating Implementation Dependencies

Shortcomings

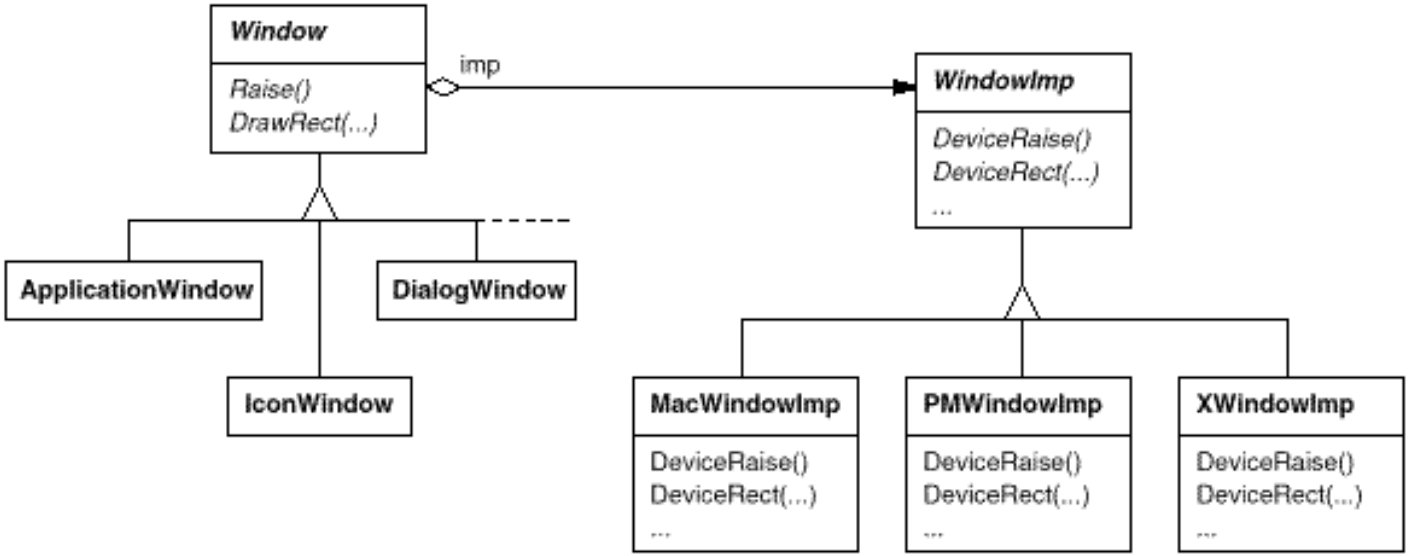
- Maintenance cost due to class explosion
- Not possible to change the Window System after compiling

- What can we do?

Encapsulate the *concept* that varies

Design Issue #7

Supporting Multiple Window Systems



Design Issue #7

Supporting Multiple Window Systems

- Meet the **Bridge** pattern to
  - Defines a uniform set of windowing abstractions (common interface)
  - Clients deal only with Window abstractions, not with the impl.
  - Configure window objects (possible at run-time as well!) to the window system we want simply by passing them the right window system-encapsulating object
    - » **See `AbstractFactory` / `FactoryMethod` patterns**
  - Hide the individual implementations
  - Window hierarchy is not polluted with implementation details

Design Issue #7

Supporting Multiple Window Systems

- Bridge pattern is used up-front in a design to let abstractions and implementations vary independently.

- On the other hand, but similar to Bridge, the **Adapter** pattern is geared toward making unrelated classes work together. It is usually applied to systems after they're designed, during implementation phase.

- Exists 2 variants of Adapter pattern
  - Class Adapter
  - Object Adapter