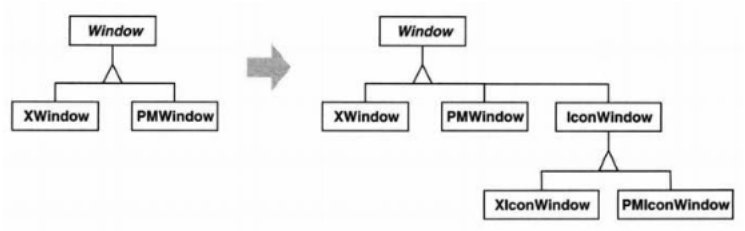
Bridge

Bridge

- Intent
  - Decouple an abstraction from its implementation so that the two can vary independently.
- Motivation
  - When an abstraction can have one of several possible implementations, the usual way to accommodate them is to use inheritance.
  - But inheritance binds an implementation to the abstraction permanently, which makes it difficult to modify, extend, and reuse abstractions and implementations independently.

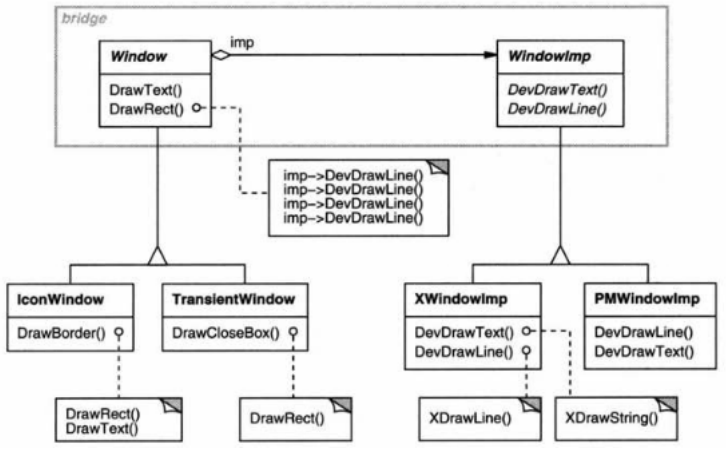
Bridge

- E.g: It's inconvenient to extend the Window abstraction to cover different kinds of windows or new platforms.
- Clients should be able to create a window without committing to a concrete implementation.



Bridge

- Put the Window abstraction and its implementation in separate class hierarchies.



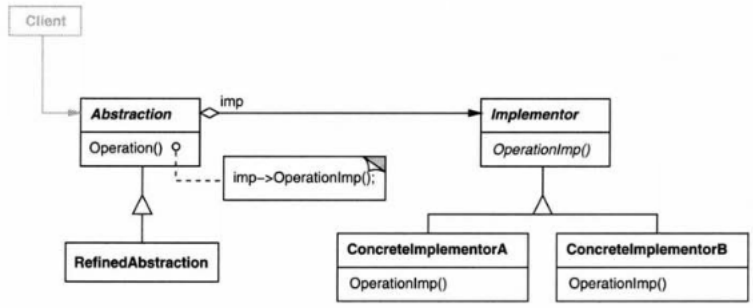
Applicability

- Use the Bridge pattern when
  - you want to avoid a permanent binding between an abstraction and its implementation.
  - both the abstractions and their implementations should be extensible by subclassing.
  - changes in the implementation of an abstraction should have no impact on clients

Applicability

- Use the Bridge pattern when
  - you want to hide the implementation of an abstraction completely from clients.
  - you have a proliferation of classes as shown earlier in the first Motivation diagram.
  - you want to share an implementation among multiple objects (perhaps using reference counting), and this fact should be hidden from the client.

Structure



Consequences

- Decoupling interface and implementation.
  - It's even possible for an object to change its implementation at run-time.
  - Also eliminates compile-time dependencies on the implementation. (essential when you must ensure binary compatibility between different versions of a class library.)
- Can extend the Abstraction and Implementor hierarchies independently.
- Hiding implementation details from clients.

Implementation

- Only one Implementor.
  - creating an abstract Implementor class isn't necessary.
- Creating the right Implementor object.
  - How, when, and where do you decide which Implementor class to instantiate?
    - Abstraction's constructor
    - Abstract Factory

Implementation

- Using multiple inheritance.
  - Inherit publicly from Abstraction and privately from a ConcreteImplementor.
    - relies on static inheritance
    - binds an implementation permanently to its interface.
    - can't implement a true Bridge with multiple inheritance