Bridge Pattern

Xinran Wang
Jiankun Tao

Bridge Pattern

- **Three kinds of patterns:**

- Creational Pattern
- Structural Pattern
- Behavioral Pattern

Bridge Pattern

- **Three kinds of patterns:**

- Creational Pattern
- Structural Pattern
- Behavioral Pattern

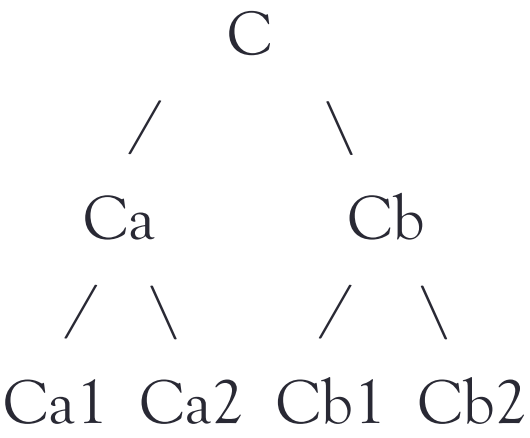
Bridge Pattern

- Decouple an *abstraction* from its *implementation* so that the two can *vary independently*.

Motivation

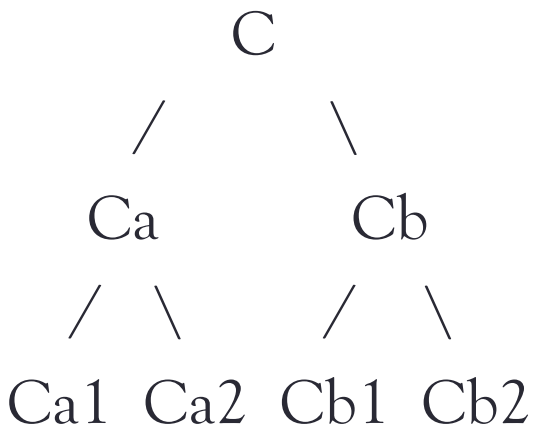
- When an abstraction has several possible implementations, the usual way to accommodate them is to use *inheritance*.
- An *abstract class* defines the interface to the abstraction, while *concrete subclasses* implement it in different ways.
- But, this approach isn't always **flexible** enough.
- Why?
- Because inheritance binds an implementation to the abstraction permanently, which makes it difficult to *modify*, *extend*, and *reuse* abstractions and implementations **independently**.

Benefits

- Inheritance only handles the variation in one dimension
- 

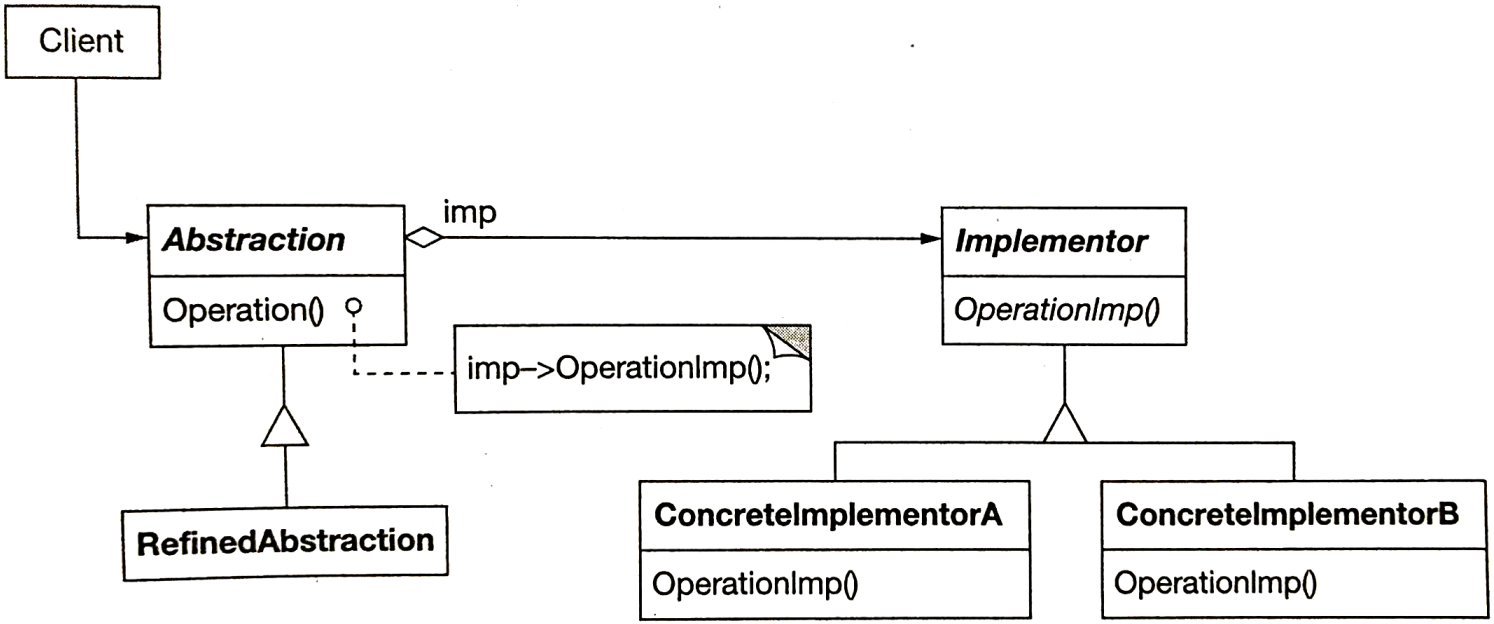
Benefits

- Inheritance only handles the variation in one dimension



- Refactor to

UML

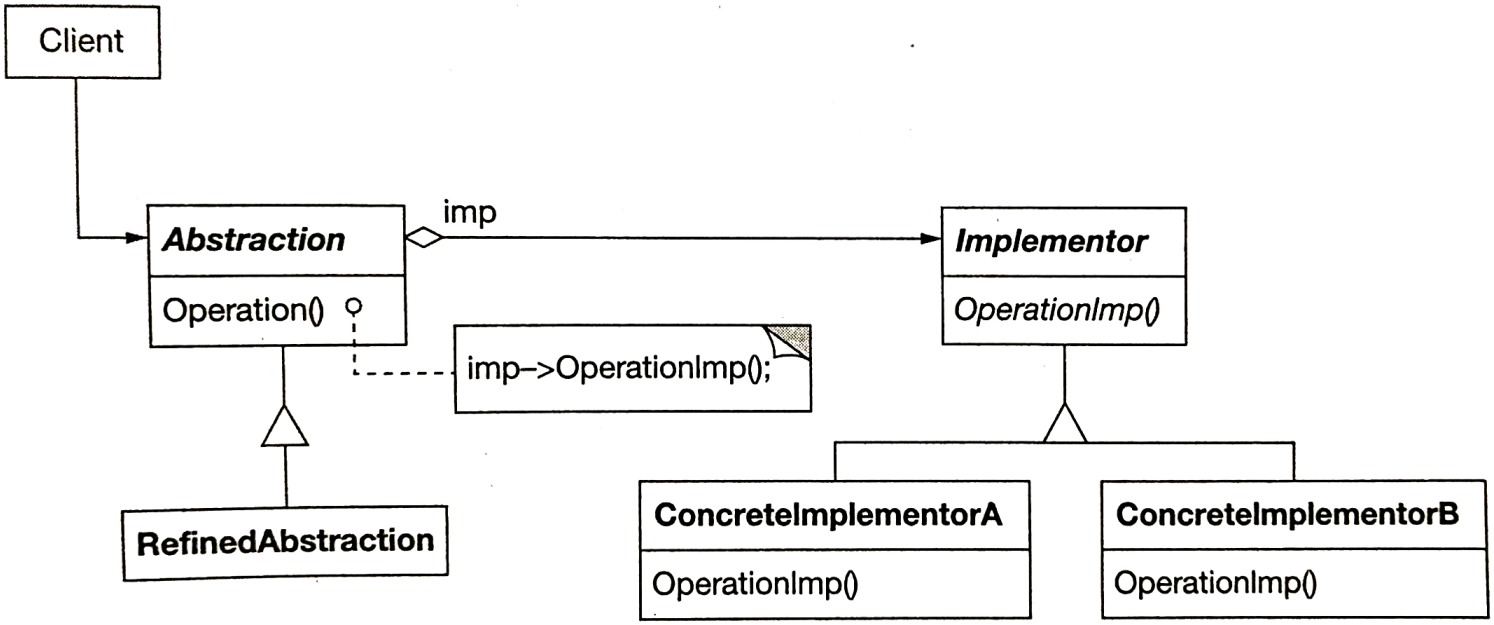

Client
Abstraction
Operation()
imp
Implementor
OperationImp()
imp->OperationImp();
RefinedAbstraction
ConcreteImplementorA
OperationImp()
ConcreteImplementorB
OperationImp()

UML

http://www.turbosquid.com/3d-models/3d-wood-wooden-bridge/772462

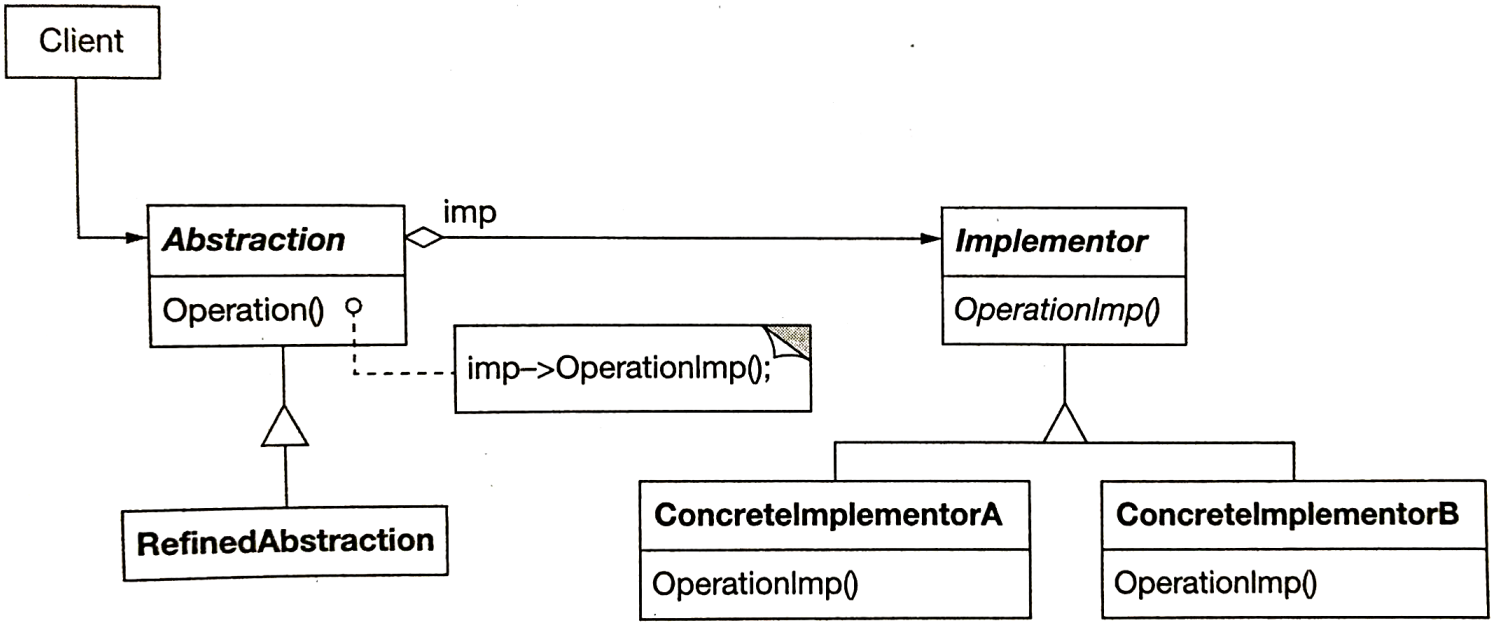
UML

How to implement the bridge?
Aggregation. (“uses”, but not “owns”)



UML

Interfaces of abstraction and interfaces of implementor: The same or different. (High level v.s. Low level)



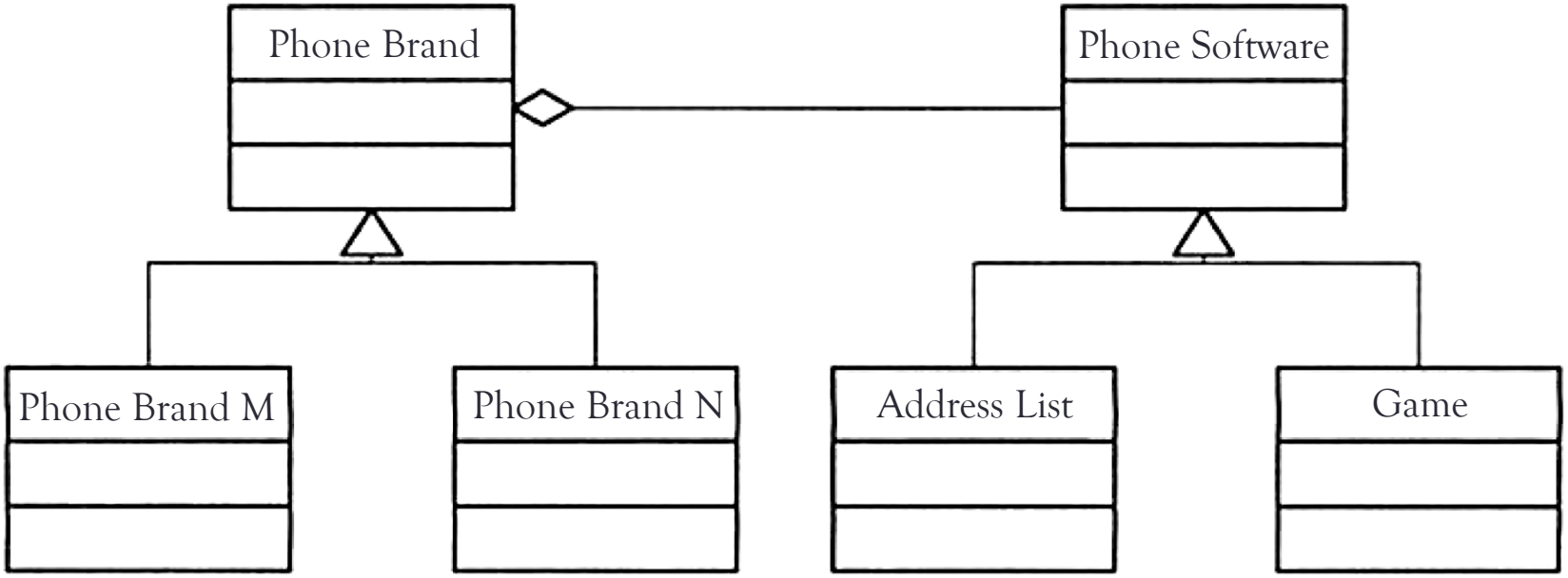
Consequences

- 1. **Decoupling interface and implementation.**
- not bound permanently
- can be configured / changed at run-time
- eliminate compile-time dependencies
- no need to recompile abstraction and clients
- 2. **Improved extensibility.**
- extend the two hierarchies independently
- 3. **Hiding implementation details from clients.**
- e.g. the sharing of implementor objects and reference count

Related Patterns

- *Abstract Factory* can create and configure a particular *Bridge*.
- *Adapter* pattern makes unrelated classes work together. It is usually applied to systems **after** they are designed. While *Bridge* pattern is used **up-front** in a design to let abstractions and implementations vary independently.

Example


Phone Brand
Phone Software
Phone Brand M
Phone Brand N
Address List
Game