Ingegneria del Software

Corso di Laurea in Informatica per in Management

Design Patterns
part 3

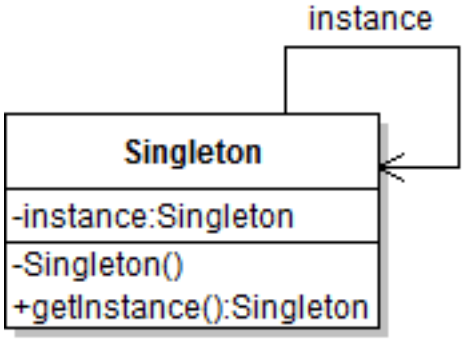
Davide Rossi

Dipartimento di Informatica – Scienze e Ingegneria

Università di Bologna

GoF: Singleton

- Problem: how go guarantee only one instance for a given class is ever created (and all clients see that same instance)?
- Singleton: ensure a class only has one instance, and provide a global point of access to it.



Singleton

- Beware of singletons (code smell)!
  - *How do you provide global variables in languages without global variables? Don't*. [K. Beck]
- Ensure that only one instance of a class is created
- Provide a global point of access to the object

Singleton candidates

- Factories
- Loggers
- Configuration classes
- Resource access
- Classes that have no non-static attributes nor any associations that are navigable away from their instances

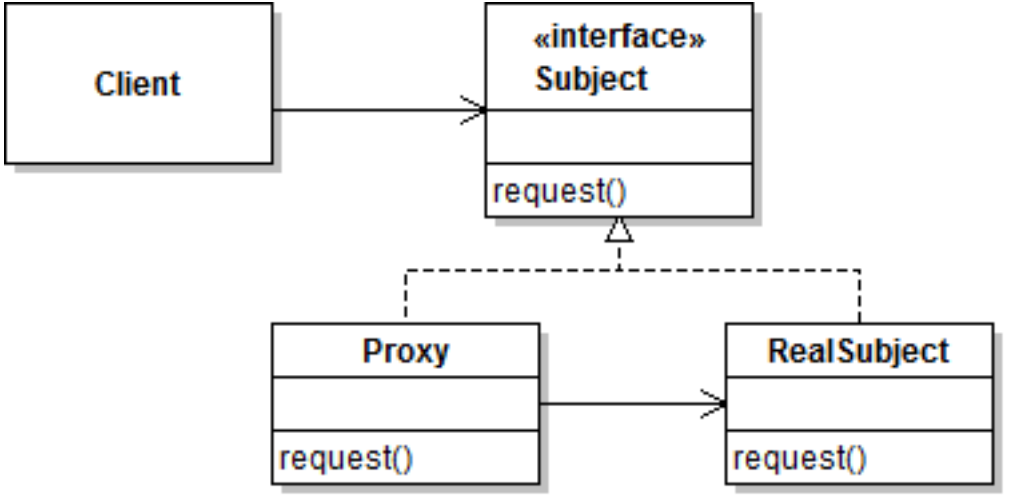
Singleton unlikely candidates

- Classes for which a single instance is part of the specification but is not intrinsic to the problem domain
- Objects that should be globally accessible

GoF: Proxy

- Problem: how to *intercept* the access to on object to address orthogonal concerns?
- Proxy: provide a surrogate or placeholder for another object to control access to it.
- The proxy can add behavior without adding responsibilities (the basic task of a proxy is delegating to a real subject)

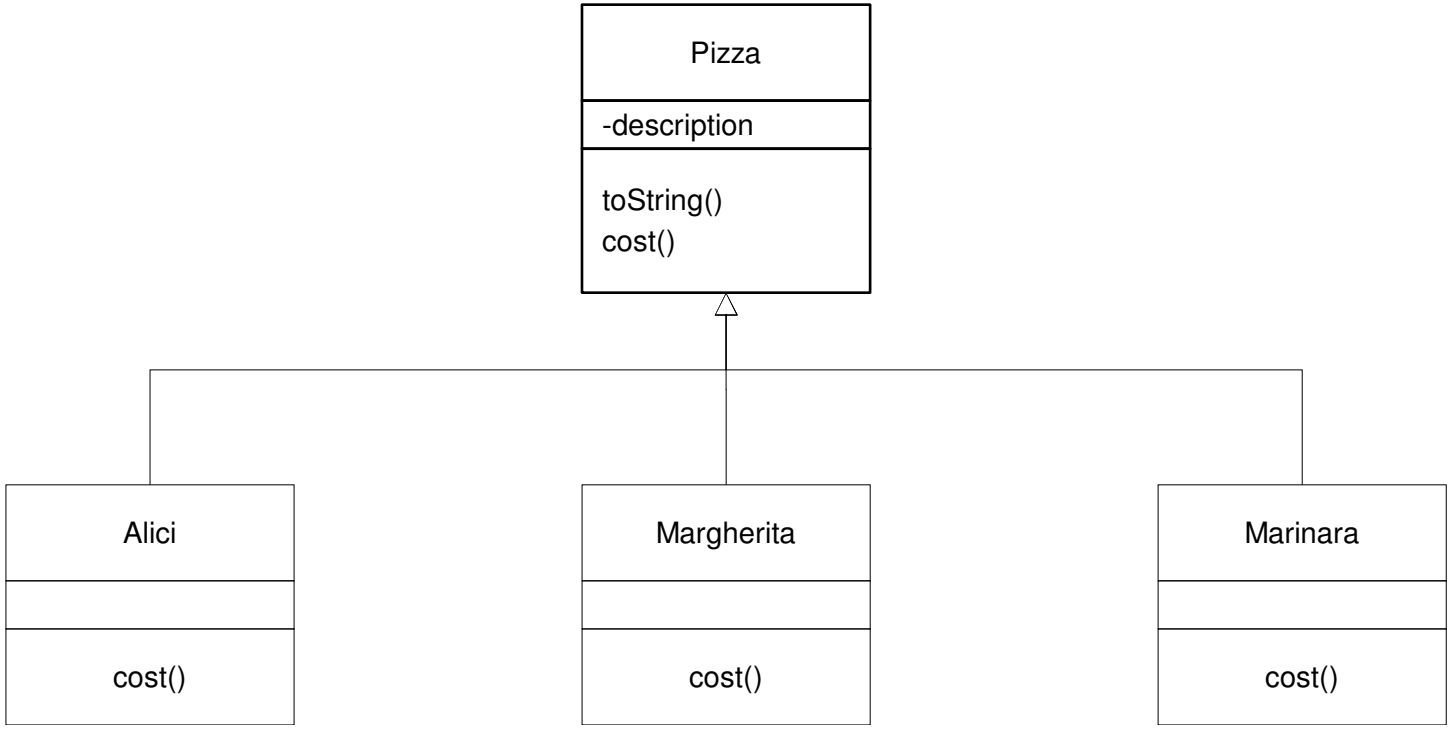
GoF: Proxy


Client
«interface»
Subject
request()
Proxy
request()
RealSubject
request()

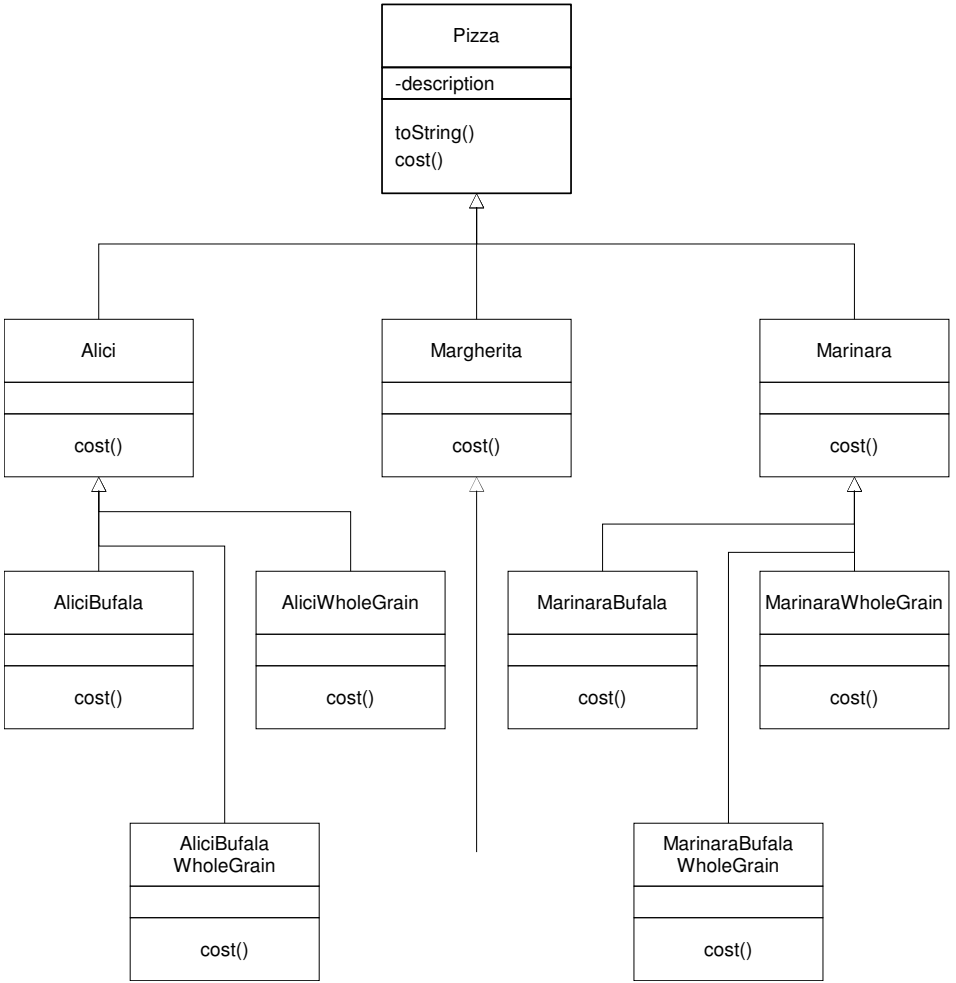
Examples of proxy uses

- Access control
- Access counter
- Access logger
- Access to remote objects (possibly with caching)
- Smart references (reference counter, load a persistent subject on demand, lock checking, ...)

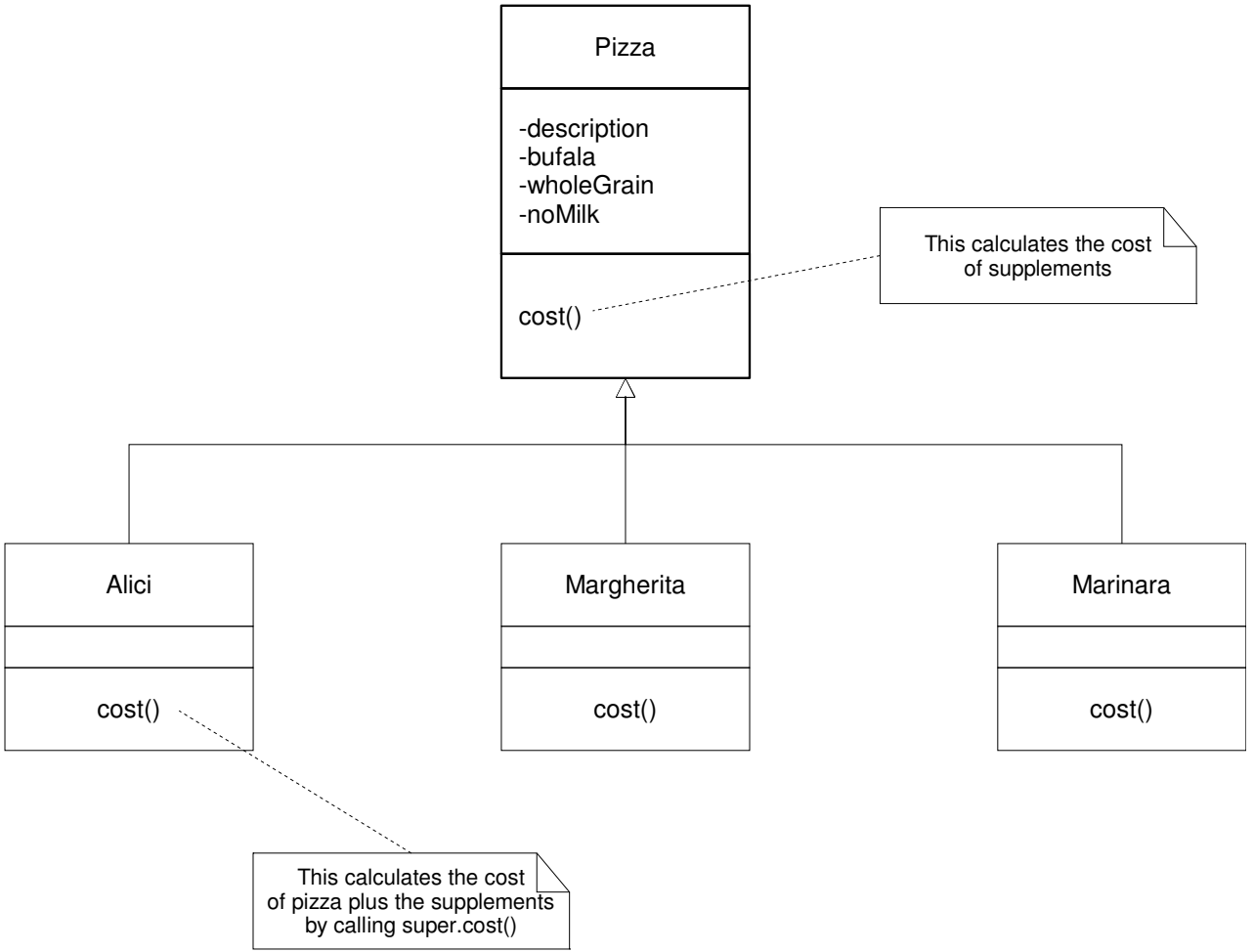
Too many dimensions



Too many dimensions



Too many dimensions

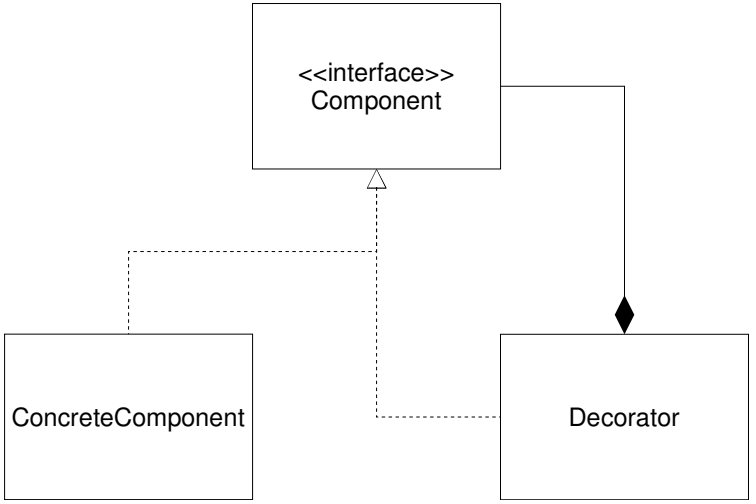


Too many dimensions

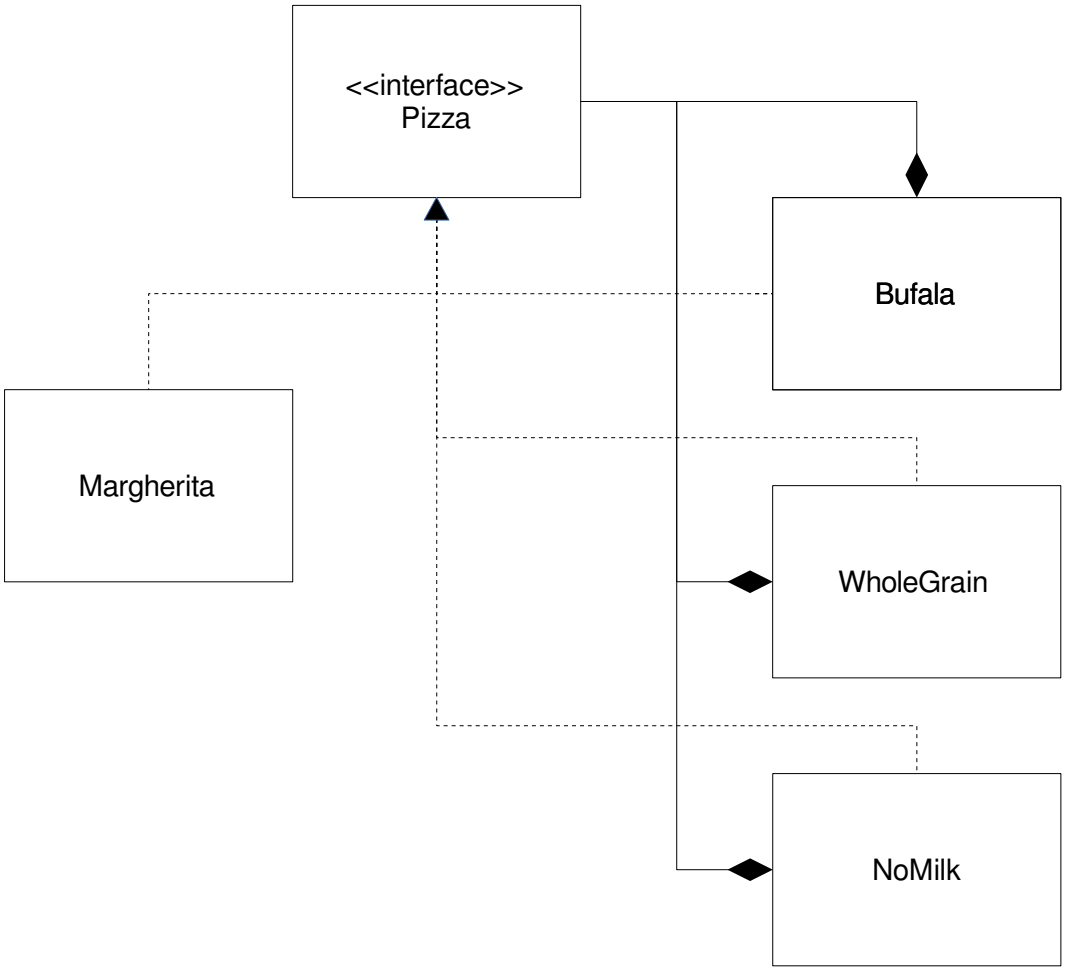
- What about new options?
- What about a price change for an option?
- What about “double bufala”?

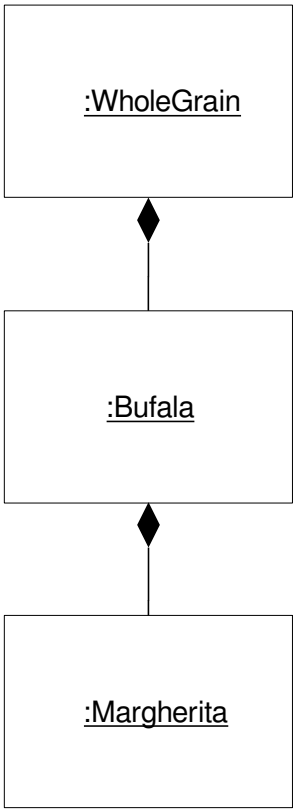
GoF: Decorator

- Attach additional responsibilities to an object dynamically.
- Decorators provide a flexible alternative to sub-classing for extending functionality (think of this as a wrapper).



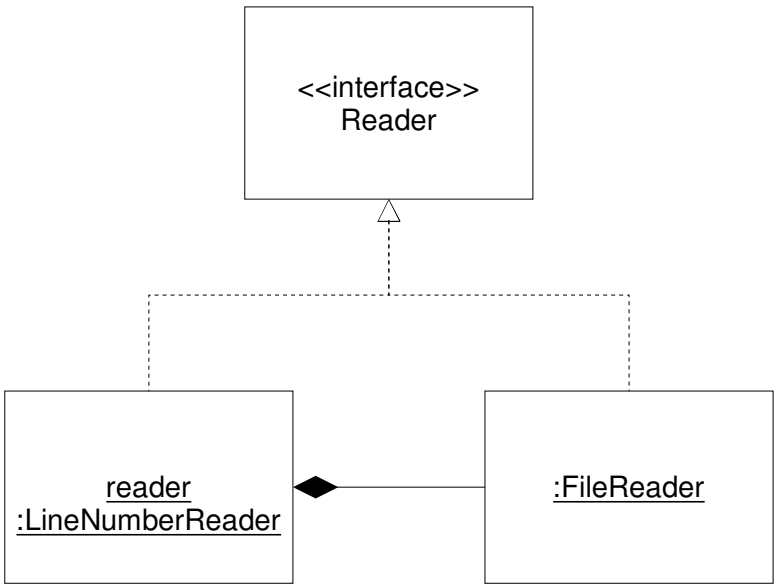
Decorated pizza





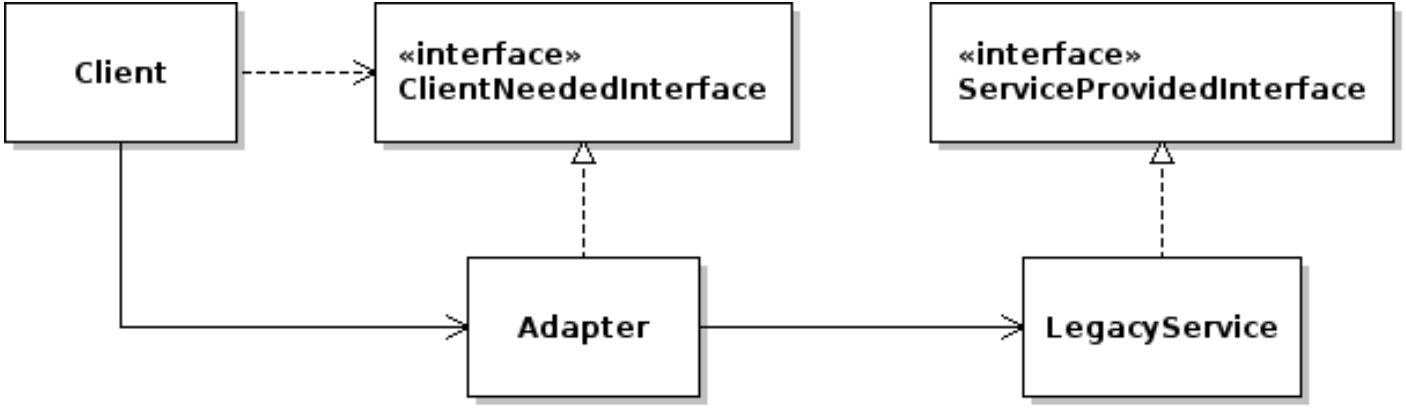
Decorator in the Java API

```
java.io.Reader
Reader reader =
     new LineNumberReader(
          new FileReader(“myfile”));
```



GoF: Adapter

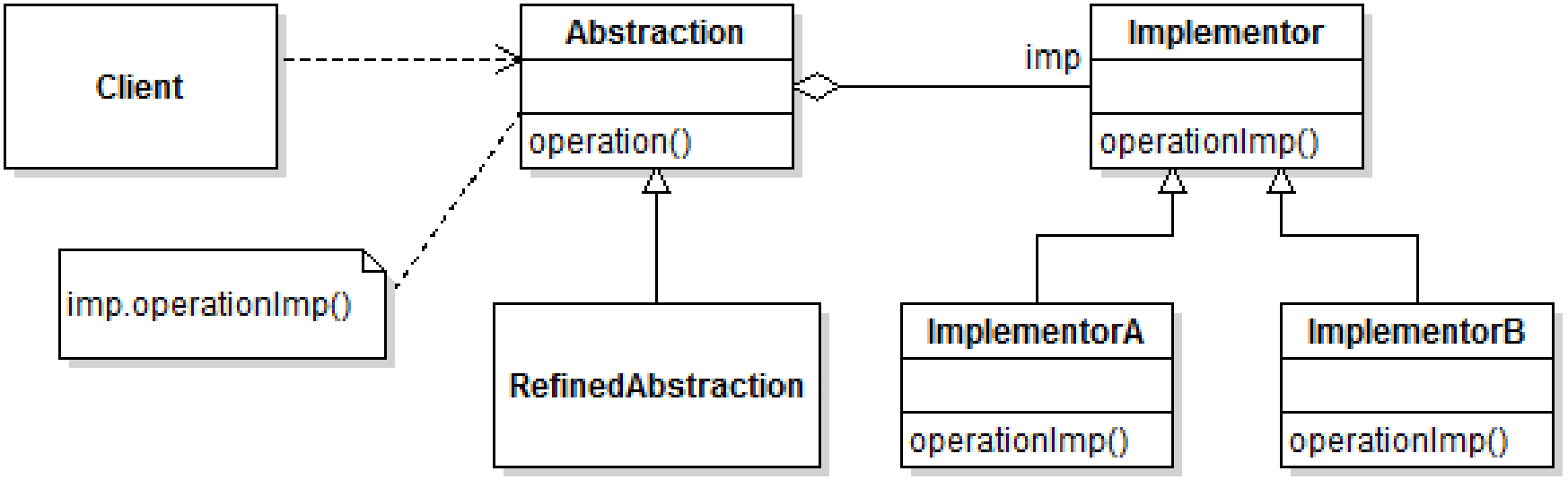
- Problem: how to access a class whose methods does not align with client's expectations (coding styles, parameter types, ...)
- Adapter: convert the interface of a class into another interface clients expect.
- Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.



GoF: Bridge

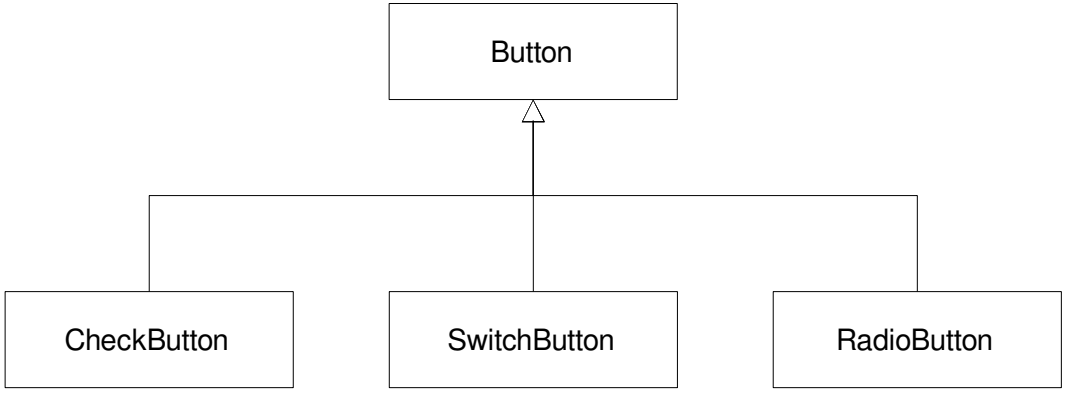
- Problem: how to break the tyranny of the client’s abstractions?
- Bridge: decouple an abstraction from its implementation so that the two can vary independently.
- The Bridge pattern puts abstractions and their implementations in separate class hierarchies. Delegation is used to bind the two.

Bridge

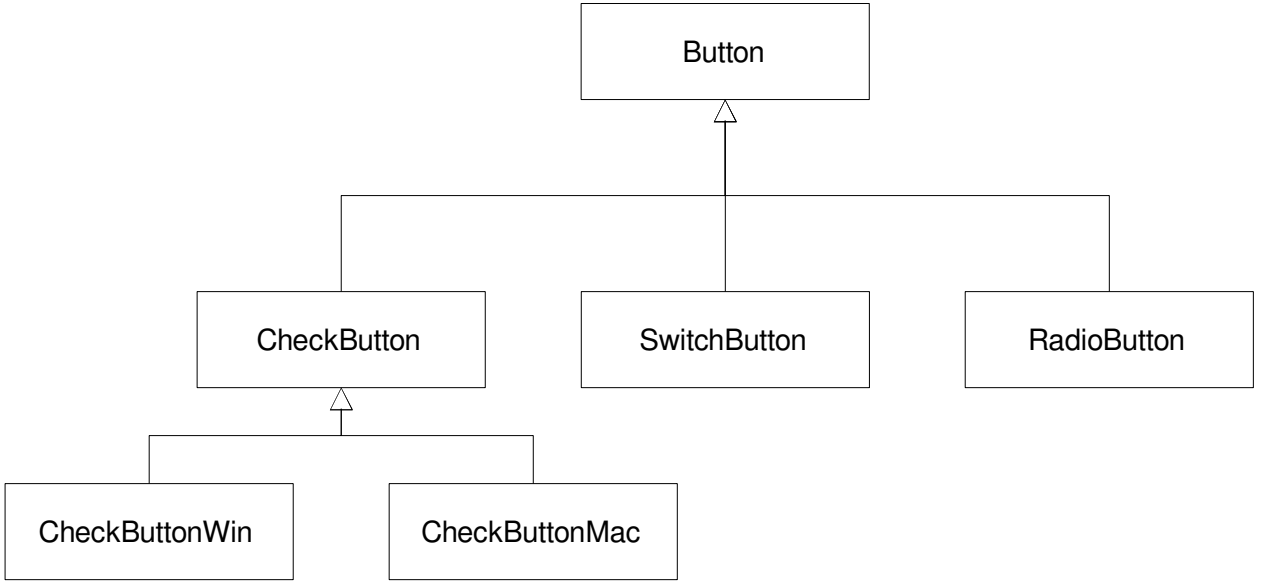


Adapter makes things work after they're designed; Bridge makes them work before they are. [GoF]

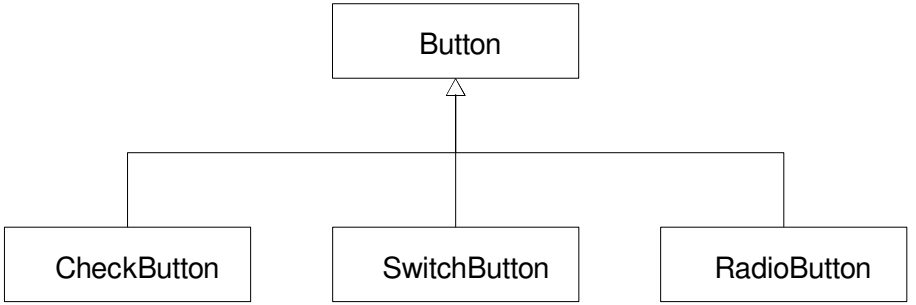
Bridge

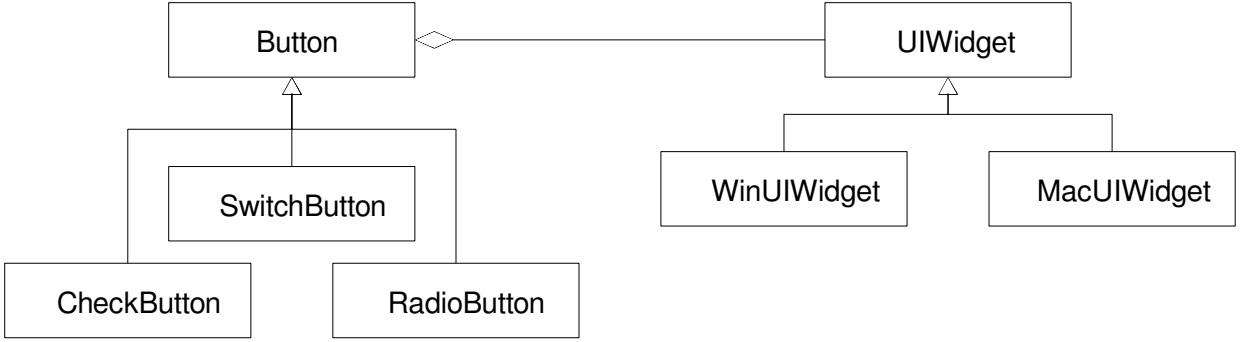


Bridge


Button
CheckButton
SwitchButton
RadioButton
CheckButtonWin
CheckButtonMac

Bridge


Button
CheckButton
SwitchButton
RadioButton


Button
UIWidget
SwitchButton
WinUIWidget
MacUIWidget
CheckButton
RadioButton

Resources

Books

- Eric Freeman & Elisabeth Robson, *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software* (2nd Edition), O'Reilly

Online:

- http://www.vincehuston.org/dp/
- http://www.oodesign.com/
- https://refactoring.guru/design-patterns/
- http://www.informit.com/articles/article.aspx?p=1404056