OO Design Patterns

Coding Cleverly

Why design pattern

Design patterns make code :

Reusable

Common building blocks most programmers know

Add **functionality** in the simplest way

See Repository for code examples!

Types of Design patterns

Creational

Create objects but hide details of **constructors**

Structural

Concern **composition**, **aggregation**, **inheritance**

Behavioral

Concerning **interfaces** between objects

Today’s Agenda

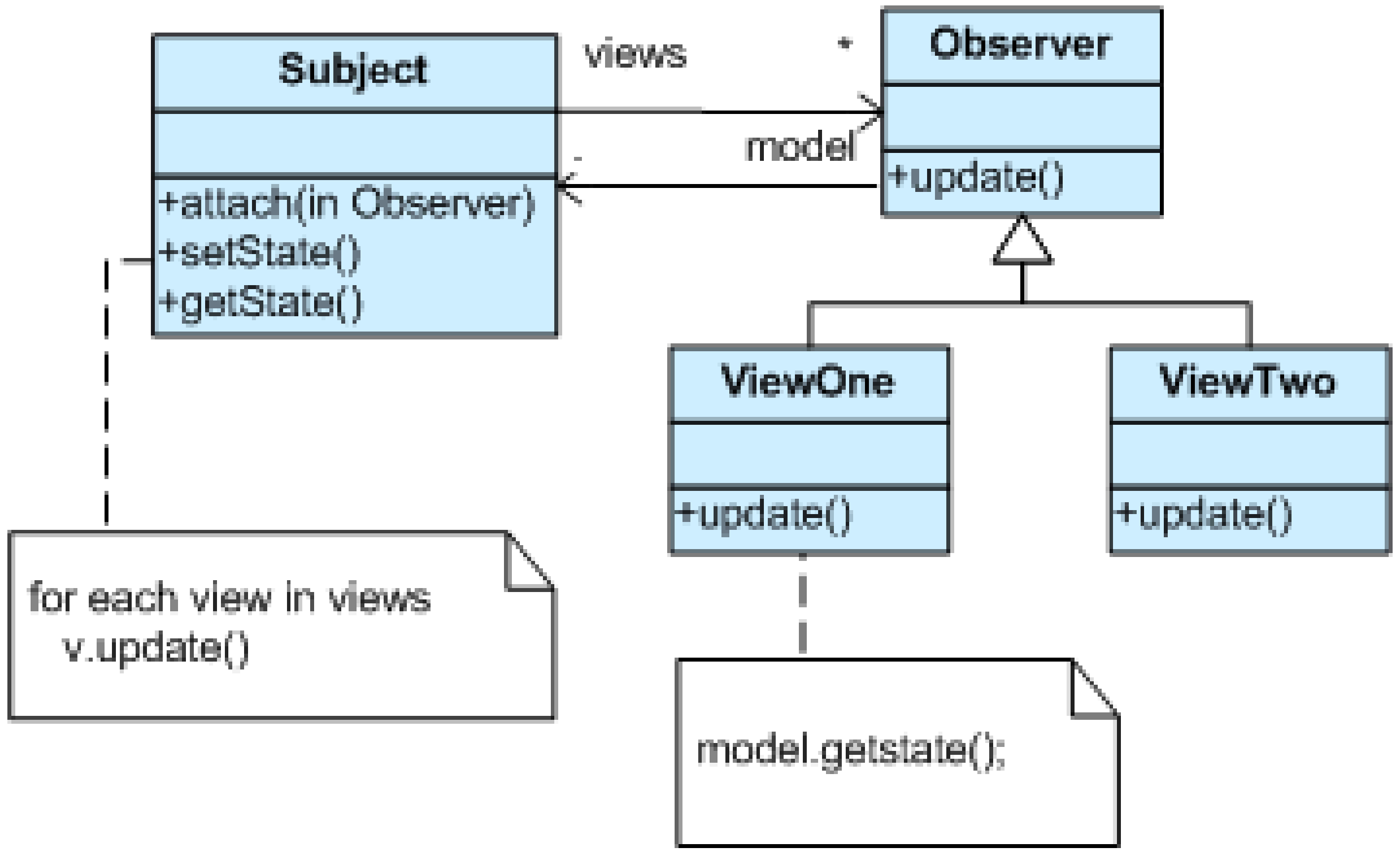
Patterns we will cover

- Observer
- Strategy
- Factory
- Decorator
- Adapter
- Singleton
- Proxy

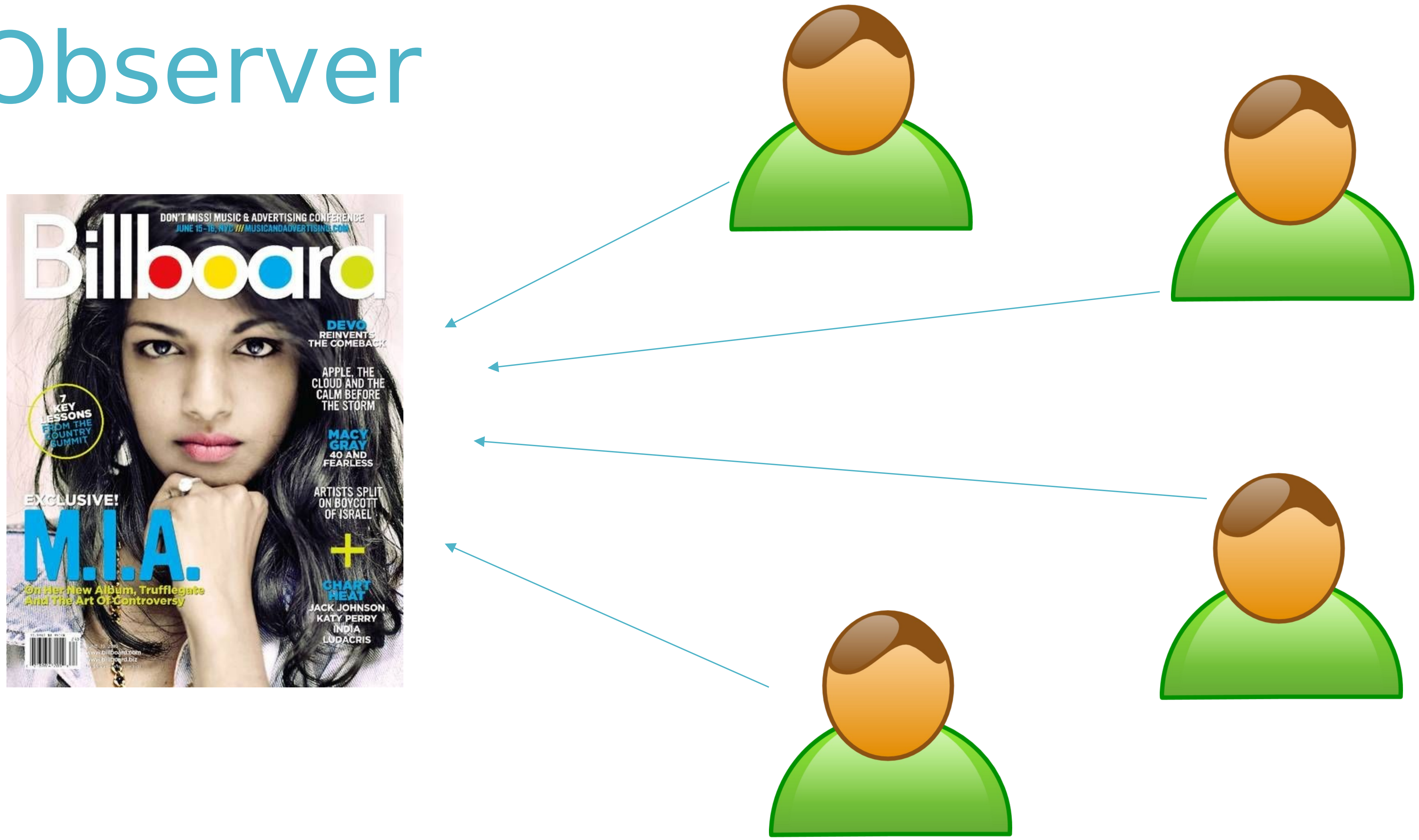
Observer

Problem:
There is a **one to many** relationship where the many need to know information from the one. Is it **scalable** for many to be polling the one?
Example: Think of designing **push notifications** on mobile phones

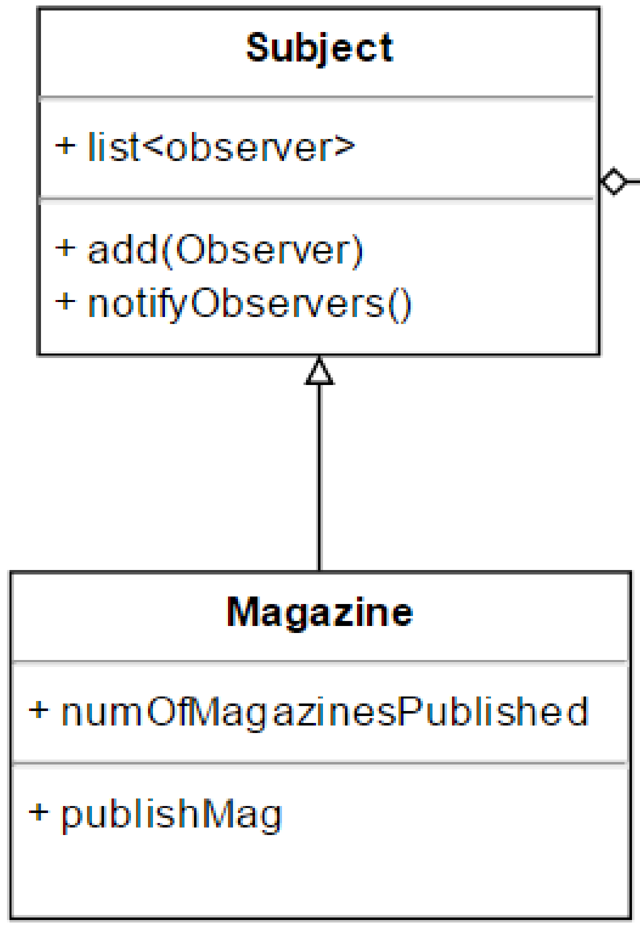
Observer

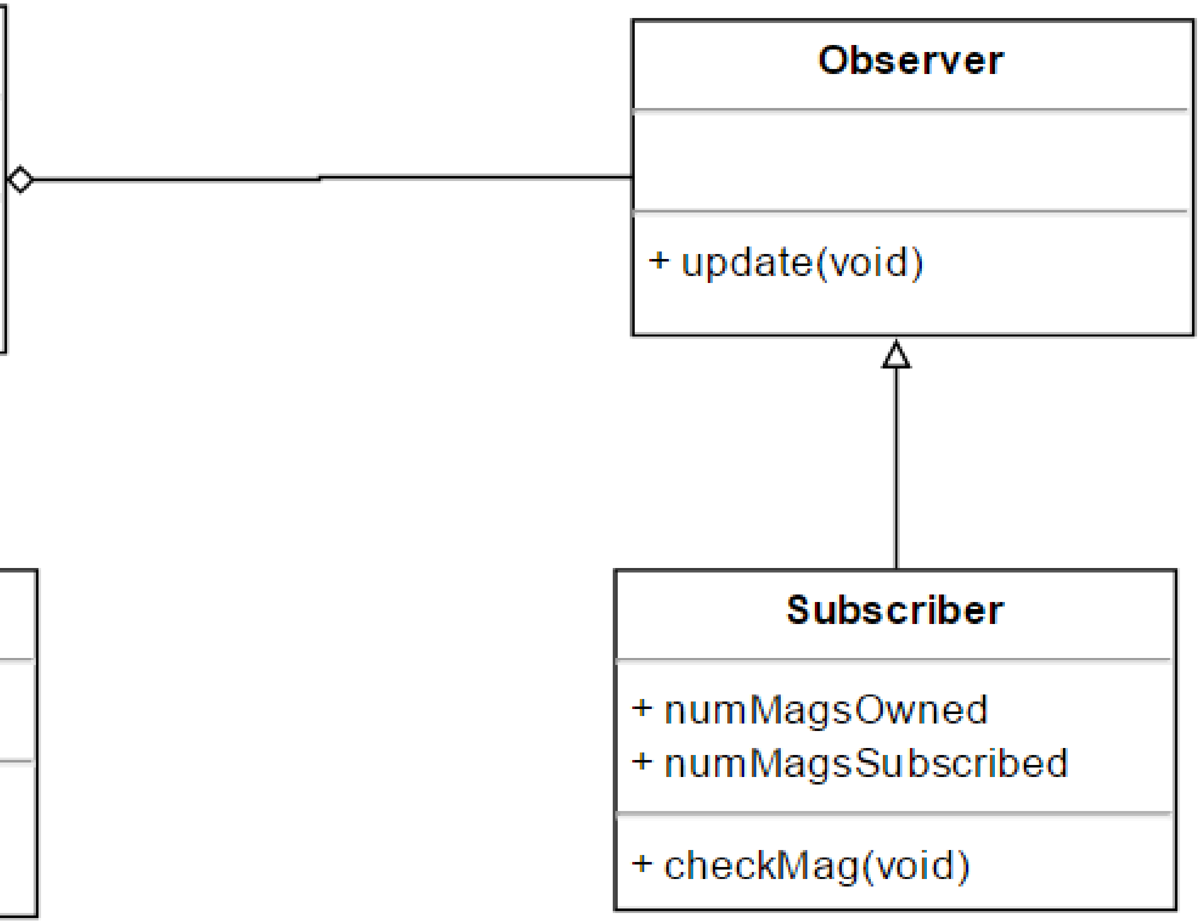

Subject
+attach(in Observer)
+setState()
+getState()
views
*
model
Observer
+update()
ViewOne
+update()
ViewTwo
+update()
for each view in views
v.update()
model.getstate();

Observer


DON'T MISS! MUSIC & ADVERTISING CONFERENCE
JUNE 15-16, NYC /// MUSICANDADVERTISING.COM
Billboard
DEVO REINVENTS THE COMEBACK
APPLE, THE CLOUD AND THE CALM BEFORE THE STORM
7 KEY LESSONS FROM THE COUNTRY SUMMIT
MACY GRAY 40 AND FEARLESS
ARTISTS SPLIT ON BOYCOTT OF ISRAEL
EXCLUSIVE!
M.I.A.
On Her New Album, Trufflegate And The Art Of Controversy
+
CHART HEAT
JACK JOHNSON
KATY PERRY
INDIA
LUDACRIS

Observer

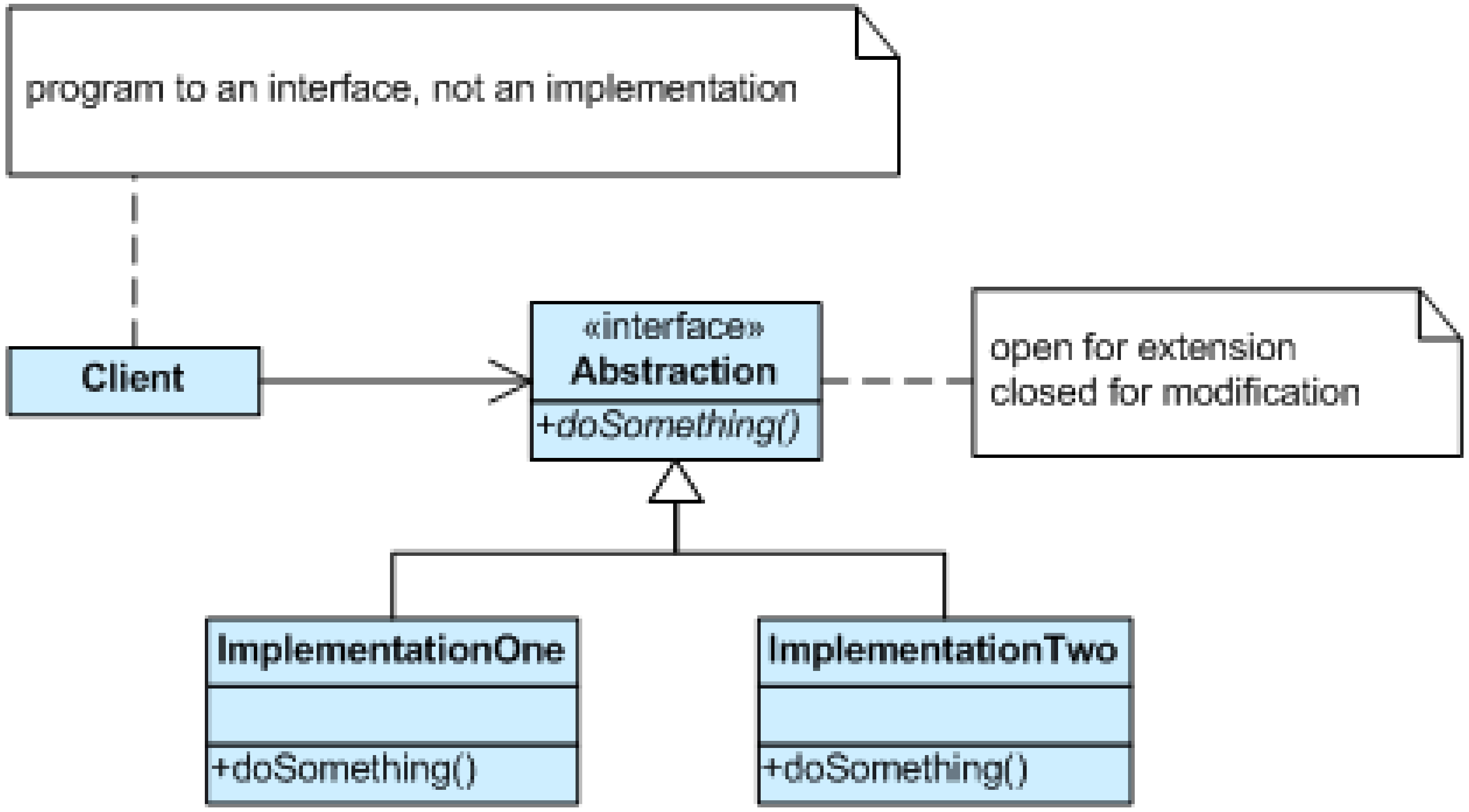

Subject
+ list<observer>
+ add(Observer)
+ notifyObservers()
Magazine
+ numOfMagazinesPublished
+ publishMag


Observer
+ update(void)
Subscriber
+ numMagsOwned
+ numMagsSubscribed
+ checkMag(void)

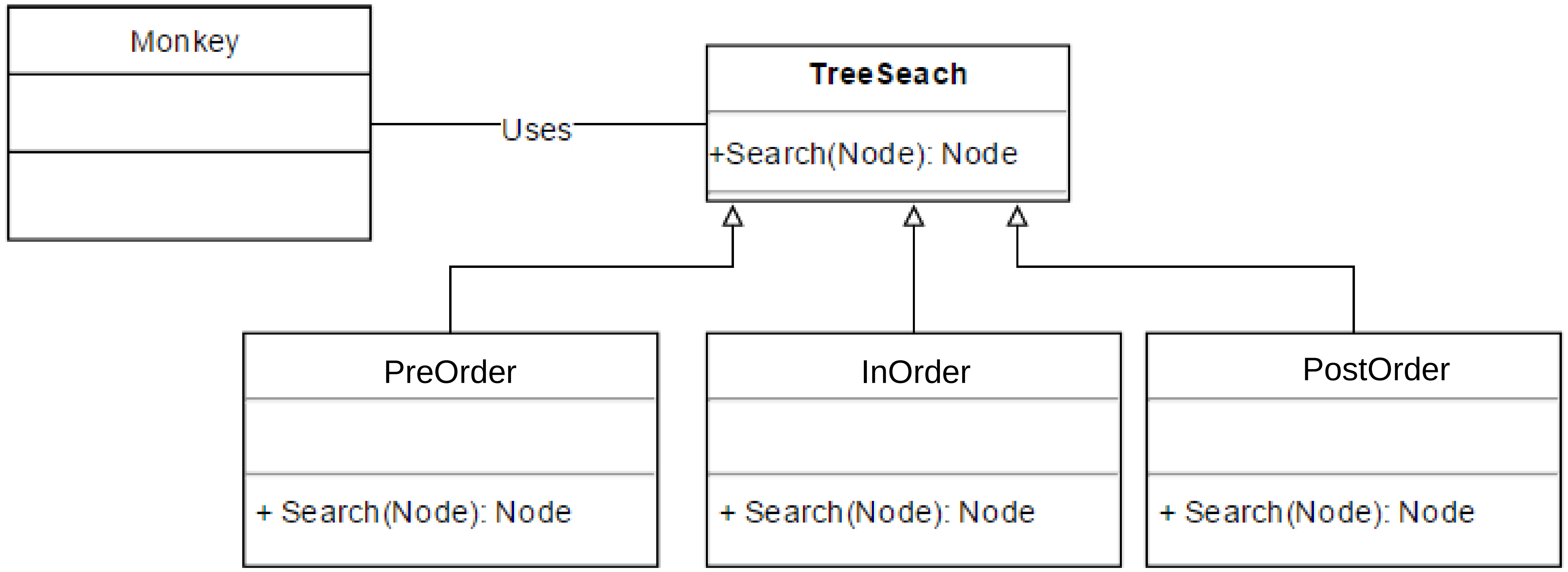
Strategy

Problem:
The **same object** needs to have **different behaviours** at different times when running the program

Strategy


program to an interface, not an implementation
Client
«interface»
Abstraction
+doSomething()
open for extension
closed for modification
ImplementationOne
+doSomething()
ImplementationTwo
+doSomething()

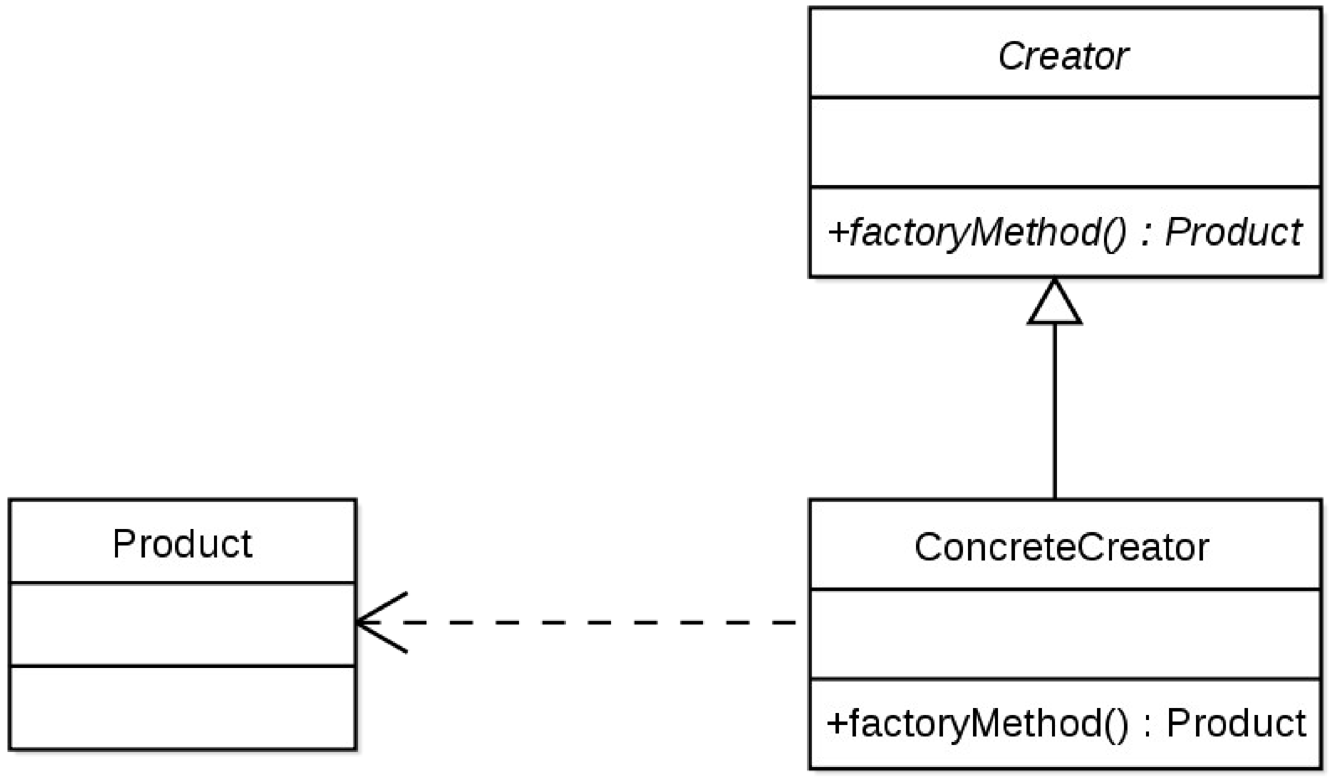
Strategy


Monkey
Uses
TreeSeach
+Search(Node): Node
PreOrder
+ Search(Node): Node
InOrder
+ Search(Node): Node
PostOrder
+ Search(Node): Node

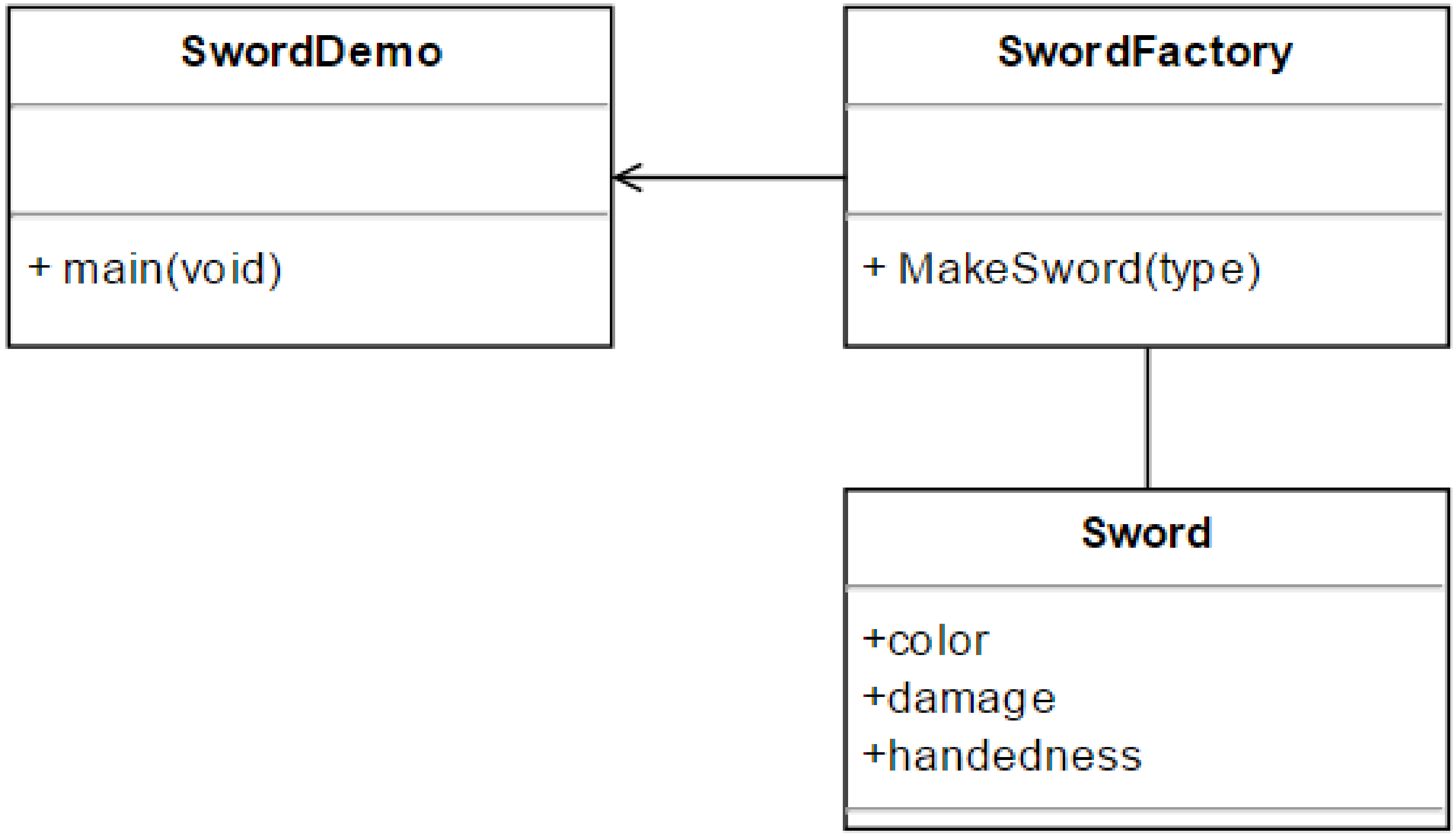
Factory

Problem:
How can we **create objects at runtime** without cluttering code with many if else statements

Factory


Creator
+factoryMethod() : Product
Product
ConcreteCreator
+factoryMethod() : Product

Factory

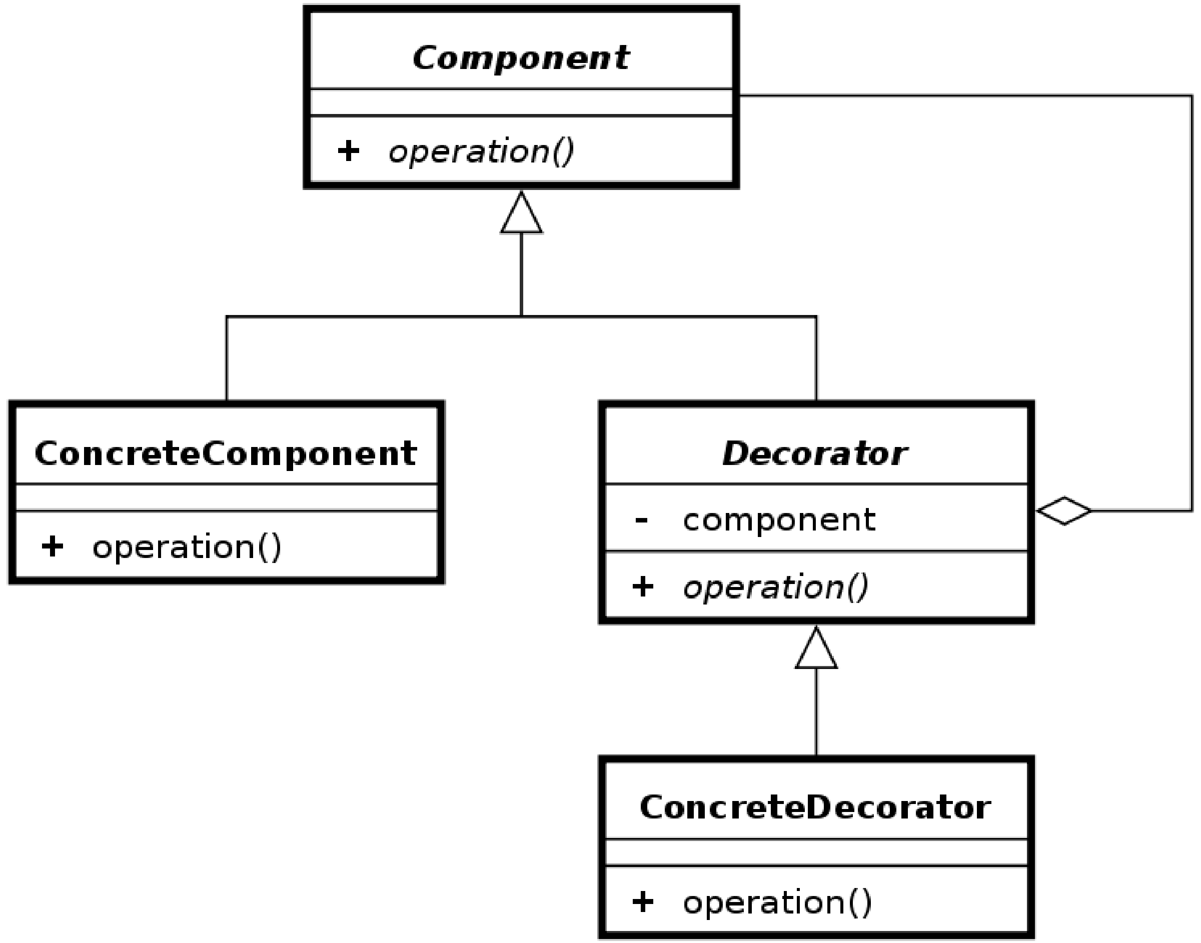

SwordDemo
+ main(void)
SwordFactory
+ MakeSword(type)
Sword
+color
+damage
+handedness

Decorator Pattern

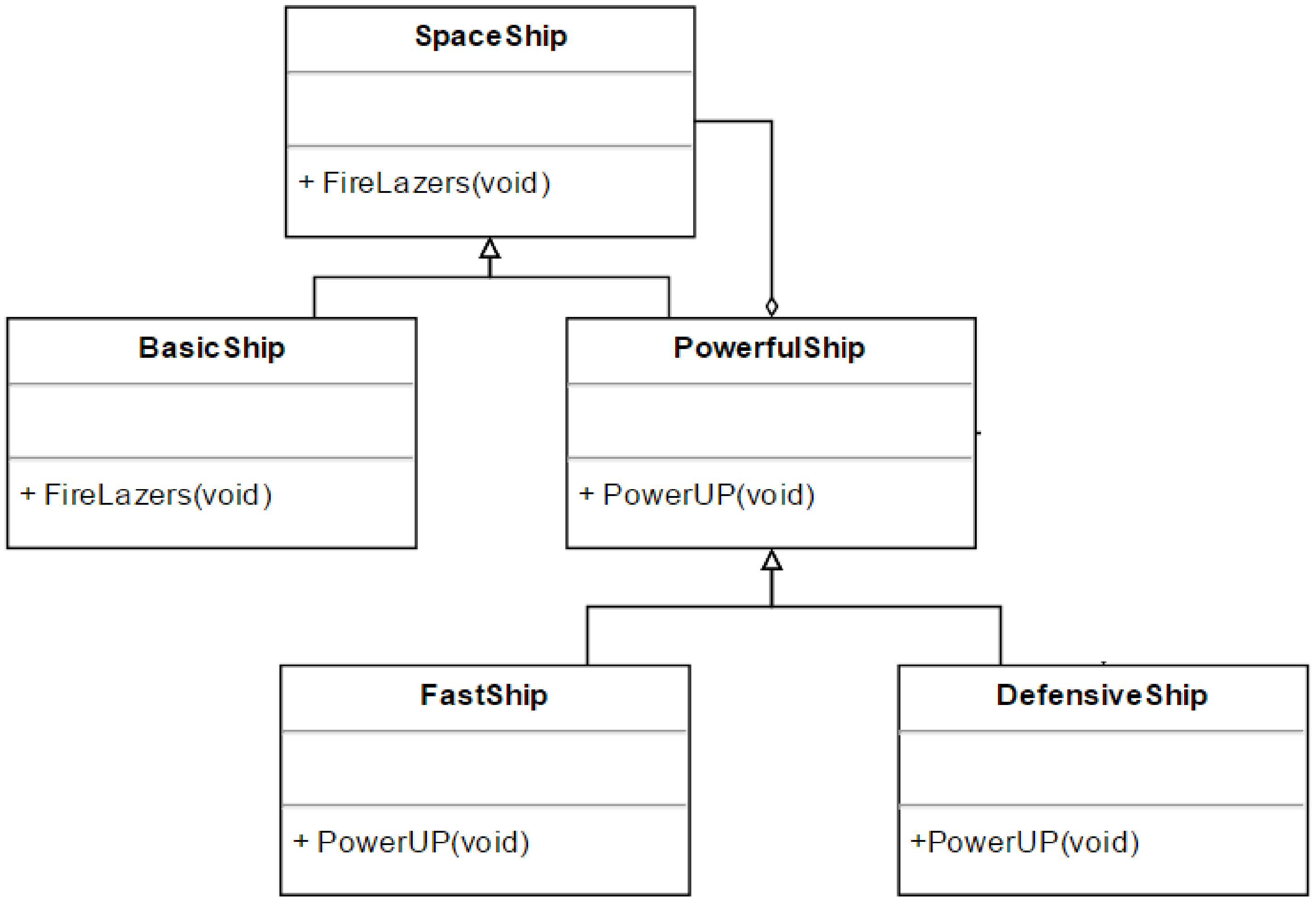
Problem:
How can we add **functionality** to a class at **runtime** instead of using inheritance at compiletime?

Example: See Java inputstream

Decorator Pattern



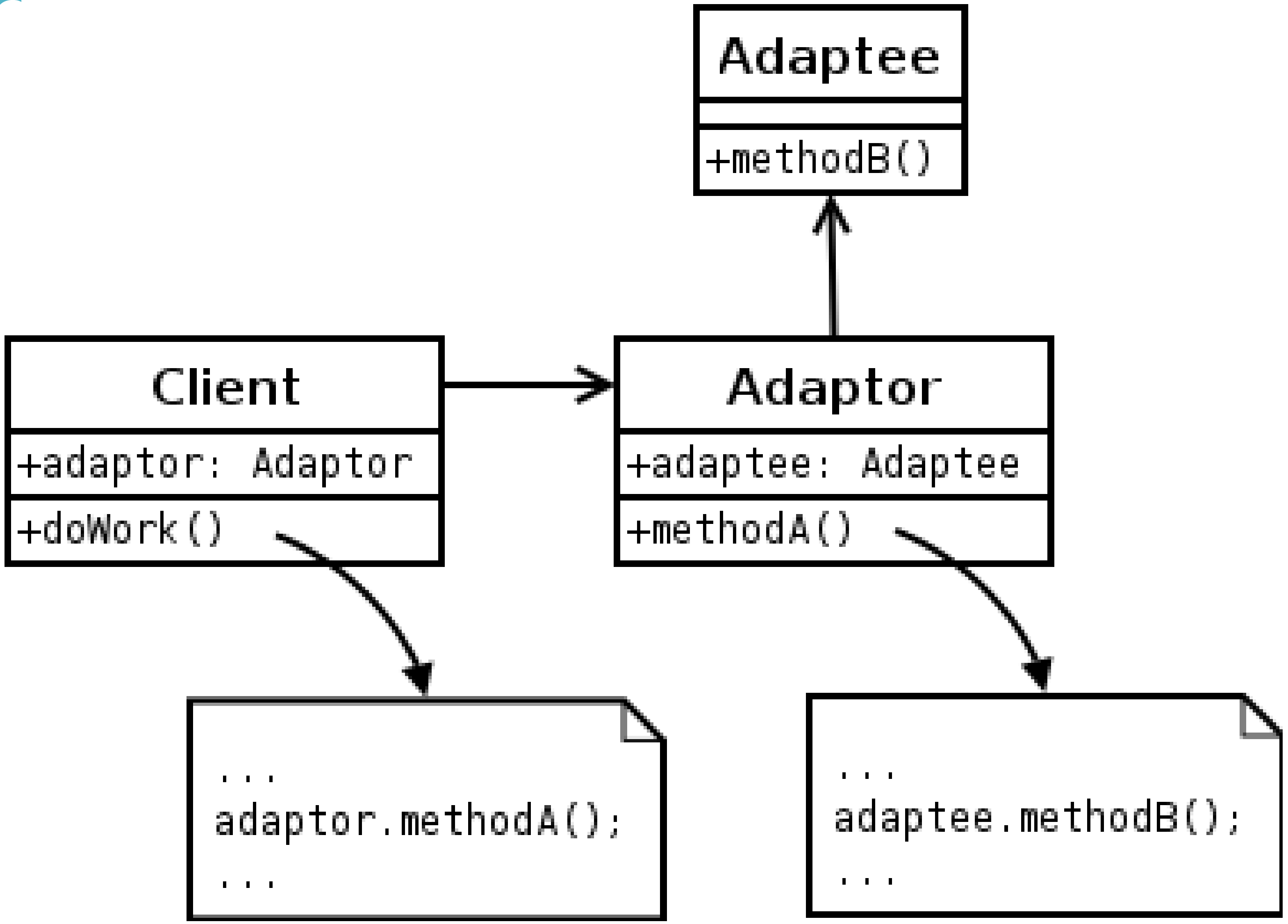
Decorator Pattern



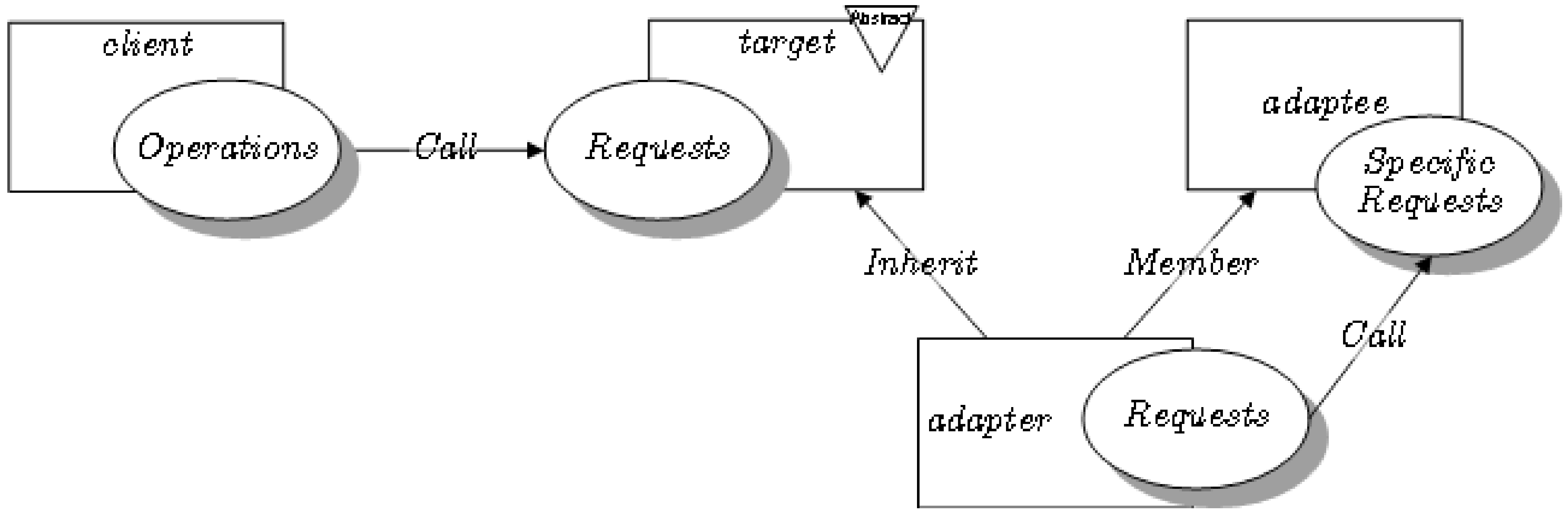
Adapter

Problem:
How can we make an **old piece of code** (eg. code taken from the internet) function with our **current project**?

Adapter


Adaptee
+methodB()
Client
+adaptor: Adaptor
+doWork()
Adaptor
+adaptee: Adaptee
+methodA()
...
adaptor.methodA();
...
...
adaptee.methodB();
...

Adapter

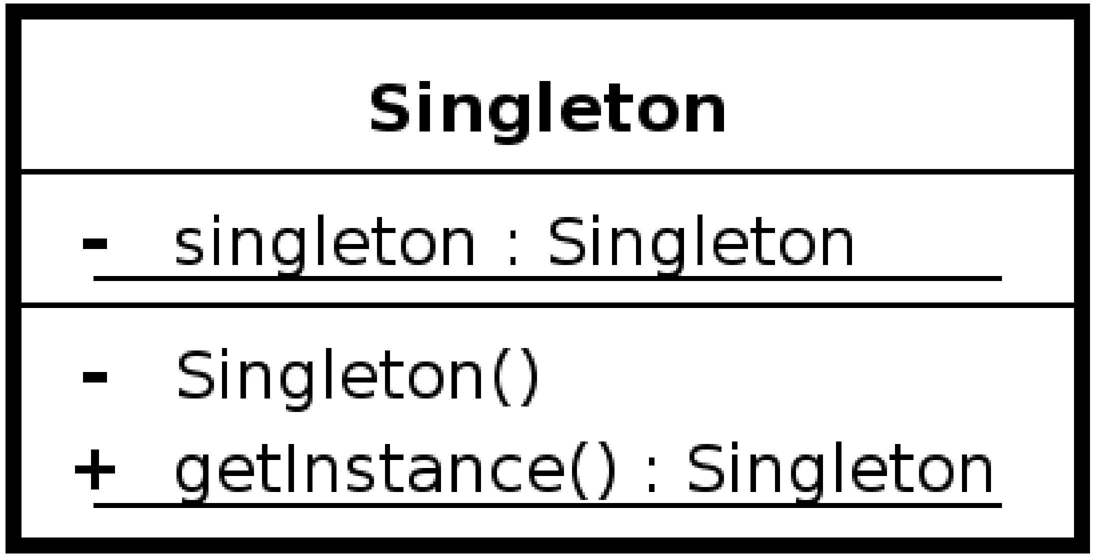

client
Operations
Call
Requests
target
Abstract
Inherit
Member
adaptee
Specific Requests
Call
adapter
Requests

Singleton

Problem:
How can we make sure an object gets **created once** only during the program.

Bonus: if its created once only, can all other classes know and use this object?

Singleton


Singleton
- singleton : Singleton
- Singleton()
+ getInstance() : Singleton

Singleton

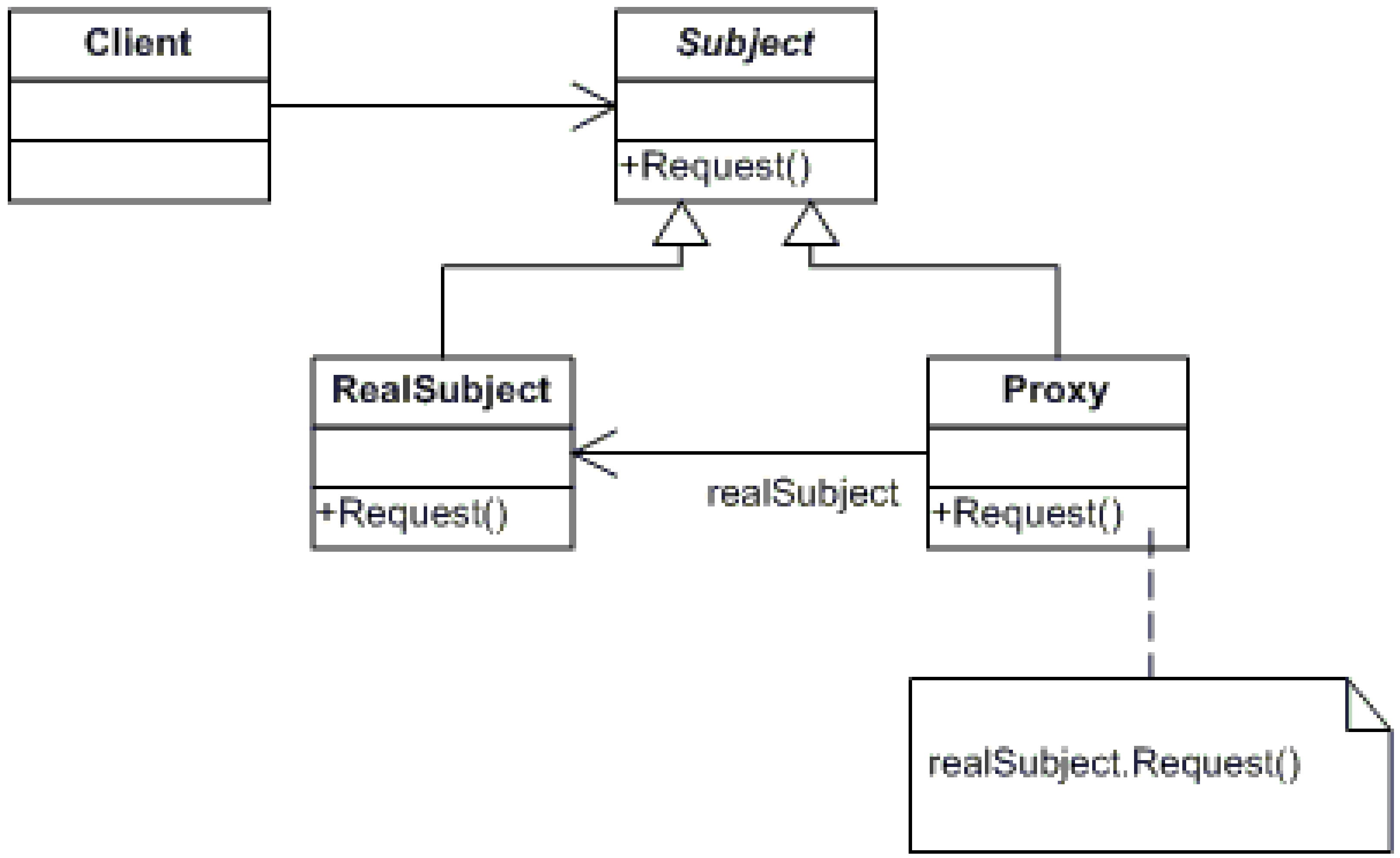
Steps:

1. Add a static instance of the class
2. Make constructor Private
3. Make a static getter method for the instance

Proxy

Problem:
How can we hide complexity of a class?

Proxy



Client
Subject
+Request()
RealSubject
+Request()
Proxy
+Request()
realSubject
realSubject.Request()

Proxy

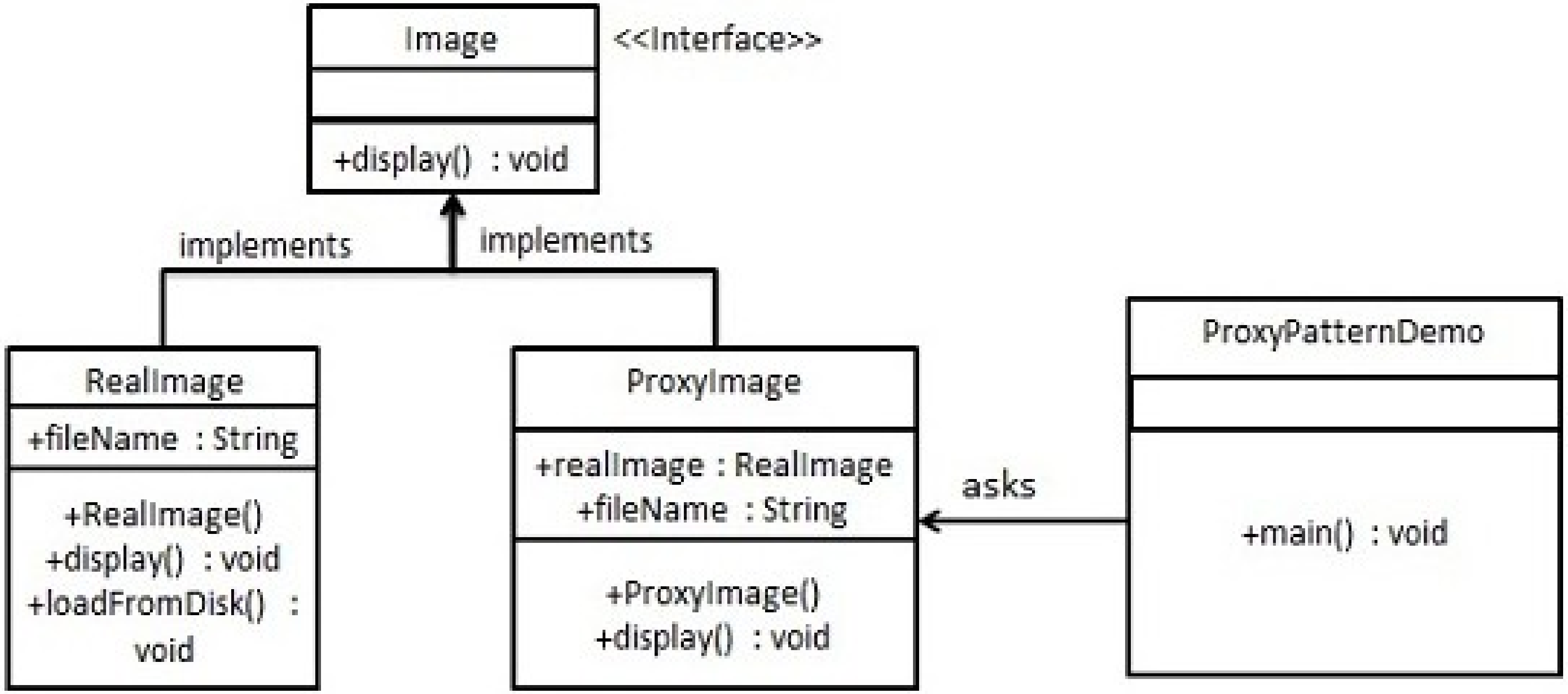


Image
<<Interface>>
+display() : void
implements
implements
RealImage
+fileName : String
+RealImage()
+display() : void
+loadFromDisk() : void
ProxyImage
+realImage : RealImage
+fileName : String
+ProxyImage()
+display() : void
asks
ProxyPatternDemo
+main() : void

References

1. http://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
2. Factory: https://www.youtube.com/watch?v=ub0DXaeV6hA