Strategy & Bridge ~~& Factory~~

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 10 — 02/22/19

Goals of the Lecture

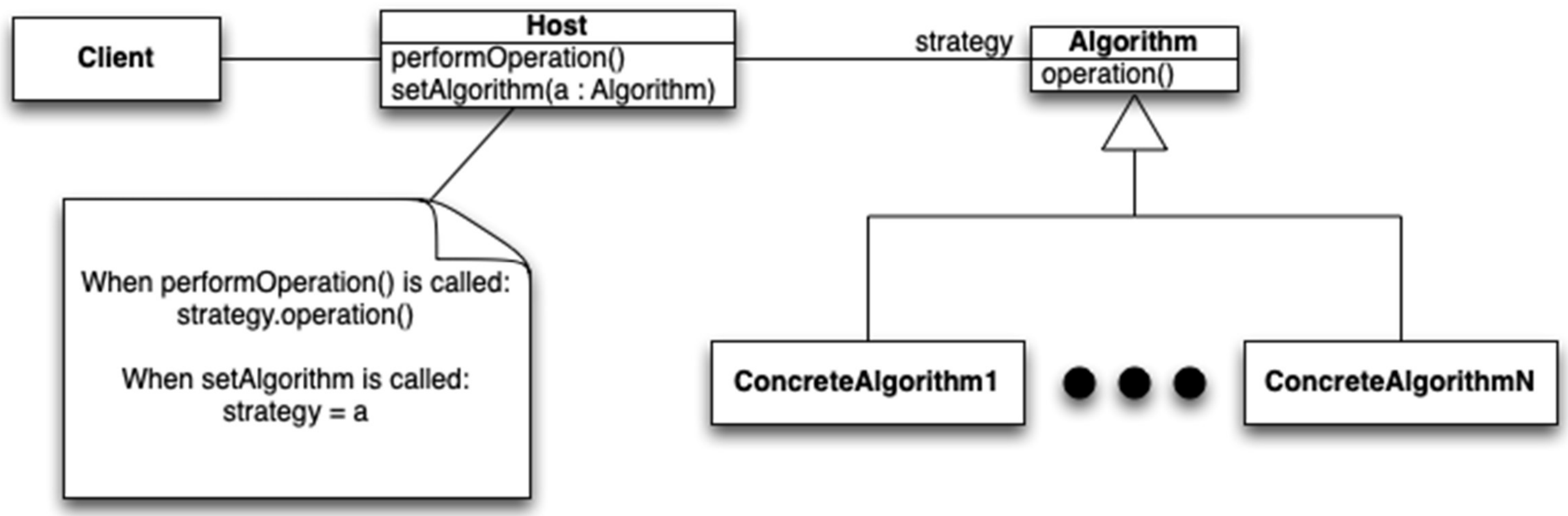
- Cover the material in Chapters 9, 10 & 11 of our textbook
  - The Strategy Pattern
  - The Bridge Pattern
  - ~~The Abstract Factory Pattern~~

- We won’t get to the factory patterns today, we’ll do them next time

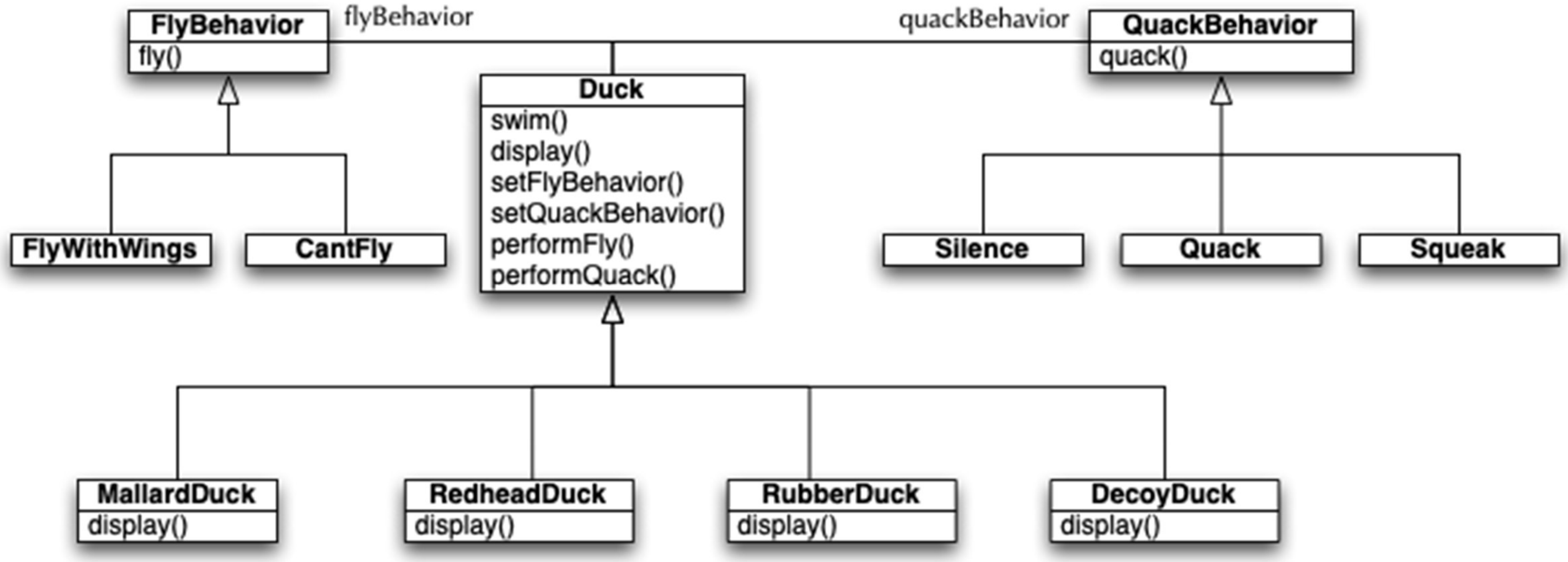
The Strategy Pattern

- We already covered this pattern in earlier lectures
- Intention
  - Lets you use different algorithms depending on context
  - Or lets you vary behavior across subclasses of some abstract class

The Structure of the Strategy Pattern



SimUDuck's use of Strategy



Duck's Fly and Quack behaviors are decoupled from Duck

The Bridge Pattern

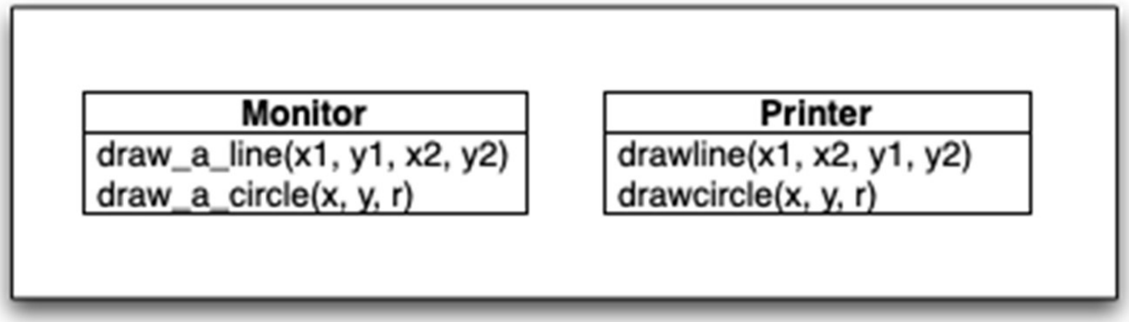
- The Gang of Four book says the intent of the pattern is to "decouple an abstraction from its implementation so that the two can vary independently"
- Tough one to get a handle on - break it down
  - Decouple – things behaving independently
  - Abstraction – conceptual relation between things
  - Implementation – here, speaking to objects the abstract class and derivations of that class are using (not the derivations of the abstract class – i.e. concrete classes)
- What it doesn't mean
  - Allow a C++ header file to be implemented in multiple ways
- What it does mean
  - Allows a set of abstract objects to implement their operations in a number of ways **in a scalable fashion**

Bottom-Up Design

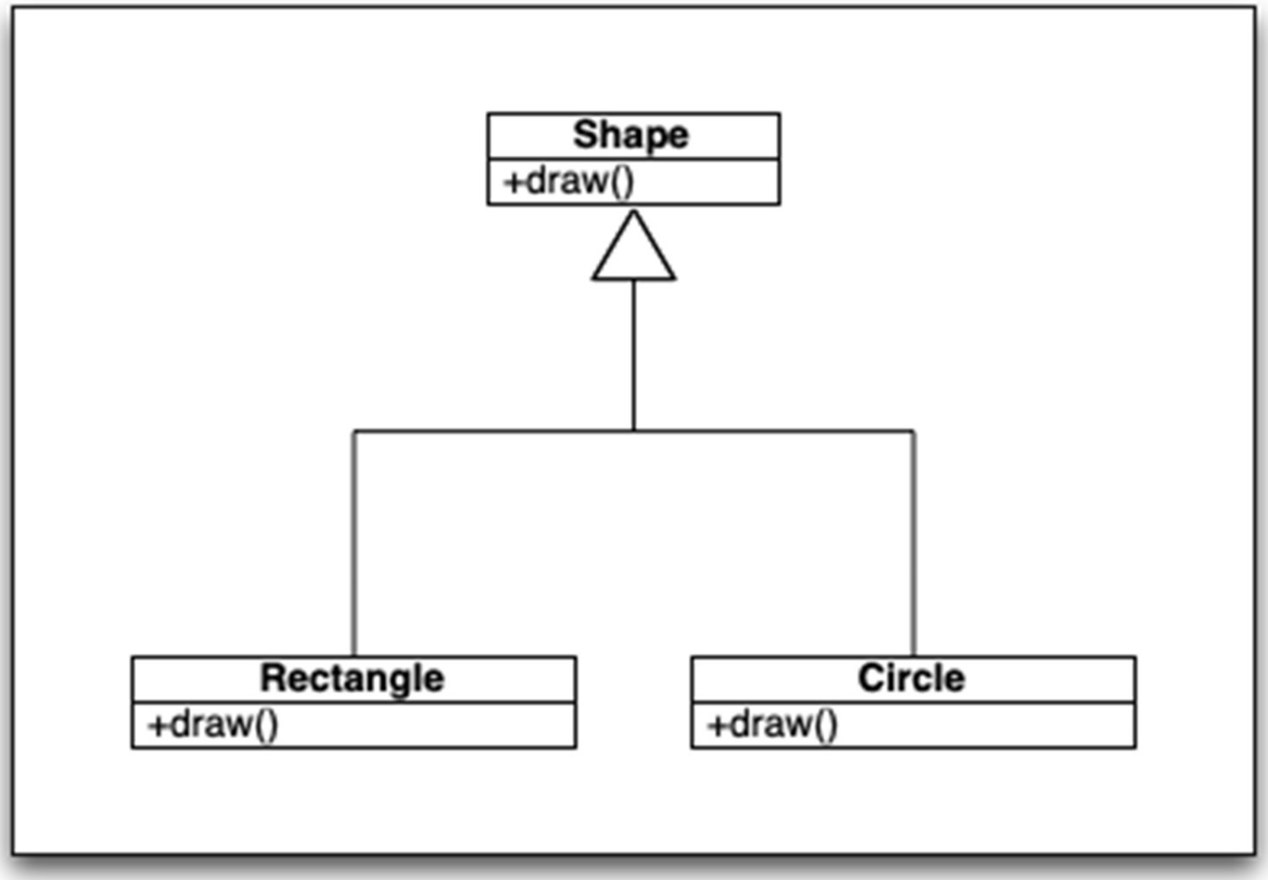
- The book presents an example that derives the bridge pattern
  - The goal of the example is to consider
  - Variations in concept abstractions
  - Variations in implementing the concepts
- Let a set of shapes draw themselves using different drawing libraries
  - Think of the libraries as items such as Monitor, Printer, OffScreenBuffer, etc.
  - Imagine a world where each of these might have slightly different methods and method signatures

Examples of Drawing Library

- The drawing library for Monitor has these methods
  - draw_a_line(x1, y1, x2, y2)
  - draw_a_circle(x, y, r)
- The drawing library for Printer has these methods
  - drawline(x1, x2, y1, y2)
  - drawcircle(x, y, r)

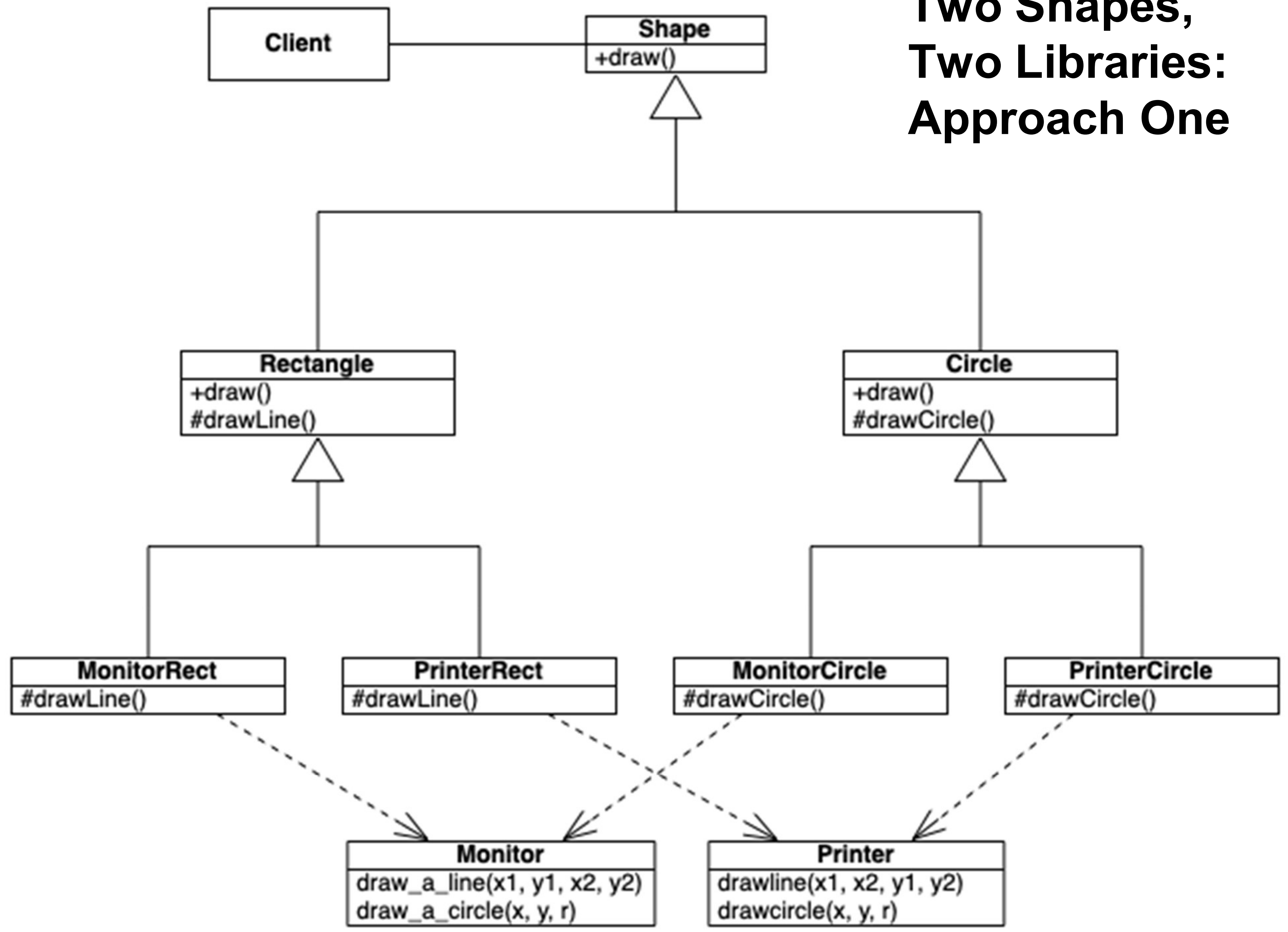


Examples of Shape

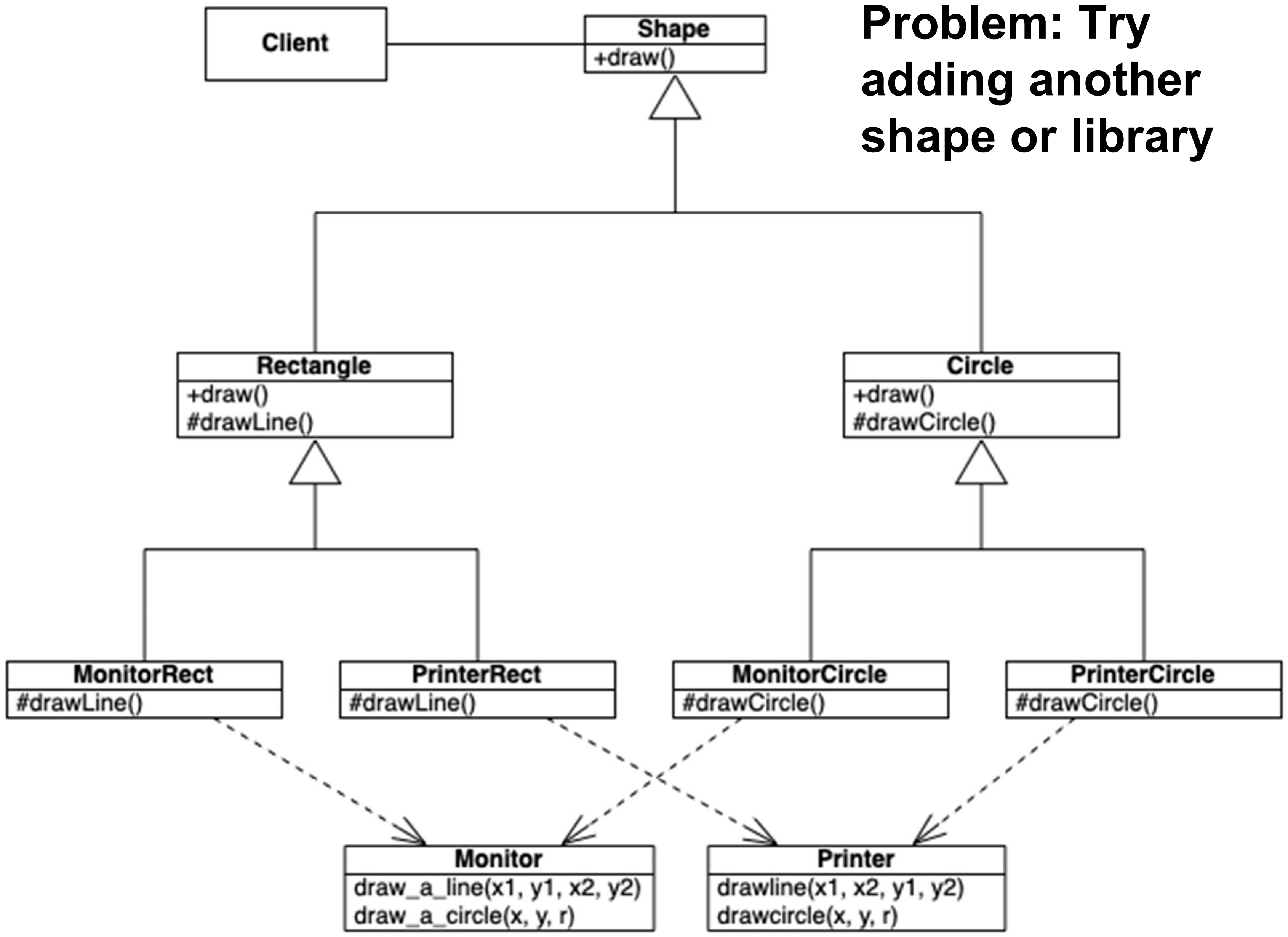


We want to be able to create collections of rectangles and circles and then tell the collection to draw itself and have it work regardless of the medium

Two Shapes, Two Libraries: Approach One



Problem: Try adding another shape or library



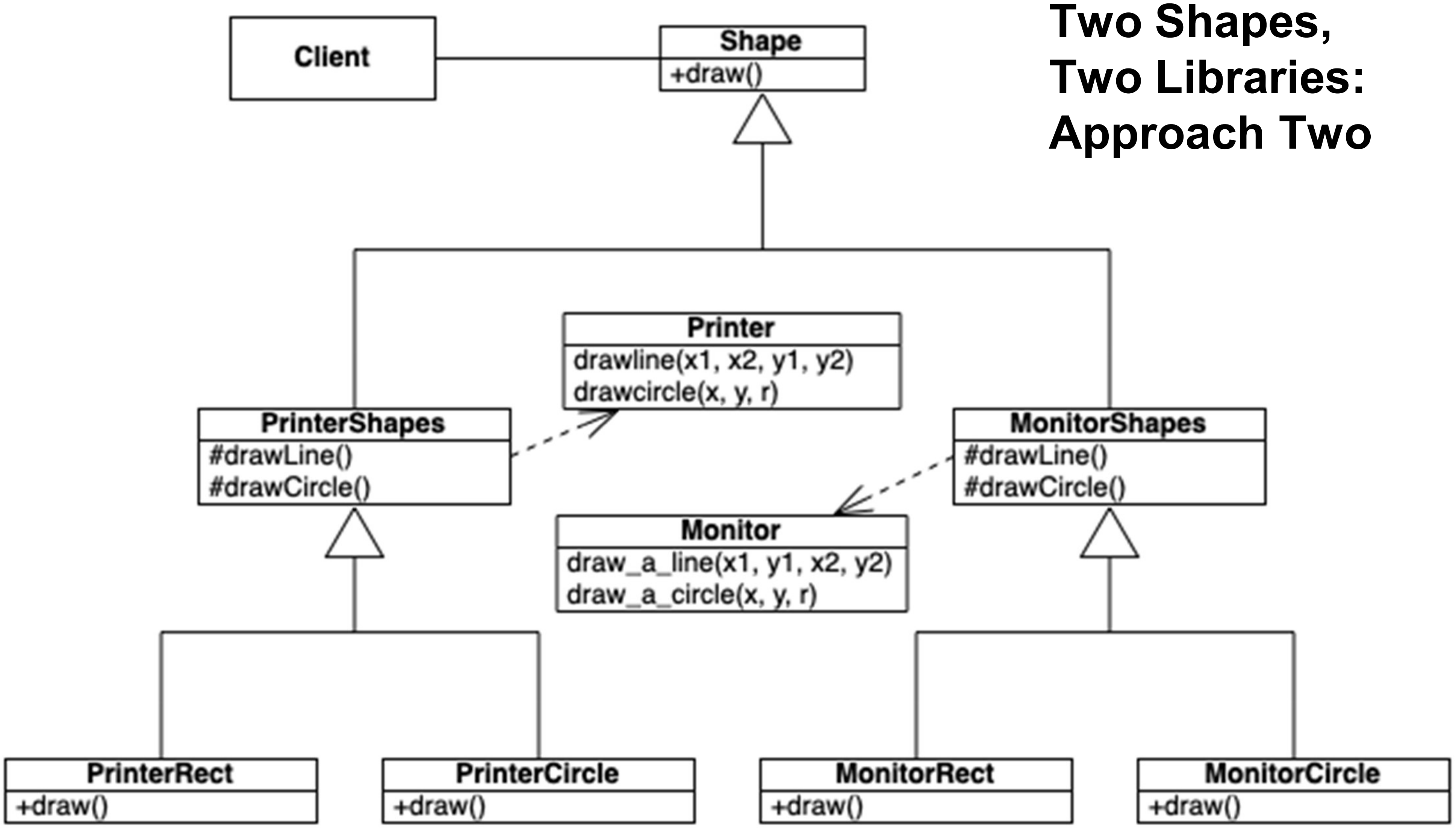
Emphasis of Problem (I)

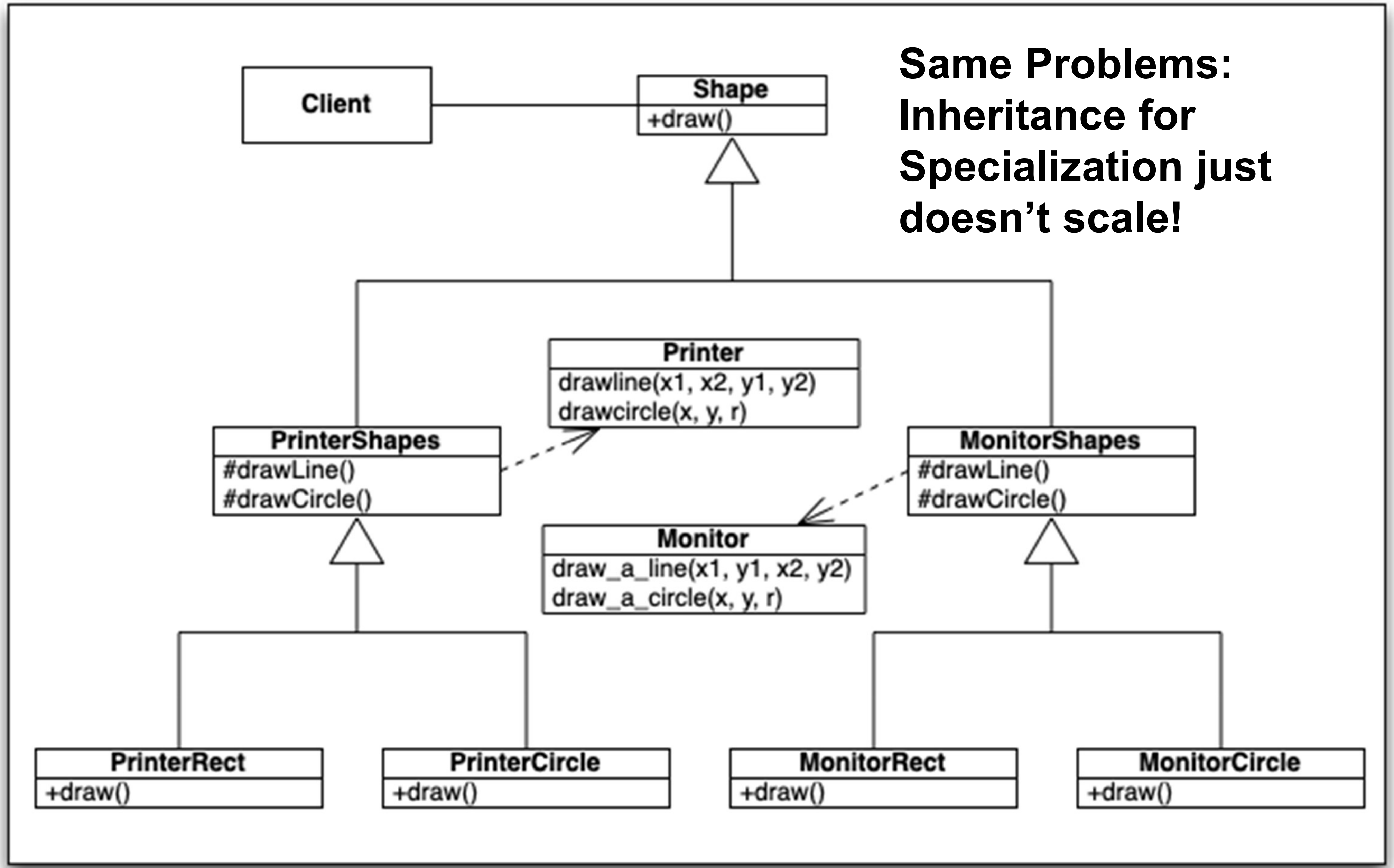
- We are using inheritance to specialize for implementation
  - And, surprise, we encounter the combinatorial subclass program once again
  - The book terms this a “class explosion”
  - 2 shapes, 2 libraries: 4 subclasses
  - 3 shapes, 3 libraries: 9 subclasses
  - 100 shapes, 10 libraries: 1000 subclasses
- Use inheritance for behavior, not specialization

Emphasis of Problem (II)

- Is there redundancy (duplication) in this design?
  - Yes, each subclass method is VERY similar
- Tight Coupling
  - You bet… each subclass highly dependent on the drawing libraries
    - change a library, change a lot of subclasses
- Strong Cohesion? Not completely, shapes need to know about their drawing libraries; no single location for drawing
- Would you want to have to maintain this code?

Two Shapes, Two Libraries: Approach Two





Same Problems: Inheritance for Specialization just doesn't scale!

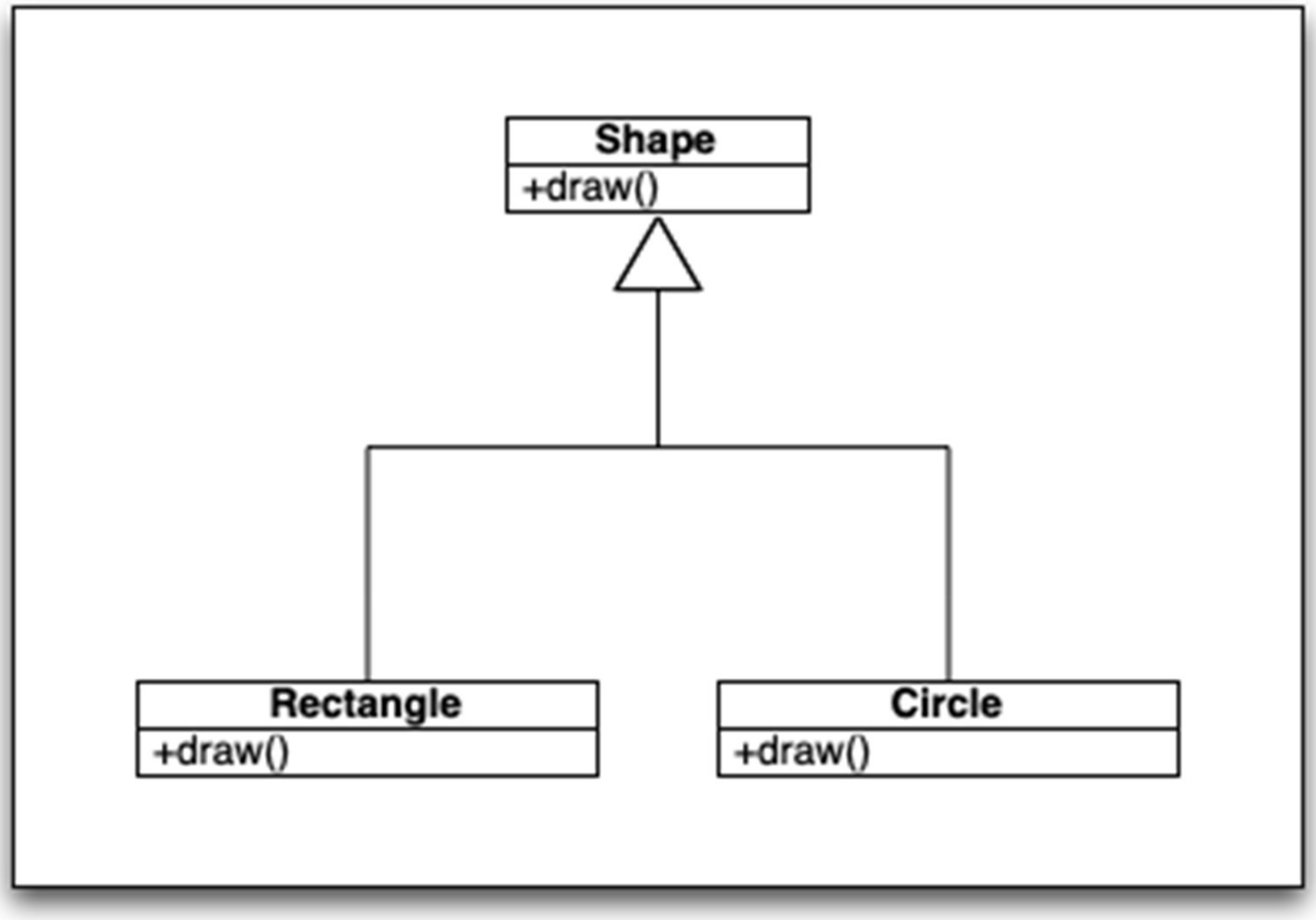
Finding a Solution

- The textbook offers two strategies to find the right solution
  - Find what varies and encapsulate it
  - Favor delegation over inheritance (book: aggregation)
  - Recognize those? Two OO Principles
- What varies?
  - Shapes and Drawing Libraries
- We’ve seen two approaches to using inheritance
  - But neither worked, let’s try delegation instead

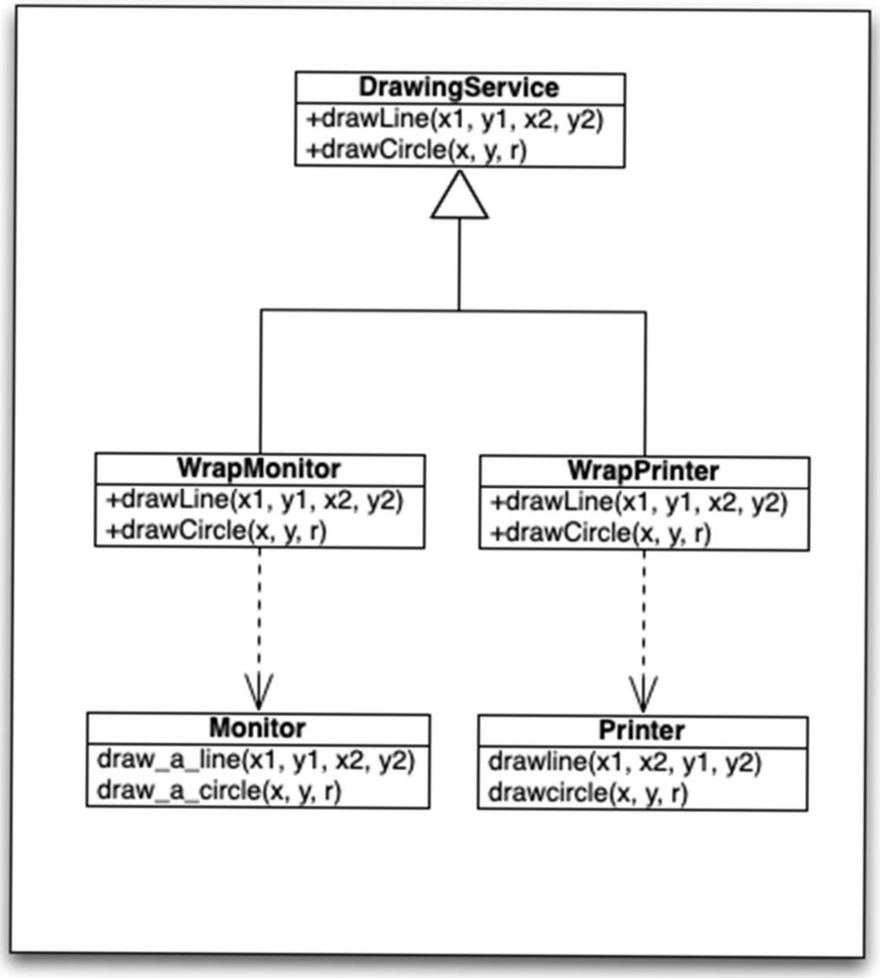
An aside: Using Design Patterns

- With patterns, designers often focus on the solutions the patterns present
- The book highlights this is starting at the wrong end
- First, understand the problem
    - Looking for where to apply a pattern tells you what to do, but not when or why
- Try to focus instead on the context of a pattern, that is, what problem is the pattern trying to solve?
    - Understand the when and why I would use this
- For the Bridge, it’s useful for abstractions that have different implementations, and it allows the abstractions and implementations to vary independently…

What varies? Shapes



What varies? Drawing Libraries

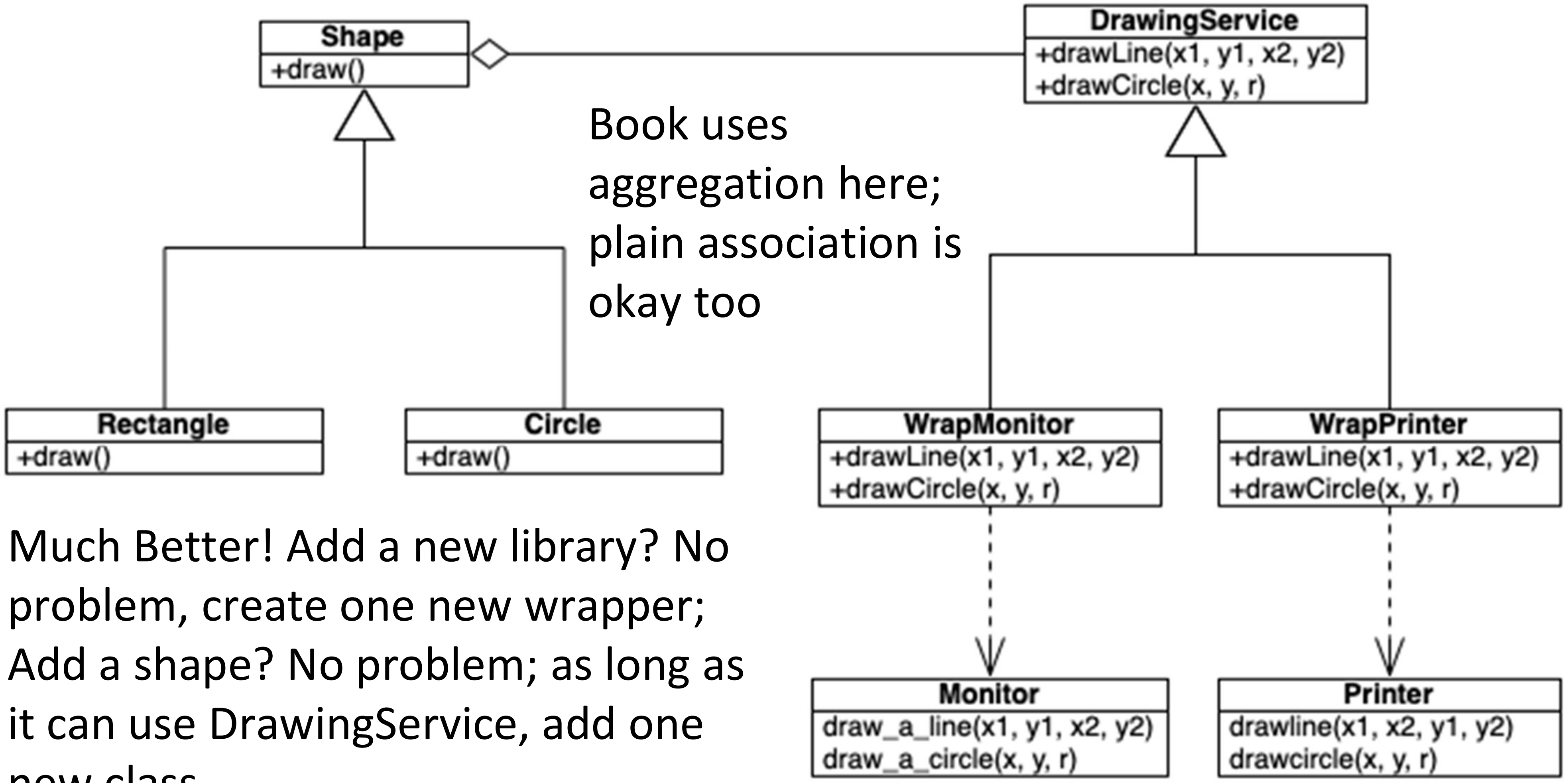


One abstract service which defines a uniform interface

The next level down consists of classes that wrap each library behind the uniform interface

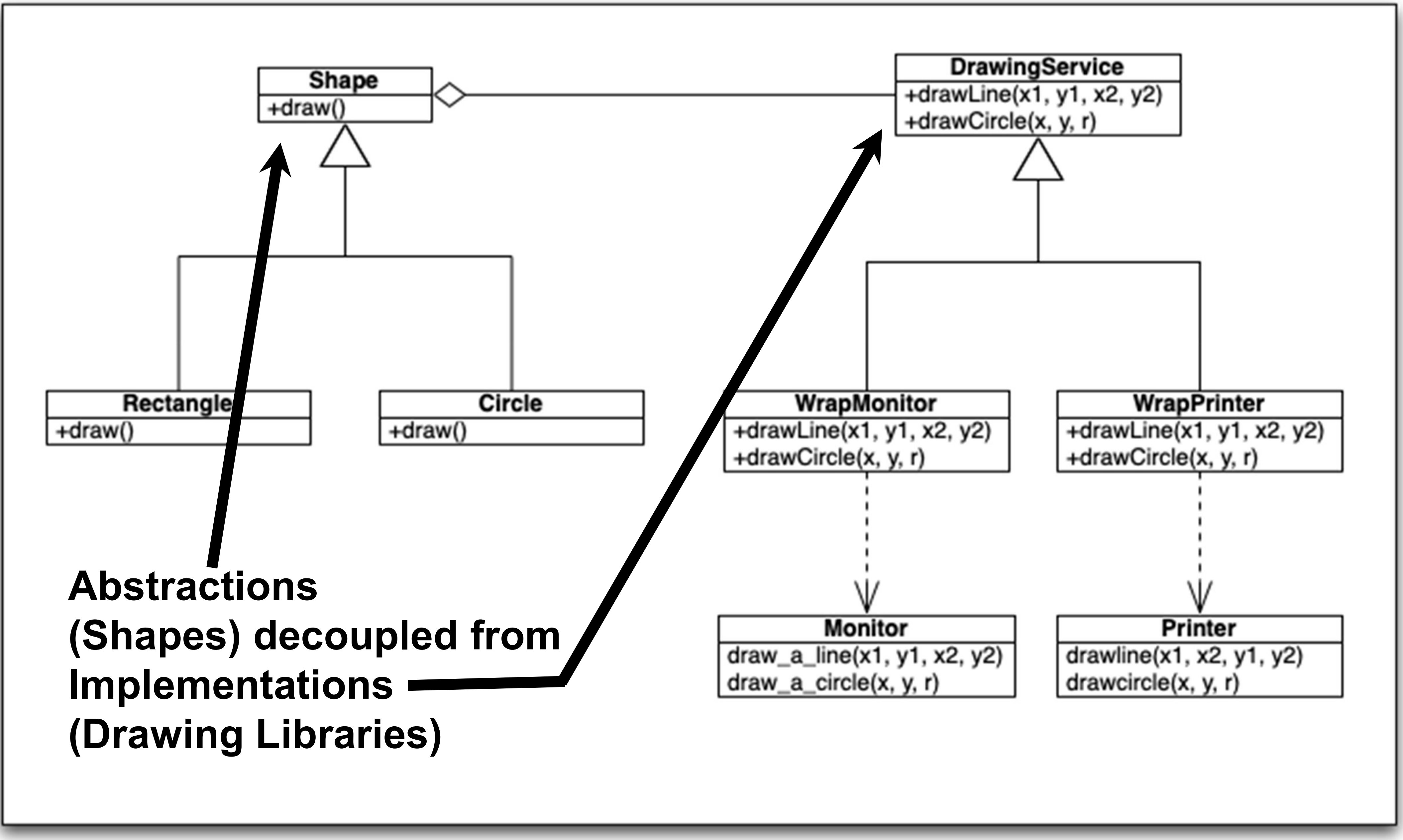
Favor delegation

- Two choices
  - DrawingLibrary delegates to Shape
    - That doesn’t sound right
  - Shape delegates to DrawingLibrary
    - That sounds better
- So…



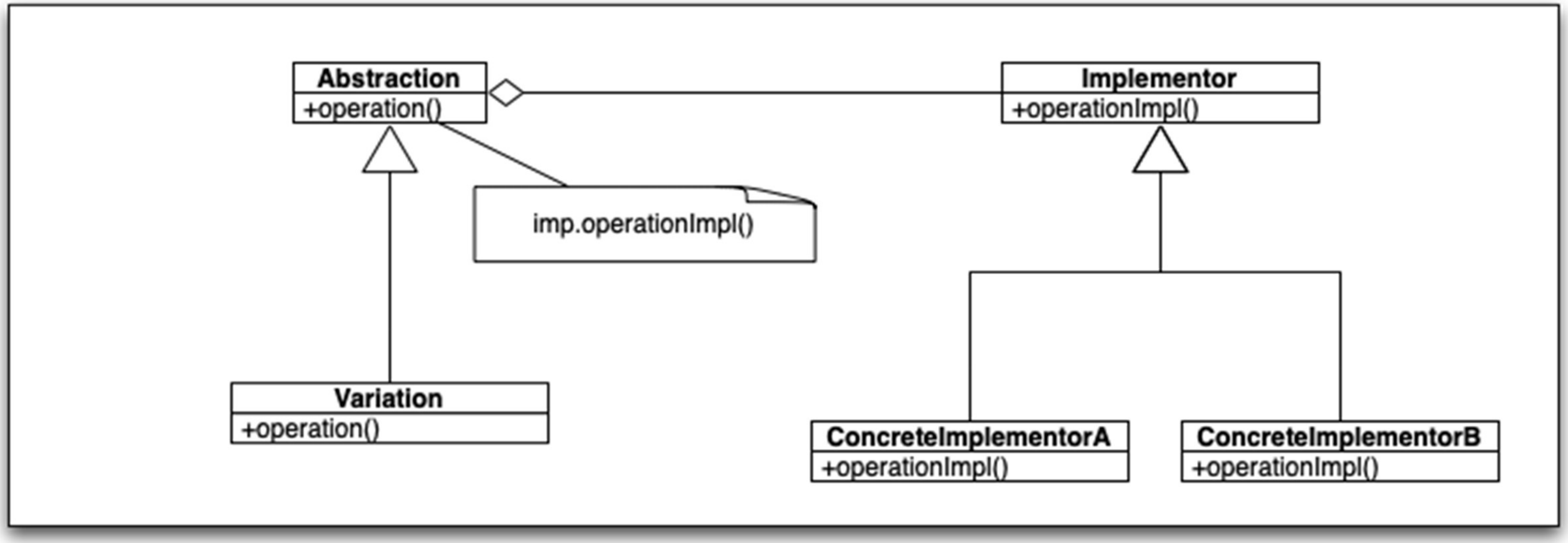
Book uses aggregation here; plain association is okay too

Much Better! Add a new library? No problem, create one new wrapper; Add a shape? No problem; as long as it can use DrawingService, add one new class



Abstractions (Shapes) decoupled from Implementations (Drawing Libraries)

The Structure of the Bridge Pattern



One Rule, One Place

- The book highlights that an important implementation strategy is to have only one place where a given rule is implemented
- This lines up with prior discussions about the “bad smell” of duplicate code (or logic)
- Generally, following this rule may result in code with a greater number of smaller methods
- This is a minimal cost that eliminates duplication and prevents future scaling and maintenance problems

- This Bridge design example also highlights the prior lecture on commonality and variability analysis…

Next time: Two Factory Patterns

- We’ll use an example from Head First Design Patterns to introduce the concept of Abstract Factory
- The example uses the Factory Method, so you get an early introduction to this pattern
    - The textbook holds off on Factory Method until Chapter 23 – we’ll get there…
- Do read chapter 11 of the textbook to prepare…

Summary

- We reviewed Strategy and worked on applying it
- We learned about Bridge and saw how it allows a set of abstractions to make use of multiple implementations in a scalable way
- We used OO Principles as well as Commonality and Variability Analysis
  - Find what varies and encapsulate it
  - Favor delegation (aggregation) over inheritance

Next Steps

- Optional additional material
  - Dr. Anderson’s lecture on Strategy, Bridge, Abstract Factory
    - You can find it on the class Canvas site under Media Gallery
    - Again, very similar material as I’m using versions of his slides…
- Next week
  - Monday 2/25 – Lecture: Finish up Abstract Factory
    - This will be the last lecture material covered on the midterm exam!
  - Wednesday 2/27 – Recitation w/Manjunath (optional)
    - Good time to review anything you’re fuzzy on for the midterm…
  - Friday 3/1 – Lecture: OO in Java, Homework 4: Design for Semester Project
    - Material is <u>not</u> on midterm
  - Monday 3/4 – Midterm Exam
- Things that are due
  - Grad Presentation Outline is due Mon 2/25 at 11 AM
  - Quiz 5 due Wed 2/27 11 AM, up on Saturday
  - Class Semester Project topic Canvas submission is due Friday 3/1 11 AM
  - Homework 3 is due Friday 3/1 11 AM
  - Distance students – be sure I have your proctor info
  - Accommodations for extended time exams – email me for room/time info