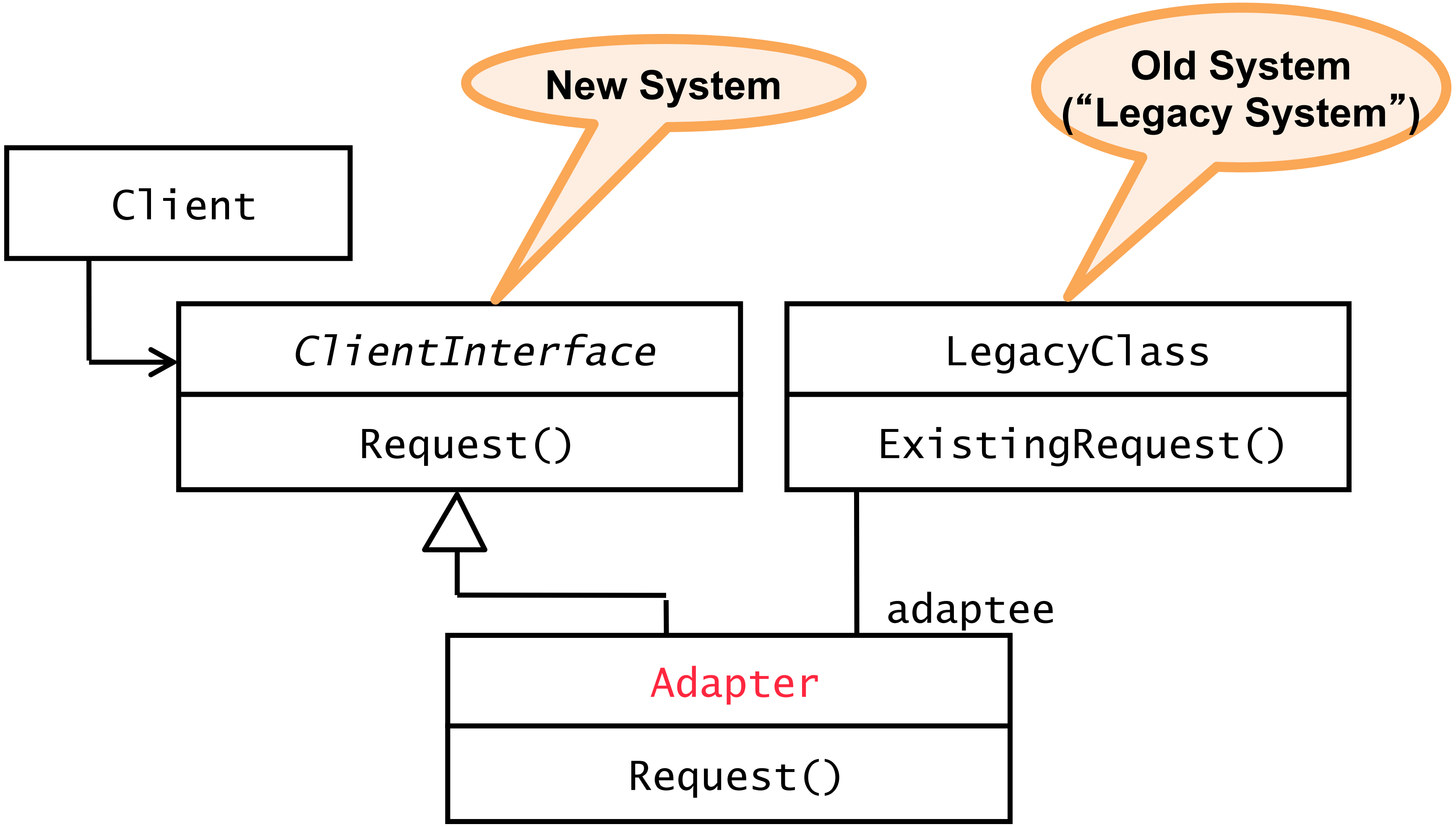
Adapter Pattern

CSCI-4448 - Boese

University of Colorado **Boulder**

Adapter Pattern



Objectives

- Problem
- Definition
- Why
- How
  - Class Adapter
  - Object Adapter
- Examples
- Design Considerations
- Façade vs. Adapter

Problem

Problem

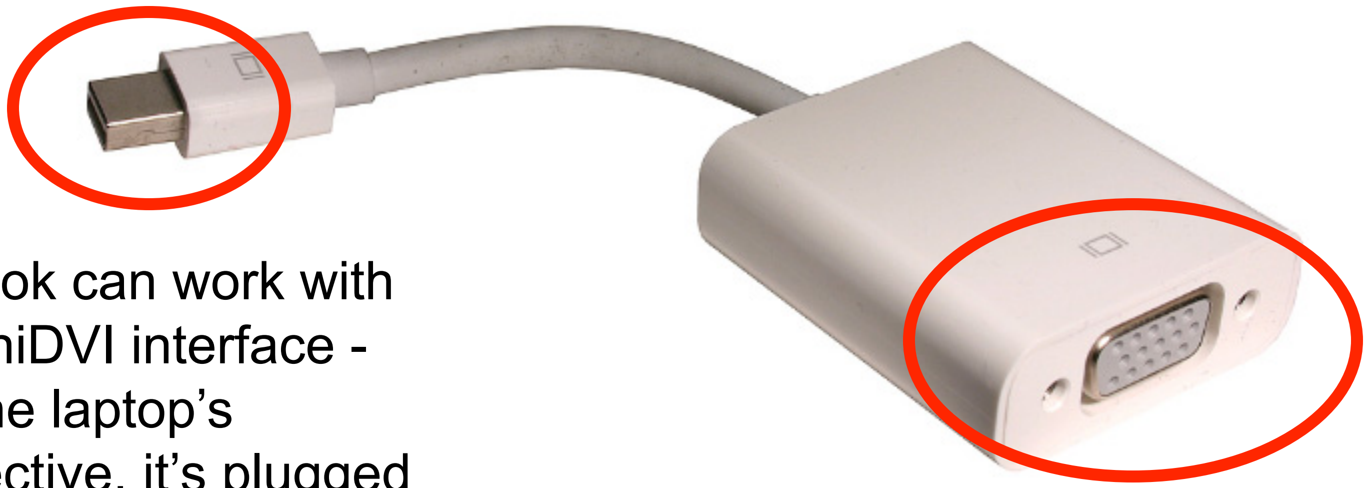
The laptop expects to work with a MiniDVI interface

Projector’s interface is a VGA port

Solution 1

Throw out the Macbook Pro, buy a Sony Viao X ($1,149.99)

Solution 2



Macbook can work with the MiniDVI interface - from the laptop's perspective, it's plugged into a MiniDVI port on the projector

The projector receives its instructions from a VGA cable - does not have to change

Buy a MiniDVI to VGA adapter ($8.99)

Definition

Definition

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces."

-Gang of Four

Definition

- **Name** “Adapter”
  - A device that is used to connect two pieces of equipment that were not designed to be connected

- **Intent**
  - Convert the interface a class to the interface some client (code) expects
  - Allows classes to work together which otherwise would not be able to due to incompatible interfaces

Why

Why use Adapter Pattern?

- You wish to incorporate some class into your project, but the existing code does not interact with the class's interface
  - You have a **Shape** superclass, with a *draw* method
    - *Circle, Square, Triangle, etc. subclasses*
  - Want to add **Text** as a shape, with *type* method
- Want to enable <u>polymorphism</u> with several classes with different interfaces.
- <u>Cost</u> of writing this new class is less than the cost of
  - Rewriting your software to be able to be used with the new class
  - Maintaining several versions of your software

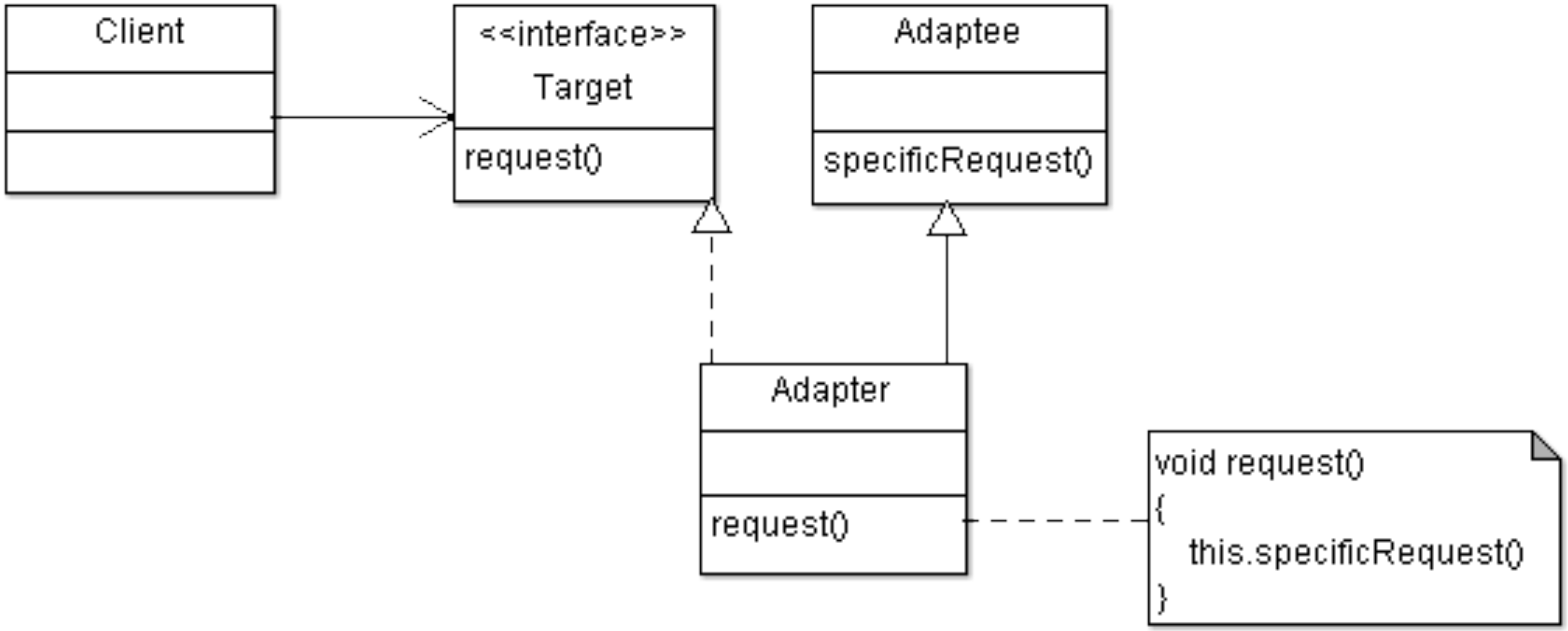
How

How

- Participants
  - **Client** – collaborates with objects conforming to the Target interface
  - **Target** – defines the domain-specific interface that the Client uses
  - **Adaptee** – defines an existing interface that needs adapting
  - **Adapter** – adapts the interface of the Adaptee to the Target interface

Option 1: Class Adapter

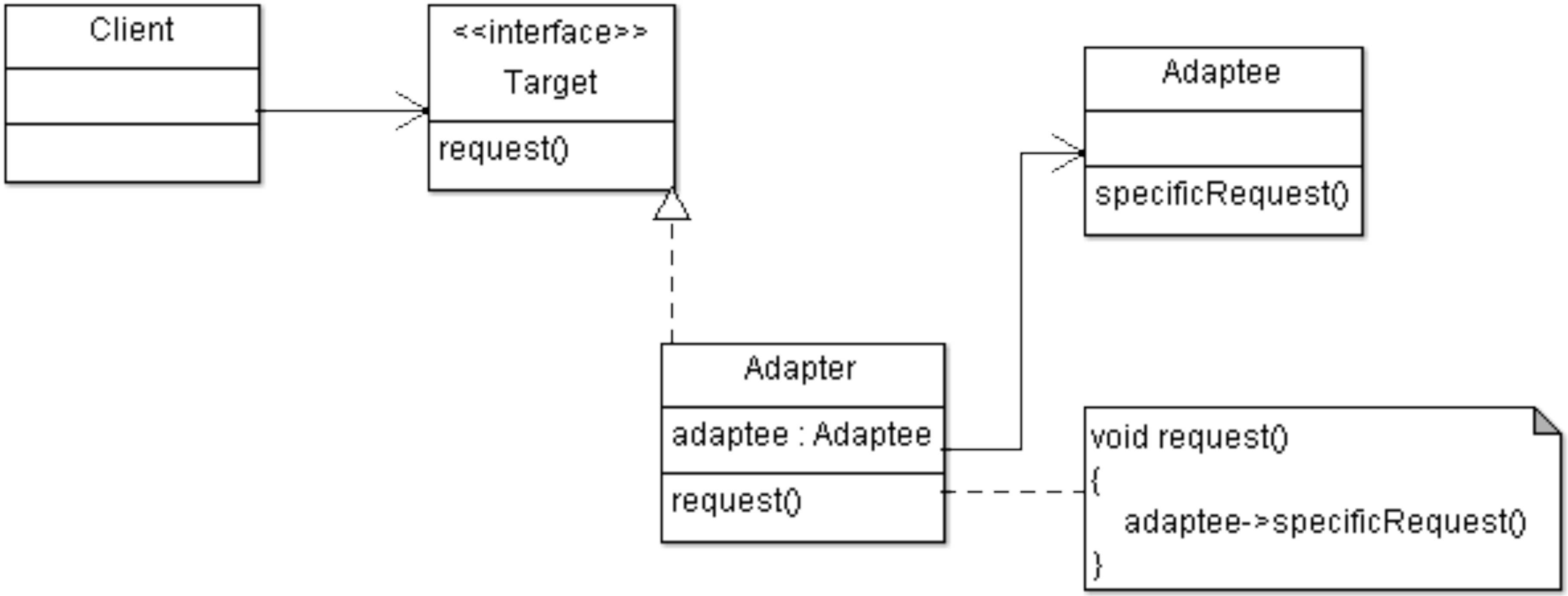
Uses (multiple) inheritance to adapt one interface to another



```
void request()
{
   this.specificRequest()
}
```

Option 2: Object Adapter

Uses delegation to adapt one interface to another



```
void request()
{
    adaptee->specificRequest()
}
```

Adapter Collaboration

- The Client object calls a method on the Adapter instance, using the Target interface
    - `myAdapter->request( );`
- The Adapter then passes this message to the adaptee
    - Class Adapter:
      Calls the desired method of the Adaptee superclass
        - `this.specificRequest( );`
    - Object Adapter:
      Delegates the call to the Adaptee object
        - `this.adaptee->specificRequest( );`

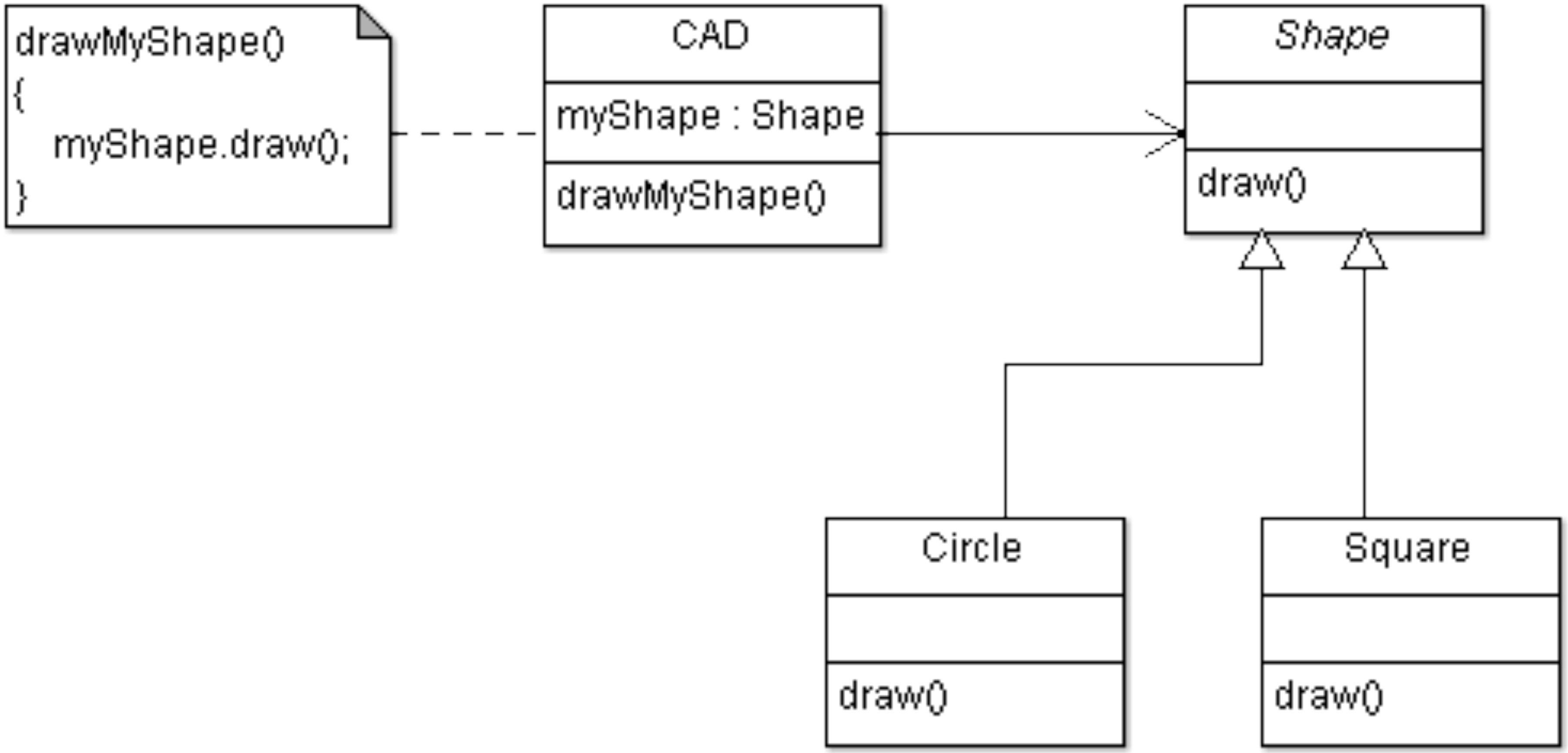
Example
Adapt to a 3rd Party API

CAD Tool

- You are developing a CAD tool, which draws various shapes
- *Shape* is an abstract class with method *draw*, and has subclasses *Circle* and *Square*
- Want to extend our CAD program to include text
  - Text is difficult to code – lots of spline math and fonts
  - Expensive to implement
- 3rd party *Text* class does exist!
  - API indicates that method *renderText* exists which performs our desired functionality

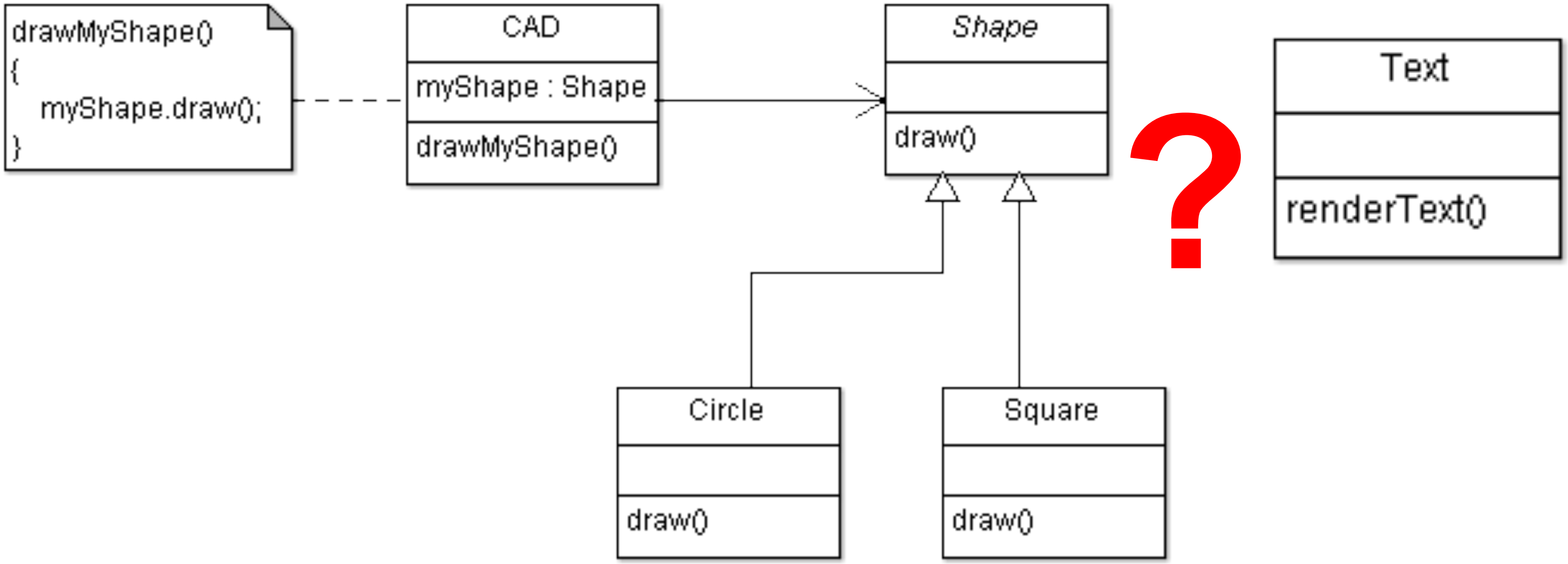
Problem

The current system implement Circle and Square



Problem

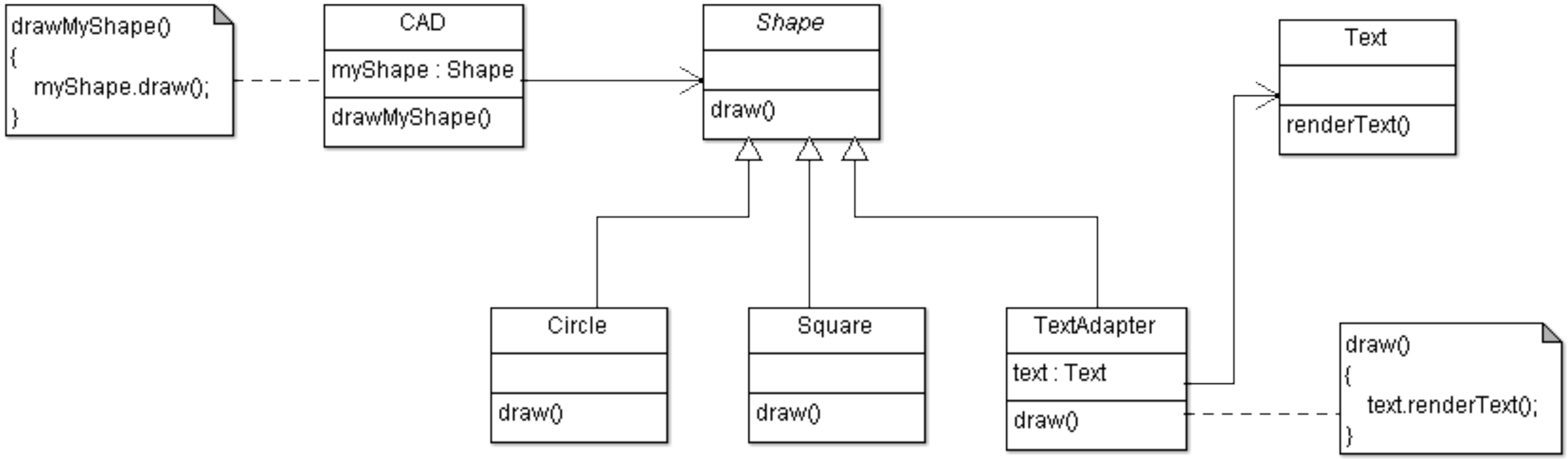
How can we incorporate Text as a subtype of Shape?



Solution – Object Adapter

Apply the object pattern solution!

– Question: What are the Client, Target, Adapter and Adaptee classes in this example?



CAD Implementaiton

```
public class CAD
{
    Shape myShape;
    public void drawTest()
    {
        // test circle
        myShape = new Circle();
        myShape.draw();
        // test square
        myShape = new Square();
        myShape.draw();
        // test text
        myShape = new TextAdapter();
        myShape.draw();
    }
}
```

TextAdapter is used the same way as Circle and Square now!

Class vs. Object Adapter

Class vs. Object Adatper

- At first glance, there seems to be little difference between a *class adapter* and *object adapter*
  - Class adapter uses inheritance to adapt the Adaptee
  - Object adapter uses delegation to adapt the Adaptee
- However, there *are* important considerations and trade-offs when selecting between the two

Tradeoffs

Class Adapter

- Commits Adaptee to a concrete Adapter class
  - Wont work when we want to adapt a class *and* its sublasses
- Let's the Adapter override some of the Adaptee's behavior
- Introduces only one object
  - no additional pointer indirection to get to adaptee

Object Adapter

- Let a single Adapter work with many Adaptees
  - Since we use delegation, we can replace adaptee with any subclass of Adaptee class
- Harder to override Adaptee behavior
  - need to subclass Adaptee

Façade vs. Adapter

Façade vs. Adapter

	Façade	Adapter
Works with existing classes?	**YES.** A *Façade* class is created to provide a ***<u>simple interface</u>*** to several already existing classes	**YES.** An *Adapter* class is created to provide a ***<u>common interface</u>*** to one or more already existing classes
Requires a superclass?	**NO.** A *Façade* class is created to interact with several different objects, and delegates work to these objects	**YES.** Each *Adapter* class will need to subclass or implement a *Target* superclass or interface
Requires polymorphism?	**NO.** A single *Façade* class is created with only the required interface to the subsystem	**Probably.** The *Target* will likely have several *Adapter* subclasses / realizations. Client selects appropriate *Target* at runtime
Simpler Interface	**YES.** The purpose of the *Façade* is to provide a more simple interface to the subsystem	**NO.** The *Target* interface is likely designed to implement to use full functionality (interface) of each *Adaptee*.

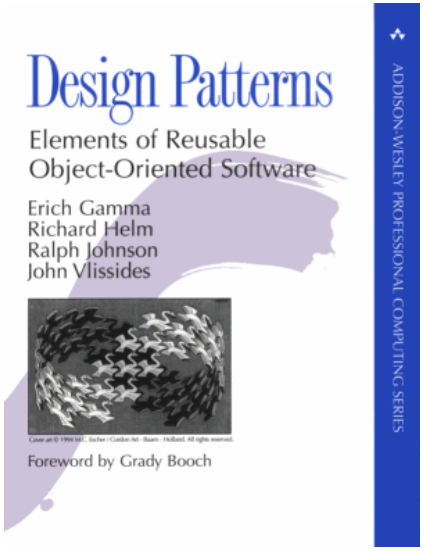
Façade vs. Adapter

- In general, the Façade pattern and Adapter pattern both interact with several already existing classes
  - A Façade is build on a subsystem consisting of many classes
    - *These classes likely interact or perform specific parts of a larger task*
  - An Adapter is build for one of many existing classes
    - *Each class performs the same or a similar task*
    - *Adaptees are unlikely to interact with one another*

Façade vs. Adapter

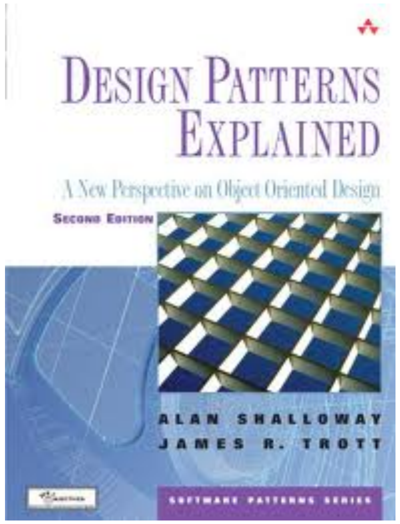
- The main purpose of a Façade pattern is to provide a ***simplified*** interface to a complex subsystem
    - *Interface leaves out many of the details of the subsystem*
    - *Each method in the interface may do a bit of work, delegating the task among several objects in the subsystem*
- The main purpose of an Adapter pattern is to provide a ***common*** interface to several individual classes
    - *Each Adaptee class performs essentially the same functionality, but using a different interface*
    - *The Adapter is designed to ensure that each of the Adaptee classes can be used interchangeably in the Client system*
    - *Adaptees are independent – they do not compose some larger subsystem*

Further Reading


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

- **Design Patterns**
  pp. 139 - 150

- **Design Patterns Explained**
  Chapter 7
  pp. 101-112


DESIGN PATTERNS EXPLAINED
A New Perspective on Object Oriented Design
SECOND EDITION
ALAN SHALLOWAY
JAMES R. TROTT
SOFTWARE PATTERNS SERIES