# Design Patterns II

## In our last episode ...

- Singleton
- Facade
- Bridge

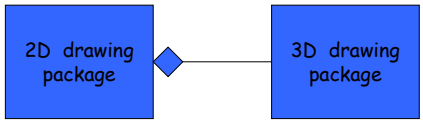
## Singleton

Problem: Want global access to a one-of-a-kind object (class)

## Facade

Problem: Want a simplified interface to a complicated subsystem.

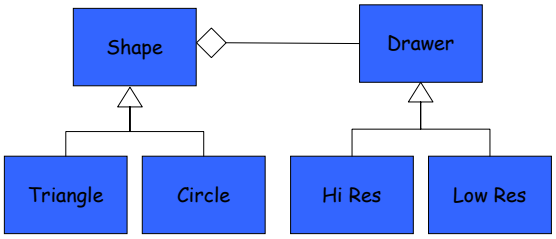
example facade




## Bridge

Problem: Want to support multiple implementation that have different interfaces in an extensible way.

example bridge




# Problem

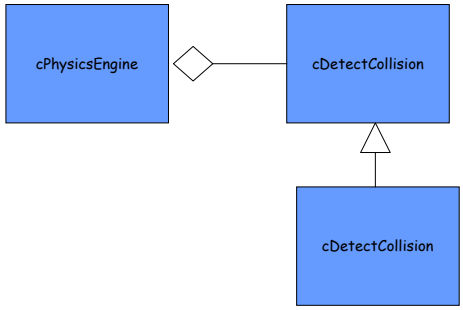
I am building a physics engine that performs collision detection between a sphere and some triangles. My physics engine class contains the following method:

```
cCollision cPhysicsEngine::detectCollision(cPath p, cTriangles t)
```

Later I plan to implement a faster but less robust algorithm that could be used on systems with slow processors.

Come up with a design that will allow for future variations in my collision detection algorithm.

Strategy Design Pattern



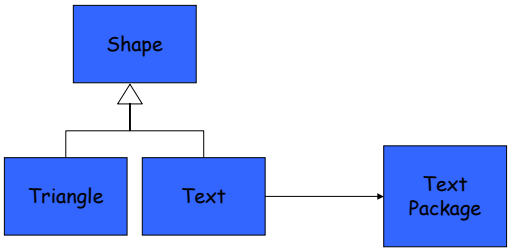

What difference (if any) is there between the bridge and the strategy design pattern?

when and how do you decide the type of constructor to evoke?

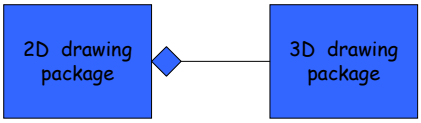
# adapter

Problem: A subsystem has the right behavior but the wrong interface.

example adapter

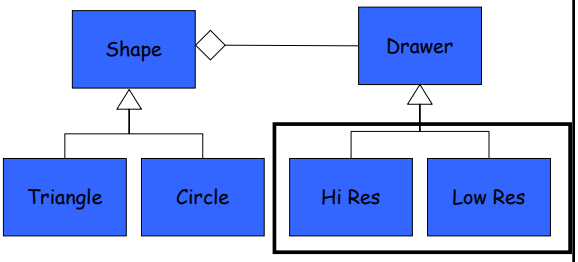



example facade




facade is a special case adapter

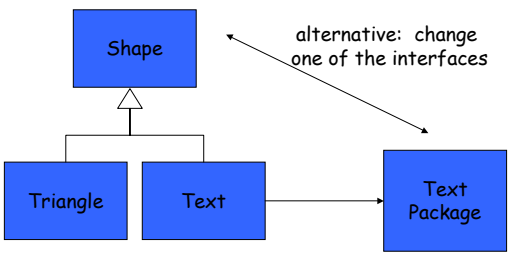
example bridge




bridge uses adapters

when is it garbage and when is it art?

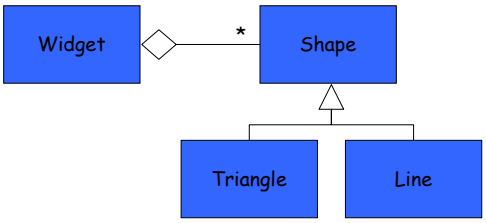
example adapter




# New problem

- I want a 2D drawing program that supports triangle and lines
- I want to be able to add, delete, draw, and move primitives.
- I want to be also want to be able to group primitives into a "widget" and treat the widget as a primitive.
- I want to be able to add and delete primitives from a widget

# Solution




# Design Principles

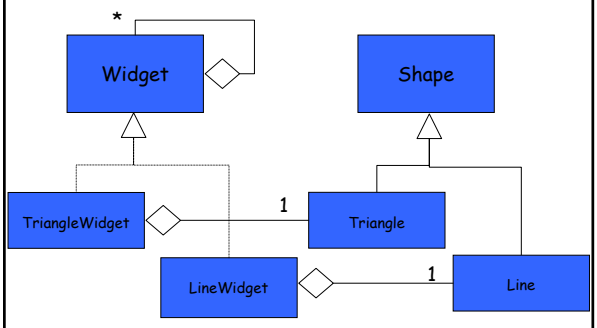
- Design to interfaces not implementation
- Favor composition over inheritance
- Find out what varies and encapsulate it
- Design highly cohesive classes that are loosely coupled
- Think like an object
- Do not forge data
- Once rule one place

# Client's code

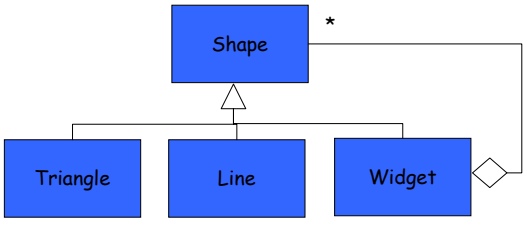
If widget then ...
Else ...

Solution: Only use widgets!

# Solution?




# Composite




We'll use a List class to manage a list of Shape pointers.

- Should List shapes be a member of Shape or Widget?
- Should add(Shape *sPtr) be a member of Shape?