Design Patterns II

In our last episode ...

- Singleton
- Facade
- Strategy

Singleton

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

Facade

Problem: Want a simplified interface to a complicated subsystem.

Strategy

Problem: Want to use a collection of interchangeable algorithms.

another problem

- I want a 2d shape class that supports lines.
- I want to draw these lines using one of two drawing programs. The exact choice of drawing program will be decided when the class is instantiated. The drawing calls are:
  - Draw program 1: drawLine(x1,y1,x2,y2)
  - Draw program 2: drawALine(x1,x2,y1,y2)
- In the future I may want to add other shapes and other drawing programs.

exercise

- draw UML diagrams for two different designs
- describe the tradeoffs

solution 1

```
draw() {
  if (drawPackage == 1)
      drawLine(v1.x, v1.y, v2.x, v2.y)
  else
      drawALine(v1.x, v2.x, v1.y, v2.y)
}
```

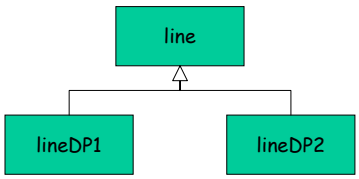
solution 1

- Advantages
- Disadvantages

solution 1

- Advantages
  - simple to implement
  - simple to understand
- Disadvantages
  - as additional shapes are added we violate a variation on the "No Forgery" principle called "One Rule, One Place"

solution 2

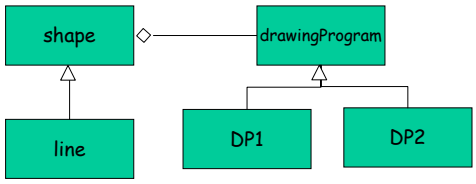


solution 2

- Advantages
  - simple to implement
  - simple to understand
- Disadvantages
  - as additional shapes and drawing programs are added the number of classes becomes LARGE

solution 3: bridge

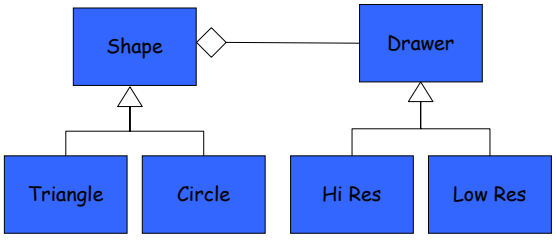
the drawingProgram class provides a uniform interface to the various drawing programs



Bridge

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

example bridge



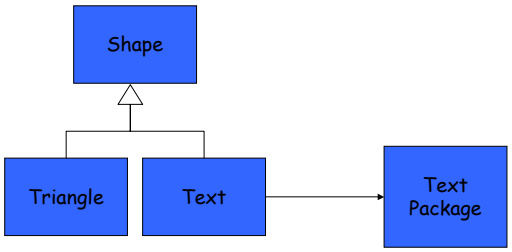
Compare/contrast

- Facade
- Strategy
- Bridge

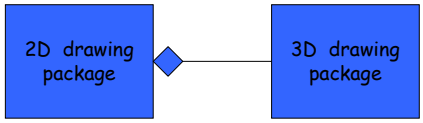
adapter

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

example adapter

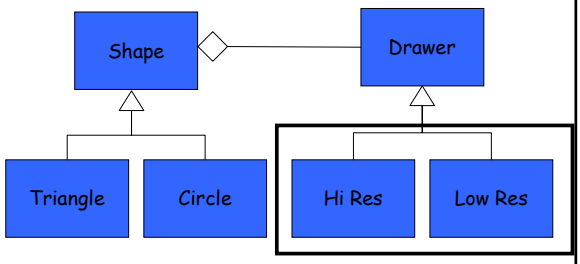


example facade



facade is a special case adapter

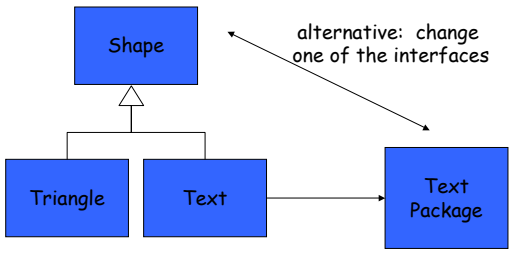
example bridge



bridge uses adapters

when is it garbage and when is it art?

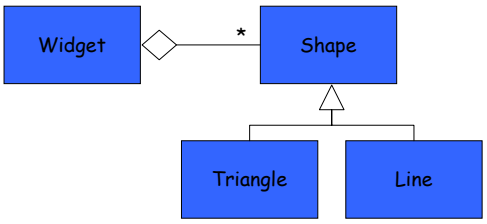
example adapter



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

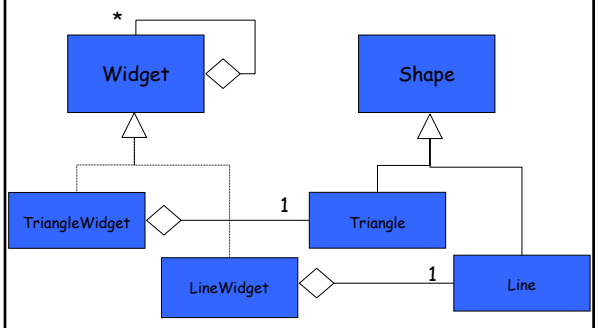
- Design to interfaces not implementation
- Favor composition over inheritance
- Find out what varies and encapsulate it
- Design highly cohesive classes that are loosely coupled
- Think like an object
- Do not forge data
- Once rule one place

Client's code

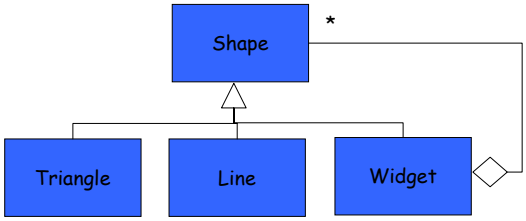
If widget then ...
Else ...

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?