design patterns 101

design patterns 101

- façade
- adapter
- bridge

problem 1

I want a 2D graphics library that supports the following functions for triangles:

- set color to r,g,b
- translate vertices by dx, dy, dz
- rotate α degrees about the origin
- draw

help

I have a 3D graphics library that has a triangle class with the following interface

- triangle()
- triangle(v1x, v1y, v1z, v2x, v2y, v2z, v3x, v3y, v3z)
- ~triangle()
- set color(r, g, b)
- rotate(vector, angle)
- translate(dx, dy, dz)
- scale(sx, sy, sz)
- draw()
- flip(planeA, planeB, planeC, planeD)
- texture(textureMap)
- standardize()

exercise 1

Design a 2D triangle class that uses the 3D class to do the work!

façade

- Scenario — You need to use a subset of a complex system or you need to interact with the system in a particular way.
- Problem — You want to simplify how to use the existing system. You can define your own interface.

problem 2

I want a 2D graphics library that supports shapes. Each shape has a "center" and an HSV "color". The shape interface includes a default constructor, destructor, and the following:

- set color(h, s, v)
- rotateAboutCenter(angle)
- translate(dx, dy)
- draw()

problem 2 cont

The abstract shape class is already written but now we need a triangle class. Rather than rewriting everything from scratch, we want to use the triangle class you designed in problem 1! But the interface is wrong.

exercise 2

Design a triangle class derived from shape that uses your triangle class from problem 1 to do the work.

adapter

- Scenario — A system has the right data and behavior but the wrong interface.
- Problem — You want to use a system object but with different interface.

problem 3

- 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: drawTriange(x1,y1,x2,y2)
  - Draw program 2: drawATriangle(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)
}
```

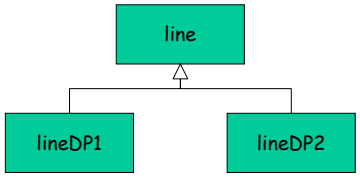
solution 1

- 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"

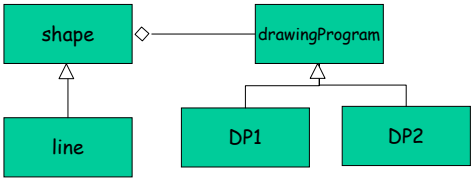
solution 2



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



bridge

- scenario: derived class need to use multiple implementation
- problem: you want to follow "one rule, one place" and still avoid an explosion in number of classes