Design and Design Patterns

Design Patterns

Design Patterns use OO-principles to solve common problems.

Singletons: The OO answer to global variables.

Why not use globals?

A. They make code hard to understand.
B. They make code hard to debug.
C. They make code hard to modify.

Why not use globals?

D. Profs O’Neill and Kuenning with haunt your dreams if you do.

Answer

All of the above.
Singleton Pattern

• Problem: Ensure a class has only one instance and provide a global point of access to that instance.

Singleton Class

class Singleton
{
public:
  static Singleton* Instance();
private:
  static Singleton* theSingletonInstance;
  Singleton();
};
Singleton::Singleton* theSingletonInstance = NULL;

Instance Implementation

Singleton* Instance()
{
  if (theSingletonInstance == NULL)
    theSingletonInstance = new Singleton;
  return theSingletonInstance;
}

Access

Singleton* ptrTheSingleton = Singleton::Instance;

Example

class Ball
{
public:
  static Ball* theBall();
private:
  Sphere theSphere;
  Ball();
};
Ball::Ball* theBall = NULL;

problem

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()`
- `setColor(r, g, b)`
- `rotate(vector, angle)`
- `translate(dx, dy, dz)`
- `scale(sx, sy, sz)`
- `draw()`
- `flip(planeA, planeB, planeC, planeD)`
- `texture(textureMap)`
- `standardize()`

exercise

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

- 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: `drawTriangle(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

```c
void draw() {
    if (drawProgram == 1)
        drawLine(v1.x, v1.y, v2.x, v2.y)
    else
        drawATriangle(v1.x, v2.x, v1.y, v2.y)
}
```
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

- 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

- 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

bridge