OBJECT ADAPTER

By

Sachin Bhargava

Structural Object Patterns

- Concerned with how classes and objects are composed to form larger structures
- Structural object patterns describe ways to compose objects to realize new functionality
- This added flexibility of object composition comes from the ability to change the composition at run-time, which is impossible with static class composition

Adapter

Intent

- Convert the Interface of class into another interface that the client expects
- Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces
- Basically, this means that we need a way to create a new interface for an object that does the right stuff but has the wrong interface

- AKA Wrapper

When Do I Use This?

- You want to use a method that someone else has written because it does something that you need
- You can’t directly incorporate the routine into your program
- The interface or the way of calling the code is not exactly equivalent to the way that its related objects need to use it

Example

- Say you have 2 Requirements:
  - You have to create classes for points, triangles, and squares that have the behavior “display” or “undisplay”
  - The client objects should not have to know whether they actually have a point, a triangle, or a square. They just want to know that they have one of these shapes
  - Basically, you want to free the client object from knowing the details so that it can treat the details in a common way

Example Continued…

- 2 Benefits:
  - Allows client object to deal with these objects in the same way – freed from having to pay attention to their differences
  - Enables one to add different kinds of shapes in the future without having to change the clients

Solution

- Create a Shape class and derive Points, Lines, and Squares from it
- Define an interface in the Shape class for its behavior and then implement it in each of the derived classes
- Some behaviors of the Shape class:
  - Set Shape location, get Shape location, display Shape, fill Shape, set Shape color, undisplay Shape

Requirements Change!

- Now, I am asked to implement a circle, a new kind of Shape
- I do not want to write all of the methods (display, fill, undisplay, etc..) for circle
- I find out that a friend already wrote a MyCircle Class for circles that pretty much does what I need to do, but all the method names are different

- Cannot directly use the MyCircle class because we want to preserve polymorphic behavior with Shape
- Probably not the best idea to go go ahead and change her method names – may cause unanticipated side effects

Solution

- Make a new class that does derive from shape and therefore implements shape's interface but avoids rewriting the circle implementation in MyCircle
  - Class Circle derived from Shape
  - Circle contains MyCircle
  - Circle passes request made to the Circle object on through to the MyCircle object

Discuss Code Example

References

- Design Patterns Elements of Reusable Object-Oriented Software
  - Erich Gamma
  - Richard Helm
  - Ralph Johnson
  - John Vlissides
- Design Patterns Explained A New Perspective on Object Oriented Design
  - Alan Shalloway
  - James R. Trott