Structural Design Patterns

Motivation and Examples

Ahmed Khattab

Technische Universität München

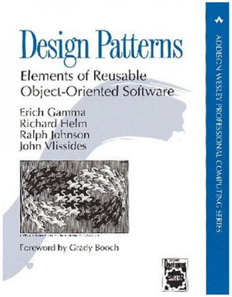
Patterns and Anti-Patterns, 28th of April 2015

Outline

Outline

Introduction

Design Patterns: Elements of Reusable Object-Oriented Software, 1994.[1]


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON WESLEY PROFESSIONAL COMPUTING SERIES

Written by: Gamma, Helm, Johnson and Vlissides (aka. The Gang of four)

[1] http://c2.com/cgi/wiki?DesignPatternsBook

Categorizing patterns

In the GoF book, patterns were classified according to their *p*urpose. According to this classification, patterns can have either a **creational**, **structural**, or **behavioral** purpose.

Outline

Structural Patterns

Definition

- Concerned with how object are composed to form more complex structures
- Provide simple ways to realize relationships between objects

Structural Patterns

Definition

- Concerned with how object are composed to form more complex structures
- Provide simple ways to realize relationships between objects

Motivation

- Flexibility to change
- Extensibility
- Structured code reuse

Examples of structural patterns

- Adapter Pattern
- Composite Pattern
- Decorator Pattern
- Bridge Pattern
- Façade Pattern
- Flyweight Pattern
- Proxy Pattern
- Aggregate Pattern
- ...

Our focus

Our focus will be on:

- Adapter Pattern
- Composite Pattern
- Decorator Pattern
- Bridge Pattern
- Façade Pattern
- Flyweight Pattern
- Proxy Pattern
- Aggregate Pattern
- ...

Outline

Proxy Pattern

Definition

- Sits between the client of an object and the object itself
- Controls access to the object

Proxy Pattern

Definition

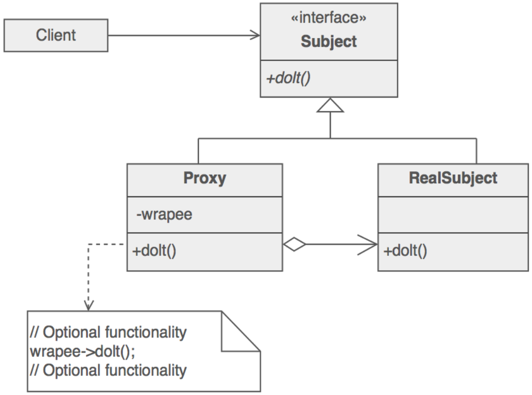
- Sits between the client of an object and the object itself
- Controls access to the object

Common Scenarios

- Controlling the instantiation of an expensive object
- Making a remote object seem local
- Caching (web service requests, rendering of graphical elements, ...

Proxy Pattern - The analogy

Proxy Pattern - In Detail



```
// Optional functionality
wrapee->doIt();
// Optional functionality
```

Outline

Decorator Pattern

Definition

- Allowing the addition of functionality to an object dynamically
- Provide a flexible alternative to subclassing for extending functionality

Decorator Pattern

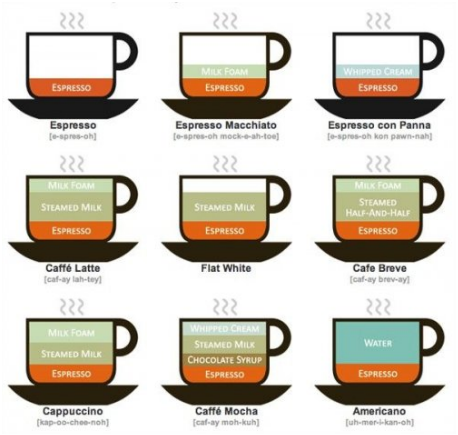
Definition

- Allowing the addition of functionality to an object dynamically
- Provide a flexible alternative to subclassing for extending functionality

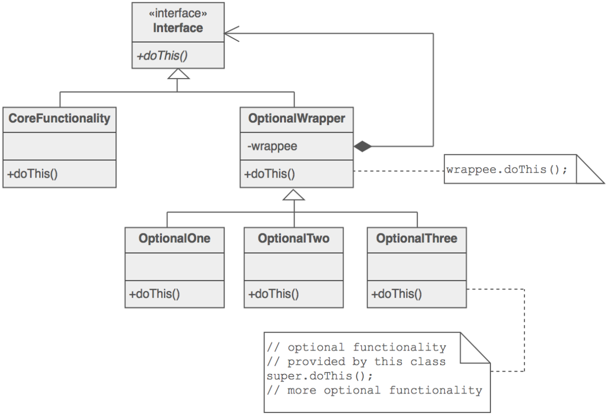
Common Scenarios

- Adding additional features to objects without heavily modifying the code using them
- Too many dynamic options that can be added, making subclassing a headache
- e.g. Lord of the rings game, different roles (elf, orc, hobbit, etc..)

Decorator Pattern - The analogy



Decorator Pattern - In Detail



```
wrappee.doThis();
```

```
// optional functionality
// provided by this class
super.doThis();
// more optional functionality
```

Outline

Façade Pattern

Definition

- Provides a simpler abstracted interface to a larger (potentially more complex) body of code.

Façade Pattern

Definition

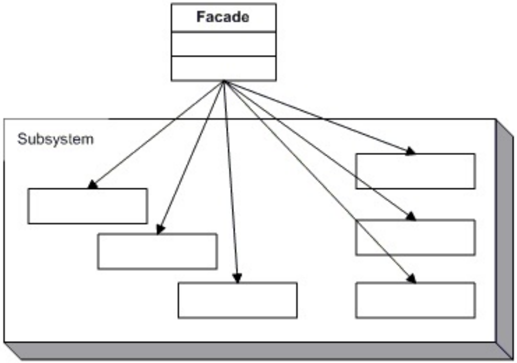
- Provides a simpler abstracted interface to a larger (potentially more complex) body of code.

Common Scenarios

- Interface to abstract access to several complex subsystems
- Wrap a poorly designed collection of APIs with a single well-designed API

Façade Pattern - The analogy

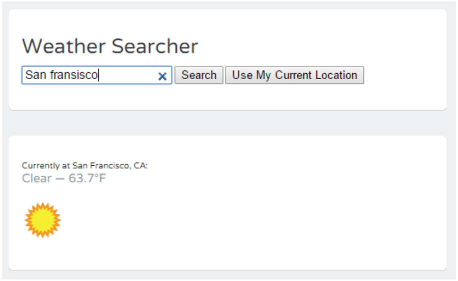
Façade Pattern - In Detail



Outline

The Weather App[2]

A simple application for fetching location-specific weather information from a web service

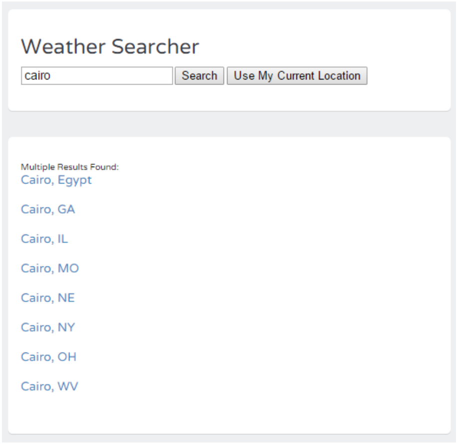


Currently at San Francisco, CA:
Clear — 63.7°F

[2] https://github.com/flamingveggies/weathersearcher

The Weather App

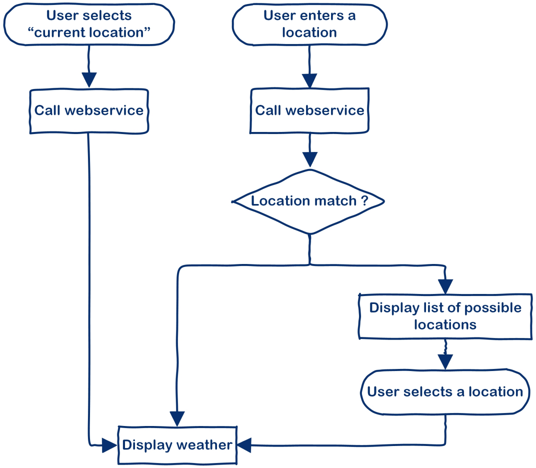
Upon no location match: a list of suggested locations is returned



Multiple Results Found:
Cairo, Egypt
Cairo, GA
Cairo, IL
Cairo, MO
Cairo, NE
Cairo, NY
Cairo, OH
Cairo, WV

The Weather App

The flow:



The Weather App - analysis

The callback function handles everything

- Receiving response from webservice calls
- Parses and displaying weather info when available
- Otherwise displays list of possible locations and attaches event handlers for the displayed list

Disadvantages

- Tight coupling between view and control (handling web service calls and rendering of the output in one place)
- Suggestions for locations are fetched and the whole list is rendered every time from the web service (Caching ?)

The Weather App - refactoring I

The proxy:

```
function API() {
    this.cache = {};

    this.addToCache = function(location, data){
      this.cache[location] = data;
    }
    this.searchLocation = function(searchLocation, callback){
      if(this.cache[searchLocation])
         // cache hit, load from cache
      else
         // cache miss, call web service
    }
    this.getWeatherByID = function(searchID, callback){
      var url = " ... ";
        ...
```

The Weather App - refactoring I

The proxy: a class API that provides the following:

- Encapsulates actual calls to the web service
- Manages the cache for caching the returned lists of locations

The Weather App - refactoring II

The decorator:

```
//declare the different decorators
decorators = {};
decorators.locationsResponse = {
   render: function(data){
    //do custom rendering for locations suggestions list...
    }};

...

var weatherResponse = new BasicResponse(data);
if(data.locations != '') {
    weatherResponse.decorate('locationsResponse');
}
weatherResponse.render();
```

The Weather App - refactoring II

The decorator:

```
function BasicResponse(data) {

    this.decorator;

    this.render = function(){
      var resultHTML;
      if(this.decorator)
      // a decorator exists, use it to render
      decorators[this.decorator].render(this.data);
      else {
      // no decorators added, render normally...
  } } }
```

The Weather App - refactoring II

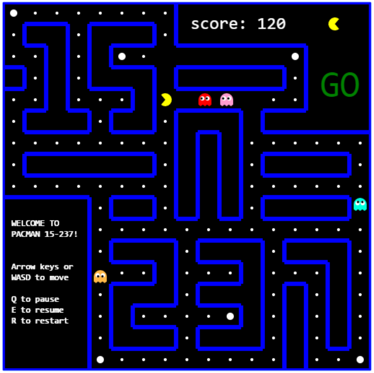
The BasicResponse class provides the following:

- An object oriented representation for each response from the web service
- A render method to parse and display normal responses
- The render method can be decorated to display special responses differently

Outline

Pacman[3]

The famous Pacman implemented in JavaScript and HTML5 canvas



WELCOME TO
PACMAN 15-237!

Arrow keys or
WASD to move

Q to pause
E to resume
R to restart

[3]https://github.com/bxia/Javascript-Pacman

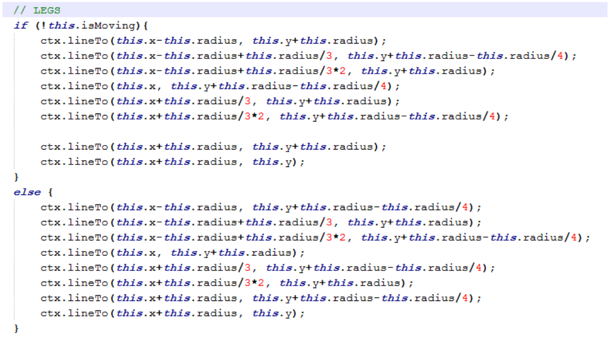
Pacman - ghosts.js

- Script responsible for representing the Ghost object
- Handles rendering of the object using HTML5 canvas methods
- Every ghost can take different forms (color, eye position, ...)

Pacman - analysis

A huge Ghost.draw() function:

- Checks the status of the ghost (weak or strong, moving or not, ..)
- Draws every detail (the eyes, mouth, legs, ...)



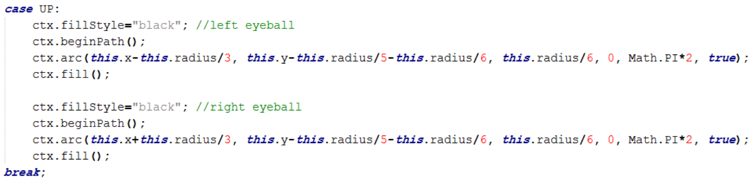
```
// LEGS
if (!this.isMoving){
    ctx.lineTo(this.x-this.radius, this.y+this.radius);
    ctx.lineTo(this.x-this.radius+this.radius/3, this.y+this.radius-this.radius/4);
    ctx.lineTo(this.x-this.radius+this.radius/3*2, this.y+this.radius);
    ctx.lineTo(this.x, this.y+this.radius-this.radius/4);
    ctx.lineTo(this.x+this.radius/3, this.y+this.radius);
    ctx.lineTo(this.x+this.radius/3*2, this.y+this.radius-this.radius/4);

    ctx.lineTo(this.x+this.radius, this.y+this.radius);
    ctx.lineTo(this.x+this.radius, this.y);
}
else {
    ctx.lineTo(this.x-this.radius, this.y+this.radius-this.radius/4);
    ctx.lineTo(this.x-this.radius+this.radius/3, this.y+this.radius);
    ctx.lineTo(this.x-this.radius+this.radius/3*2, this.y+this.radius-this.radius/4);
    ctx.lineTo(this.x, this.y+this.radius);
    ctx.lineTo(this.x+this.radius/3, this.y+this.radius-this.radius/4);
    ctx.lineTo(this.x+this.radius/3*2, this.y+this.radius);
    ctx.lineTo(this.x+this.radius, this.y+this.radius-this.radius/4);
    ctx.lineTo(this.x+this.radius, this.y);
}
```

Pacman - analysis

A huge Ghost.draw() function:

- Checks the status of the ghost (weak or strong, moving or not, ..)
- Draws every detail (the eyes, mouth, legs, ...)



```
case UP:
    ctx.fillStyle="black"; //left eyeball
    ctx.beginPath();
    ctx.arc(this.x-this.radius/3, this.y-this.radius/5-this.radius/6, this.radius/6, 0, Math.PI*2, true);
    ctx.fill();

    ctx.fillStyle="black"; //right eyeball
    ctx.beginPath();
    ctx.arc(this.x+this.radius/3, this.y-this.radius/5-this.radius/6, this.radius/6, 0, Math.PI*2, true);
    ctx.fill();
break;
```

Pacman - analysis

A huge Ghost.draw() function:

- Checks the status of the ghost (weak or strong, moving or not, ..)
- Draws every detail (the eyes, mouth, legs, ...)

Disadvantages

- Hard to understand, to maintain, or to debug.
- Repetitive very similar lines of code, no reuse.

Pacman - refactoring

Façade:

- Re-structured the draw() function by using several helper functions
- Reuse of code for drawing eyes at different positions, as well as legs

```
Ghost.prototype.eyeBlack = function (offsetX, offsetY){ ... };

Ghost.prototype.eyeWhite = function (){ ... };

Ghost.prototype.legs = function (){ ... };

Ghost.prototype.mouth = function (){ ... };
```

Pacman - refactoring

Façade:

```
Ghost.prototype.eyeBlack = function (offsetX, offsetY){
  ctx.fillStyle="black";
  ctx.beginPath();
  ctx.arc(this.x+offsetX, this.y+offsetY, this.radius/6, 0,
      Math.PI*2, true);
  ctx.fill();
};

Ghost.prototype.draw = function (){
...
case UP:
  this.eyeBlack(-this.radius/3, -this.radius/5-this.radius/6);
  this.eyeBlack(this.radius/3, -this.radius/5-this.radius/6);
break;
...
}
```

Summary I

The decorator pattern:

- Objects can be 'decorated' and used with new behavior, without worrying about modifying the base object.
- Excessive use is not advised, managing them becomes a headache (instantiation of objects, decorators interdependence, ..)

Summary II

The proxy pattern:

- Introduces a level of indirection that helps in regulating or optimizing access to objects
- While making access to remote objects completely transparent, inefficient uses can occur

```
 if (account.getBalance() > 0 && account.getBalance() < MAX) {
    transferAmount(account.getBalance() / 2);
}
```

Summary III

The façade pattern:

- Promotes decoupling and reuse, enhances structure and maintainability of code.
- Need to be aware of the performance costs of the abstraction offered by the façade

Summary IV

Interesting to compare:

- **Adapter** provides a different interface to its subject. **Proxy** provides the same interface. **Decorator** provides an enhanced interface.
- `https://sourcemaking.com/design_patterns/structural_patterns`

References

Stefanov, S.
JavaScript Patterns.
O'Reilly Media, 2010.

Osmani, A.
Learning JavaScript Design Patterns.
O'Reilly Media, 2012.

Gamma, E. and Helm, R. and Johnson, R. and Vlissides, J.
Design Patterns: Elements of Reusable Object-oriented Software.
Addison-Wesley Longman Publishing Co., 1994.

Sourcemaking
Design Patterns Explained Simply.
https://sourcemaking.com/design_patterns

Zaikin, M.
Benefits and Drawbacks of Design Patterns.
http://java.boot.by/scea5-guide/ch07s03.html