

Structural Patterns

Chrysa Papadaki and Nishant Gupta

Department of Computer Science, Boltzmannstr. 3, 85748 Garching

Associate Editor: Guy Yachdav

ABSTRACT

1 INTRODUCTION

Introduction to structural patterns

2 APPROACH

In order to demonstrate the application of general JavaScript patterns several projects on GitHub have been reviewed (see Section ??). In each project some code parts have been optimized by applying one or multiple structural patterns.

3 STRUCTURAL PATTERNS

There are multiple structural patterns that can be applied in any JavaScript project. In this paper the following structural patterns have been reviewed [[1]]:

- Proxy pattern
- Decorator pattern
- Facade pattern
- Composite pattern

3.1 Proxy Pattern

Proxy Pattern

3.2 Decorator Pattern

Decorator Pattern

3.3 Facade Pattern

Facade design patterns Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use [1]. Facade pattern hides the complexities of the system and by providing an interface to the client using which the client can access the system.

3.3.1 Applicability Facade pattern should be used when it is required to provide a simple interface to a large and complex system. It can also be used when there are many dependencies between clients and the implementation classes of an abstraction. If multiple subsystems are to be layered, Facade classes can define entry points to each subsystem layer.

3.3.2 Structure Typical structure of Facade usage can be depicted as shown in the figure 1.

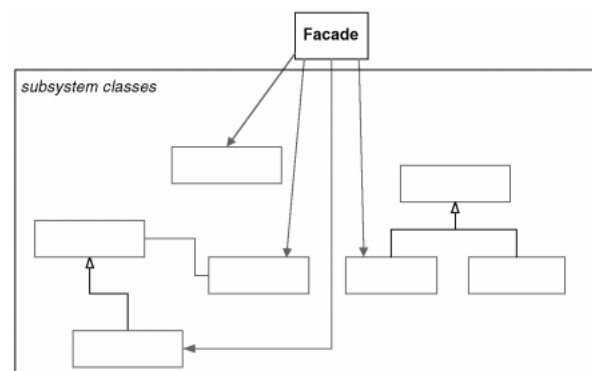


Fig. 1. Generic structure of the Facade pattern [1]

Here, subsystem classes implement respective subsystem's functionality and facade has the responsibility to delegate client requests to appropriate subsystem objects. Subsystem classes don't have any knowledge of the facade, but handle the work assigned by the Facade object.

3.3.3 Benefits By providing abstraction to clients from subsystem components, facade makes the subsystem easier to use for clients. Facade also promotes the concept of changing the components of a subsystem without affecting its clients, hence providing a weak coupling between the subsystem and its clients. Besides all these features, facade doesn't prevent applications from using subsystem classes if they need to.

3.4 Composite Pattern

Composite Pattern

4 REFACTORING EXAMPLES

4.1 songFinder

4.2 drawr-bootstrap

5 CONCLUSION

Conclusion

REFERENCES

- [1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.