Design Patterns

Seb Arnold

5 Mai 2015

Overview

This lecture presents the origin and the definition of design patterns. It then explains the differences between architectural and design patterns, and goes in the description of a few common ones. Finally it provides a methodology to find bad structure (through anti-patterns and bad smells), and explains the process of refactoring code to avoid it.

Important Slides

A list of the important slides, and an explanation of why they are important. (At least one slide per concept explained, easy or difficult)

Slide n°2 Design patterns are common design decisions for well known tasks.

Slide n°6 Explains the definition of a design pattern. A pattern relies on abstraction, documentation as well as other well defined properties.

Slide n°8 Architectural vs Design. The difference lies in the level of abstraction.

Architectural Patterns:

- apply to the system as a whole (multiple components)
- are applicable to problems with common application domain

In contrast, Design Patterns:

- apply to specific part of system (a single component)
- are applicable across arbitrary application domains.

Slide n°9 State-Logic-Display Pattern: (Also called three-tiered)

- Applies to most software application
- Analogous to MVC M: State, V: Display, C: Logic (See 10-11)
- Analogous to Sense-Compute-Control (See 12)

Slide n°16 Categories of patterns: Creational, Structural, Behavioral

Slide n°18 Elements defining a pattern: Name, Problem Description, Solution Description, Consequences

Slide n°19-21 Observer Pattern: A subject (ie, variable, object, ...) is linked to some observers. When the subject is updated, the observers are notified and can act upon this change. Often used for multiple displays of data (Think excel, table, graph, pie) Examples: $scope variables in AngularJS, Events in Java and JavaScript (onClick, Listeners)

Slide n°22-23 Mediator Pattern: Used when several components need to interact with each other. Instead of coding the interactions within the objects, they are implemented in the mediator. (See 23) Examples: Thread synchronization mechanisms (Barriers, Semaphores), Angular's Services

Slide n°24-27 Façade Pattern: Imagine an object who needs to communicate with several subsystems. Instead of dealing with each of them separately, you create a "Façade" that unifies the interface and takes care of the interactions. Examples: Load Balancers, EchoNest/IEEE API, scikit-learn .fit() and .score() methods.

Slides n°28-29 Proxy Pattern: Instead of accessing the real object, you access a proxy to it. The advantages include access control of the original object through the proxy, management of pointer changes, and creation of expensive objects. Examples: internet proxies

Slide n°30-31 Adapter Pattern: It is essentially a converter of interfaces. It allows two classes to communicate together, despite different APIs. Human example: a translator

Slide n°32-33 Bridge Pattern: Separates an abstraction from its implementation. In Java, this would be an extra layer between an interface and the class implementation. Doing so allows you to change the abstraction and then you only need to change the bridge, and not every time you access the class, nor the class itself.

Slide n°34-37 Anti-Pattern Description: Anti-patterns are useful for spotting errors in design decisions.

- Bad Smell: If something seems "fishy", there is probably a problem
- Slide 37: List of examples that are fishy. (essentially the opposite of all good coding practices)

Slide n°38-40 Refactoring: is changing what looks wrong in order to approach the perfect code. Note that refactoring doesn't change the observable behavior but only the internals of the program. The challenge here is to make those changes without messing up previous functionalities. See 40 for a list describing the refactoring process.

Exam Questions

A list of questions that will probably appear in the final exam.

- Give N examples of architectural and design patterns, and explain the conceptual differences.
- What are the differences/similarities between MVC and State-logic-display ?
- What are the three categories of design patterns ? How do they differ from each other ?
- Give an example of an observer pattern.
- How are the Façade, Mediator, and proxy patterns different ?
- What is bad smell ? How to detect it ?
- Give three anti-patterns and explain them.
- State the stages of the refactoring process.