Design Patterns

Mikael Svahnberg[*]

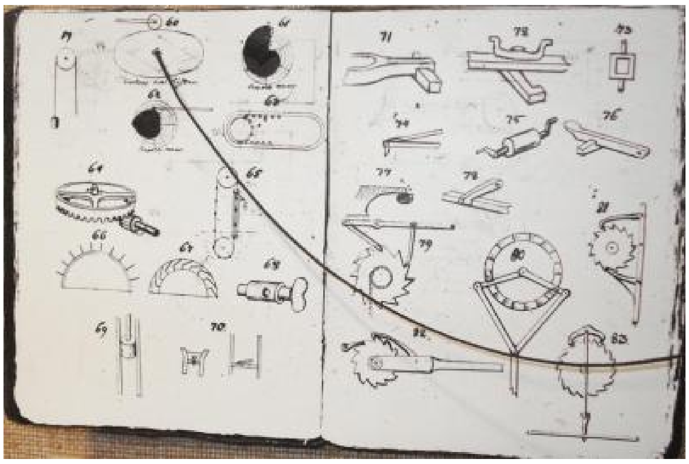
2021-03-03

Contents

[*]Mikael.Svahnberg@bth.se

1 Introduction

1.1 Christopher Polhem, father of Design Patterns?



1.2 Patterns – a brief introduction

- Design patterns are reusable solutions to known problems
  - With known consequences
  - “encoded experience”
  - Codified in a structured format
  - named
- There is nothing that *requires* you to use design patterns; they are a convenience.
- Design patterns focus primarily on structure (class view), and interaction (sequence diagrams).

Responsibility Driven Design

1.3 Levels of Patterns

Different levels:

- Architecture
  - Systems, subsystems
- Design
  - Classes, groups of classes
- Idioms
  - One class, functions within one class
- GRASP

  - In some sense orthogonal
  - Learning aid for OO Design
  - Advice for Assigning Responsibilities

1.4 Some Common Design Patterns

From Gamma, E., Helm, R., Johnson, R., & Vlissides, J., *Design patterns: elements of reusable object-oriented languages and systems* (1994), Reading MA: Addison-Wesley.

1. Strategy
2. State
3. Observer
4. Facade && Adapter
5. Abstract Facory
6. Builder
7. Command
8. Interpreter
9. Visitor

2 GRASP Patterns

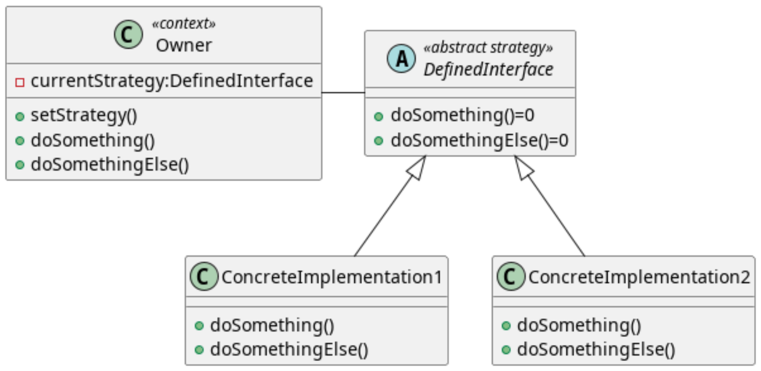
Larman, C., Applying uml and patterns: an introduction to object-oriented analysis and design and iterative development, 3/e (2012), : Pearson Education.

- Low Coupling
- High Cohesion
- Information Expert
- Creator
- Controller
- Polymorphism
- Indirection
- Pure Fabrication
- Protected Variations

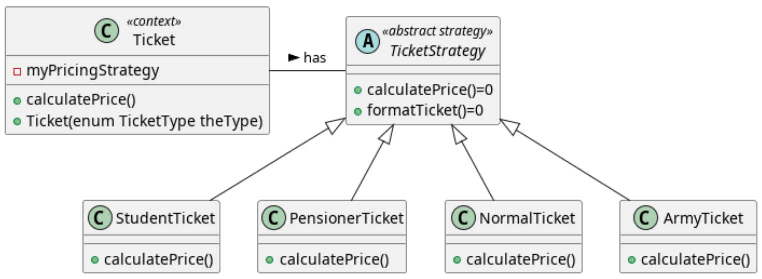
3 Design Patterns

3.1 Strategy

1. Generic Structure

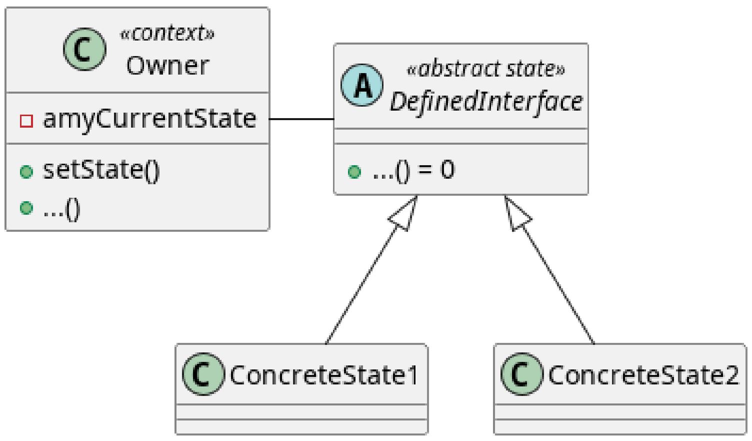


2. Example



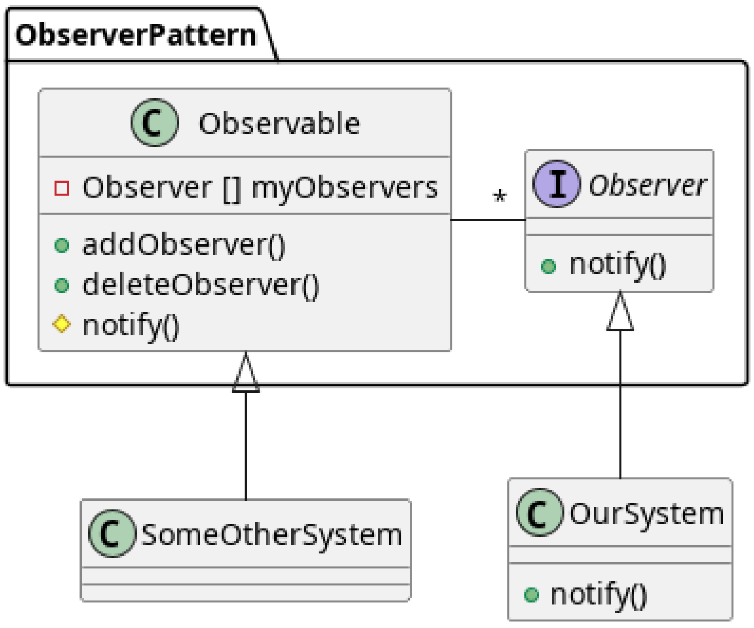
3.2 State

1. Generic Structure

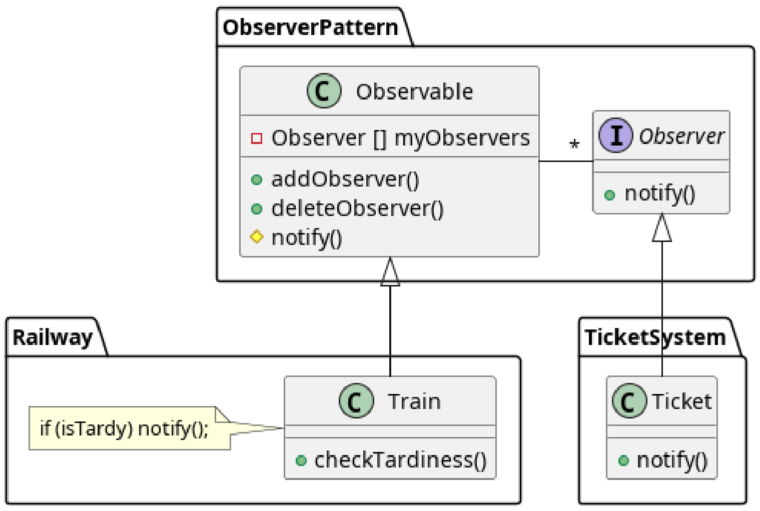


3.3 Observer

1. Generic Structure

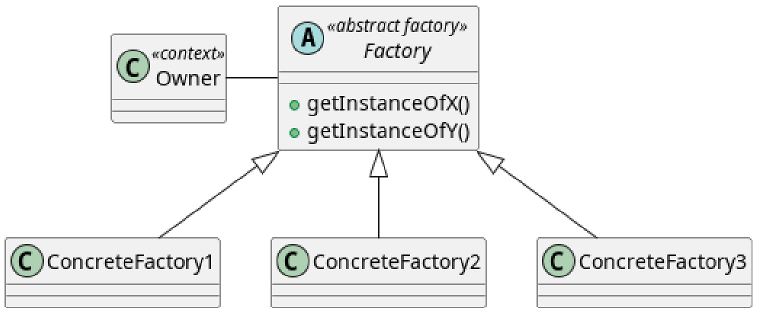


2. Example

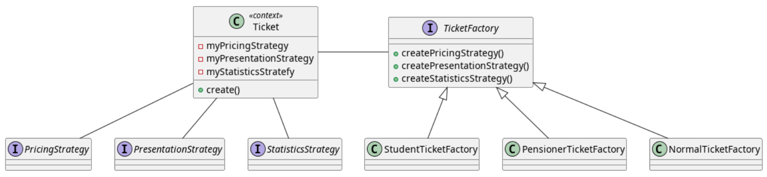


3.4 Abstract Factory

1. Generic Structure



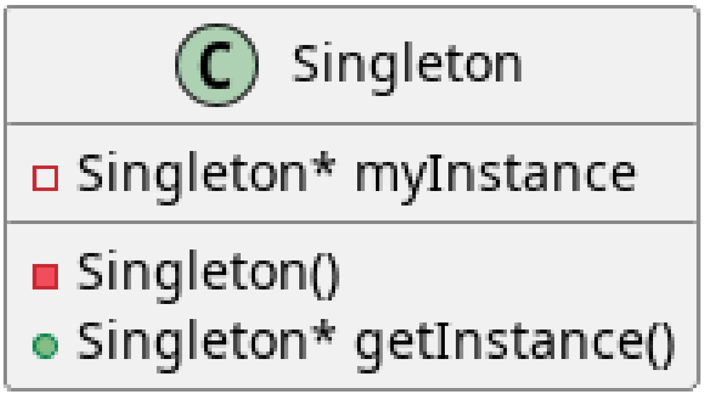
2. Example 1



```
create() {
  if(student) myFactory = new StudentTicketFactory();

  ...

  myPricingStrategy = myFactory.getPricingStrategy();

}
```

3.5 Singleton

1. Generic Structure

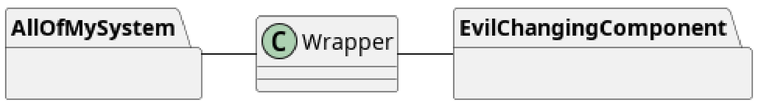


2. getInstance()

```
static Singleton* Singleton::getInstance() {
  if(null == myInstance) {
    myInstance = new SIngleton();
  }

  return myInstance;

}
```

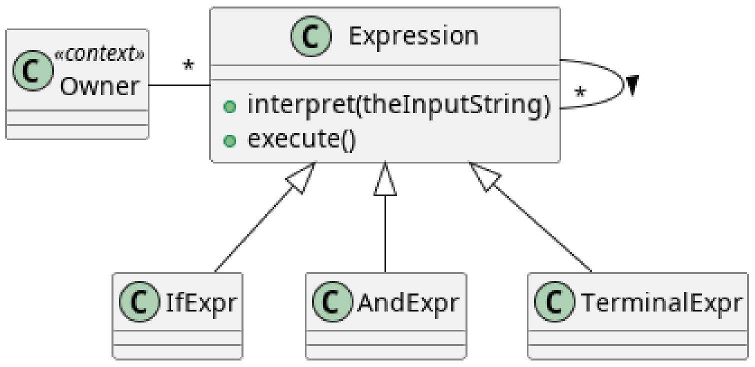
3.6 Facade, Wrapper, Adapter

1. Generic Structure



3.7 Interpreter

1. Generic Structure



2. Example Lisp grammar:

   sexpr list || atom

   list '(' sexpr* ')'

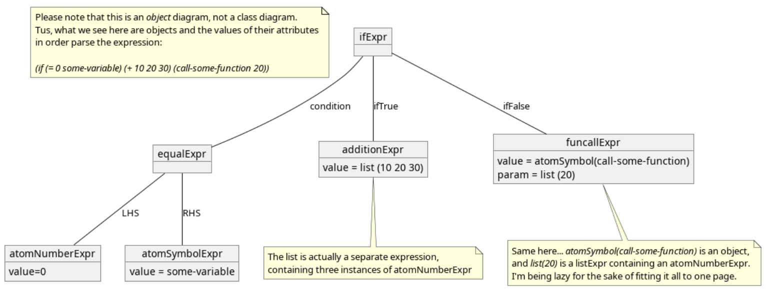
   atom number || symbol

   number [0-9]+

   symbol ^[()]+

   Let's parse an expression: `(if (= 0 some-variable) (+ 10 20 30) (call-some-function 20))`

   Object Diagram



Please note that this is an *object* diagram, not a class diagram.
Tus, what we see here are objects and the values of their attributes
in order parse the expression:

(if (= 0 some-variable) (+ 10 20 30) (call-some-function 20))

The list is actually a separate expression,
containing three instances of atomNumberExpr

Same here... *atomSymbol(call-some-function)* is an object,
and *list(20)* is a listExpr containing an atomNumberExpr.
I'm being lazy for the sake of fitting it all to one page.

4 Design Antipatterns

- Design Patterns == Solutions to commonly recurring problems
- Programming **antipatterns** == What **not** to do.
- High and low, ranging from process, to design, to programming

4.1 Design Antpattern Examples

Process antipatterns

Premature optimisation Optimising your code before you know if, when, and where a particular quality attribute is going to be a problem.

Bikeshedding Spend time discussing the *colour* of the bikeshed instead of figuring out whether you really need a bikeshed in the first place.

Analysis Paralysis The most perfect analysis in the world is useless unless you have actually built the system. *Good enough* is often plenty.

Gold Plating Adding things to your project/design/code that might someday come in handy even if no-one has ever asked for it – or is willing to pay for it.

Design antipatterns

The God class The one central class that knows too much about the rest of the system, with too many dependencies and responsibilities.

Fear of adding classes More classes doth not always a more complex design make.

Programming antipatterns

Magic Numbers and Strings string and number literals that are thrown into the middle of the code means that it is (a) more difficult to find them when they change, and (b) there is no encoded decision rationale what thet mean. Use a constant instead.

Copypasta When you copy code from one function to another that *almost* do the same thing, you are creating twice the maintenance headache.

5 Security Antipatterns

Pardon the Malintent You should use the same standard of throwing exceptions and notifications when you detect a hacking attempt as in your own code. If you can't open a file, you throw an exception so that the user can know about it (hopefully you do not terminate unless it is critical), but if you notice a hacking attempt you just swallow it without even logging the attempt?

Incomplete Mediation When you authenticate some types of requests (the common ones), but forget other access routes.

Insufficient Knowledge of Protocols Just because you have never seen a HTTP request/response (it is fully possible to implement a whole web application without evver seeing one), it doesn't mean that the hacker doesn't know how to exploit it. Do you know how all parameters in the HTTP header are treated in your application?-

Secure Library Ignorance Make sure you security audit any third party libraries that you use. This includes all third party libraries, not just the ones you think might be at risk.

Mixing Code and Data Make sure that all and any inputs are sanitised before you concatenate them into a a string that is going to be executed. Fun fact: Did you know that someone voted for `pwn'); DROP TABLE VOTERS;--` in the 2010 Swedish elections?

Confirmation Bias We all have blind spots where we assume but do not verify certain things. In security engineering, this can be particularly problematic.