



What are Design Patterns?

Definitions and Template

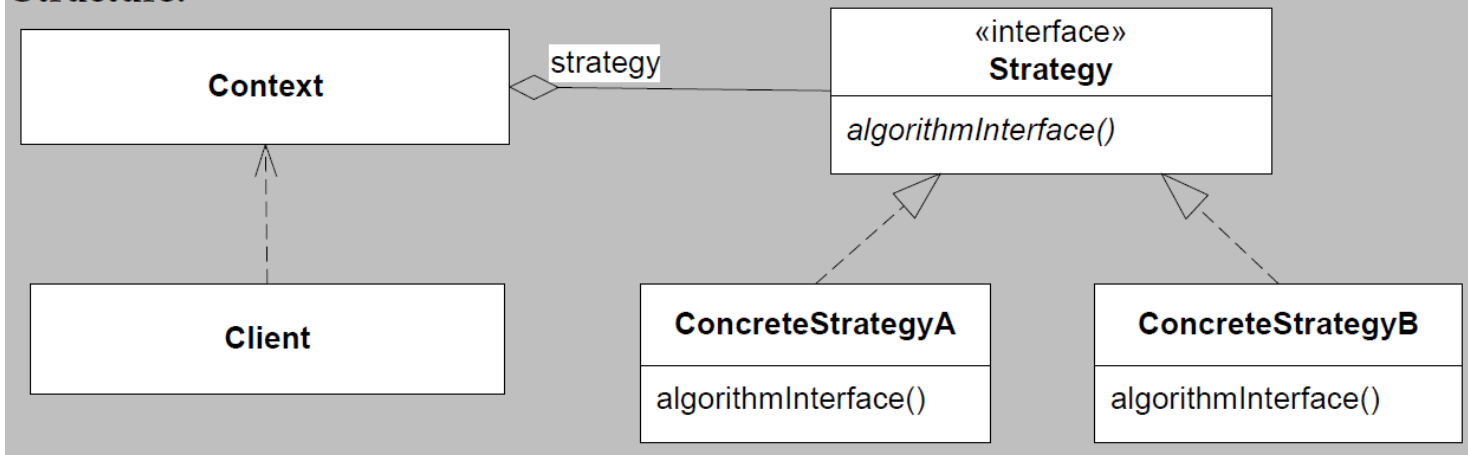
Gamma et al.'s definition

Definition: Design Pattern (Gamma et al.)

Patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.

Exercise: How does it relate to...

Structure:



Beck et al.'s Definition

Definition: Design Pattern (Beck et al.)

A design pattern is a particular prose form of recording design information such that designs which have worked well in the past can be applied again in similar situations in the future.

Prose form = “writing template”

The template varies from author to author.

However, must contain

- Name
- Problem
- Solution
- Consequences

FRSE's template

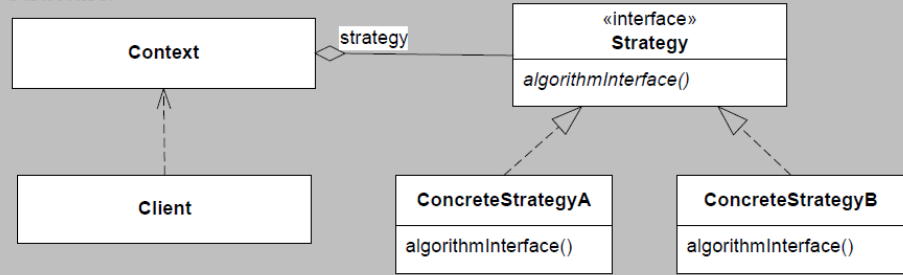
Intent

- Short description

Roles

- Responsibilities of each participating object/abstraction in the pattern

[7.1] Design Pattern: Strategy

Intent	Define a family of business rules or algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithms vary independently from clients that use it.
Problem	Your product must support variable algorithms or business rules and you want a flexible and reliable way of controlling the variability.
Solution	Separate selection of algorithm from its implementation by expressing the algorithms responsibilities in an interface and let each implementation of the algorithms realize this interface.
Structure:	 <pre> classDiagram class Context class Client class Strategy { <<interface>> algorithmInterface() } class ConcreteStrategyA { algorithmInterface() } class ConcreteStrategyB { algorithmInterface() } Context o--> Strategy : strategy Client ..> Context Strategy < .. ConcreteStrategyA Strategy < .. ConcreteStrategyB </pre> <p>The diagram illustrates the Strategy Design Pattern structure. It features a Context class, a Client class, a Strategy interface, and two concrete strategy classes, ConcreteStrategyA and ConcreteStrategyB. The Client has a dashed dependency arrow pointing to the Context. The Context class has a solid association arrow pointing to the Strategy interface, labeled with the role strategy. The Strategy interface defines the algorithmInterface() method. Both ConcreteStrategyA and ConcreteStrategyB implement this interface, as indicated by dashed generalization arrows pointing from them to the Strategy interface.</p>
Roles	Strategy specifies the responsibility and interface of the algorithm. ConcreteStrategies defines concrete behavior fulfilling the responsibility. Context performs its work for Client by delegating to an instance of type Strategy .
Cost - Benefit	<p>The benefits are: <i>Strategies eliminate conditional statements. It is an alternative to subclassing. It facilitates separate testing of Context and ConcreteStrategy.</i></p> <p>The liabilities are: <i>Increased number of objects. Clients must be aware of strategies.</i></p>

Differentiating Patterns

Be aware that many patterns are *structurally equal* – their UML class diagrams are more or less identical!

Patterns are defined by the *problem they solve!*

Strategy is the problem of

Handling variability of algorithms / business rules, making them interchangeable.