Strategy Design Pattern

EECS 6431- Software Re-engineering

Nasim Razavi

YORK UNIVERSITÉ UNIVERSITY

Table of Content

- Strategy pattern
- Motivation
- Intent
- Applicability
- Structure
- Participants
- Implementation
- Sequence diagram
- Consequences
- References

Strategy pattern

- **Strategy pattern** (known as policy) is a behavioral software design pattern that enables a behavior (an algorithm) to be selected at runtime.

- A Strategy defines a set of algorithms that can be used interchangeably.

Motivation

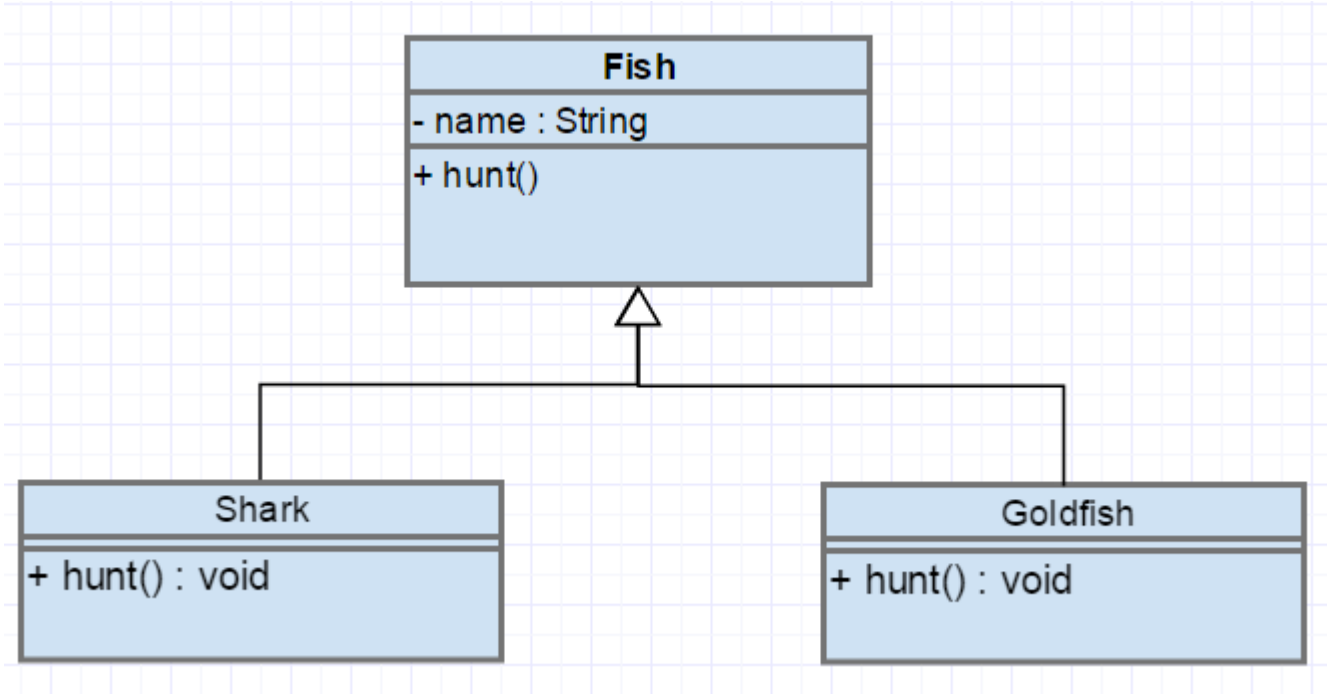
- A superclass or an interface may offer features that are not applicable for all subclass. In this case, for instance, we have to overwrite those features for some subclasses.

- Having optional features in the supper classes that are only relevant to some subclasses causes many duplicate codes and long lists of conditionals.

- A class have different behaviors. Multiple algorithms can be implemented for a specific task.

Motivation

- Example



Intent

- The behavior of a class to be independent from the clients that use it.
  - By encapsulating each algorithm and make them interchangeable.
  - Put the abstraction in an interface, push implementation details down to derived classes.

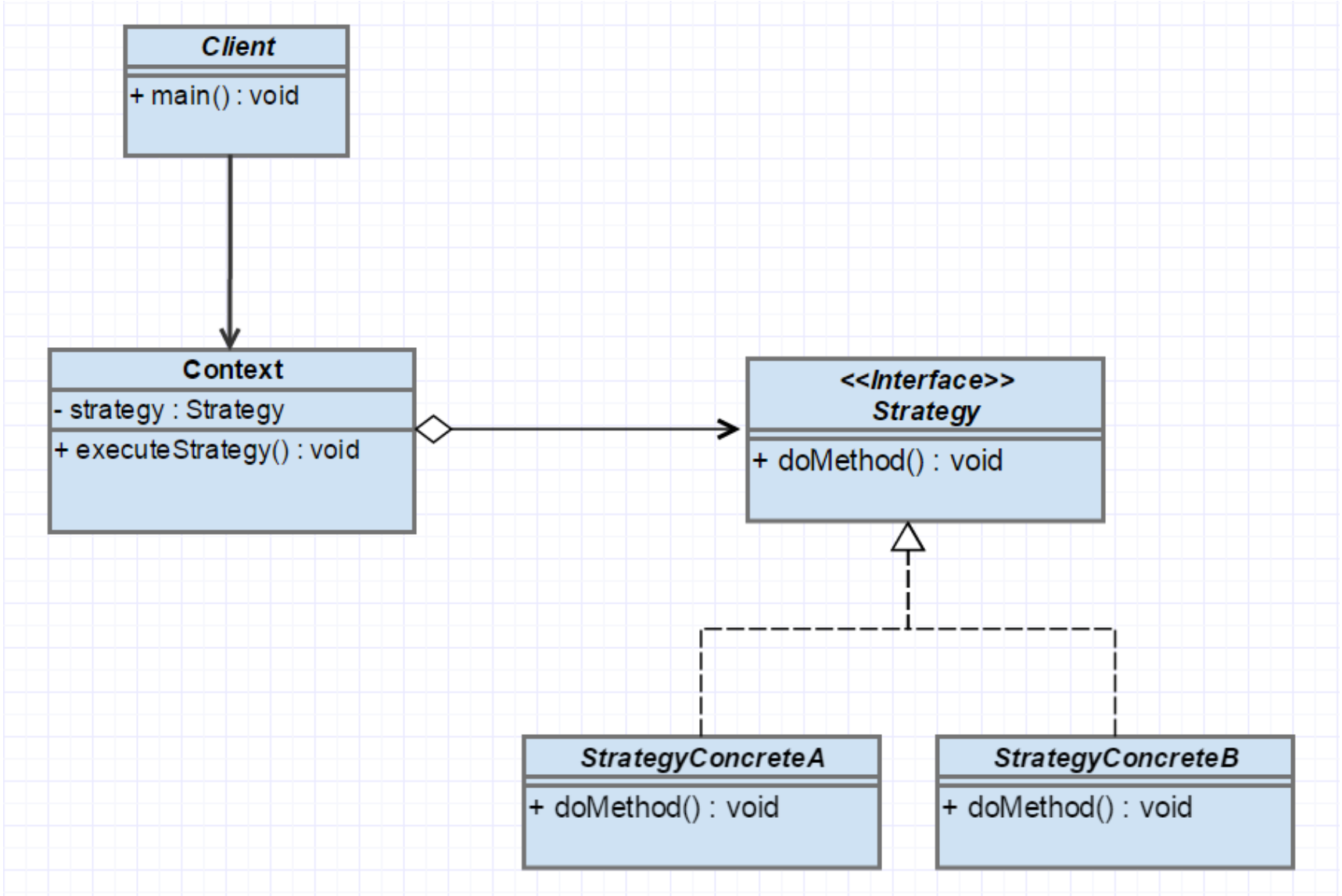
- The behavior can change in subclasses without side effects.

Applicability

Use the Strategy pattern when

- Many related classes differ only in their behavior.
- To avoid complex code.
- We have multiple algorithms for a specific task and we want the client decides what strategy to be used at runtime.

Structure



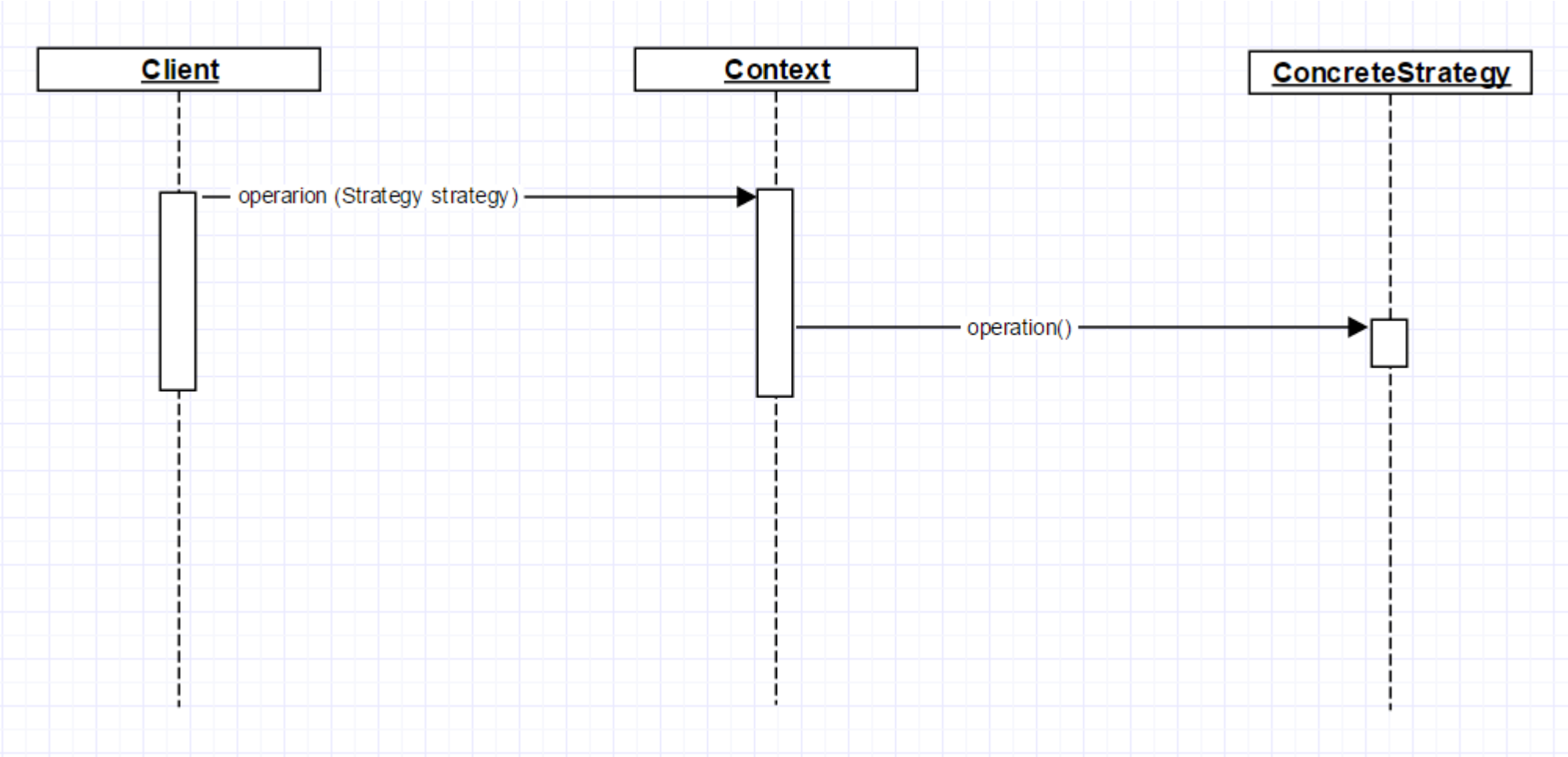
Participants

- Strategy
  - Declares an interface common to all supported algorithms.
  - Different behaviors can be selected by swapping between **ConcreteStrategy**s without any side effect in Context.
- ConcreteStrategy
  - Defines an algorithm by implementing the Strategy interface.
- Context
  - Is composed of a strategy. It is a class that requires changing behaviors.
- Client
  - Defines what behavior the context uses.

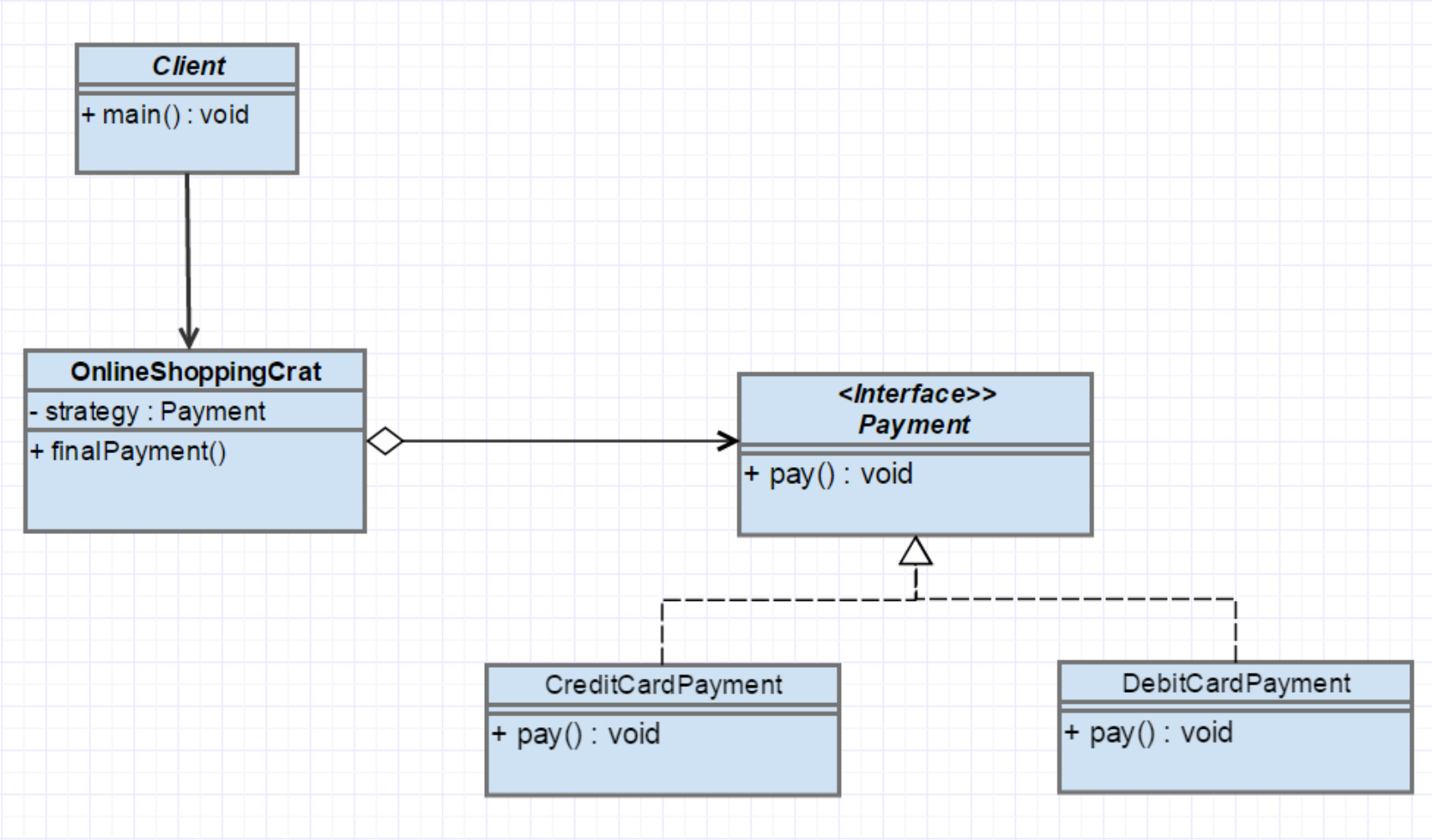
Implementation

- Define the Strategy and Context. Make Strategy objects of related algorithms.
- The strategies can be defined as a hierarchy of classes offering the ability to extend and customize the existing algorithms.
- The context object can have a default algorithm. In runtime, if it does not contain a strategy object, it will execute the default algorithm.

Sequence Diagram



Example



Consequences

- Benefits
  - Encapsulates families of related algorithms into Strategy classes.
  - Keeps class changes from forcing other class changes.
  - Implements algorithms independent from the Context class.
  - Simplifies switching, understanding, and extending the algorithms.
  - Avoids duplicate codes.
  - Eliminates conditional statements.
  - Is flexible. The client can choose among different strategies.

Consequences

- Drawbacks
  - Communication overhead between Strategy and Context.
  - The increased number of objects.
  - The client has to be aware of strategies and their difference.

Refrences

- Vlissides, John, et al. "Design patterns: Elements of reusable object-oriented software." *Reading: Addison-Wesley* 49.120 (1995): 11.

Exercise

Imagine you are working on a system that needs to compress text using different compression algorithms such as RunLengthEncoding, HuffmanCoding. Implement this system which allows users to flexibly compress a text or decompress a compressed text using different methods.