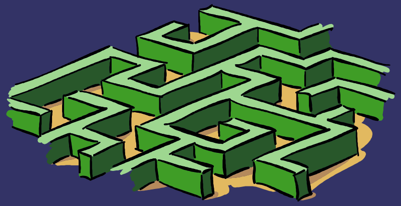
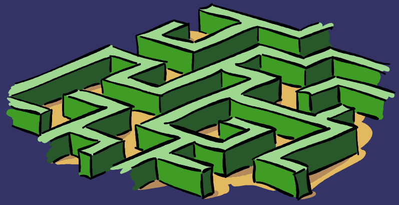
Strategy Patterns

Strategy Pattern Basics and Implementation



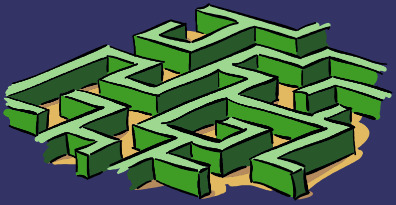
Origin

- First defined in the 1994 book *Design Patterns: Elements of Reusable Object-Oriented Software*
- Strategy pattern encapsulates alternative algorithms (or strategies) for a particular task.



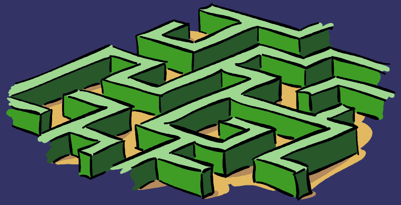
Overview

- The ***strategy pattern*** (also known as the ***policy pattern***) is a behavioral software de-sign pattern that enables selecting an algo-rithm at run-time.
- Instead of implementing a single algorithm directly, code receives run-time instructions as to which in a family of algorithms to use.
- Strategy lets the algorithm vary indepen-dently from clients that use it.



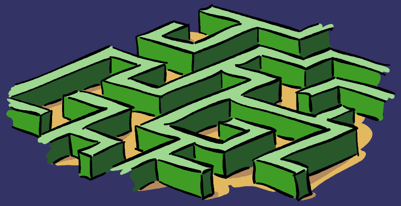
Wait...what?

- Basically strategy pattern helps us separate the parts of an object which are subject to change from the rest of the static bits.



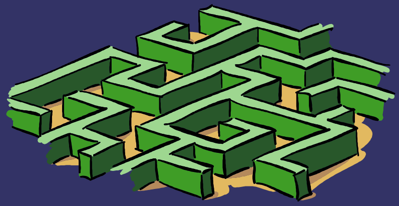
But Why?

- Deferring the decision about which algo-rithm to use until run-time allows the calling code to be more flexible and reusable.
- Used to manage algorithms, relationships and responsibilities between objects.
- Using Strategy objects versus subclasses can often result in much more flexible code since we're creating a suite of easily swap-pable algorithms.

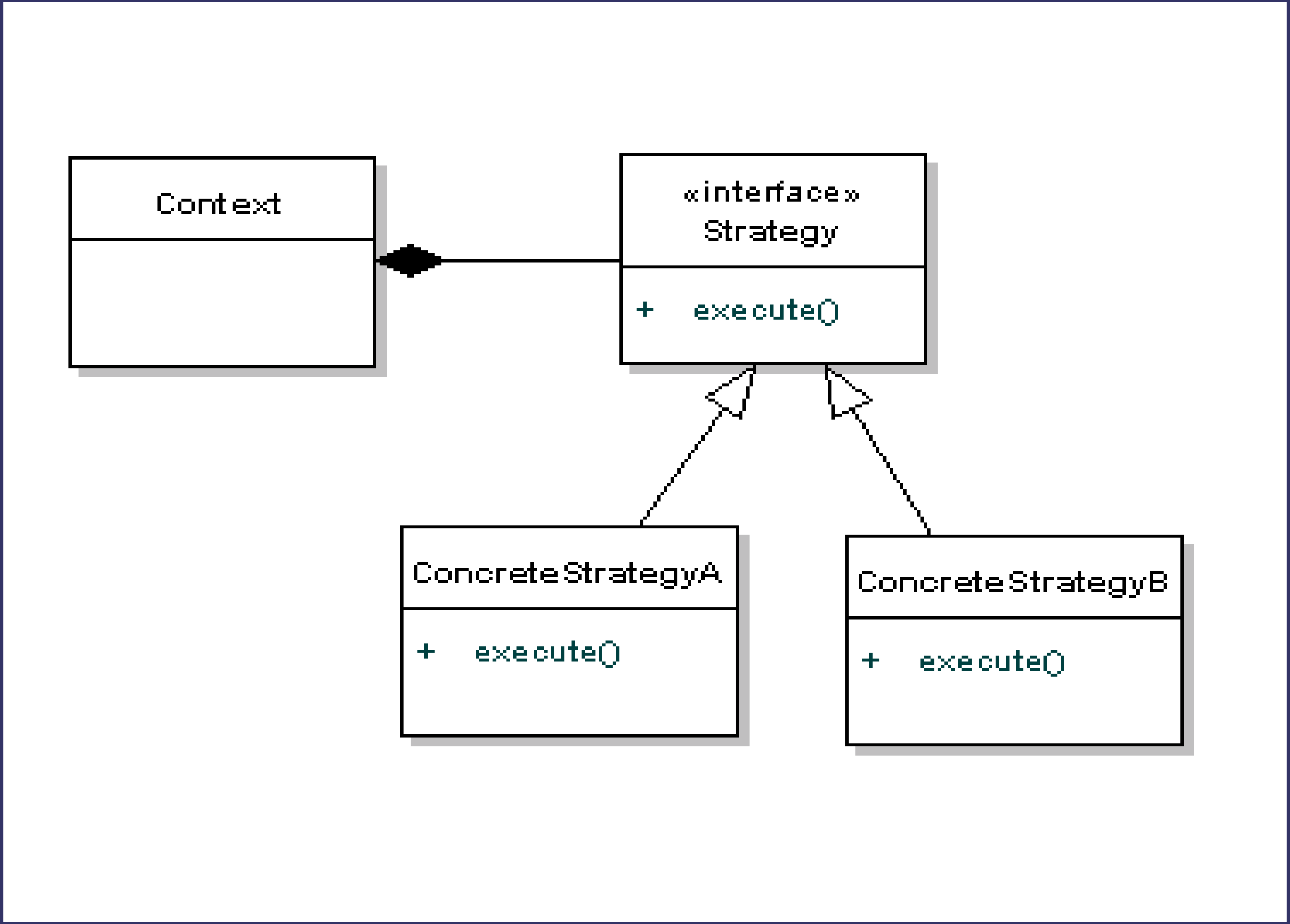


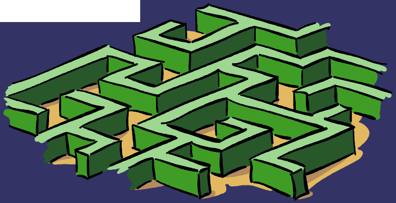
In other words...

- The Strategy pattern is to be used where you want to choose the algorithm to use at run-time.
- A good use would be saving files in different formats, running various sorting algorithms, or file compression.
- Provides a way to define a family of algo-rithms, encapsulate each one as an object, and make them interchangeable.

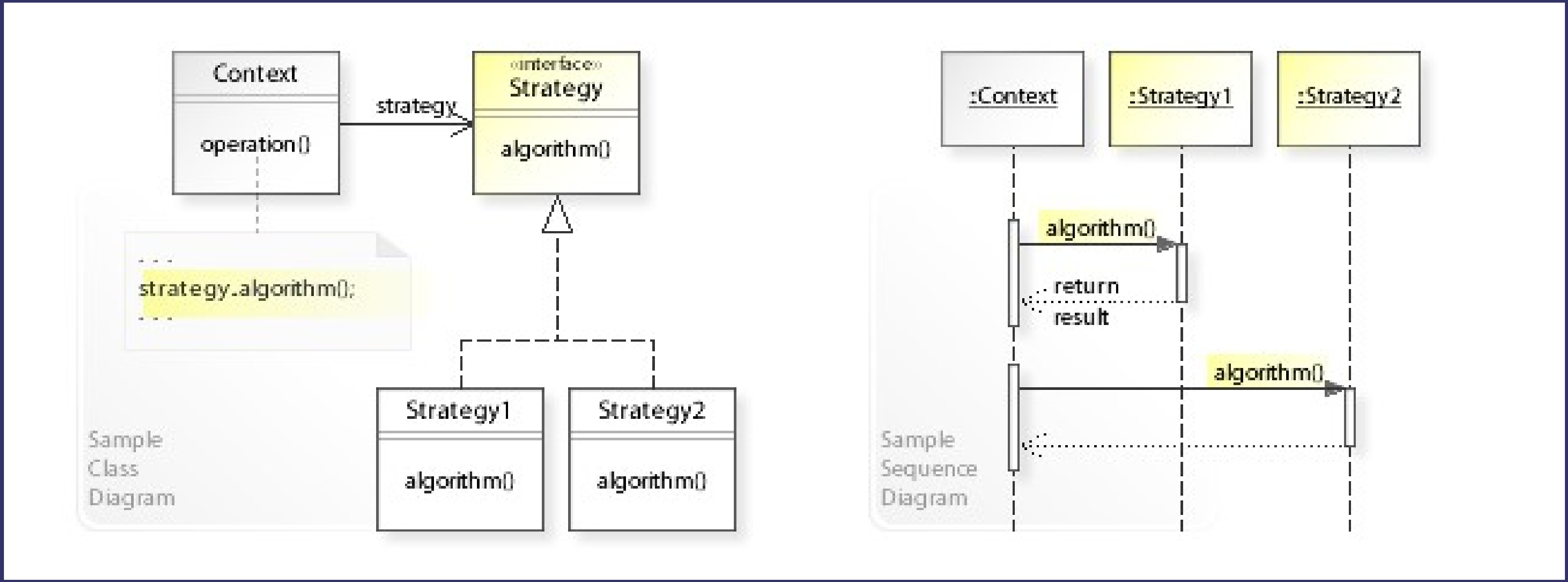


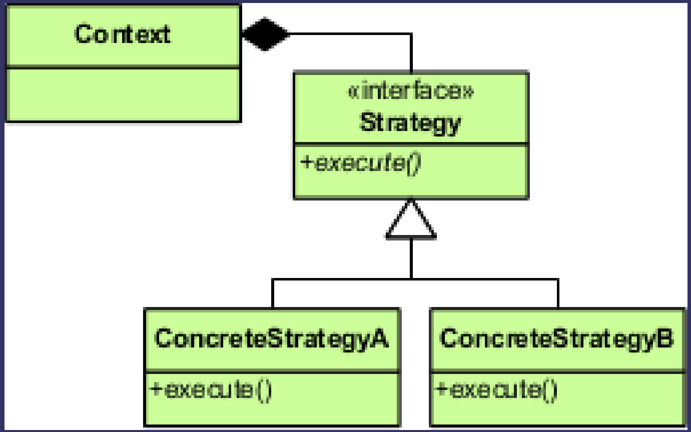
Visuals!

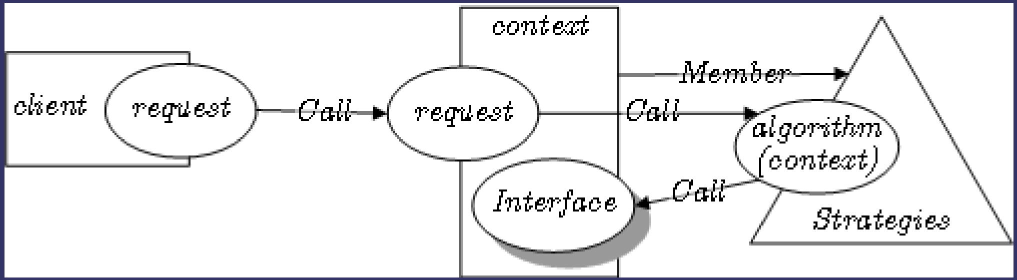

Context
«interface»
Strategy
+ execute()
ConcreteStrategyA
+ execute()
ConcreteStrategyB
+ execute()

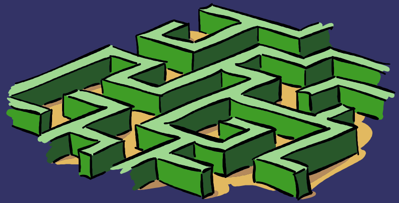


More Visuals!



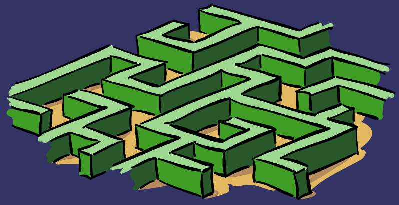






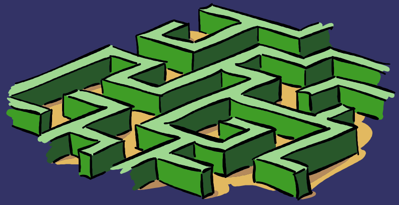
When should I use it?

- When you have a part of a "Class" that's subject to change frequently
- When you have many related sub-classes which only differ in behavior, it's a good time to consider using a Strategy pattern.
- When you want to hide complex logic or data that the client doesn't need to know about.



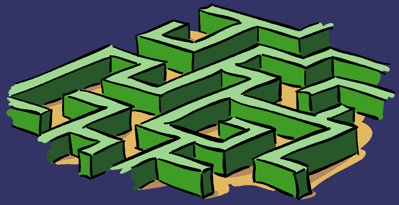
Strategy & Open/Closed Principle

- According to the strategy pattern, the behaviors of a class should not be inherited. Instead they should be encapsulated using interfaces.
- This is compatible with the open/closed principle (OCP), which proposes that classes should be open for extension but closed for modification.



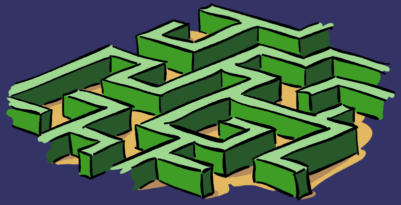
TL;DR

- We define a strategy pattern, we define a family of algorithms and encapsulate them (the behaviors) into classes so that they are interchangeable
- The benefit of using strategy pattern is the independent algorithms and behaviors can vary independently of the clients that are consuming them



References

- https://en.wikipedia.org/wiki/Strategy_pattern
- http://robdodson.me/javascript-design-patterns-strategy/
- http://www.blackwasp.co.uk/gofpatterns.aspx
- https://sourcemaking.com/design_patterns/strategy
- https://www.youtube.com/watch?v=QZIvIny1Onk
- https://www.youtube.com/watch?v=Nx8iUv-ZnPw



Good Luck!