Strategy ID

Student:

Izabela Kuźniar

Teacher:

Andrea Corradini

Course:

Software Design Patterns

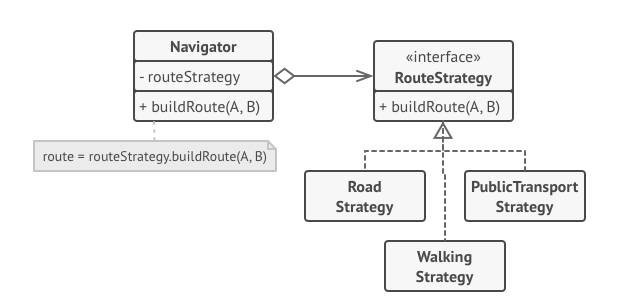
# Name and category

Strategy is a behavioral pattern.

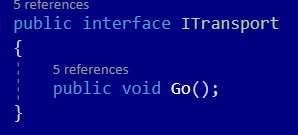
# Intent:

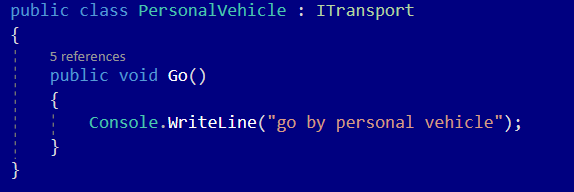
Defines set of behaviors that can be plugged into the application and changing the functionality on the run time.

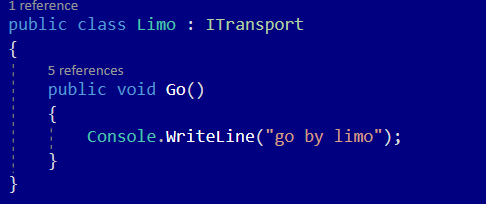
# Structure as a UML class diagram

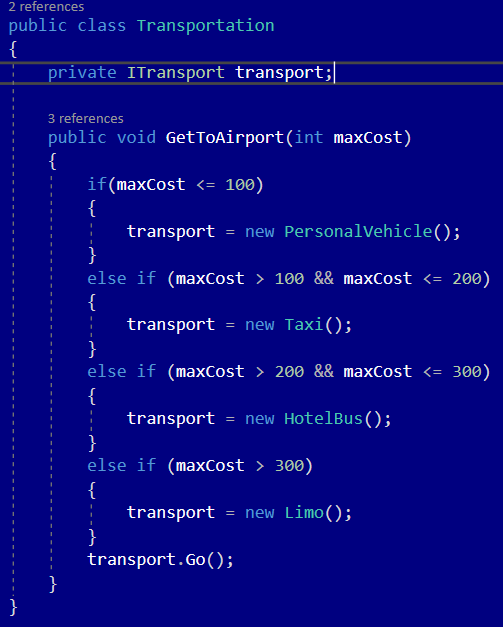


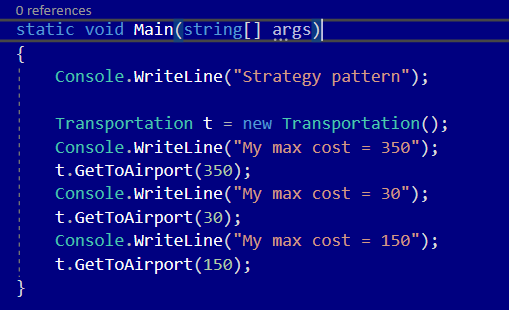
# Implementation:











# Consequences:

Benefits:

* Ability to swap algorithms used inside an object at runtime.
* Ability to isolate the implementation details of an algorithm from the code that uses it.
* Ability to replace inheritance with composition
* Open/Closed Principle. Ability to introduce new strategies without having to change the context.

Drawbacks:

* If you only have a couple of algorithms and they rarely change, there’s no real reason to overcomplicate the program with new classes and interfaces coming along with it.
* Clients must be aware of the difference between strategies to be able to select a proper one.
* A lot of modern programming languages have functional type support that lets you implement different versions of an algorithm inside a set of anonymous functions. Then you could use these exactly as you’d have used the strategy objects, but without bloating your code with extra classes and interfaces.

# Known uses

* Manage algorithms

# Related patterns

1. Bridge, State, Strategy (and to some degree Adapter) have very similar structures. Indeed, all of these patterns are based on composition, which is delegating work to other objects. However, they all solve different problems. A pattern isn’t just a recipe for structuring your code in a specific way. It can also communicate to other developers the problem the pattern solves.
2. Command and Strategy may look similar because you can use both to parametrize an object with some action. However, they have very different intents.
   1. Command can be used to convert any operation into an object. The operation’s parameters become fields of that object. The conversion lets you defer execution of the operation queue it, store the history of commands, send commands to remote services, etc.
   2. On the other hand, Strategy usually describes different ways of doing the same thing, letting you swap these algorithms within a single context class.
3. Decorator lets you change the skin of an object, while Strategy lets you change the guts.
4. Template Method is based on inheritance: it lets you alter parts of an algorithm by extending those parts in subclasses. Strategy is based on composition: you can alter parts of the object’s behavior by supplying it with different strategies that correspond to that behavior. Template Method works at the class level, so it’s static. Strategy works on the object level, letting you switch behaviors at runtime.
5. State can be considered as an extension of Strategy. Both patterns are based on composition: they change the behavior of the context by delegating some work to helper objects. State doesn’t restrict dependencies between concrete states, letting them alter the state of the context at will.