sages

Python: Wzorce Projektowe

Termin:
18-20.12.2024

Prowadzi:

Paweł Cyrta

Harmonogram szkolenia

Dzień 1

Wstęp
Rodzaje wzorców
UML
Idiomy w Pythonie
OOP w Pythonie
Protokoły
Dekoratory

Dzień 2

Behawioralne wzorce

Strukturalne wzorce

Konstrukcyjne wzorce

Dzień 3

Refaktoryzacja
Dobre praktyki
Code review
Paradygmaty
Profilowanie i Testy
Optymalizacja
Praca z AI

sages

Day 2

Harmonogram szkolenia

Dzień 2

Konstrukcyjne wzorce
2.1. Singleton
2.2. Borg
2.3. Factory Method
2.4. Abstract Factory
2.5. Builder
2.6. Prototype

Strukturalne wzorce
2.7. Composite
2.8. Adapter
2.9. Decorator
2.10. Façade
2.11. Flyweight
2.12. Bridge
2.13. Proxy

Behawioralne wzorce
2.14. Memento
2.15. State
2.16. Iterator
2.17. Strategy
2.18. Template Method
2.19. Command
2.20. Observer
2.21. Mediator
2.22. Chain of Responsibility
2.23. Visitor
2.24. Gateway
2.25. Interpreter
2.26. State Machine

Paradigms
2.27. Callback
2.28. Event Programming

sages

Structural - Relationships between objects

Behavioral - Communication and interaction between objects

Creational - Creating new objects

sages

Design Patterns

Gang-Of-Four Design Patterns

Creational	Structural	Behavioral
Abstract Factory Builder Factory Method Object Pool Prototype	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain Of Responsibility Command Interpreter Iterator Mediator Memento Null Object Observer Strategy Template Method

sages

Behavioral Design Patterns

- Chain of Responsibility (polish: Łańcuch zobowiązań)
- Command (polish: Polecenie)
- Interactor (polish: Interactor)
- Mediator (polish: Mediator)
- Memento (polish: Pamiątka)
- Observer (polish: Obserwator)
- State (polish: Stan)
- Strategy (polish: Strategia)
- Visitor (polish: Odwiedzający)
- Interpreter (polish: Interpreter)
- Template Method (polish: Metoda szablonowa)

sages

Structural Design Patterns

Bridge (polish: Most)

Composite (polish: Kompozyt)

Decorator (polish: Dekorator)

Façade (polish: Fasada)

Flyweight (polish: Pyłek)

Proxy (polish: Pełnomocnik)

Adapter (polish: Adapter)

sages

Creational Design Patterns

Abstract Factory (polish: Fabryka Abstrakcyjna)

Builder (polish: Budowniczy)

Prototype (polish: Prototyp)

Singleton (polish: Singleton)

Factory Method (polish: Metoda wytwórcza)

sages

1.3.6. Paradigm

EFAP (Easier to ask for forgiveness than permission)

Metaclasses

Borg

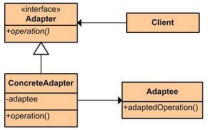
Mixin

sages

Day 1
part 1

sages

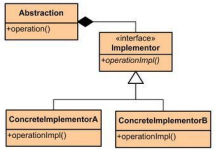
Adapter



Type: Structural

What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

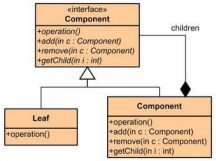
Bridge



Type: Structural

What it is:
Decouple an abstraction from its implementation so that the two can vary independently.

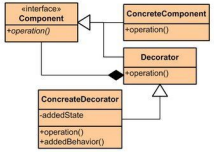
Composite



Type: Structural

What it is:
Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions of objects uniformly.

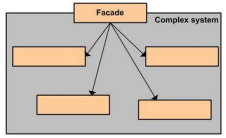
Decorator



Type: Structural

What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.

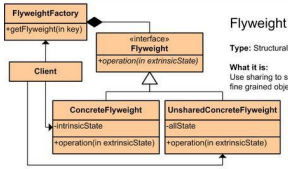
Facade



Type: Structural

What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

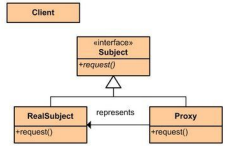
Flyweight



Type: Structural

What it is:
Use sharing to support large numbers of fine grained objects efficiently.

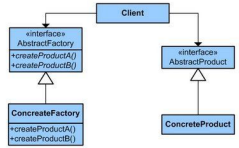
Proxy



Type: Structural

What it is:
Provide a surrogate or placeholder for another object to control access to it.

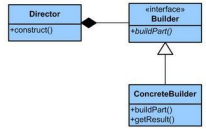
Abstract Factory



Type: Creational

What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.

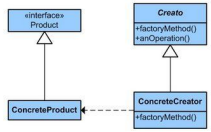
Builder



Type: Creational

What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.

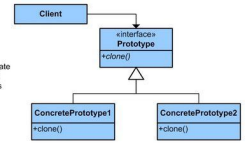
Factory Method



Type: Creational

What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.

Prototype



Type: Creational

What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc.

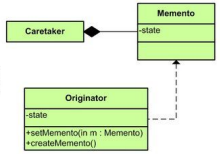
sages

- C Abstract Factory
- S Adapter
- S Bridge
- C Builder
- B Chain of Responsibility
- B Command
- S Composite
- S Decorator
- S Facade
- C Factory Method
- S Flyweight
- B Interpreter
- B Iterator
- B Mediator
- B Memento
- C Prototype
- S Proxy
- B Observer
- C Singleton
- B State
- B Strategy
- B Template Method
- B Visitor

Memento

Type: Behavioral

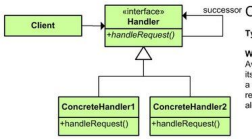
What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.



Chain of Responsibility

Type: Behavioral

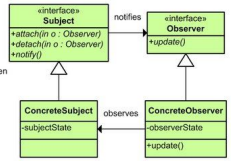
What it is:
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.



Observer

Type: Behavioral

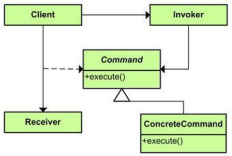
What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



Command

Type: Behavioral

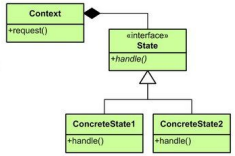
What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.



State

Type: Behavioral

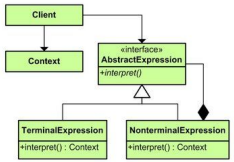
What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.



Interpreter

Type: Behavioral

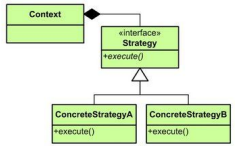
What it is:
Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.



Strategy

Type: Behavioral

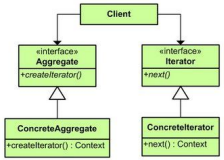
What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



Iterator

Type: Behavioral

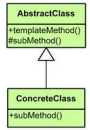
What it is:
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.



Template Method

Type: Behavioral

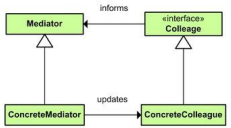
What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



Mediator

Type: Behavioral

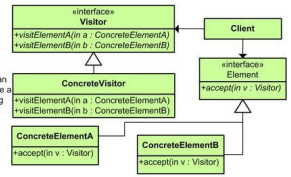
What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.



Visitor

Type: Behavioral

What it is:
Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.



Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc.

sages

Singleton - wzorzec konstrukcyjny

To ensure a class has a single instance

Database connection pool

~~HTTP Gateway~~

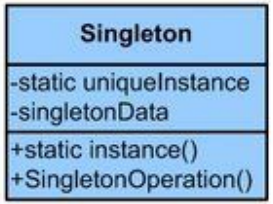
Settings

Main game/program window

Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.



sages

Borg - wzorzec konstrukcyjny

The real reason that borg is different comes down to subclassing.
If you subclass a borg, the subclass' objects have the same state
as their parents classes objects, unless you explicitly override
the shared state in that subclass.

Each subclass of the singleton pattern has its own state
and therefore will produce different objects.
Also in the singleton pattern the objects are actually the same,
not just the state (even though the state is the only thing that really matters).

https://stackoverflow.com/a/1318444

```
class Borg:
    shared_state: dict = {}

    def __init__(self):
        self.__dict__ = self.shared_state
```

sages

Factory Method - wzorzec konstrukcyjny

Defer the creation of an object to subclasses

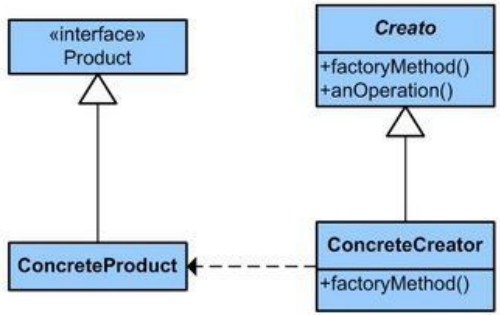
Relays on inheritance and polymorphism

Adds flexibility to the design

Factory Method

Type: Creational

What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.



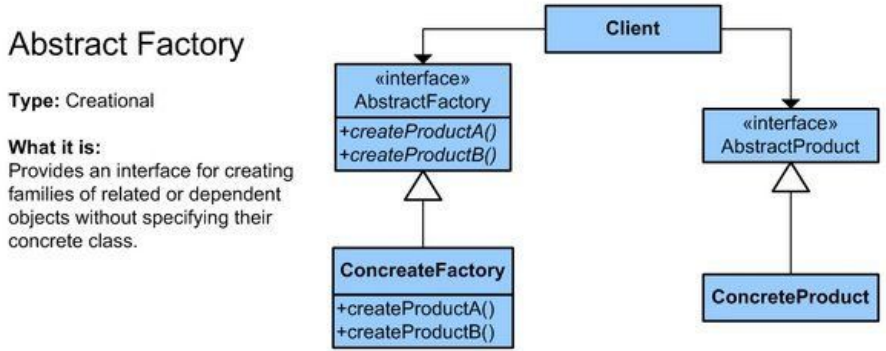
Abstract Factory - wzorzec konstrukcyjny

Provide an interface for creating families of related objects

Factory Method is a method

Abstract Factory is an abstraction (interface)

Used for theme support (which generates buttons, inputs etc)



sages

Builder- wzorzec konstrukcyjny

To separate the construction of an object from its representation

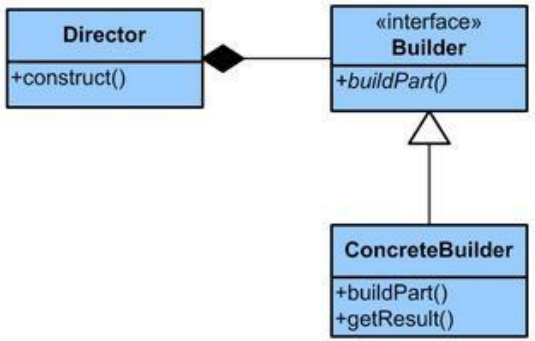
The same construction algorithm can be applied to different representations

Usecase: Export data to different formats

Builder

Type: Creational

What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.



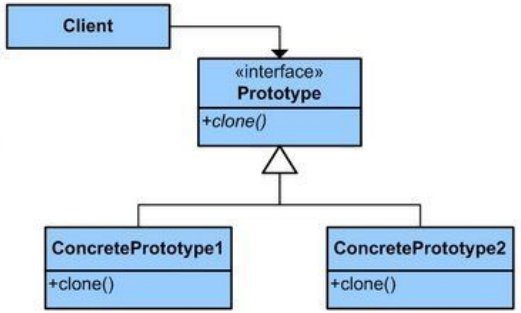
Prototype - wzorzec konstrukcyjny

Create new object by copying an existing object

Prototype

Type: Creational

What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.



sages

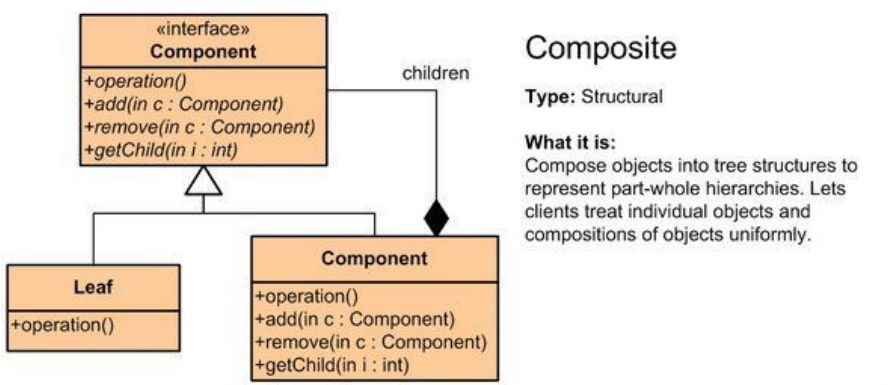
Day 1
part 2

Composite - wzorzec strukturalny

Represent a hierarchy of objects

Groups (and subgroups) objects in Keynote

Files in a Folder; when you move folder you also move files
allows you to represent individual entities
and groups of entities in the same manner.
is a structural design pattern that
lets you compose objects into a tree.



is great if you need the option of swapping hierarchical relationships around.

makes it easier for you to add new kinds of components

conform to the Single Responsibility Principle in the way that
it separates the aggregation of objects from the features of the object.

Adapter - wzorzec strukturalny

Convert an interface of an object to a different form

Like power socket adapter for US and EU

Refactoring of a large application

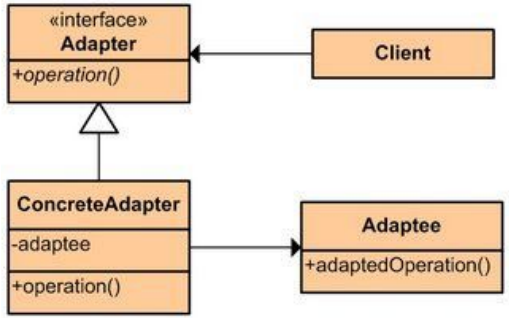
Working with legacy code / database

Niekompatybilne API dwóch systemów

Wymagające różnych sposobów uwierzytelniania (OAuth2, BasicAuth)

Tłumaczenie pomiędzy różnymi formatami danych
(SOAP/XML, REST/JSON)

Iteracyjne przepisywanie legacy systemu na nowy,
ale tak, aby móc wciąż korzystać ze starego



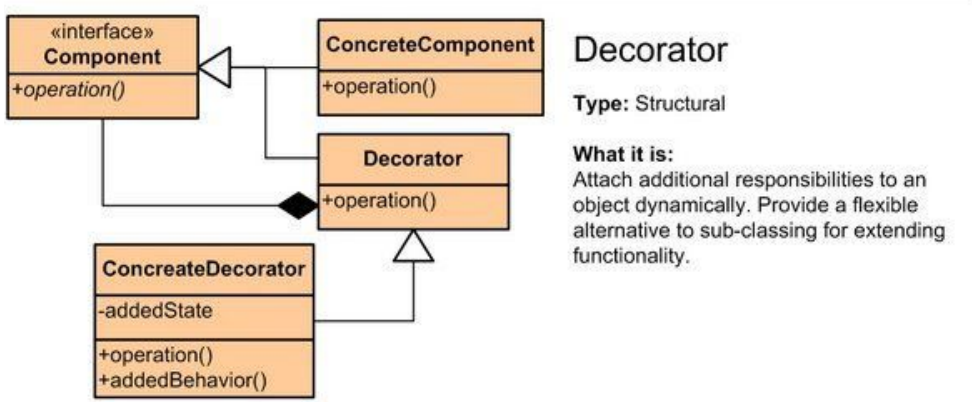
Adapter

Type: Structural

What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

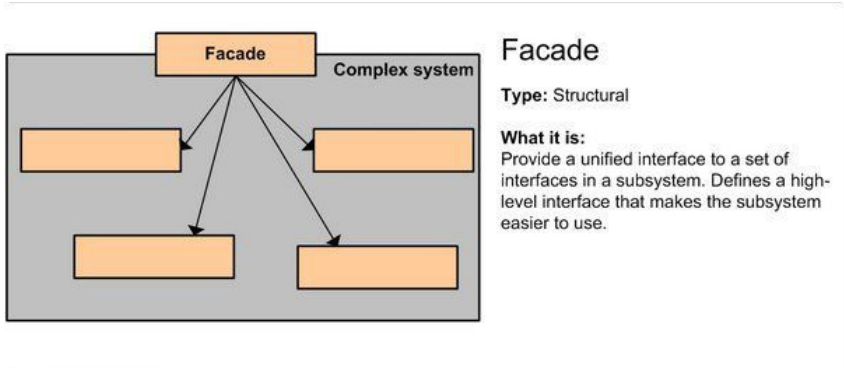
Decorator - wzorzec strukturalny

Add additional behavior to an object



Façade - wzorzec strukturalny

Provide simple interface to complex system



Flyweight - wzorzec strukturalny

In applications with large number of objects

Objects take significant amount of memory

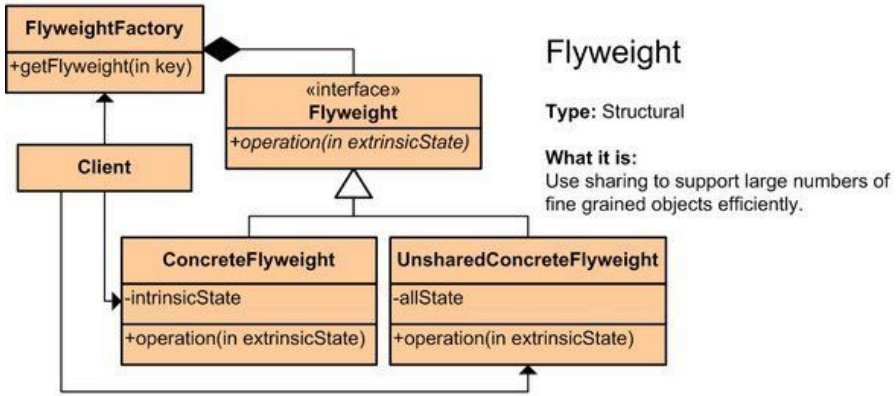
Reduce memory consumed by objects

Imagine mapping application, such as: Open Street Maps, Google Maps, Yelp, Trip Advisor etc.

There are thousands points of interests such as Cafe, Shops, Restaurants, School etc.

Icons can take a lot of memory, times number of points on the map

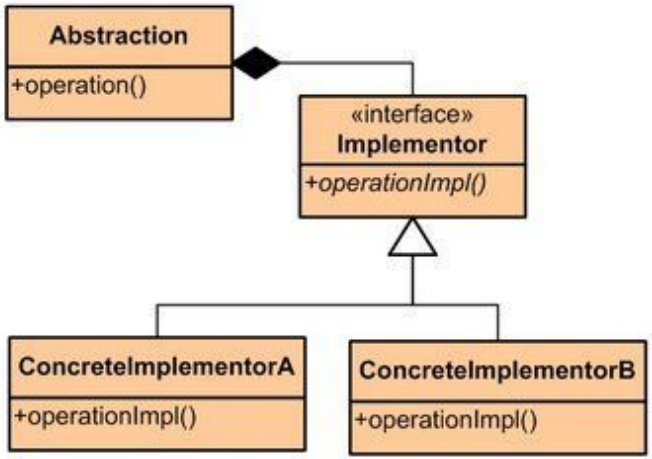
It might crash with out of memory error (especially on mobile devices)



sages

Bridge - wzorzec strukturalny

Nested hierarchies of classes



Bridge

Type: Structural

What it is:
Decouple an abstraction from its implementation so that the two can vary independently.

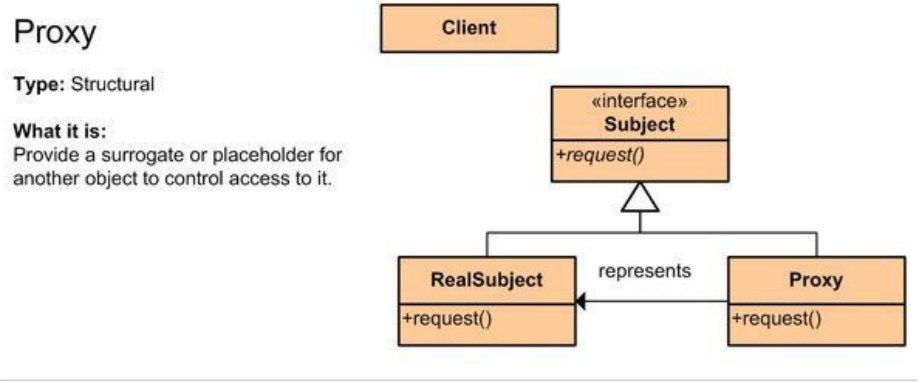
sages

Proxy - wzorzec strukturalny

Create a proxy, or agent for a remote object

Agent takes message and forwards to remote object

Proxy can log, authenticate or cache messages



sages

Day 1
part 3

Memento - wzorzec behawioralny

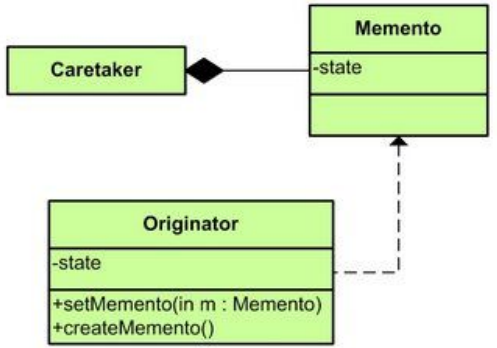
Undo operation

Remembering state of objects

Memento

Type: Behavioral

What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.



State- wzorzec behawioralny

Changes based on class

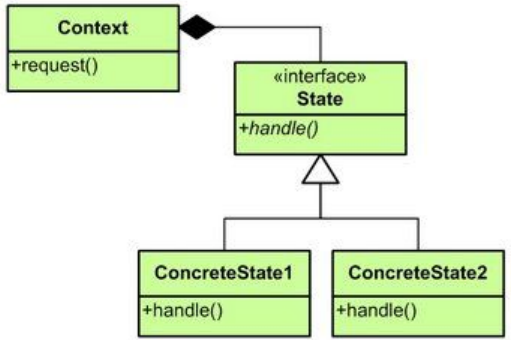
Open/Close principle

Using polymorphism

State

Type: Behavioral

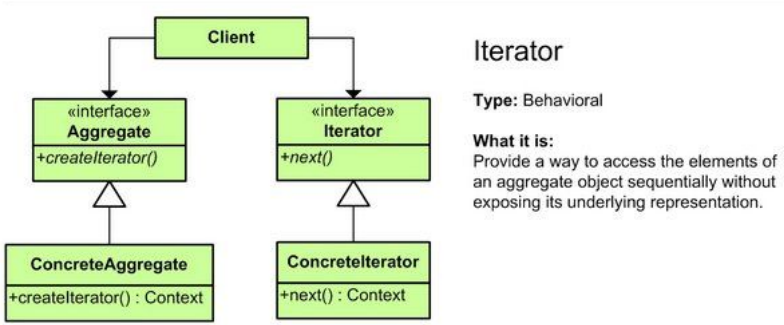
What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.



Iterator - wzorzec behawioralny

Mechanizm ułatwiający przechodzenie po kolekcji

History (like browser history)



sages

Strategy - wzorzec behawioralny

Similar to State Pattern

No single states

Can have multiple states

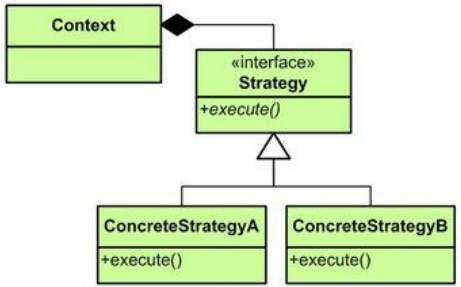
Different behaviors are represented by strategy objects

Store images with compressor and filters

Strategy

Type: Behavioral

What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



Template Method - wzorzec behawioralny

Duplicated code

Bank application with audit trail (all actions)

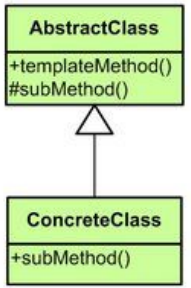
Record task history

Audits
Not enforced to record in audit trail

Template Method

Type: Behavioral

What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



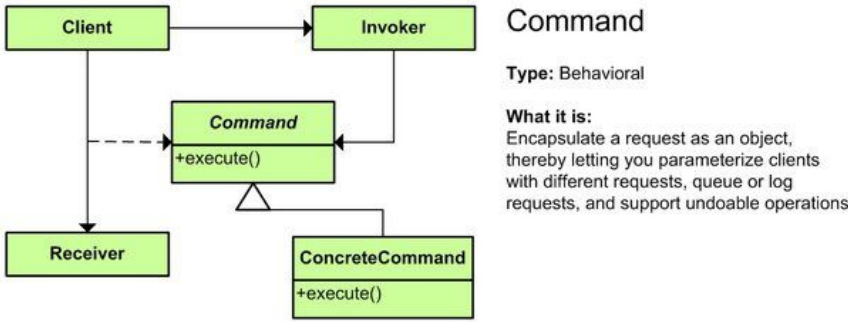
Command - wzorzec behawioralny

Parallel processing or thread pools

Transactional behaviour
Rollback whole set of commands, or defer till later

Wizards

Receiver — The Object that will receive and execute the comman



Invoker — Which will send the command to the receiver

Command Object — Itself, which implements an execute, or action method, and contains all required information

Client — The main application or module which is aware of the Receiver, Invoker and Commands

GUI Buttons, menus

Macro recording

Multi level undo/redo

Networking — send whole command objects across a network, even as a batch

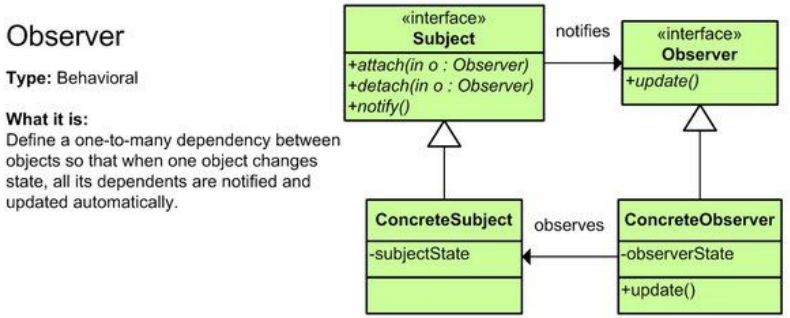
Observer - wzorzec behawioralny

When the state of the object changes
and you need to notify other objects about this change

Notify chart about changes in data to refresh

Spreadsheet formulas

Push or pull style of communication



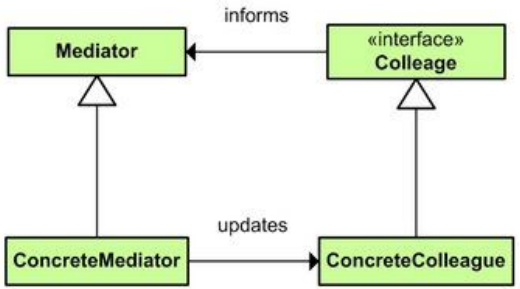
Mediator - wzorzec behawioralny

Input fields which needs to collaborate

Cannot submit form if all required fields are not filled

If you select article in list of articles, editor form with curre

Auto slug-field based on title content



Mediator

Type: Behavioral

What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

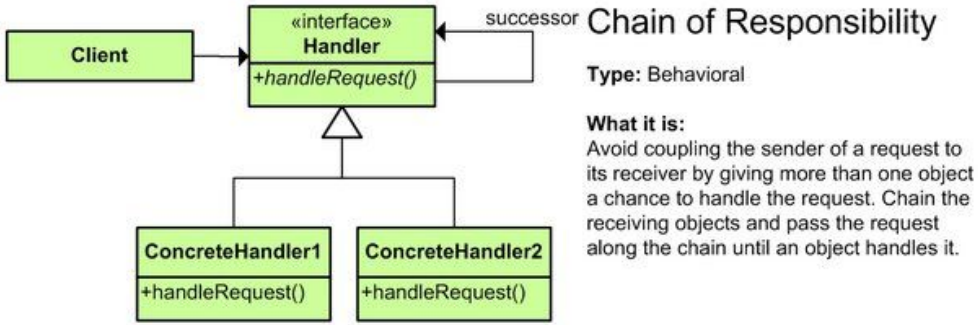
sages

Chain of Responsibility - wzorzec behawioralny

Chain of objects

Create a pipeline of classes with different responsibilities

Open/Close Principle for adding new handlers

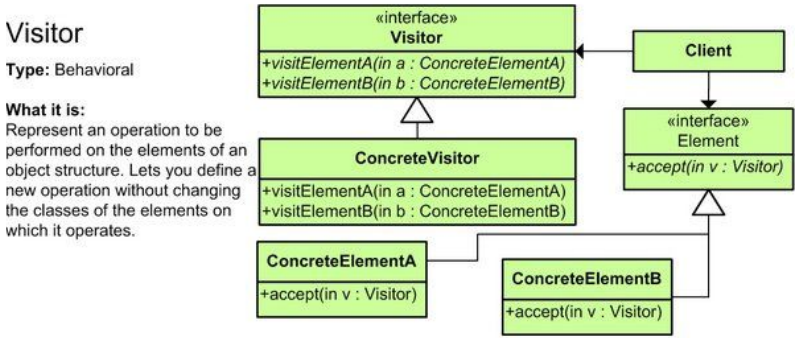


Visitor - wzorzec behawioralny

Add new operations to an object structure without modifying it

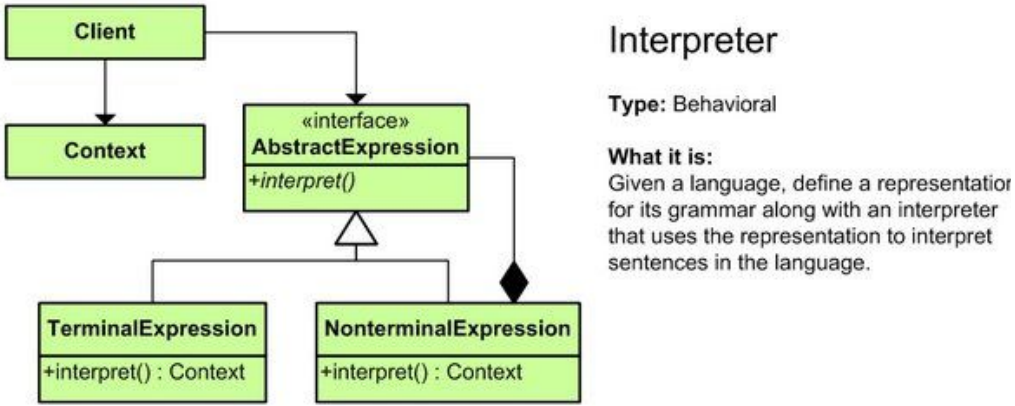
For building editors

Open/Close Principle

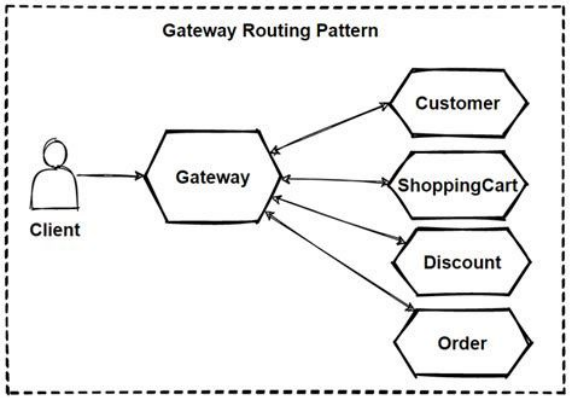


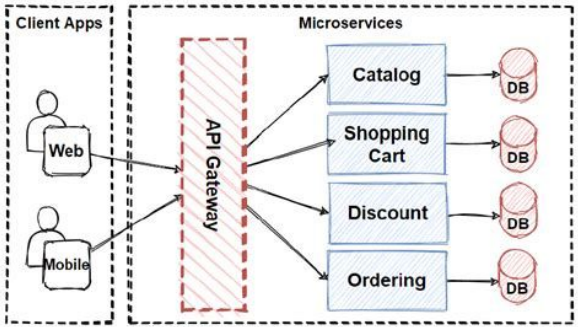
sages

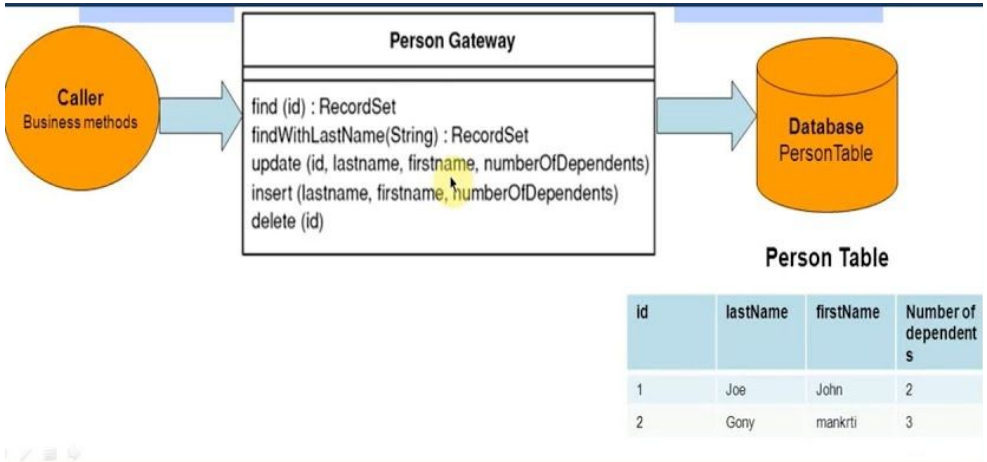
Interpreter - wzorzec behawioralny



Gateway - wzorzec behawioralny







sages

State Machine - wzorzec behawioralny

State Machine imposes a structure to automatically
change the implementation from one object to the next

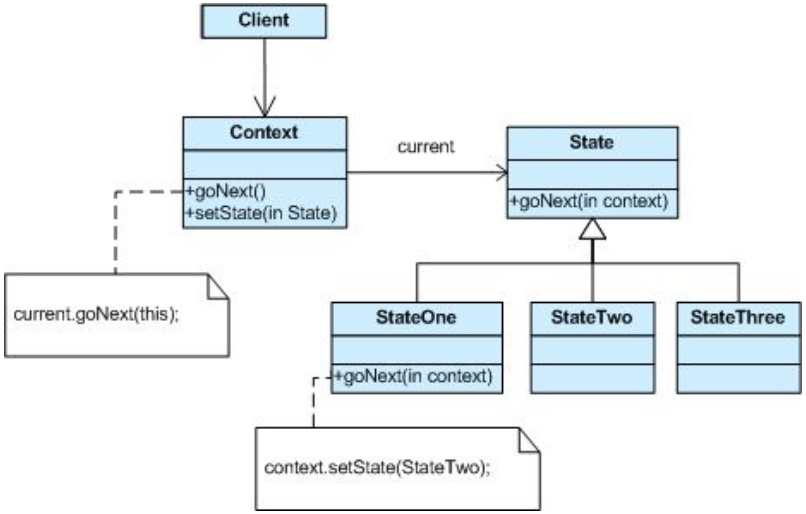
The current implementation represents the state that a system is in.

System behaves differently from one state to the next

The code that moves the system from one state to the next

Each state can be run() to perform its behavior

You can pass it an input object so it can tell you
what new state to move to based on that input



Each State object decides what other states it can move to, based on the input

Each State object has its own little State table

There is a single master state transition table for the whole system

sages

Koniec

Materiały do samodzielnej nauki: