

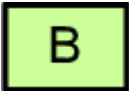
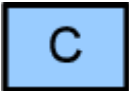
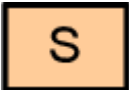
# Шаблон проектирования

Шаблон проектирования или паттерн — повторяемая архитектурная конструкция, используемая при проектировании программного обеспечения. Представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Паттерны не привязаны к какому-либо конкретному языку программирования. Это просто подход к проектированию чего-либо. Если смотреть глубже, то многие паттерны ООП были созданы на основе реальных жизненных ситуаций в проектировании вполне себе осязаемых объектов нашего мира.

## Виды паттернов

-  — поведенческие (behavioral);
-  — порождающие (creational);
-  — структурные (structural).

## Список шаблонов

C	Абстрактная фабрика	S	Фасад	S	Прокси
S	Адаптер	C	Фабричный метод	B	Наблюдатель
S	Мост	S	Приспособленец	C	Одиночка
C	Строитель	B	Интерпретатор	B	Состояние
B	Цепочка обязанностей	B	Итератор	B	Стратегия
B	Команда	B	Посредник	B	Шаблонный метод
S	Компоновщик	B	Хранитель	B	Посетитель
S	Декоратор	C	Прототип		

### Порождающие паттерны:

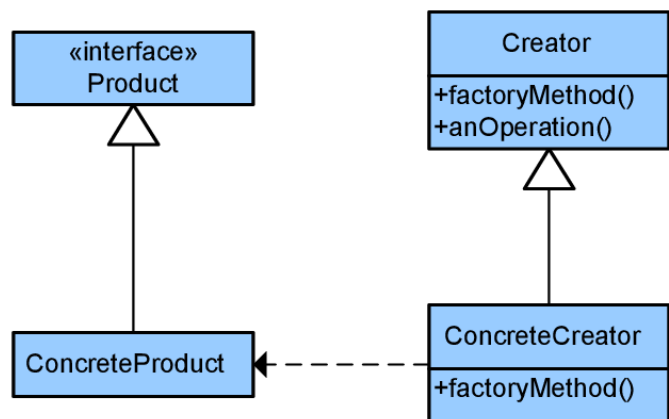
Паттерны которые создают новые объекты, или позволяют получить доступ к уже существующим. То есть те шаблоны, по которым можно создать новый автомобиль и как это лучше сделать.

### Фабричный метод *Factory method*

Тип: Поведенческий

Что это:

Определяет интерфейс для создания объекта, но позволяет подклассам решать, какой класс инстанцировать. Позволяет делегировать создание объекта подклассам.



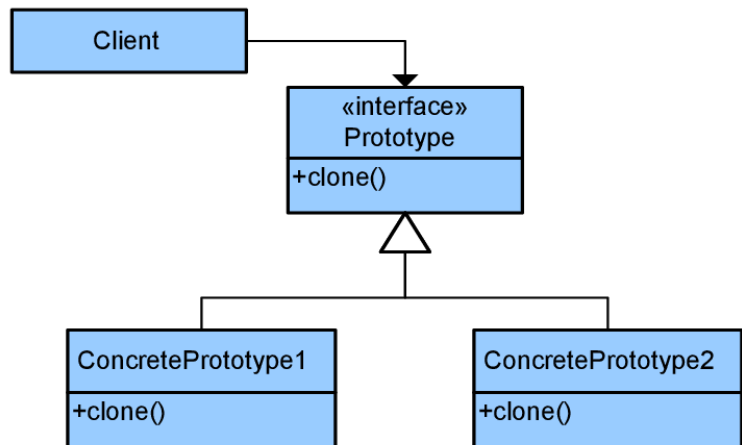
## Прототип

*Prototype*

**Тип:** Поведенческий

**Что это:**

Определяет несколько видов объектов, чтобы при создании использовать объект-прототип и создаёт новые объекты, копируя прототип.



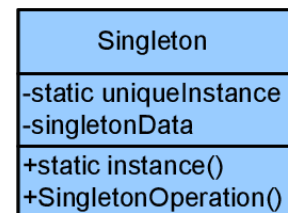
## Одиночка

*Singleton*

**Тип:** Поведенческий

**Что это:**

Гарантирует, что класс имеет только один экземпляр и предоставляет глобальную точку доступа к нему.



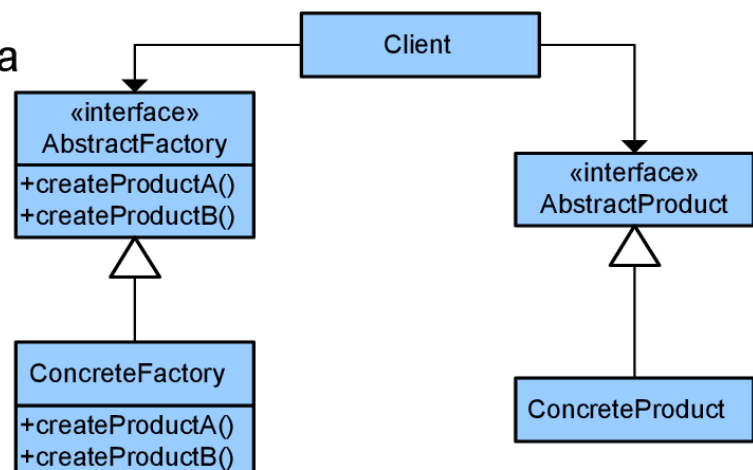
## Абстрактная фабрика

*Abstract factory*

**Тип:** Поведенческий

**Что это:**

Предоставляет интерфейс для создания групп связанных или зависимых объектов, не указывая их конкретный класс.



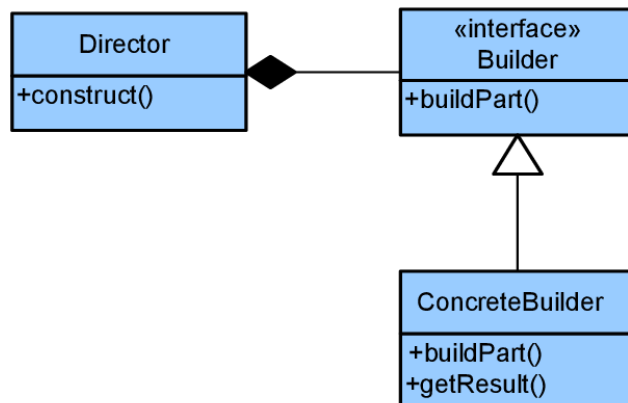
## Строитель

*Builder*

Тип: Поведенческий

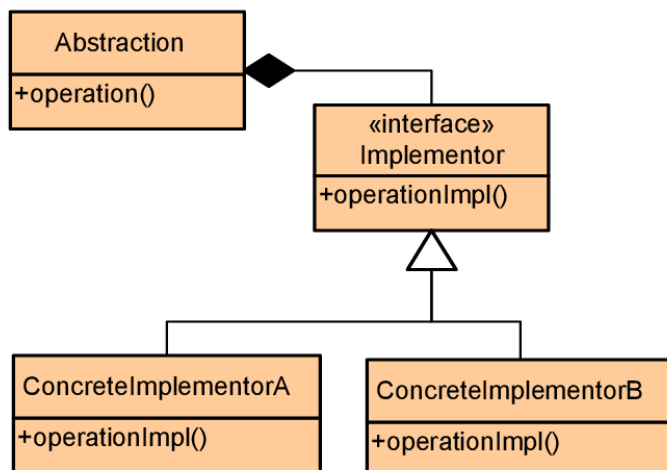
Что это:

Разделяет создание сложного объекта и инициализацию его состояния так, что одинаковый процесс построения может создать объекты с разным состоянием.



## Структурирующие паттерны

Данные паттерны помогают внести порядок и научить разные объекты более правильно взаимодействовать друг с другом.



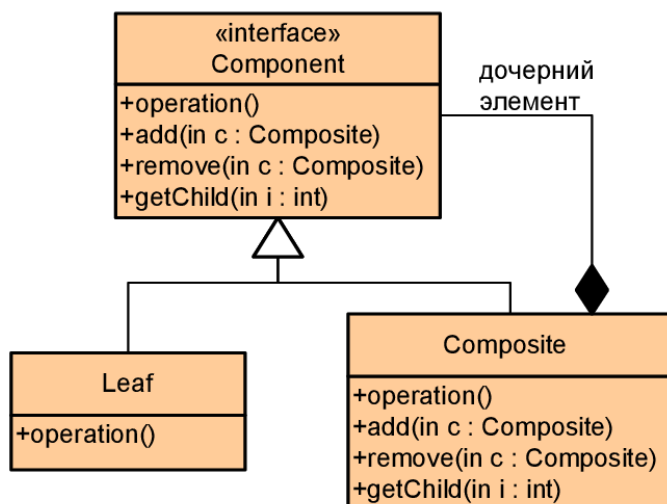
## Мост

*Bridge*

Тип: Поведенческий

Что это:

Разделяет абстракцию и реализацию так, чтобы они могли изменяться независимо.



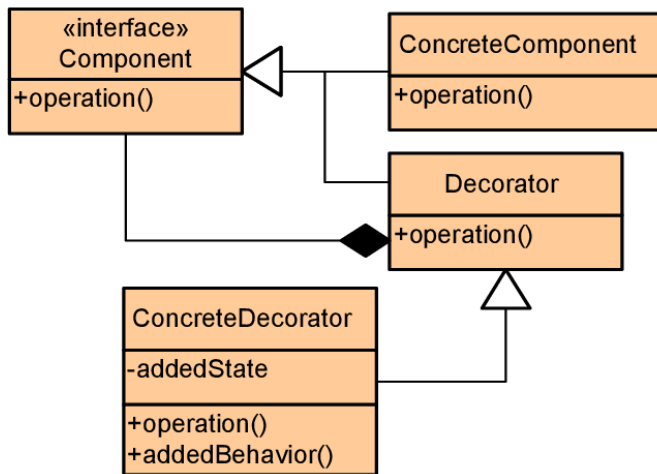
## Компоновщик

*Composite*

Тип: Поведенческий

Что это:

Компонует объекты в древовидную структуру, представляя их в виде иерархии. Позволяет клиенту одинаково обращаться как к отдельному объекту, так и к целому поддереву.

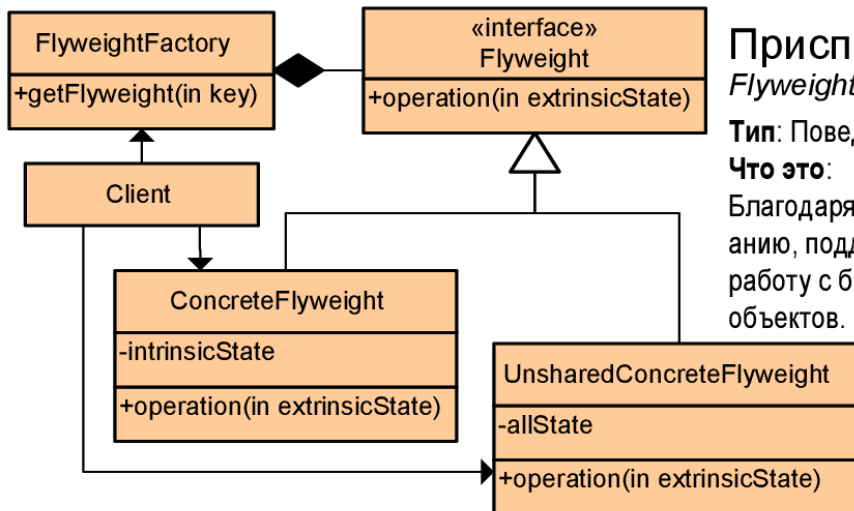


## Декоратор *Decorator*

Тип: Поведенческий

Что это:

Динамически предоставляет объекту дополнительные возможности. Представляет собой гибкую альтернативу наследованию для расширения функциональности.

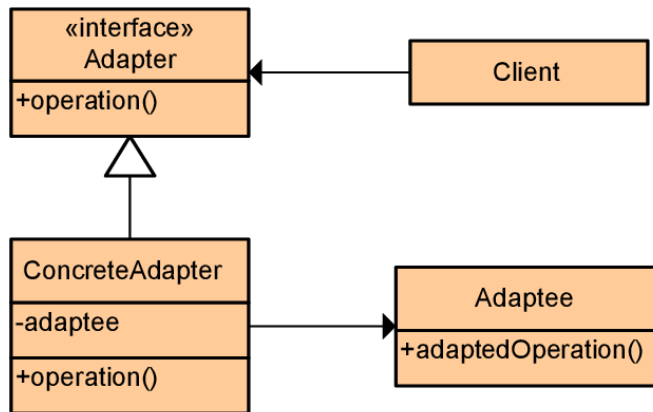


## Приспособленец *Flyweight*

Тип: Поведенческий

Что это:

Благодаря совместному использованию, поддерживает эффективную работу с большим количеством объектов.



## Адаптер *Adapter*

**Тип:** Поведенческий

**Что это:**

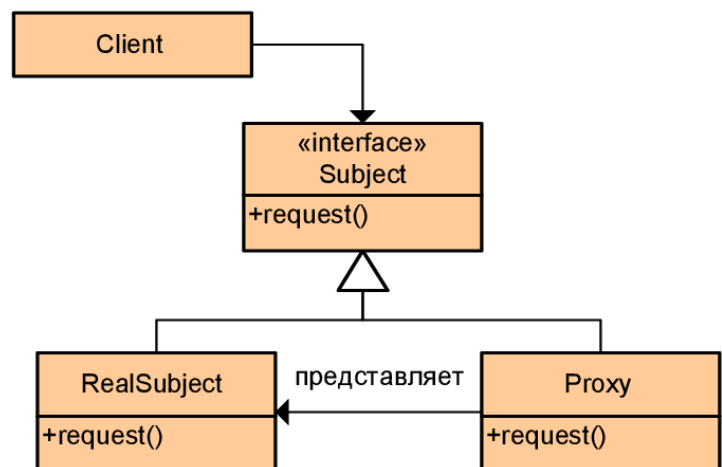
Конвертирует интерфейс класса в другой интерфейс, ожидаемый клиентом. Позволяет классам с разными интерфейсами работать вместе.

## Прокси *Proxy*

**Тип:** Поведенческий

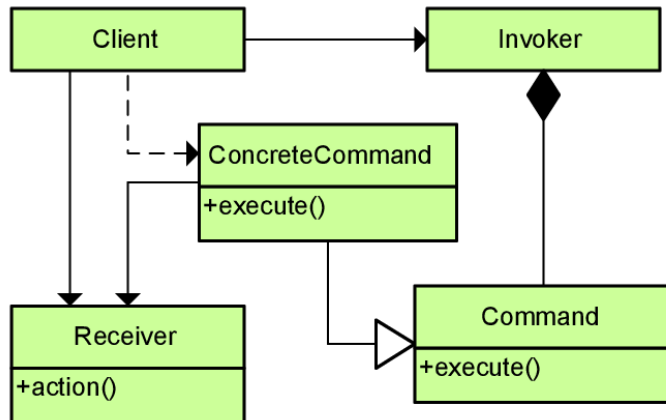
**Что это:**

Предоставляет замену другого объекта для контроля доступа к нему.



## Паттерны поведения

Эта группа паттернов позволяет структурировать подходы к обработке поведения и взаимодействия объектов. Проще говоря, как должны проходить процессы в которых существует несколько вариантов протекания событий.



### Команда *Command*

Тип: Поведенческий

Что это:

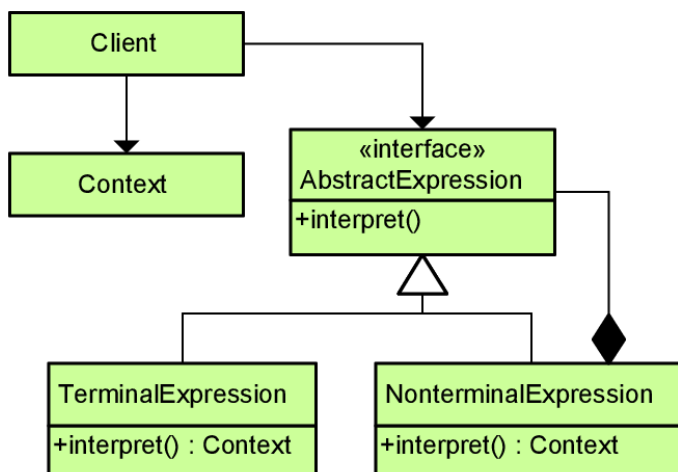
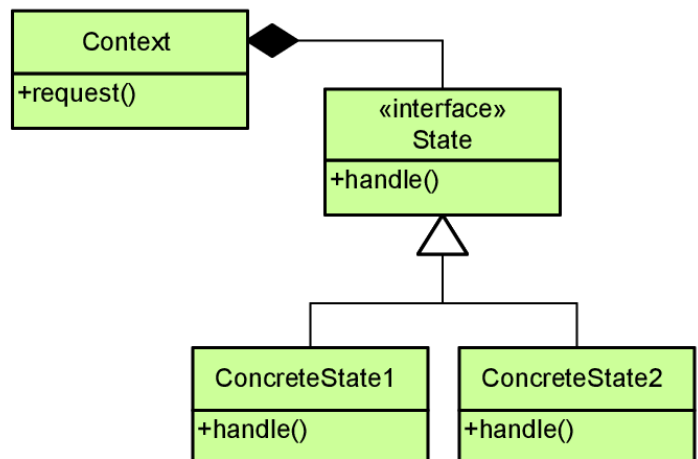
Инкапсулирует запрос в виде объекта, позволяя передавать их клиентам в качестве параметров, ставить в очередь, логировать а также поддерживает отмену операций.

### Состояние *State*

Тип: Поведенческий

Что это:

Позволяет объекту изменять своё поведение в зависимости от внутреннего состояния.



### Интерпретатор *Interpreter*

Тип: Поведенческий

Что это:

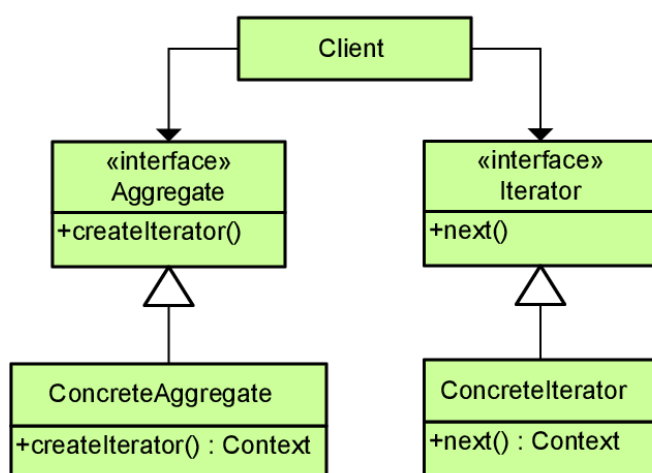
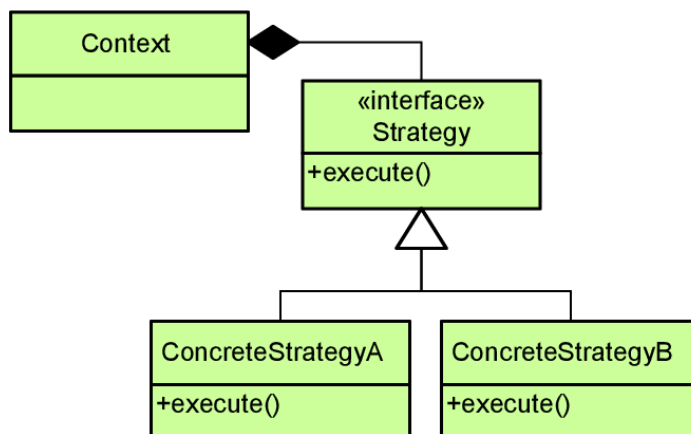
Получая формальный язык, определяет представление его грамматики и интерпретатор, использующий это представление для обработки выражений языка.

## Стратегия *Strategy*

**Тип:** Поведенческий

**Что это:**

Определяет группу алгоритмов, инкапсулирует их и делает взаимозаменяемыми. Позволяет изменять алгоритм независимо от клиентов, его использующих.



## Итератор *Iterator*

**Тип:** Поведенческий

**Что это:**

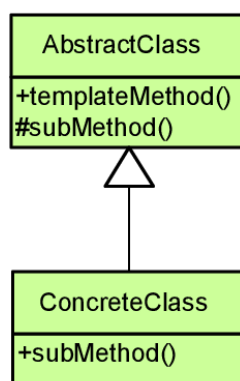
Предоставляет способ последовательного доступа к элементам множества, независимо от его внутреннего устройства.

## Шаблонный метод *Template method*

**Тип:** Поведенческий

**Что это:**

Определяет алгоритм, некоторые этапы которого делегируются подклассам. Позволяет подклассам переопределить эти этапы, не меняя структуру алгоритма.



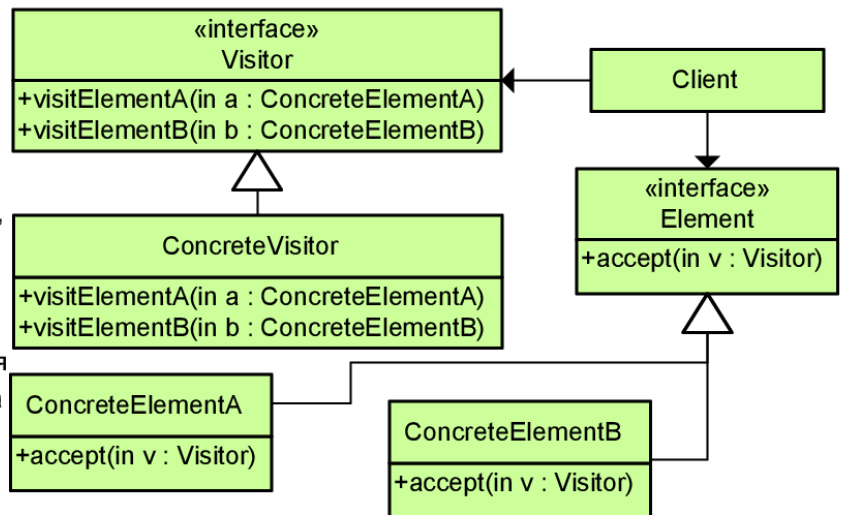


## Посетитель *Visitor*

**Тип:** Поведенческий

**Что это:**

Представляет собой операцию, которая будет выполнена над объектами группы классов. Даёт возможность определить новую операцию без изменения кода классов, над которыми эта операция проводится.

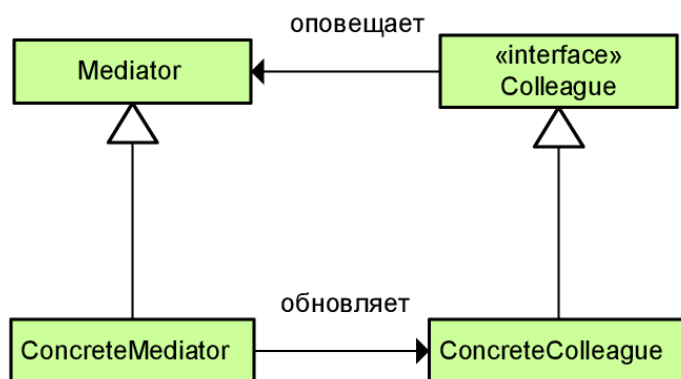


## Посредник *Mediator*

**Тип:** Поведенческий

**Что это:**

Определяет объект, инкапсулирующий способ взаимодействия объектов. Обеспечивает слабую связь, избавляя объекты от необходимости прямо ссылаться друг на друга и даёт возможность независимо изменять их взаимодействие.

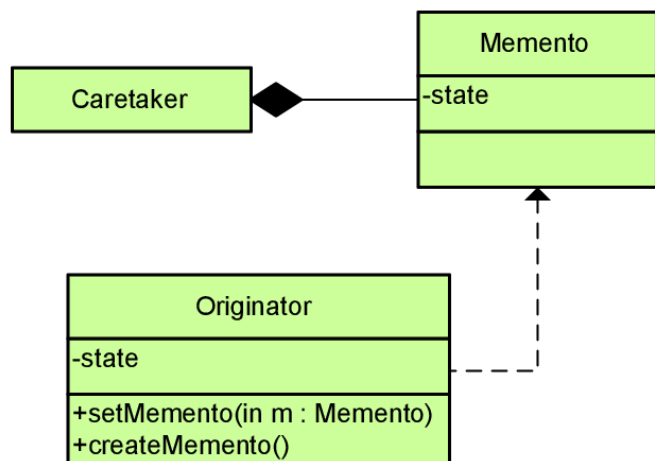


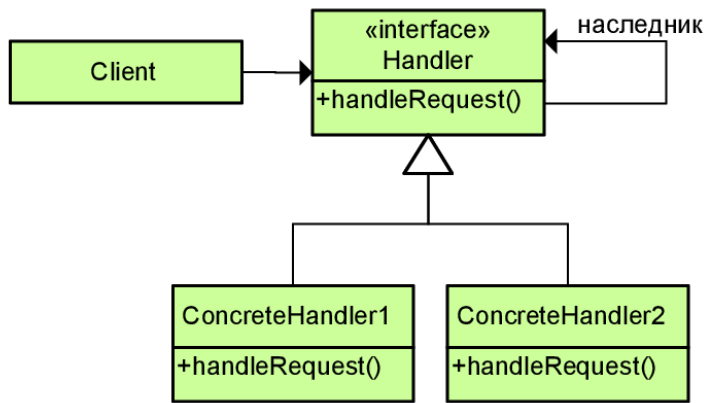
## Хранитель *Memento*

**Тип:** Поведенческий

**Что это:**

Не нарушая инкапсуляцию, определяет и сохраняет внутреннее состояние объекта и позволяет позже восстановить объект в этом состоянии.





## Цепочка обязанностей *Chain of responsibility*

**Тип:** Поведенческий

**Что это:**

Избегает связывания отправителя запроса с его получателем, давая возможность обработать запрос более чем одному объекту. Связывает объекты-получатели и передаёт запрос по цепочке пока объект не обработает его.

## Наблюдатель *Observer*

**Тип:** Поведенческий

**Что это:**

Определяет зависимость "один ко многим" между объектами так, что когда один объект меняет своё состояние, все зависимые объекты оповещаются и обновляются автоматически.

