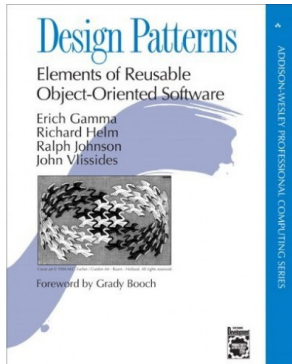




WarAngel\_alk 25 января 2014 в 21:07

# Шпаргалка по шаблонам проектирования

Разработка веб-сайтов, Анализ и проектирование систем, Проектирование и рефакторинг



Перевод pdf файла с сайта <http://www.mcdonaldland.info/> с описанием 23-х шаблонов проектирования GOF. Каждый пункт содержит [очень] короткое описание паттерна и UML-диаграмму. Сама шпаргалка доступна в pdf, в виде двух png файлов (как в оригинале), и в виде 23-х отдельных частей изображений. Для самых нетерпеливых — все файлы в конце статьи.

**Под катом — много картинок.**

## Условные обозначения

### Отношения между классами

- ◁ — агрегация (aggregation) — описывает связь «часть»—«целое», в котором «часть» может существовать отдельно от «целого». Ромб указывается со стороны «целого».
- ◼ — композиция (composition) — подвид агрегации, в которой «части» не могут существовать отдельно от «целого».
- > — зависимость (dependency) — изменение в одной сущности (независимой) может влиять на состояние или поведение другой сущности (зависимой). Со стороны стрелки указывается независимая сущность.
- > — обобщение (generalization) — отношение наследования или реализации интерфейса. Со стороны стрелки находится суперкласс или интерфейс.

### Виды паттернов

- B** — поведенческие (behavioral);
- C** — порождающие (creational);
- S** — структурные (structural).

### Список шаблонов

**C** Абстрактная фабрика

**S** Адаптер

**S** Мост

**C** Строитель

**B** Цепочка обязанностей

**B** Команда

**S** Компоновщик

**S** Декоратор

**S** Фасад

**C** Фабричный метод

**S** Приспособленец

**B** Интерпретатор

**B** Итератор

**B** Посредник

**B** Хранитель

**C** Прототип

**S** Прокси

**B** Наблюдатель

**C** Одиночка

**B** Состояние

**B** Стратегия

**B** Шаблонный метод

**B** Посетитель

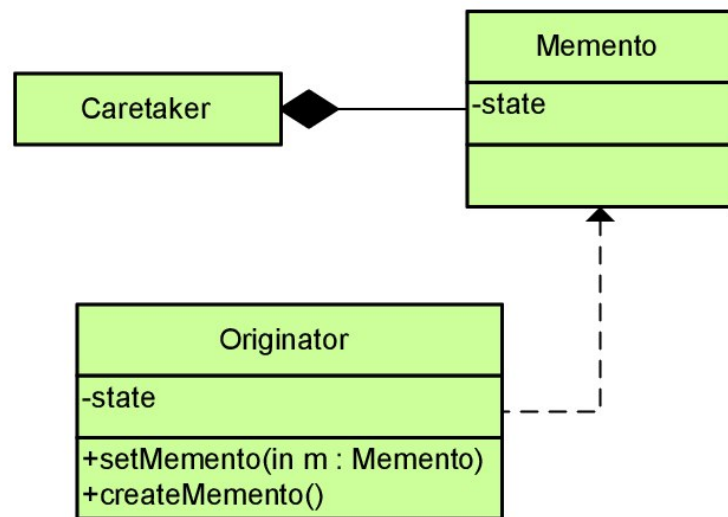
## Хранитель (memento)

### Хранитель *Memento*

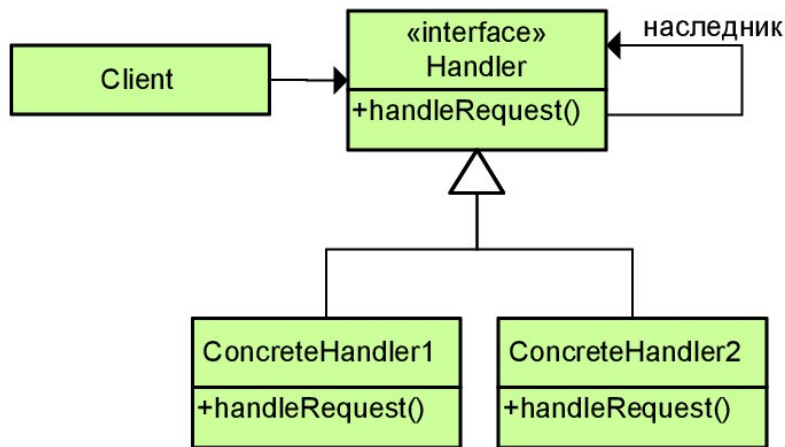
**Тип:** Поведенческий

**Что это:**

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



## Цепочка обязанностей (chain of responsibility)



## 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.

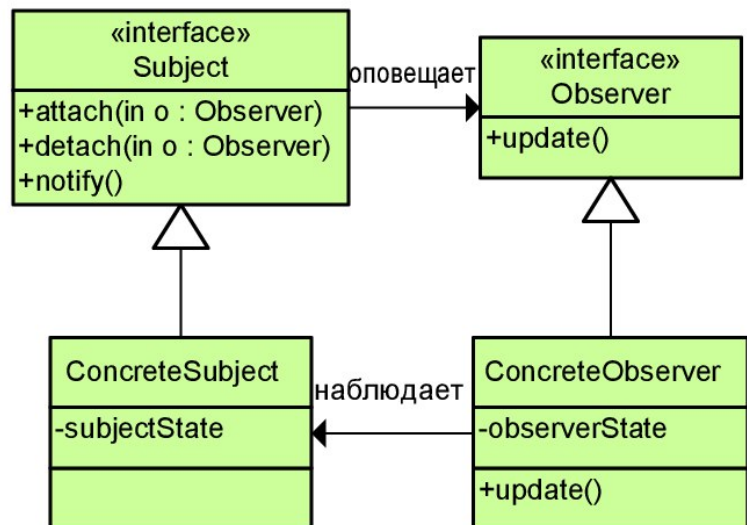
Наблюдатель (observer)

## Наблюдатель *Observer*

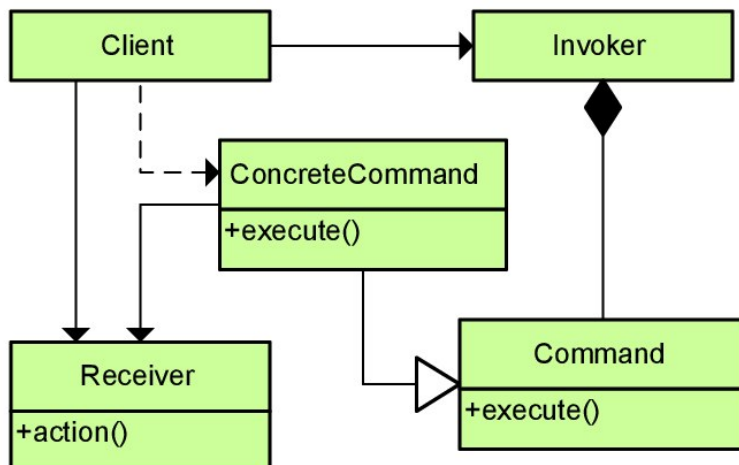
**Тип:** Поведенческий

**Что это:**

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



Команда (command)



## 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.

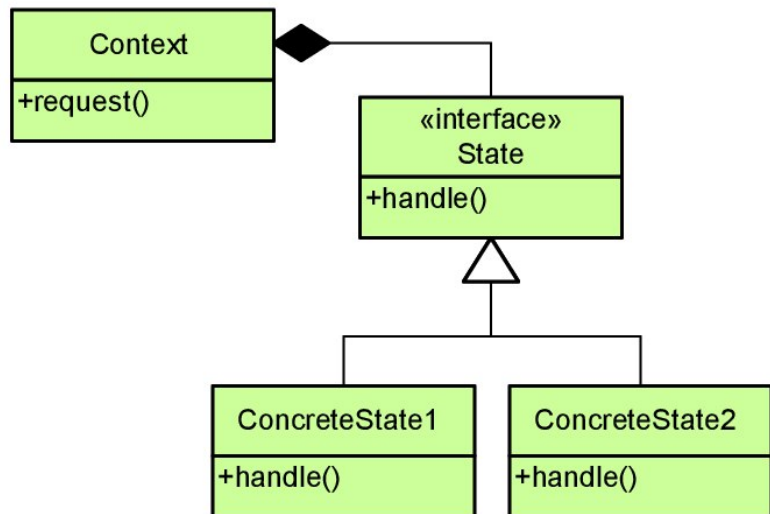
## Состояние (state)

### Состояние *State*

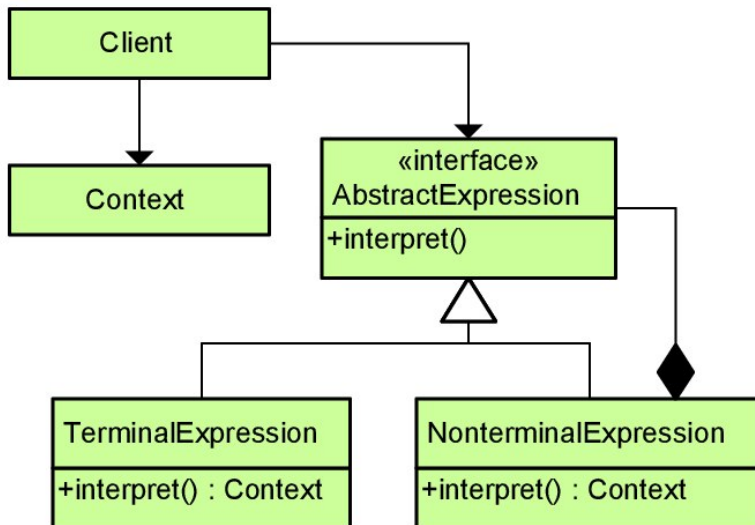
**Тип:** Поведенческий

**Что это:**

Позволяет объекту изменять своё поведение в зависимости от внутреннего состояния.



## Интерпретатор (interpreter)



### Интерпретатор *Interpreter*

**Тип:** Поведенческий

**Что это:**

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

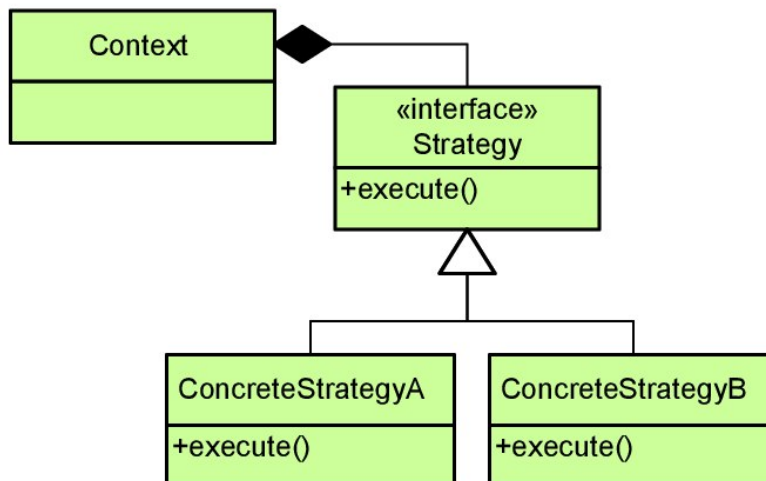
## Стратегия (strategy)

## Стратегия *Strategy*

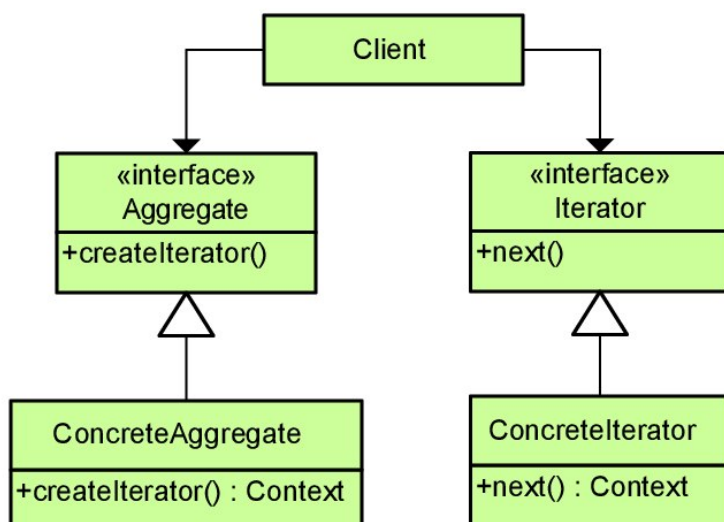
**Тип:** Поведенческий

**Что это:**

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



## Итератор (iterator)



## Итератор *Iterator*

**Тип:** Поведенческий

**Что это:**

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

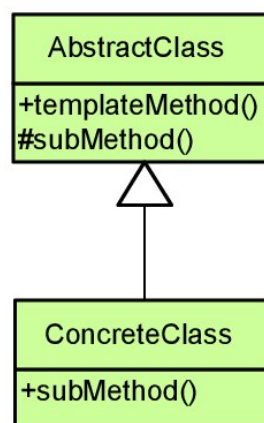
## Шаблонный метод (template method)

## Шаблонный метод *Template method*

**Тип:** Поведенческий

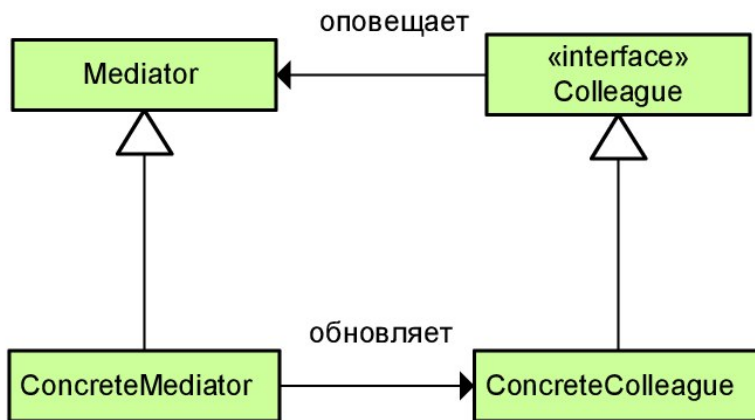
**Что это:**

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





## Посредник (mediator)



## 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.

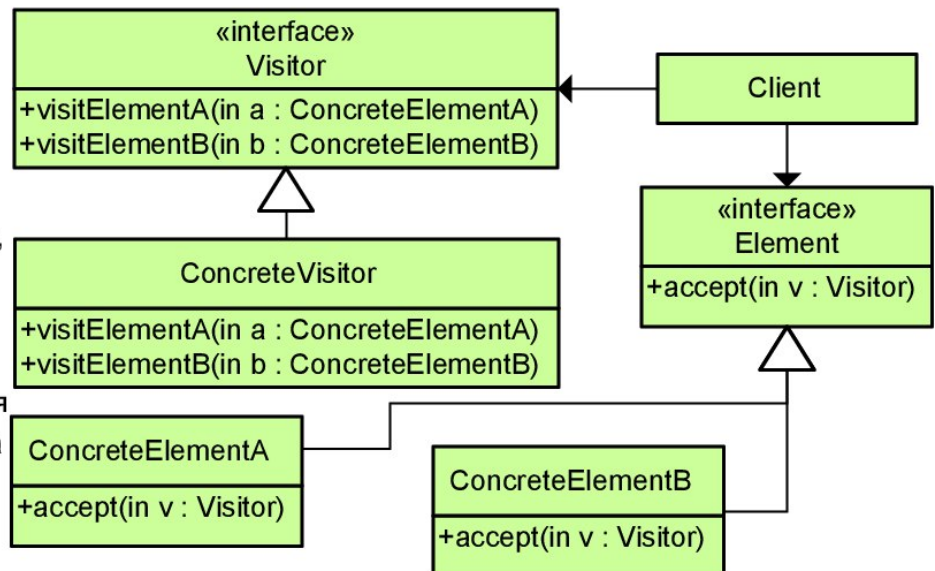
## Посетитель (visitor)

## Посетитель *Visitor*

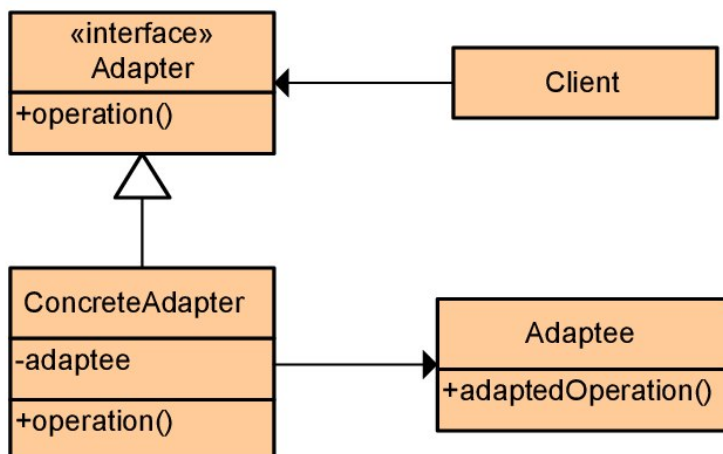
**Тип:** Поведенческий

**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.



## Адаптер (adapter)



## Адаптер *Adapter*

**Тип:** Структурный

**Что это:**

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

## Прокси (proxy)

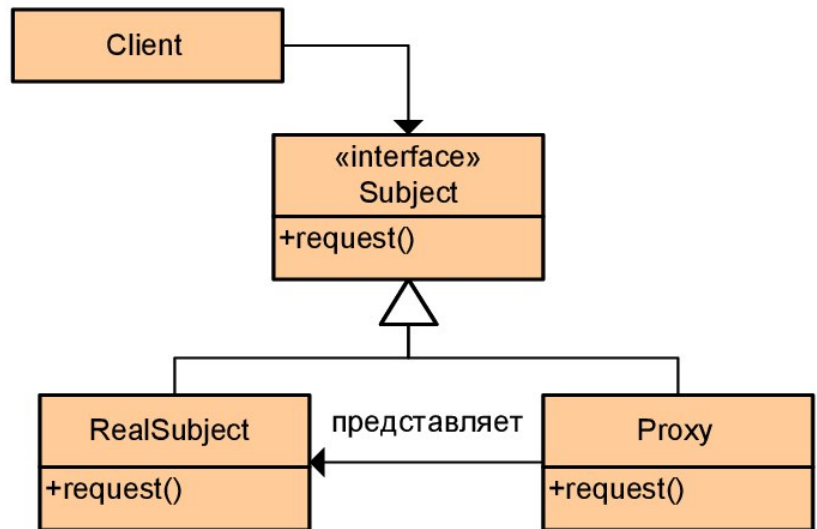
## Прокси

## Proxy

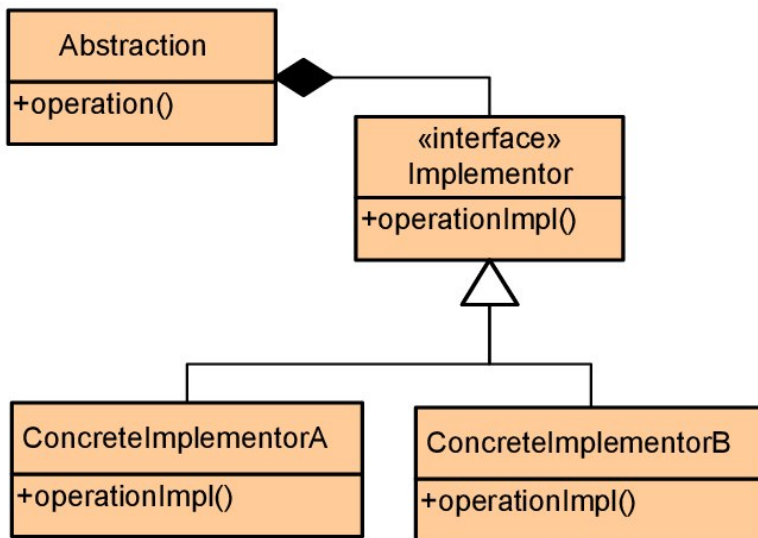
**Тип:** Структурный

### Что это:

Предоставляет замену другого объекта для контроля доступа к нему.



**Мост (bridge)**



# МОСТ

## Bridge

Тип: Структурный

**Что это:**

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

## Абстрактная фабрика (abstract factory)

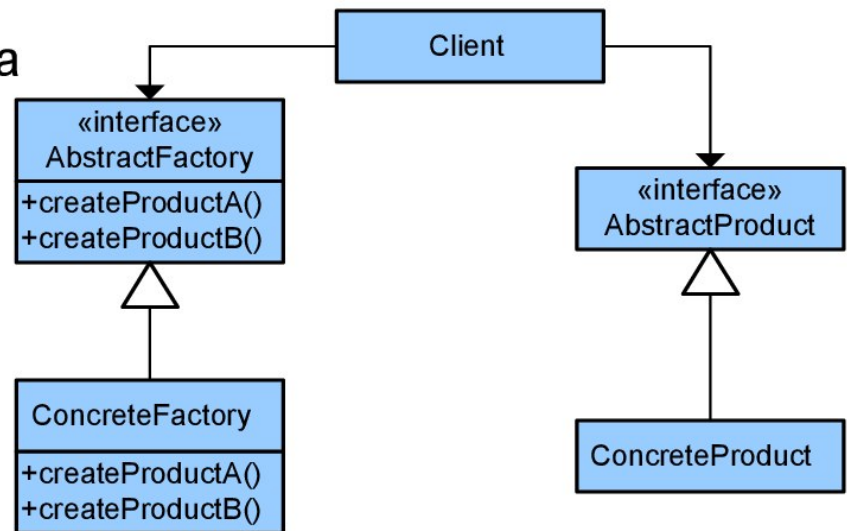
## Абстрактная фабрика

*Abstract factory*

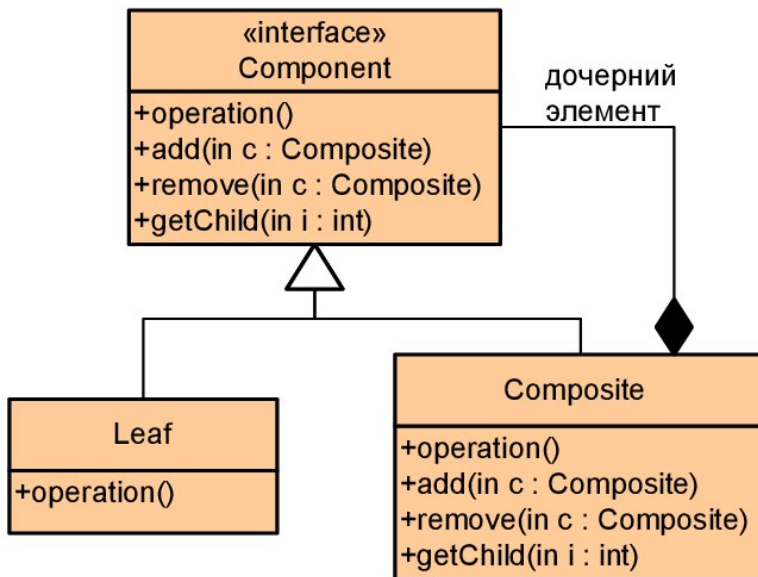
**Тип:** Порождающий

**Что это:**

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



## Компоновщик (composite)



## Компоновщик

*Composite*

**Тип:** Структурный

**Что это:**

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

## Строитель (builder)

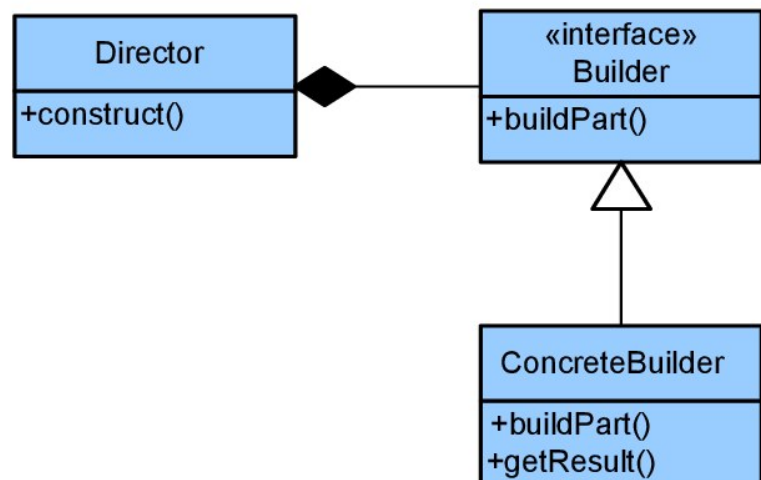
## Строитель

*Builder*

**Тип:** Порождающий

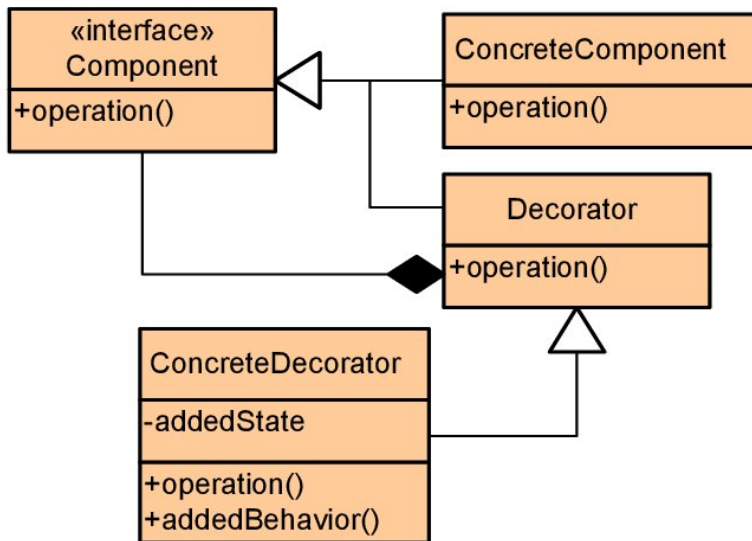
**Что это:**

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





## Декоратор (decorator)



## Декоратор *Decorator*

Тип: Структурный

Что это:

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

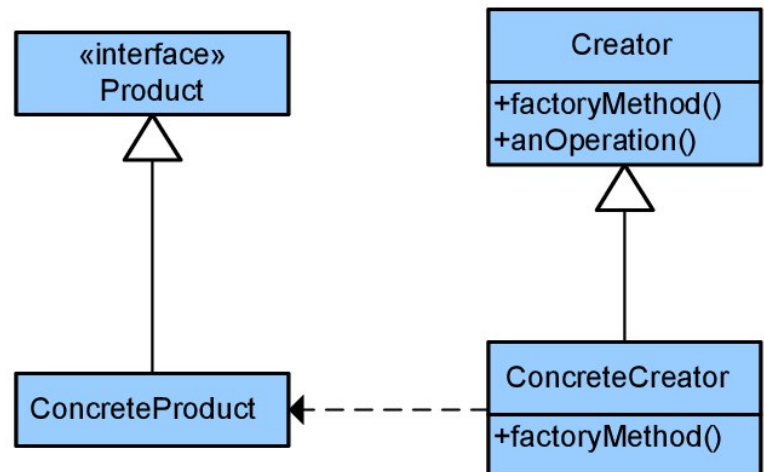
## Фабричный метод (factory method)

### Фабричный метод *Factory method*

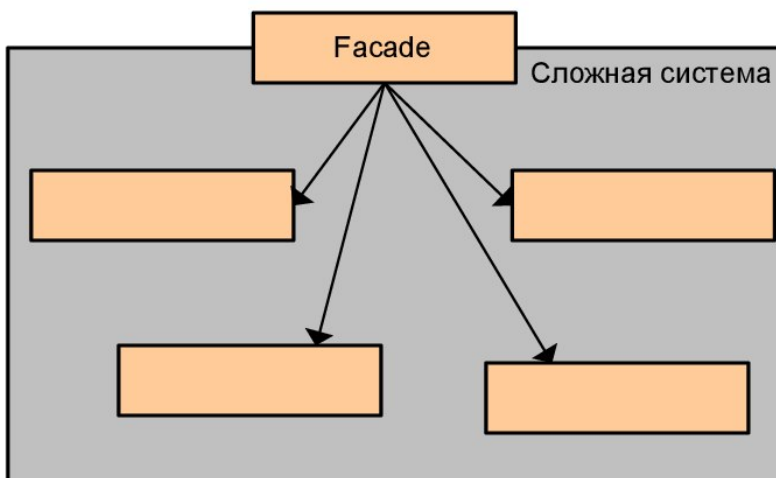
Тип: Порождающий

Что это:

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



## Фасад (facade)



## Фасад *Facade*

Тип: Структурный

Что это:

Предоставляет единый интерфейс к группе интерфейсов подсистемы. Определяет высокоуровневый интерфейс, делая подсистему проще для использования.

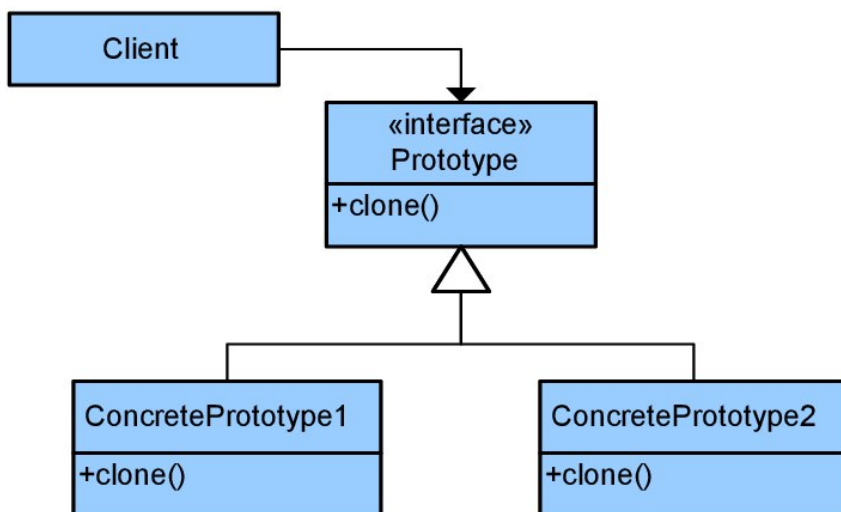
## Прототип (prototype)

## Прототип *Prototype*

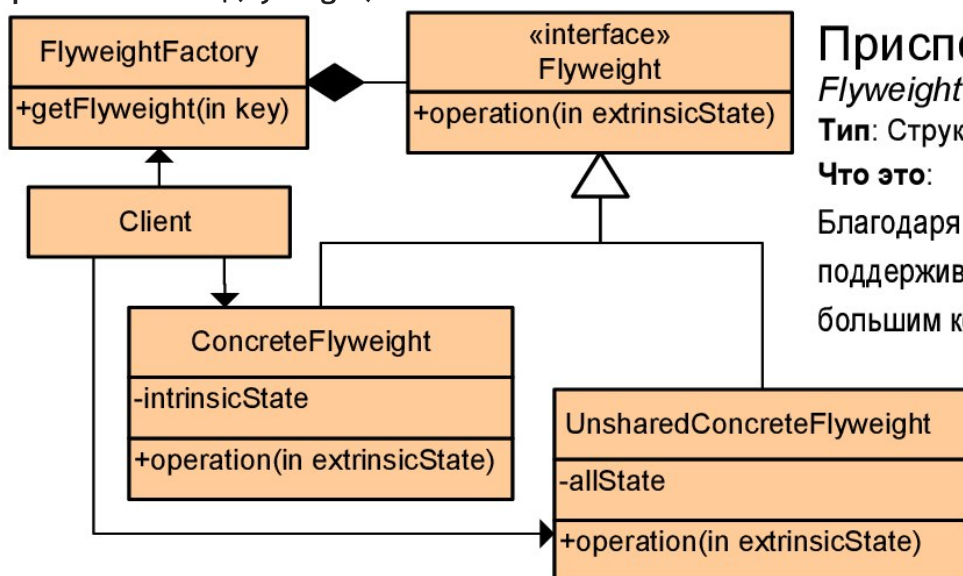
**Тип:** Порождающий

**Что это:**

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



## Приспособленец (flyweight)



## Приспособленец *Flyweight*

**Тип:** Структурный

**Что это:**

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

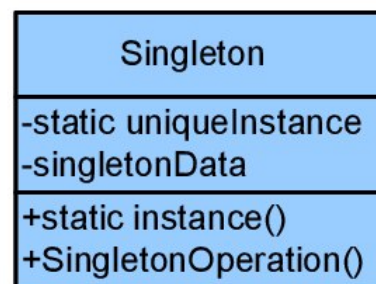
## Одиночка (singleton)

## Одиночка *Singleton*

**Тип:** Порождающий

**Что это:**

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



## Файлы

- все паттерны в pdf-файле.
- то же самое, но в png — 1 и 2 части.

- архив с нарезанными изображениями.

**Upd.** оригинальный pdf и изображения (1, 2).

**P.S.** По запросу «шаблоны проектирования» 636 топиков, а хаба нет; а по «bitcoin» — 278 топиков и хаб есть. Прошу восстановить справедливость!

## Опросы

Только зарегистрированные пользователи могут участвовать в опросе. Войдите, пожалуйста.

Стоит ли продолжать переводить шпаргалки?

95,8% Да 3979

4,2% Нет 173

Проголосовали 4152 пользователя. Воздержались 388 пользователей.

Нужен ли хаб для паттернов?

85,2% Нужен 3409

14,8% Не нужен 593

Проголосовали 4002 пользователя. Воздержались 569 пользователей.

**Теги:** паттерны, шаблоны, паттерны проектирования, шаблоны проектирования, gof, gang of four, банда четырёх, проектирование, архитектура, шпаргалка по паттернам

**Хабы:** Разработка веб-сайтов, Анализ и проектирование систем, Проектирование и рефакторинг

↑ +166 ↓ 3641 973k 64



75,7

Карма

0,0

Рейтинг

Влад @WarAngel\_alk

Пользователь

ПОХОЖИЕ ПУБЛИКАЦИИ

13 мая 2019 в 15:18

## Паттерны проектирования, используемые в Spring Framework

↑ +3    👁 13,3k    📖 81    💬 6

4 марта 2019 в 11:12

## Распределенные системы. Паттерны проектирования. Обзор книги

↑ +13    👁 16,4k    📖 109    💬 7

10 сентября 2018 в 15:53

## Паттерны проектирования в Kotlin

↑ +16    👁 17,5k    📖 113    💬 6

### ЗАКАЗЫ

#### Автоматическая обработка изображений

30 000 Р за проект • 2 отклика • 22 просмотра

#### UML проект (переделать готовый) Реферат

5 000 Р за проект • 0 откликов • 8 просмотров

#### Консультация по регистрации аккаунта разработчика Apple Developer

1 000 Р за проект • 1 отклик • 8 просмотров

#### Починить ботов

2 000 Р за проект • 2 отклика • 18 просмотров

#### Помощь в настройке 1С Битрикс

500 Р за проект • 3 отклика • 13 просмотров

Больше заказов на Хабр Фрилансе

## Комментарии 64



**IgorOleynick** 25 января 2014 в 21:28    🗨    📖

↑ +1 ↓

Отлично! Захотелось сразу даже блокнотик сделать со шпаргалками. Спасибо вам, очень полезно!



**MFomichev** 25 января 2014 в 21:39    🗨    📖

↑ +3 ↓

все читают теги



**semmaxim** 26 января 2014 в 10:56    🗨    📖    🔄    🔄

↑ +19 ↓

Прочитал только после этого комментария. А так никогда не читаю. :)



**zag2art** 26 января 2014 в 15:42    🗨    📖    🔄    🔄

↑ +4 ↓

А они и не нужны для того, чтобы их читали вместе со статьями. По ним надо искать, вводя в строку ввода "[тег]".



**Kain\_Haart** 26 января 2014 в 14:14



↑ +1 ↓

Спасибо пунктуации на новый тег «я верю» %)



**Yan169** 26 января 2014 в 15:44



↑ +8 ↓

*Сейчас будет крик души.*

Когда уже основная масса хабраавторов поймёт, что теги не для того, чтобы их читали, а для того, чтобы нужный материал было легче найти по заданному тегу как в общем поиске, так и в избранном. Поэтому нужно указывать четкие однозначные метки, а не писать сочинение в стиле «теги никто не читает, потому напишу сюда какую-нибудь лабуду».

Постоянно приходится дописывать свои теги к постам при добавлении в избранное, но проблему общего поиска по тегам это не решает.

Сейчас у меня в избранном полно информационного мусора в тегах: «никто не читает теги», «тег который никто не читает», «зачем я пишу такие подробные теги?» и пр. А по тегу «всё равно никто не читает теги» на хабре аж 43 топика.



**WarAngel\_alk** 26 января 2014 в 17:27



↑ +1 ↓

И наверняка в каждом из них есть комментарий о том, что кто-то их всё таки читает)

Но ладно, убедили, пора завязывать с этой традицией...



**ur001** 27 января 2014 в 03:35



↑ 0 ↓

А ещё, не секрет, по ним строится колонка «похожие публикации», которую многие смотрят



**vladimirkolyada** 25 января 2014 в 21:40



↑ +10 ↓

Попытки использовать какую-то ужасную нотацию, то ли куски UML, то ли сами что-то напридумывали. Где-то типы указаны, где-то нет, где то вообще поля какие-то берутся, а то вот, пустые прямоугольники. Не вижу шпаргалки тут, вижу набор субъективных представлений выраженных в плохих рисунках.



**moooV** 25 января 2014 в 21:48



↑ +14 ↓

Да и вообще, в виде схем воспринимается сложнее — привычнее такое видеть в виде кода на любом распространенном языке с Си-образным синтаксисом.



**bya** 26 января 2014 в 08:14



↑ -4 ↓

Согласен. Только лучше Python (код короче и нагляднее) и примеры на одной и той же задаче «почувствуйте разницу».  
[freehabr.ru/blog/designpatterns/2565.html](http://freehabr.ru/blog/designpatterns/2565.html)



**3ton** 26 января 2014 в 11:27



↑ +1 ↓

А это как?

Ведь каждый паттерн — это решение своей задачи, нельзя одним паттерном решить все задачи и нельзя всеми паттернами решить одну и ту же проблему...



**VoIch** 26 января 2014 в 12:37



↑ 0 ↓

Возможно имеется в виду задача как проект, в котором с помощью паттернов решаются те или иные архитектурные проблемы более удачным или, хотя бы, более понятным большинству, способом.



 **3ton** 26 января 2014 в 13:17 # 📌 🔗 ↻

↑ 0 ↓

В таком контексте поддержку просьбу.

Хотелось бы увидеть практическое применение в пределах одного проекта с объяснением бывалых почему для конкретного случая было выбрано применение определенного паттерна, потому как суть некоторых основываясь лишь на теории самому не очень ясна.

 **Yan169** 26 января 2014 в 15:32 # 📌 🔗 ↻

↑ 0 ↓

Так в оригинальной книге GoF как раз с примерами, как по отдельным паттернам, так и в пределах проекта «Текстовый редактор». Эта шпаргалка — выжимка из той книги.

 **warsoul** 29 января 2014 в 04:02 # 📌 🔗 ↻

↑ 0 ↓

Структурные паттерны на питоне? Ну-ну :)

Нет, я не спору, написать то можно что угодно, только вот показывать плохие примеры для языка это не правильно...

 **warsoul** 29 января 2014 в 04:09 # 📌 🔗 ↻

↑ 0 ↓

~~Структурные~~ Порождающие конечно же

 **bya** 29 января 2014 в 05:19 # 📌 🔗 ↻

↑ 0 ↓

У меня там [freehabr.ru/blog/designpatterns/2565.html](http://freehabr.ru/blog/designpatterns/2565.html) приведены следующие паттерны.


Behavioral: interpreter, iterator;

Creational: abstractfactory, builder, singleton;

Structural: facade, flyweight.


А так, примеры есть на все из «книги четырех» я их просто не выкладывал.

А Вы уж определитесь какие из «плохие примеры для языка». Плохо иметь единый пример, чтобы четко различать и чувствовать разницу в реализации, или надо было их делать не на Python, а на PHP, а лучше на Pascal/Delphi, которые Вы типа знаете, но испортили ими в молодости своей неокрепший разум.

 **warsoul** 29 января 2014 в 05:47 # 📌 🔗 ↻

↑ +1 ↓

После изучения Ваших примеров я просто признаю что нам с Вами нечего обсуждать.

 **Zuy** 25 января 2014 в 21:49 # 📌

↑ 0 ↓

в описании у каждого паттерна указан тип 'поведенческий'. А судя по списку в начале должны ещё быть порождающие и структурные.

 **WarAngel\_alk** 25 января 2014 в 22:10 # 📌 🔗 ↻

↑ 0 ↓

Упс... извиняюсь, глупая ошибка, сейчас исправлю.

 **x256** 25 января 2014 в 22:14 # 📌 🔗 ↻

↑ -4 ↓

Всё же очевидно, они по цвету различаются. Легенда цветов в начале. Я на неё посмотрел и дальше не читал подписи.

Ах, у кого-то проблемы с цветовосприятием? Так это их проблемы, и вообще их всего несколько процентов. Зачем ещё придумывать, как продублировать цветовую информацию?

В общем, это надо не в хаб «Паттерны», а в хаб «Анти-Accessibility». Точно будет популярным.



**Zuy** 25 января 2014 в 22:22 # 📖 🔗 ↻

↑ +1 ↓

Это точно ко мне вопросы? Для меня очевидно какой паттерн куда относится вообще без цветов и подписей. Да и без этой статьи тоже :-)

Но если автор делает шпаргалку по теме, то информация должна быть либо верной либо отсутствовать вообще. Там есть ещё файлы для скачки и печати, в которых тот же косяк. после их распечатки цветовая маркировка уже сильно не поможет



**WarAngel\_alk** 25 января 2014 в 22:36 # 📖 🔗 ↻

↑ +2 ↓

Обновил пост: поменял типы паттернов, обновил ссылки на файлы.



**Stas911** 25 января 2014 в 22:14 # 📖

↑ 0 ↓

А ссылку на исходник на английском добавьте плз



**WarAngel\_alk** 25 января 2014 в 22:27 # 📖 🔗 ↻

↑ 0 ↓

Эх, и это забыл... Добавил оригинальные файлы.



**WarAngel\_alk** 25 января 2014 в 22:27 # 📖

↑ +2 ↓

В общем, мораль понял: поспешишь — людей насмешишь :(



**Lol4t0** 25 января 2014 в 22:45 # 📖

↑ +9 ↓

А как их предлагается использовать?

Вот сидишь, значит, проектируешь новое приложение, перебираешь шпаргалки, какой паттерн куда вставить... Что-то мне кажется, что если так делать, то ничего дельного в итоге не получится.



**IgorOleynick** 25 января 2014 в 22:56 # 📖 🔗 ↻

↑ 0 ↓

Думаю, для изучения, закрепления и повторения материала пригодится.

↑36 **Error\_403\_Forbidden** 25 января 2014 в 23:55 # 📖 🔗 ↻

↑ 0 ↓

А вот в этом и заключается редкое и ценное умение использовать паттерны.



**wizardsd** 26 января 2014 в 00:11 # 📖 🔗 ↻

↑ 0 ↓

Программисту их надо просто знать, чтобы не изобретать велосипед и уметь применять там, где в этом есть смысл.

Начинающему программисту с ними надо хотя бы ознакомиться, чтобы узнать, что в его любимом языке программирования можно еще и вот такие штуки вытворять. Также начинающему программисту при ознакомлении с паттернами наставник должен сказать о том, что не стоит теперь всё писать используя только паттерны.



**Lol4t0** 26 января 2014 в 00:14 # 📖 🔗 ↻

↑ +3 ↓

Я, конечно, говорю не про то, что делать с паттернами, а что делать с такими шпаргалками по паттернам? Разобраться с шаблонами проектирования по ним явно не получится.




**alcsan** 26 января 2014 в 02:26 # 📖 🔗 ↻


↑ -1 ↓

Например, я шпаргалки по паттернам просматриваю перед собеседованиями. Книга банды четырех была прочитана и усвоена уже давно и я периодически начинаю путать названия паттернов, хотя и продолжаю помнить их смысл. Вот в такие моменты и пригодится шпаргалка.


Еще неплохо в ситуации, когда тебе коллега говорит «Вкратце, то, что я здесь реализовал похоже на паттерн N», а ты подзабыл который из паттернов называется N, но, быстро взглянув на шпаргалку, вспомнил.

 **IDVsbruck** 26 января 2014 в 00:16    0 

На английском есть очень похожая реализация (Refcardz на DZone) — [http://cdn.dzone.com/sites/all/files/refcardz/rc008-designpatterns\\_online.pdf](http://cdn.dzone.com/sites/all/files/refcardz/rc008-designpatterns_online.pdf) — подготовленная для печати версия.

 **Blumfontein**  26 января 2014 в 00:30    +3 

Честно говоря, вообще никак не воспринимаю картинки-схемы, для меня это пустое место. Вот код — другое дело.

 **xakpc** 26 января 2014 в 00:42    +1 

Я такую своим ребятам раздал, пользы от нее никакой правда )

Найти бы подобную по TDD: 5 правил TDD (типа NO PRODUCTION CODE BEFORE RED TEST), паттерны и операции рефакторинга применяемые при этом и т.п.


Т.е. 3 часть «Экстремального программирования» Кента Бека но на одну страничку :)

 **warsoul** 26 января 2014 в 03:35     0 

[orm-chimera-prod.s3.amazonaws.com/1234000000754/images/tdd\\_flowchart\\_functional\\_and\\_unit.png](http://orm-chimera-prod.s3.amazonaws.com/1234000000754/images/tdd_flowchart_functional_and_unit.png) как вариант...


 **Fally** 26 января 2014 в 01:20    -2 

вместо целого поста автор мог бы просто указать ссылку на эту книгу


 **KlonD90** 26 января 2014 в 01:48     +2 

Тоже как-то не понял в чем смысл поста. Схемки? Так особого плюса от них нет. Они ничего фактически не объясняют. Кажется очередным популистским постом, которых сейчас в интернете много из серии 10 штук, чтобы ваши штуки выглядели штуками.

НЛО прилетело и опубликовало эту надпись здесь

 **xakpc** 26 января 2014 в 02:22     +1 


Мне кажется это бесполезным. Ну будет у вас сниппет, например, на фабричный метод? Его цена равна нулю если не знать когда и зачем его применять. Это же самое главное.

 **VolCh** 26 января 2014 в 12:35     +1 

Если постоянно их проглядывать, то быстро научишься узнавать признаки паттернов в своём коде и подгонять их под стандарт.

 **IgorOleynick** 26 января 2014 в 11:50     0 

После прочтения фразы про «цветовую дифференциацию штанов» (конечно же, из многих любимого фильма «Кин-дза-дза») вспомнил лекции по мат. логике, где были задачки с этой фразой... Ностальгия.


 **Gorthauer87** 26 января 2014 в 13:24 # 📌 🔒

↑ +1 ↓

Как это в плюсах нет функций высших порядков? Есть `std::function`, есть `std::bind`, есть `std::mem_fn`, есть лямбды, есть функциональные объекты. По коду будет длиннее, чем в JS, но работать код будет заметно быстрее, особенно если функцию можно заинлайнить.

А вообще для меня эта шпаргалка скорее нужна чтобы давать более менее устоявшиеся названия сущностям в коде чтобы другие понимали, а сама по себе она не учит ничему.

Для того, чтобы научиться видеть и применять шаблоны нужен цикл задач на применения того или иного шаблона проектирования, есть ли в интернете именно задачки причем желательно с какой-нибудь возможностью тестирования эффективности полученного решения?

 **VolCh** 26 января 2014 в 14:16 # 📌 🔒

↑ 0 ↓

Как это в плюсах нет функций высших порядков? Есть `std::function`, есть `std::bind`, есть `std::mem_fn`, есть лямбды, есть функциональные объекты.

Да функция, принимающая в качестве аргумента указатель на другую функцию (или возвращающая его), уже есть функция высшего порядка. Это даже в простом Си работает.

 **Gorthauer87** 26 января 2014 в 16:54 # 📌 🔒

↑ 0 ↓

Подозреваю, что имелось в виду всё-таки `first class citizen`, таким свойством в C++ только некоторые функции обладают.

НЛО прилетело и опубликовало эту надпись здесь

 **Gorthauer87** 27 января 2014 в 14:57 # 📌 🔒

↑ +1 ↓

Лямбды страшноватые из-за статической типизации, да и не слишком это страшнее, чем

```
var result = foo(a, function(a, b, c) {...});
```

Хотя конечно это не функциональные языки и ленивые вычисления не поддерживаются а из-за отсутствия сборки мусора за временем жизни захваченных переменных нужно самому следить, но зато работают они очень быстро.

Вот раз сборника задач и нет, то, как мне кажется, он бы имел весьма большой успех. Хорошо бы их сделать вместе с тестировщиком, но я ума не приложу как тестировать архитектуру кода, это не входные и выходные данные через `stdio` подсовывать и забирать.

 **ilammy** 26 января 2014 в 14:42 # 📌 🔒

↑ 0 ↓

Выберите короче свой любимый язык и постарайтесь сделать шпаргалку именно по паттернам для него.

Естественно, ведь по сути фраза «я использовал паттерн X» — это «выразительных возможностей используемого мной языка не хватает, чтобы записать X очевидным образом». Даже сами GoF говорят о том, что их книга с паттернами направлена на языки с выразительностью уровня Java. Возможности языков различаются, так что паттерны не имеют смысла без привязки к языку. Где-то есть поддержка функций высшего порядка — там нет Strategy, там просто создаётся и передаётся функция. Где-то сигналы-слоты заменяют велосипедные Observers. Где-то есть поддержка мультиметодов — и там не нужен Visitor. Где-то есть встроенные синглтоны. Где-то классы являются объектами и Factory сводится к простому вызову функции `make`.

 **metahuman** 26 января 2014 в 04:48 # 📌

↑ +3 ↓

Такие вещи полезны, что бы вставлять их в слайды лекций.

 **CyberLight** 26 января 2014 в 08:15 # 📌

↑ 0 ↓

Вот к примеру, очень хорошо рассматриваются паттерны на [dofactory.com](http://dofactory.com) в разделе ".NET Design Patterns". Там есть и простые и из реального мира примеры на каждый паттерн :). Я пользуюсь этим ресурсом.



**kostyl** 26 января 2014 в 11:40 #

↑ +1 ↓

Вот еще шпаргалка, только без картинок [itdumka.com.ua/index.php?cmd=shownode&node=11](http://itdumka.com.ua/index.php?cmd=shownode&node=11)



**Asen** 26 января 2014 в 18:14 #

↑ 0 ↓

Было бы также не менее интересно посмотреть частотность использования тех или иных паттернов программистами. Ведь часто бывает так, что программист знаком с паттерном, но решить проблему он пытается «в лоб», т.е. не в обход с помощью паттернов.



**1111paha1111** 27 января 2014 в 12:58 #

↑ 0 ↓

В статье можно бы добавить ясности, если придерживаться группировки шаблонов как в книге. Так будет проще и быстрее найти то что нужно. В идеале, и той же последовательности паттернов лучше придерживаться. В книге они группировались на 3 раздела Creational, Structural и Behavioral patterns. Здесь же идут в перемешку... например Visitor -> Adapter -> Proxy...



**alexanderVmironenko** 27 января 2014 в 17:43 #

↑ 0 ↓

Спасибо!



**pelment** 28 января 2014 в 12:43 #

↑ 0 ↓

Шпаргалки — это прекрасно. Они структурируют все, что лежит у тебя в голове.



**ElForastero** 22 марта 2014 в 15:02 #

↑ 0 ↓

Большое спасибо!

Пожалуйста, если у Вас будет возможность, продолжайте!



**alexBondar** 22 марта 2014 в 22:48 #

↑ 0 ↓

Очень структурно. информативно и полезно!  
Thanks a million!



**KvanTTT** 9 июля 2014 в 08:18 #

↑ 0 ↓

Получается, что агрегация вообще нигде не используется, но зачем-то описана.

НЛО прилетело и опубликовало эту надпись здесь



**dim2r** 18 октября 2017 в 10:06 #

↑ 0 ↓

Жаль, что шаблоны не обозначены в языке. То есть пока комментарий не напишешь /\* это шаблон мост \*/, остается догадываться что там за шаблон.  
Было бы что-то типа «implements template Bridge» — было бы понятнее



**maydjin** 19 октября 2017 в 13:20 #

↑ 0 ↓

Тогда уж pattern а не template.



**VolCh** 20 октября 2017 в 19:59 #

↑ 0 ↓



Шаблоны как раз применяются для преодоления недостатков языка. Скажем, в языках с сильно развитой функциональной парадигмой нет нужды во многих ООП-шаблонах. А там где их используют обычно просто добавляют название шаблона к имени: *Bridge*, *Factory*, *Adapter*, (прости господи) *Singleton*



dim2r 21 октября 2017 в 00:37



0



У меня обычно с ними ассоциируется цитата Толстого «гладко было на бумаге, да забыли про овраги». :):)

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки

Неделя

Месяц

Cyberpunk 2077 игнорирует многопоточность микропроцессора AMD

↑ +16

👁 27,5k

🔖 10

💬 48

В пивной банке сделали фотографию с рекордной экспозицией — 8 лет и 1 месяц

↑ +34

👁 23,5k

🔖 11

💬 36

Как мы случайно сожгли \$72 000 за два часа в Google Cloud Platform и чуть не обанкротились

↑ +55

👁 49,4k

🔖 83

💬 154

Насколько вкусные печеньки в Яндексе?

↑ +80

👁 20,1k

🔖 48

💬 105

Как мы пофиксили Apple security-приложением

Мегапост

Ваш аккаунт

Войти

Регистрация

Разделы

Публикации

Новости

Хабы

Компании

Пользователи

Песочница

Информация

Устройство сайта

Для авторов

Для компаний

Документы

Соглашение

Конфиденциальность

Услуги

Реклама

Тарифы

Контент

Семинары

Мегaproекты

Мерч

