



Паттерны проектирования

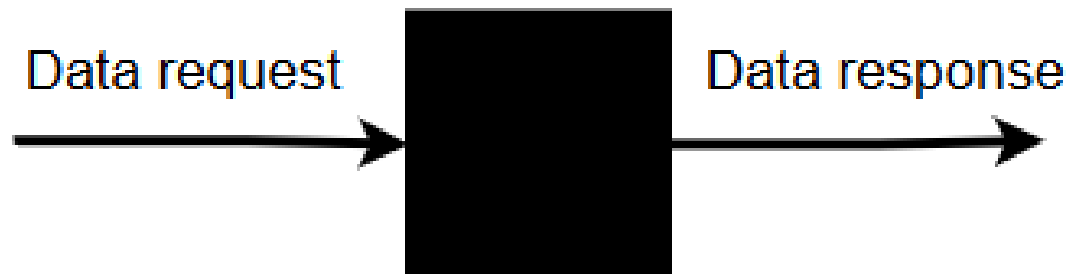
ПОНЯТИЕ "ПРОЕКТИРОВАНИЕ"

Определение понятия "проектирование"

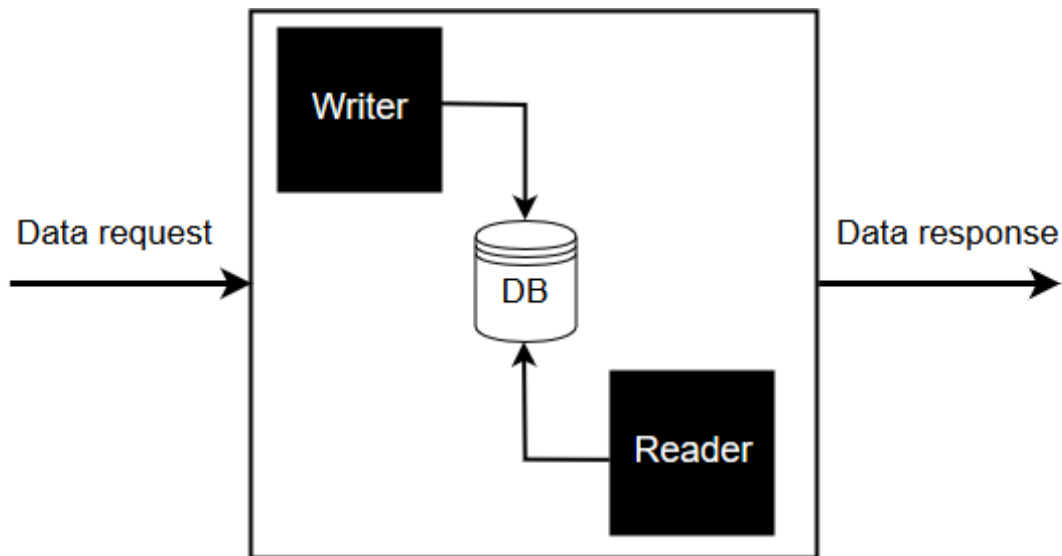
ЗАДАЧА

Спроектировать приложение, в котором возможно централизованное получение информации о текущих скидках на товары в магазинах города.

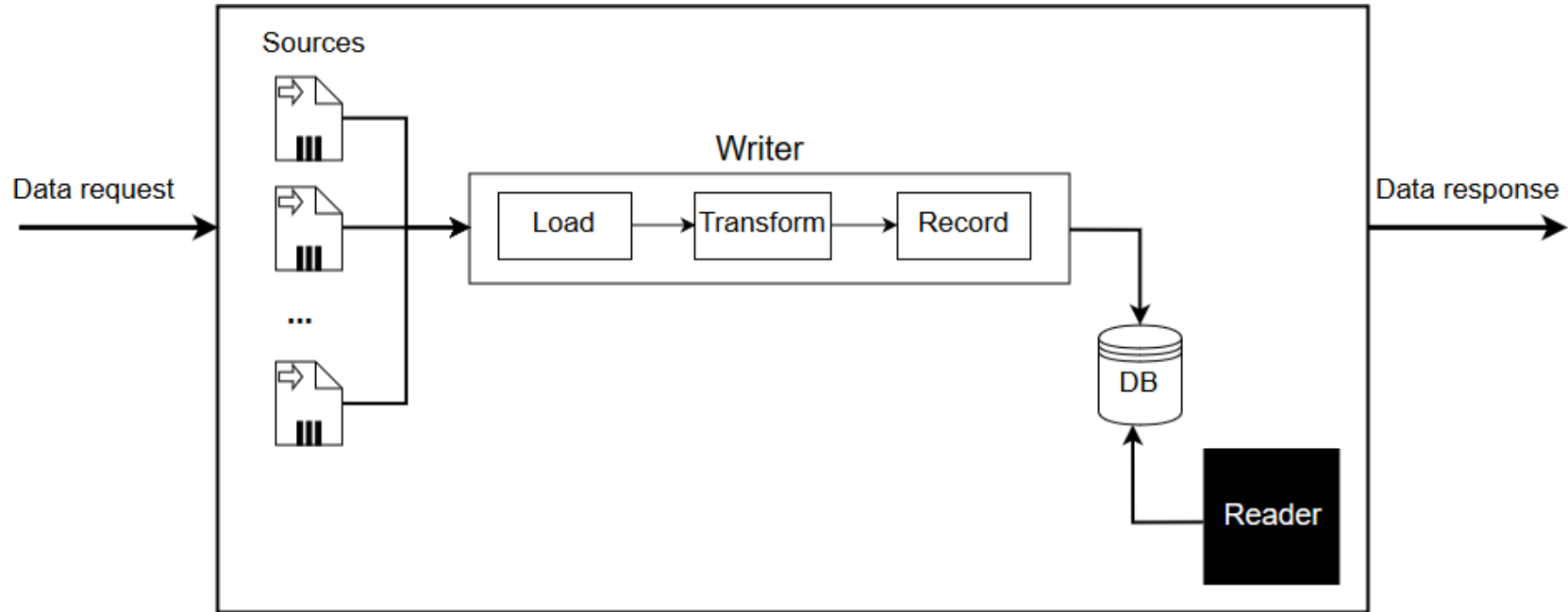
Определение понятия "проектирование"



Определение понятия "проектирование"



Определение понятия "проектирование"



Определение понятия "проектирование"

Проектирование - это процесс создания структуры приложения, определение внутренних и внешних свойств системы.

ПОНЯТИЕ "ПАТТЕРН"

Определение понятия "паттерн"



ФИЛЬМЫ УЖАСОВ

Мы пересмотрели тысячу фильмов ужасов и заметили, что везде, по сути, один и тот же сюжет: пятеро едут отдыхать в замечательное место, где от трёх до пяти из них умирают от какой-то неведомой силы с мотивом убийств ради убийств.

Определение понятия "паттерн"



МЕБЕЛЬНАЯ ФАБРИКА

Мы работаем на мебельной фабрике по производству столов. Каждый стол, который мы делаем, состоит из четырех ножек и столешницы.

Определение понятия "паттерн"

САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

КАФЕДРА ЭКСТРЕМАЛЬНОЙ
ИНФОРМАТИКИ

Лабораторная работа № _
Тема: _____

Выполнил: _____ (подпись)
Проверил: _____ (подпись)

ТИТУЛЬНЫЙ ЛИСТ

Мы пишем отчет по лабораторной работе и нам нужен титульный лист. Чтобы каждый раз не создавать разметку для наименования учебного учреждения, надписи “Лабораторная работа”, темы работы, прочерков для подписи студента и преподавателя, мы создаём шаблон, который можно взять и вписать только информацию.

Определение понятия "паттерн"

Паттерн -- это шаблон, типовое решение часто встречающейся задачи

Определение понятия "паттерн"

Паттерны проектирования -- часто встречающиеся типовые решения одной и той же проблемы в проектировании программного обеспечения.

ИСТОРИЯ


КРИСТОФЕР ВОЛЬФГАНГ АЛЕКСАНДЕР

архитектор и дизайнер, создатель более 200 архитектурных проектов

Впервые описал концепцию паттернов проектирования в книге «Язык шаблонов. Города. Здания. Строительство» в 1977 году.

В дальнейшем эта концепция получила развитие от известной четверке авторов в "Книге от банды четырёх". В этой книге они описали 23 паттерна для решения различных задач ООП-программирования.





Каждое типовое решение описывает некую повторяющуюся проблему и ключ к ее разгадке, причем таким образом, что вы можете пользоваться этим ключом многократно, ни разу не придя к одному и тому же результату.

Кристофер Александр



ИЗ ЧЕГО СОСТОИТ ПАТТЕРН?

Паттерн состоит из:



ПРОБЛЕМА, КОТОРУЮ РЕШАЕТ ПАТТЕРН



МОТИВАЦИЯ К РЕШЕНИЮ ПРОБЛЕМЫ СПОСОБОМ, КОТОРЫЙ ПРЕДЛАГАЕТ ПАТТЕРН



СТРУКТУРЫ КЛАССОВ, СОСТАВЛЯЮЩИХ РЕШЕНИЕ



ПРИМЕР НА PYTHON

КЛАССИФИКАЦИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ

Классификация паттернов проектирования

ПОРОЖДАЮЩИЕ

Решают проблемы гибкого создания объектов без внесения в программу лишних зависимостей

- Фабричный метод
- Абстрактная фабрика
- Строитель
- Прототип
- Одиночка

Классификация паттернов проектирования

СТРУКТУРНЫЕ

Показывают различные способы построения связей между объектами.

- Адаптер
- Мост
- Компоновщик
- Декоратор
- Фасад
- Легковес
- Заместитель

Классификация паттернов проектирования

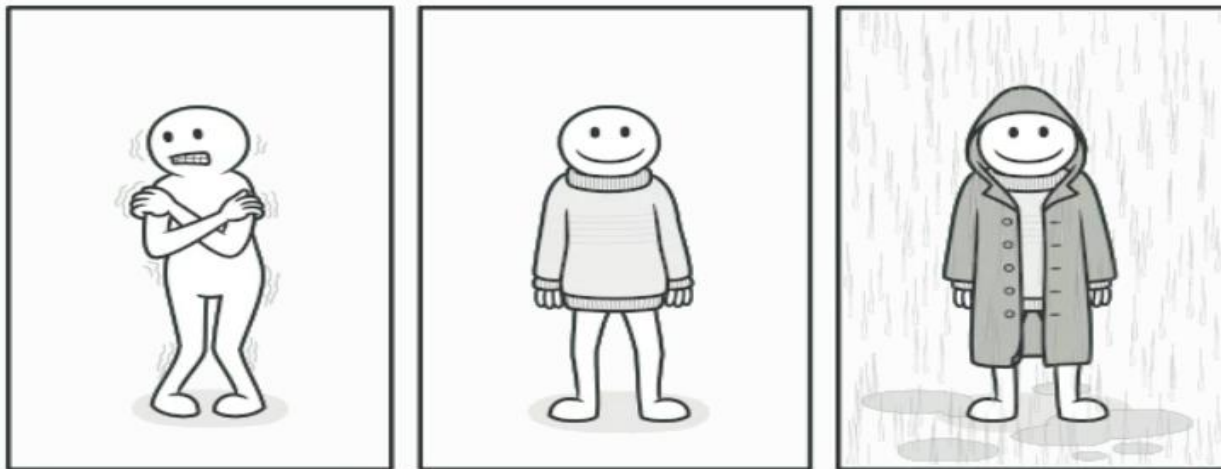
ПОВЕДЕНЧЕСКИЕ

Заботятся об эффективной коммуникации между объектами.

- Цепочка обязанностей
- Команда
- Итератор
- Посредник
- Снимок
- Наблюдатель
- Состояние
- Стратегия
- Шаблонный метод
- Посетитель

ДЕКОРАТОР

Паттерны проектирования



ДЕКОРАТОР

Декоратор - это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, добавляя им полезные «обёртки».

Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.

Паттерны проектирования: Декоратор

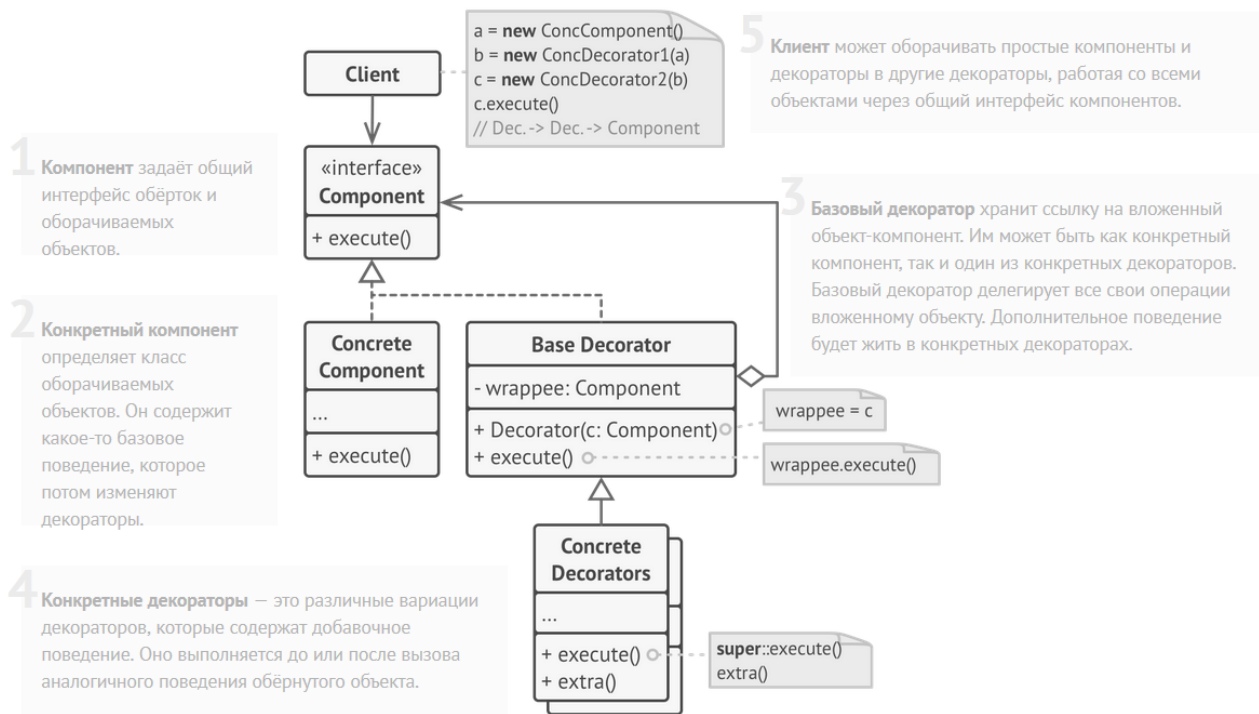
ПРОБЛЕМА

Вы хотите добавить новые обязанности в поведении или состоянии отдельных объектов во время выполнения программы. Использование наследования не представляется возможным, поскольку это решение статическое и распространяется целиком на весь класс.

МОТИВАЦИЯ ИСПОЛЬЗОВАТЬ ПАТТЕРН

Паттерн “Декоратор” позволяет добавлять к объектам дополнительную функциональность, которая будет выполняться до, после или даже вместо основной функциональности объекта. При этом нет необходимости вносить изменения в структурах, которые стоят над объектом.

Паттерны проектирования: Декоратор



АДАПТЕР

Паттерны проектирования

European Wall Outlet



AC Power Adapter



Standard AC Plug



АДАПТЕР

Адаптер — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.

Паттерны проектирования: Адаптер

ПРОБЛЕМА

Система поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс.

МОТИВАЦИЯ ИСПОЛЬЗОВАТЬ ПАТТЕРН

- Вам нужно отделить и скрыть от клиента подробности преобразования различных интерфейсов;
- Используя адаптер в структурах с множеством подклассов можно избежать создания нового уровня подклассов и дублирования кода;
- В случае изменения интерфейса и отсутствия времени, адаптер является быстрым решением;
- Адаптер не вносит изменений в существующий код, следовательно мы не сможем создать ошибок в старом коде.

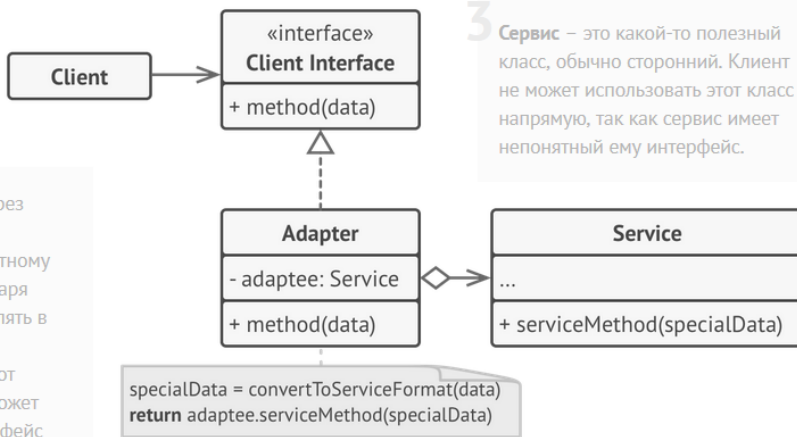
Паттерны проектирования: Адаптер

1 Клиент — это класс, который содержит существующую бизнес-логику программы.

2 Клиентский интерфейс описывает протокол, через который клиент может работать с другими классами.

3 Сервис — это какой-то полезный класс, обычно сторонний. Клиент не может использовать этот класс напрямую, так как сервис имеет непонятный ему интерфейс.

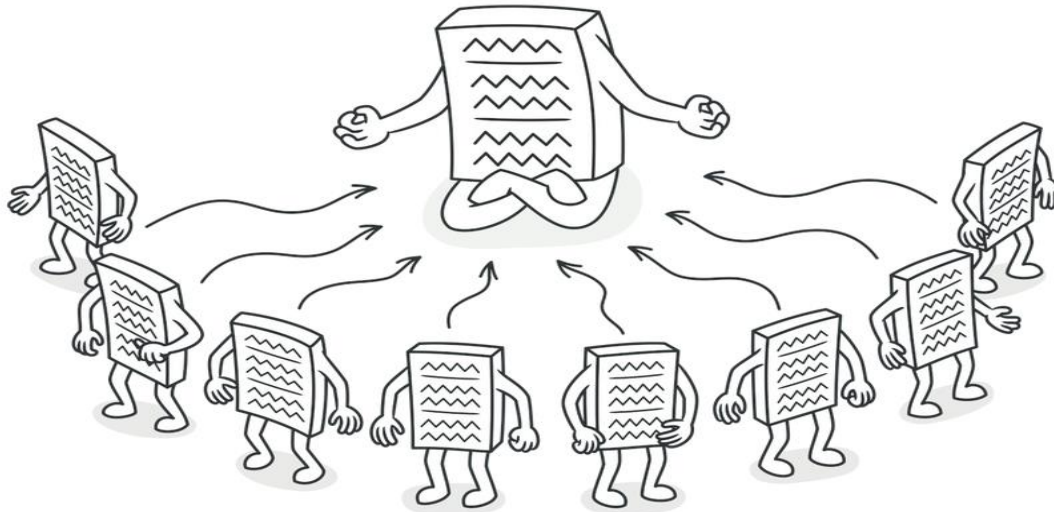
5 Работая с адаптером через интерфейс, клиент не привязывается к конкретному классу адаптера. Благодаря этому, вы можете добавлять в программу новые виды адаптеров, независимо от клиентского кода. Это может пригодиться, если интерфейс сервиса вдруг изменится, например, после выхода новой версии сторонней библиотеки.



4 Адаптер — это класс, который может одновременно работать и с клиентом, и с сервисом. Он реализует клиентский интерфейс и содержит ссылку на объект сервиса. Адаптер получает вызовы от клиента через методы клиентского интерфейса, а затем переводит их в вызовы методов обернутого объекта в правильном формате.

ОДИНОЧКА

Паттерны проектирования



ОДИНОЧКА

Одиночка — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Паттерны проектирования: Одиночка

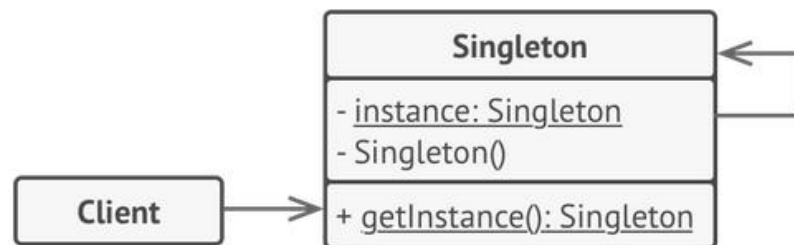
ПРОБЛЕМА

- Вам нужна гарантия того, что экземпляр класса существует в единственном экземпляре. Чаще всего это полезно для доступа к какому-то общему ресурсу, например, базе данных.
- Вам нужна глобальная точка доступа к определенному функционалу, который, возможно, хотелось бы хранить в отдельном месте.

МОТИВАЦИЯ ИСПОЛЬЗОВАТЬ ПАТТЕРН

Одиночка гарантирует, что никакой другой код не заменит созданный экземпляр класса, поэтому вы всегда уверены в наличии лишь одного

Паттерны проектирования: Одиночка



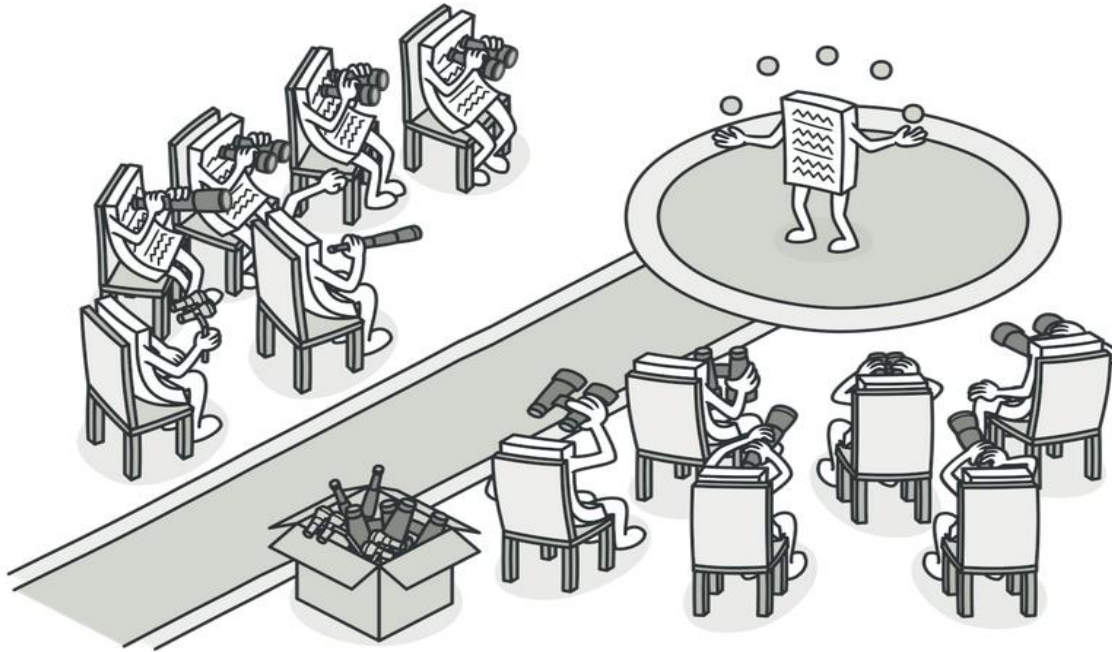
1 **Одиночка** определяет статический метод `getInstance`, который возвращает единственный экземпляр своего класса.

Конструктор одиночки должен быть скрыт от клиентов. Вызов метода `getInstance` должен стать единственным способом получить объект этого класса.

```
if (instance == null) {
    // Внимание, если вы пишете
    // многопоточный код, то здесь
    // нужно синхронизировать потоки.
    instance = new Singleton()
}
return instance
```

НАБЛЮДАТЕЛЬ

Паттерны проектирования



НАБЛЮДАТЕЛЬ

Наблюдатель — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

Паттерны проектирования: Наблюдатель

ПРОБЛЕМА

Вам нужно реализовать механизм наблюдения одного объекта за другими, где:

- После изменения состояния одного объекта требуется что-то сделать в других, но вы не знаете наперёд, какие именно объекты должны отреагировать.
- Одни объекты должны наблюдать за другими, но только в определённых случаях.

МОТИВАЦИЯ ИСПОЛЬЗОВАТЬ ПАТТЕРН

Мотивация использования шаблона "Наблюдатель" заключается в том, что нам нужно поддерживать согласованность лишь между связанными объектами, не делая классы зависимыми друг от друга. Например в случаях, когда объект должен иметь возможность уведомлять другие объекты, не делая никаких предположений, что это за объекты.

Паттерны проектирования: Наблюдатель

1 **Издатель** владеет внутренним состоянием, изменение которого интересно отслеживать подписчикам. Издатель содержит механизм подписки: список подписчиков и методы подписки/отписки.

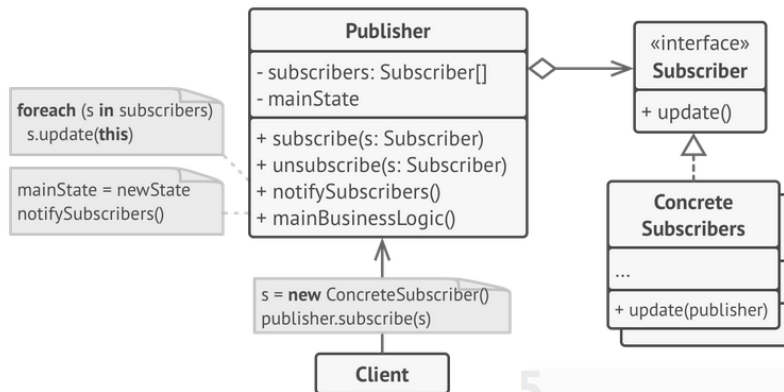
2 Когда внутреннее состояние издателя меняется, он оповещает своих подписчиков. Для этого издатель проходит по списку подписчиков и вызывает их метод оповещения, заданный в общем интерфейсе подписчиков.

3 **Подписчик** определяет интерфейс, которым пользуется издатель для отправки оповещения. В большинстве случаев для этого достаточно единственного метода.

4 **Конкретные подписчики** выполняют что-то в ответ на оповещение, пришедшее от издателя. Эти классы должны следовать общему интерфейсу подписчиков, чтобы издатель не зависел от конкретных классов подписчиков.

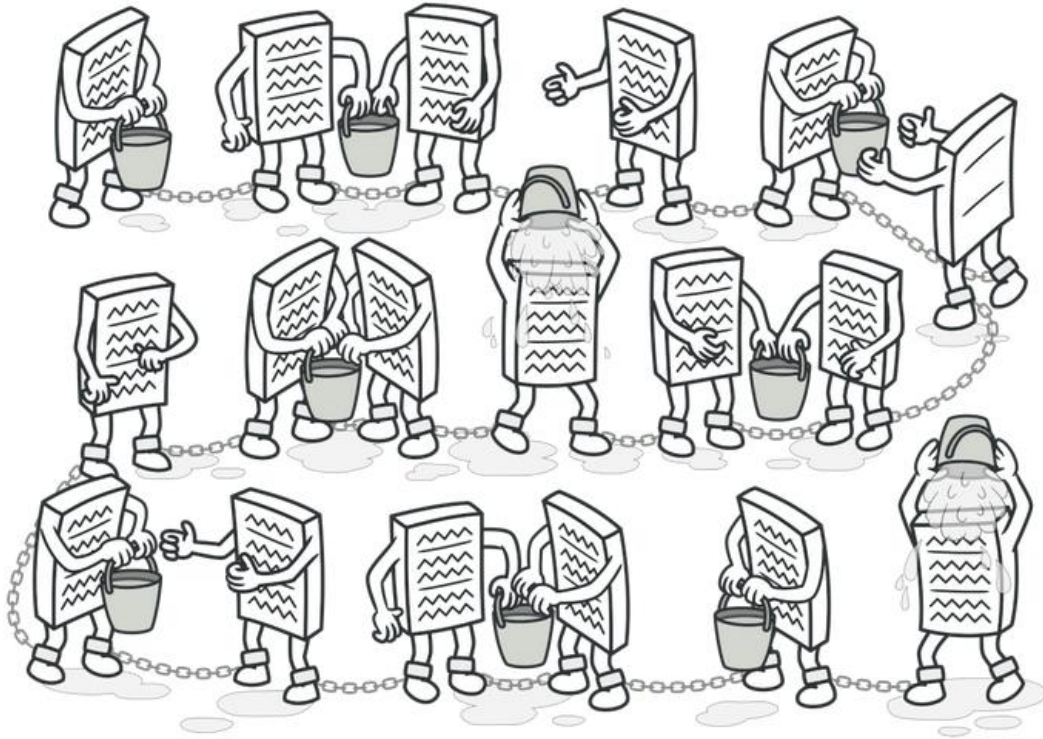
5 По приходу оповещения подписчику нужно получить обновлённое состояние издателя. Издатель может передать это состояние через параметры метода оповещения. Более гибкий вариант — передавать через параметры весь объект издателя, чтобы подписчик мог сам получить требуемые данные. Как вариант, подписчик может постоянно хранить ссылку на объект издателя, переданный ему в конструкторе.

6 **Клиент** создаёт объекты издателей и подписчиков, а затем регистрирует подписчиков на обновления в издателях.



ЦЕПОЧКА ОБЯЗАННОСТЕЙ

Паттерны проектирования



ЦЕПОЧКА ОБЯЗАННОСТЕЙ

Цепочка обязанностей — это поведенческий паттерн проектирования, который позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.

Паттерны проектирования: Цепочка обязанностей

ПРОБЛЕМА

- в разрабатываемой системе имеется группа объектов, которые могут обрабатывать сообщения определенного типа;
- все сообщения должны быть обработаны хотя бы одним объектом системы;
- сообщения в системе обрабатываются по схеме «обработай сам либо перешли другому», то есть одни сообщения обрабатываются на том уровне, где они получены, а другие пересылаются объектам иного уровня.

МОТИВАЦИЯ ИСПОЛЬЗОВАТЬ ПАТТЕРН

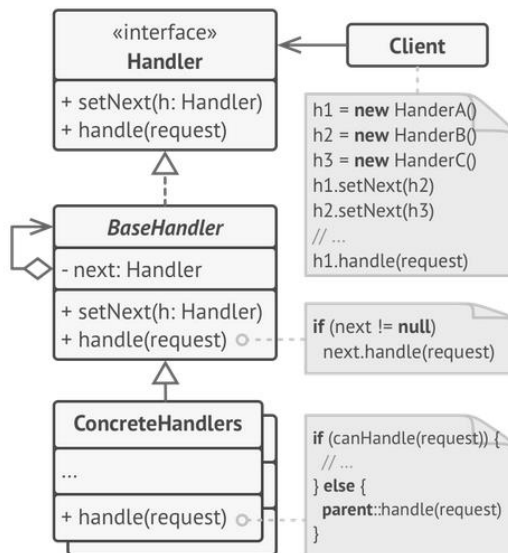
- Ваша программа должна обрабатывать разнообразные запросы несколькими способами, но заранее неизвестно, какие конкретно запросы будут приходить и какие обработчики для них понадобятся.
- Вам важно, чтобы обработчики выполнялись один за другим в строгом порядке.

Паттерны проектирования: Цепочка обязанностей

1 **Обработчик** определяет общий для всех конкретных обработчиков интерфейс. Обычно достаточно описать единственный метод обработки запросов, но иногда здесь может быть объявлен и метод выставления следующего обработчика.

2 **Базовый обработчик** — опциональный класс, который позволяет избавиться от дублирования одного и того же кода во всех конкретных обработчиках.

Обычно этот класс имеет поле для хранения ссылки на следующий обработчик в цепочке. Клиент связывает обработчики в цепь, подавая ссылку на следующий обработчик через конструктор или сеттер поля. Также здесь можно реализовать базовый метод обработки, который бы просто перенаправлял запрос следующему обработчику, проверив его наличие.



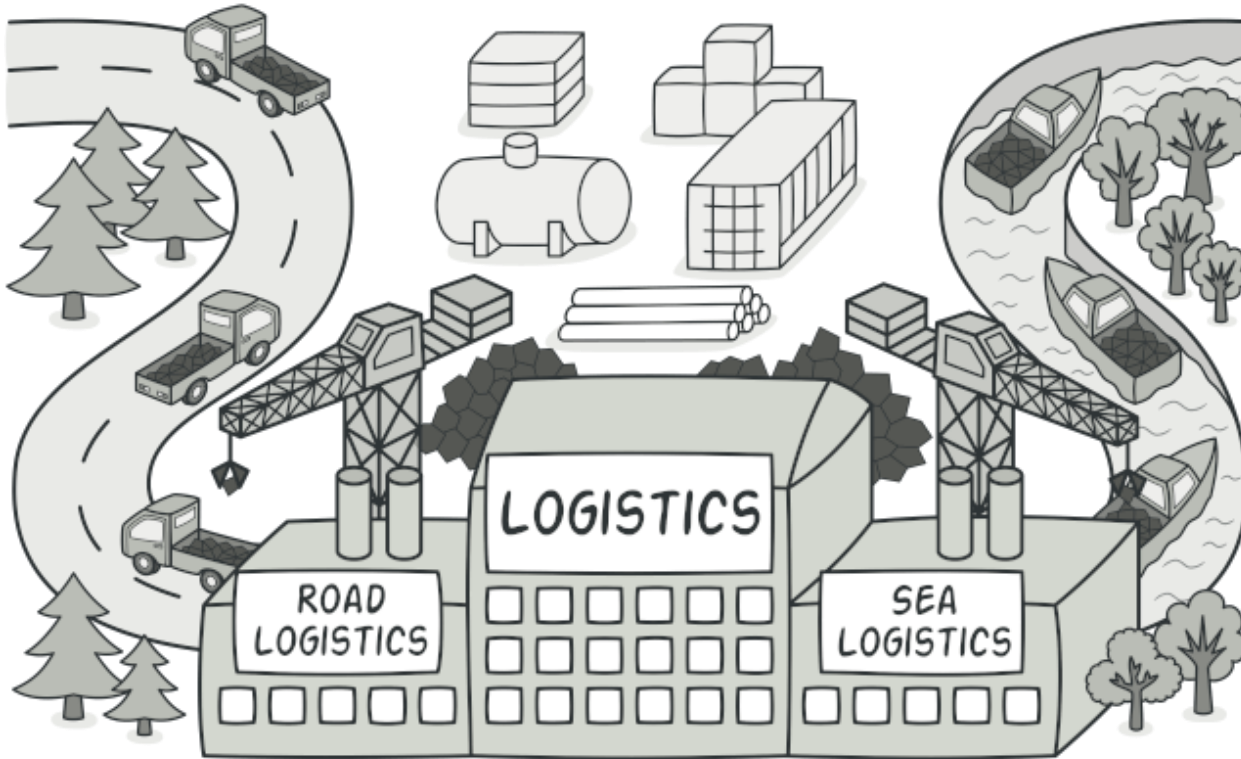
4 **Клиент** может либо сформировать цепочку обработчиков единожды, либо перестраивать её динамически, в зависимости от логики программы. Клиент может отправлять запросы любому из объектов цепочки, не обязательно первому из них.

3 **Конкретные обработчики** содержат код обработки запросов. При получении запроса каждый обработчик решает, может ли он обработать запрос, а также стоит ли передать его следующему объекту.

В большинстве случаев обработчики могут работать сами по себе и быть неизменяемыми, получив все нужные детали через параметры конструктора.

ФАБРИЧНЫЙ МЕТОД

Паттерны проектирования



ФАБРИЧНЫЙ МЕТОД

Фабричный метод — это порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Паттерны проектирования: Фабричный метод

ПРОБЛЕМА

Необходимо создать объекты разных классов с похожим функционалом для разного контекста; требуется гибкость при создании продуктов

МОТИВАЦИЯ ИСПОЛЬЗОВАТЬ ПАТТЕРН

- Когда заранее неизвестны типы и зависимости объектов, с которыми должен работать ваш код
- Фабричный метод отделяет код производства продуктов от остального кода, который эти продукты использует
- Когда вы хотите дать возможность пользователям расширять части вашего фреймворка или библиотеки
- Когда вы хотите экономить системные ресурсы, повторно используя уже созданные объекты, вместо порождения новых

Паттерны проектирования: Фабричный метод

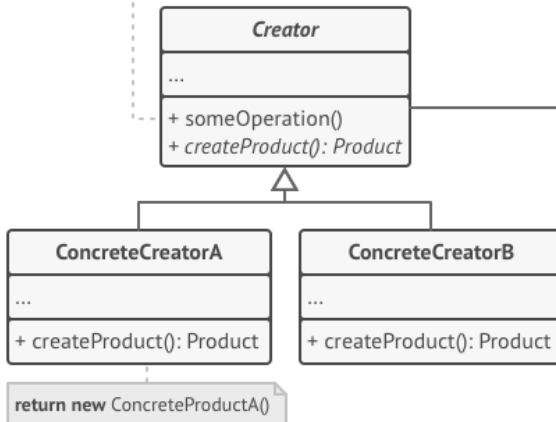
3

Создатель объявляет фабричный метод, который должен возвращать новые объекты продуктов. Важно, чтобы тип результата совпадал с общим интерфейсом продуктов.

Зачастую фабричный метод объявляют абстрактным, чтобы заставить все подклассы реализовать его по-своему. Но он может возвращать и некий стандартный продукт.

Несмотря на название, важно понимать, что создание продуктов **не является** единственной функцией создателя. Обычно он содержит и другой полезный код работы с продуктом. Аналогия: большая софтверная компания может иметь центр подготовки программистов, но основная задача компании — создавать программные продукты, а не готовить программистов.

```
Product p = createProduct()
p.doStuff()
```



4

Конкретные создатели по-своему реализуют фабричный метод, производя те или иные конкретные продукты.

Фабричный метод не обязан всё время создавать новые объекты. Его можно переписать так, чтобы возвращать существующие объекты из какого-то хранилища или кэша.

1

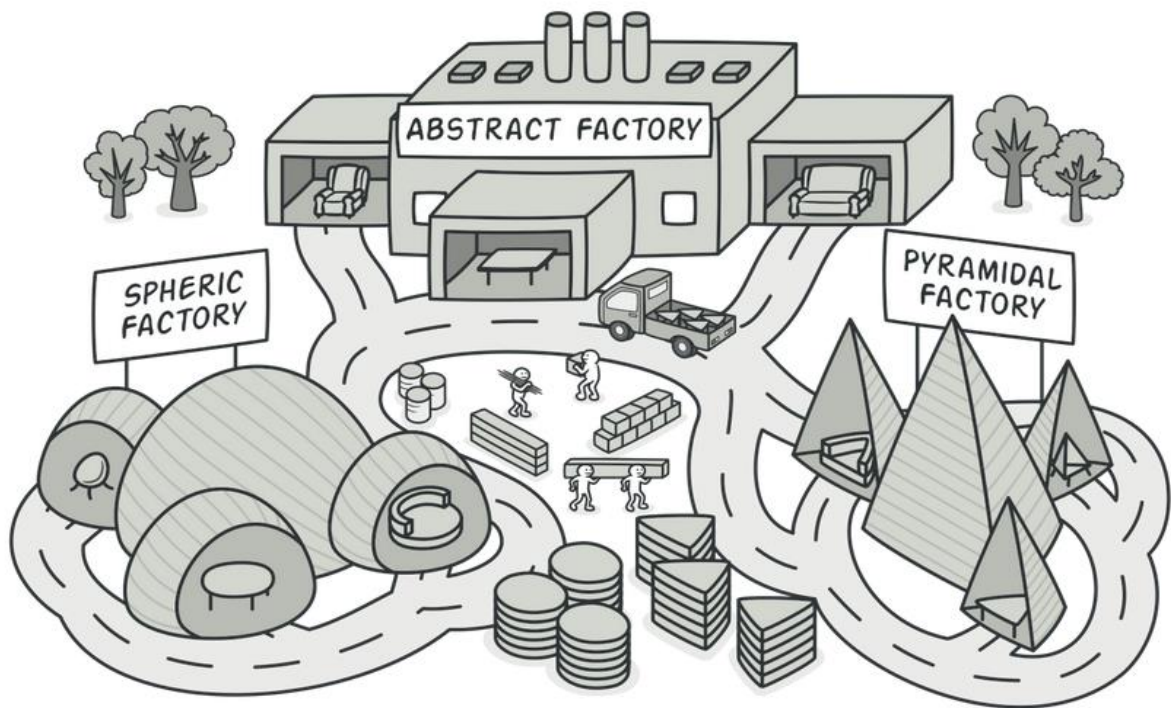
Продукт определяет общий интерфейс объектов, которые может произвести создатель и его подклассы.

2

Конкретные продукты содержат код различных продуктов. Продукты будут отличаться реализацией, но интерфейс у них будет общий.

АБСТРАКТНАЯ ФАБРИКА

Паттерны проектирования



АБСТРАКТНАЯ ФАБРИКА

Абстрактная фабрика — это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Паттерны проектирования: Абстрактная фабрика

ПРОБЛЕМА

Необходимо создать семейства или группы взаимосвязанных объектов исключая возможность одновременного использования объектов из разных этих наборов в одном контексте

МОТИВАЦИЯ ИСПОЛЬЗОВАТЬ ПАТТЕРН

- Скрывает сам процесс порождения объектов, а также делает систему независимой от типов создаваемых объектов, специфичных для различных семейств или групп.
- Позволяет быстро настраивать систему на нужное семейство создаваемых объектов.

Паттерны проектирования: Абстрактная фабрика

