

Behavioral Design Patterns

Chain of responsibility pattern

Command pattern

Interpreter pattern

Iterator pattern

Mediator pattern

Memento pattern

Null object pattern

Observer pattern

State pattern

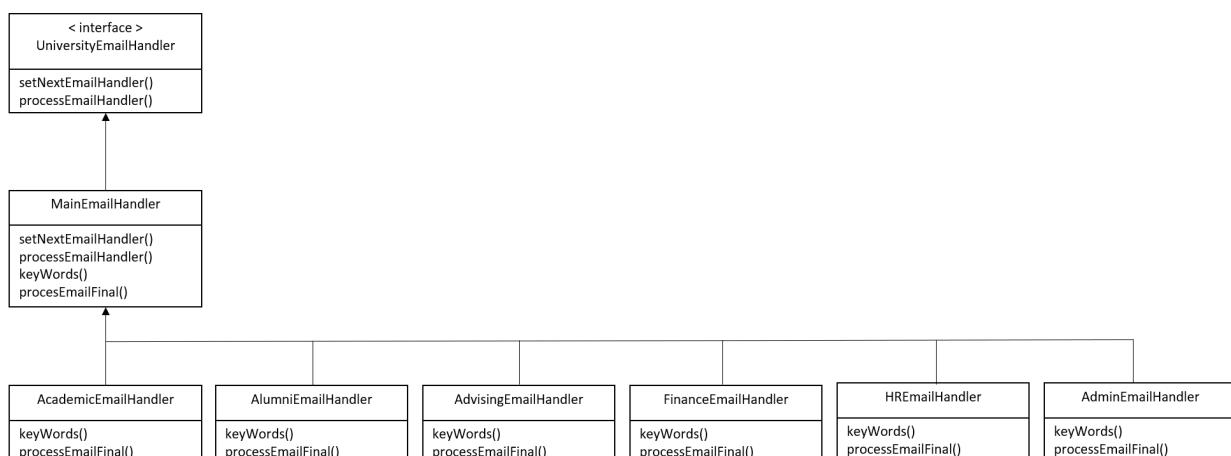
Strategy pattern

Template method pattern

Visitor pattern

Chain of responsibility pattern: An object submits a request to multiple objects

without knowing which object will handle the request - например отсылает мыла, но проверяется
дошли ли они



Command pattern: Permits the sending of requests without knowing details about the receiver or even about what is being requested проверка полномочий проходит в самом классе.

ControlBox

SLIDER_MIN
SLIDER_MAX
poweredOn
sliderValue
getSliderValue()
hasPower()
powerOn()
powerOff()
sliderIncrease()
sliderDecrease()

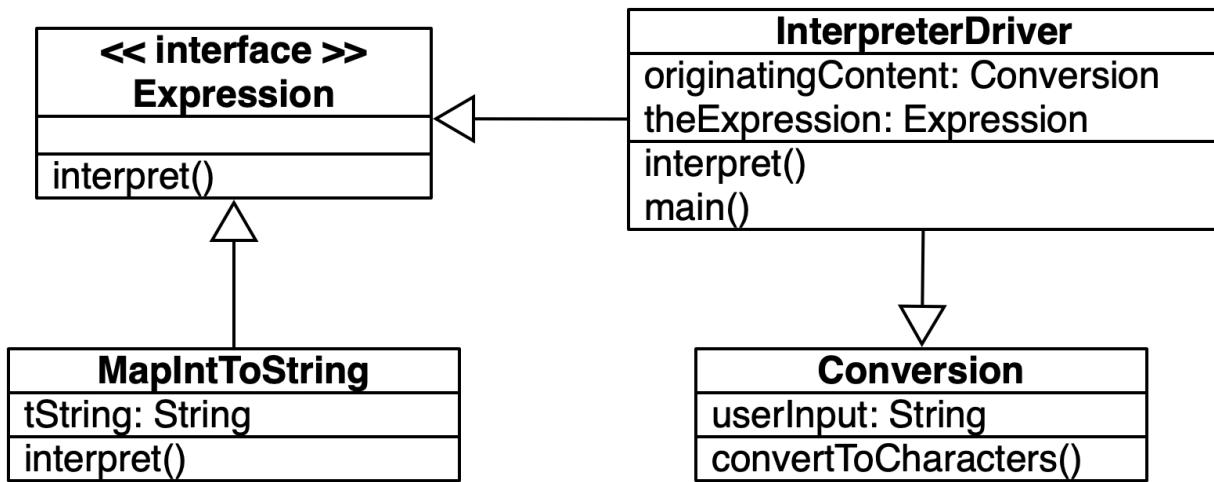
Interpreter pattern: Used to establish a grammatical representation and an interpreter that interprets language - переводит из одного языка в другой. По такому паттерну работает MSIL

CODE INTERPRETER

Enter your code: **319**

Your code: 319

Decrypted Message: YES

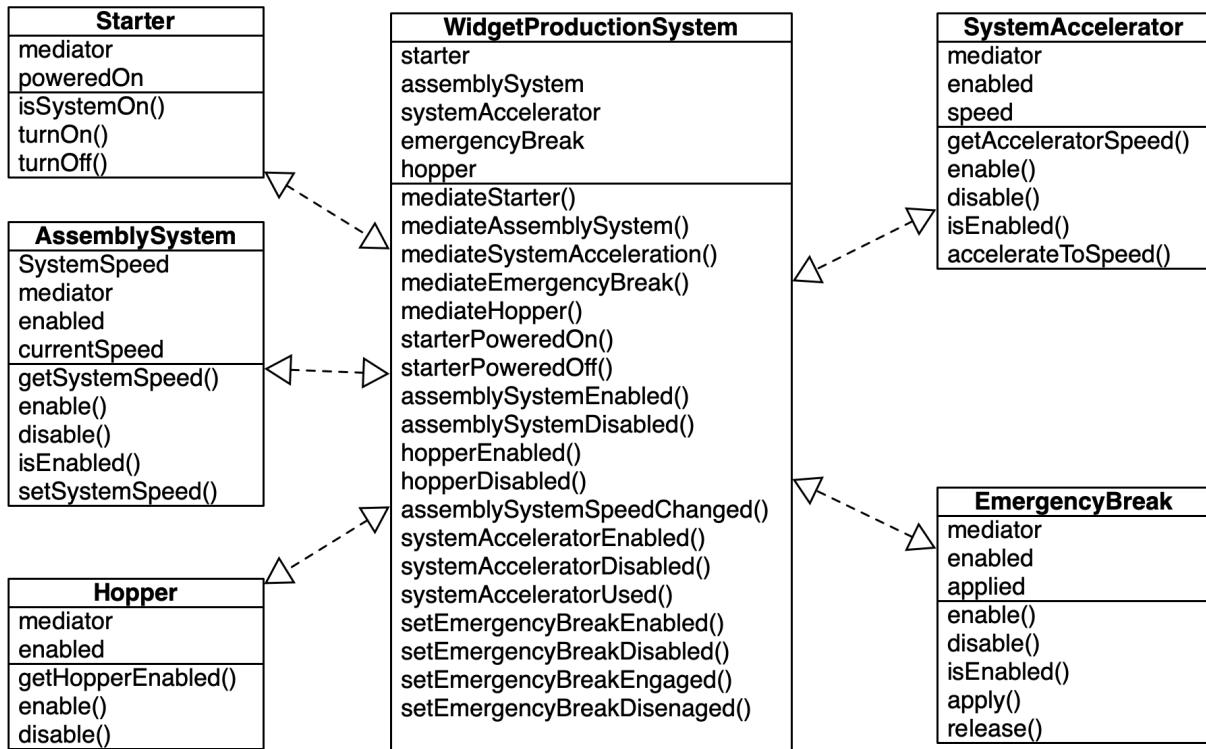


Iterator pattern: Grants access to an object's members without sharing the encapsulated data structures-предоставление доступа к членам объекта без совместное использование инкапсулированных структур данных (например доступ массиву в другом классе)

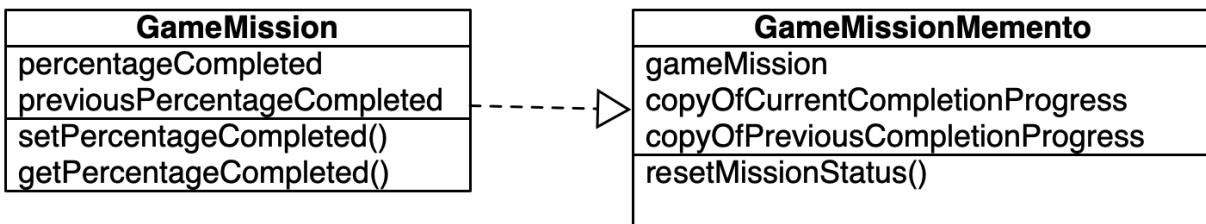
<< interface >> **Iterator**

forEachRemaining()
hasNext()
next()
remove()

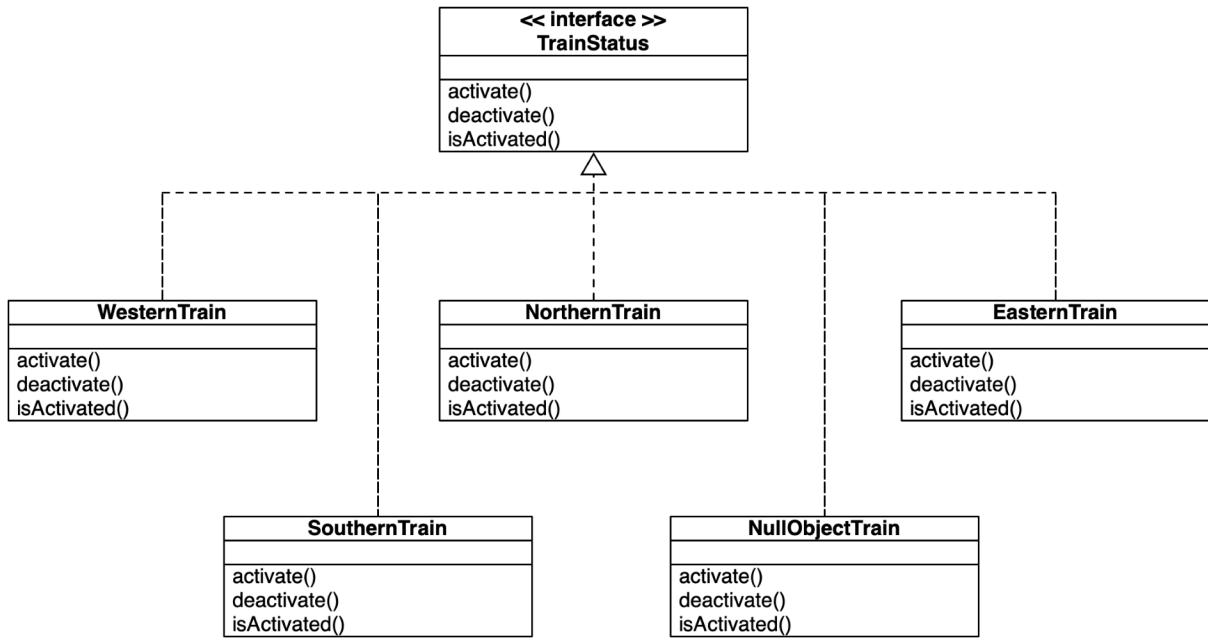
Mediator pattern: Used to permit object interactions without using explicit object References паттерн Посредник. Он между двумя классами и они не знают о его существовании.



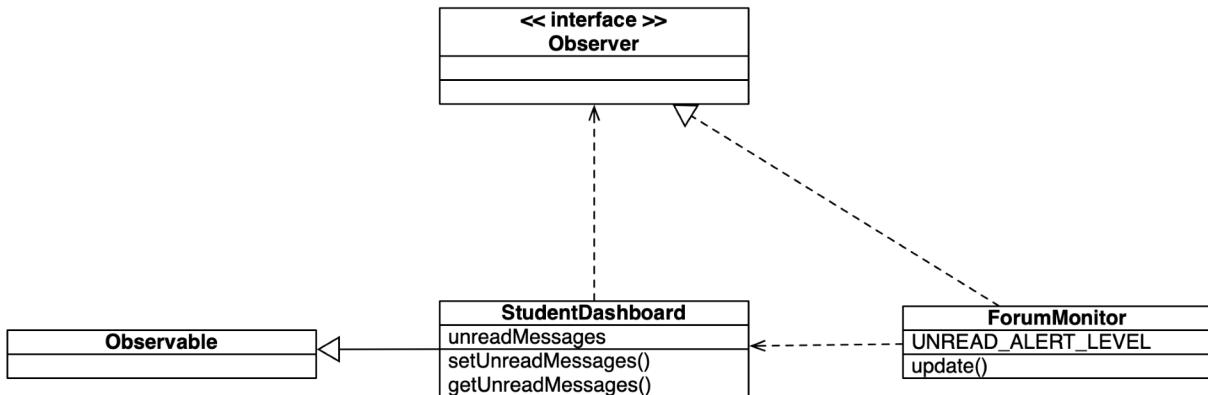
Memento pattern: Saves an object's current internal state as a memento so that it can be referred to and restored to. Паттерн проектирования «мементо» сохраняет текущее внутреннее состояние объекта как «мементо». можно сослаться и восстановить.



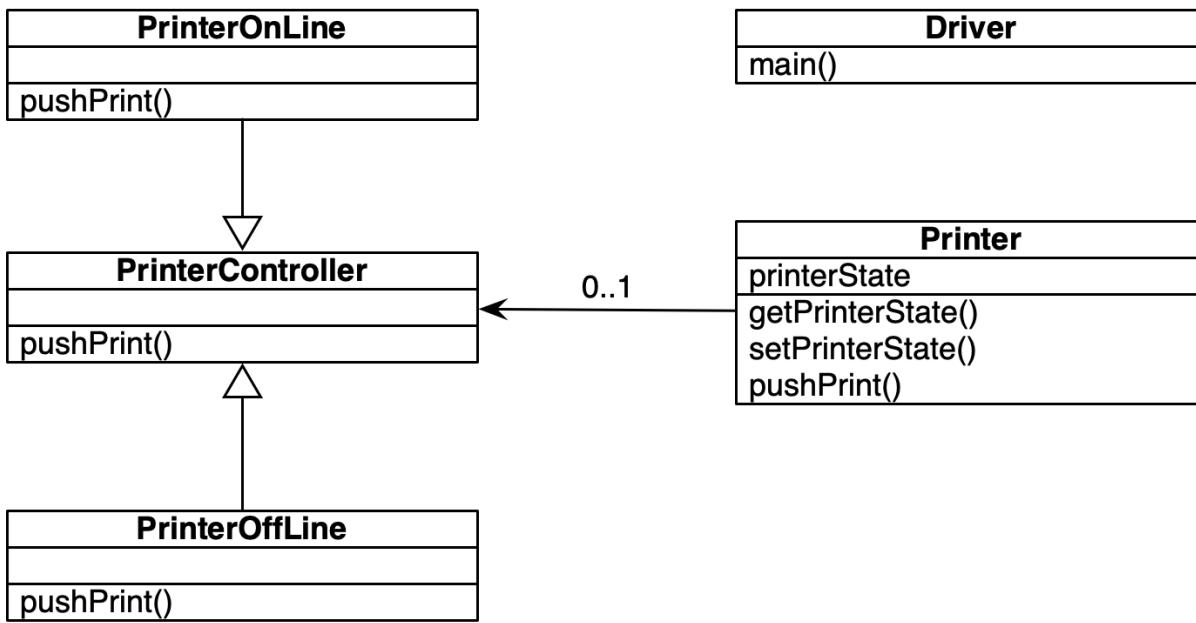
Null object pattern: Negates the need to search for the null condition. Шаблон проектирования нулевого объекта относительно прост. Обычной задачей является проверка нулевые значения во время рутинного программирования. Проверяется на null, чтобы не получить null исключение указателя из виртуальной машины Java (JVM) во время выполнения.



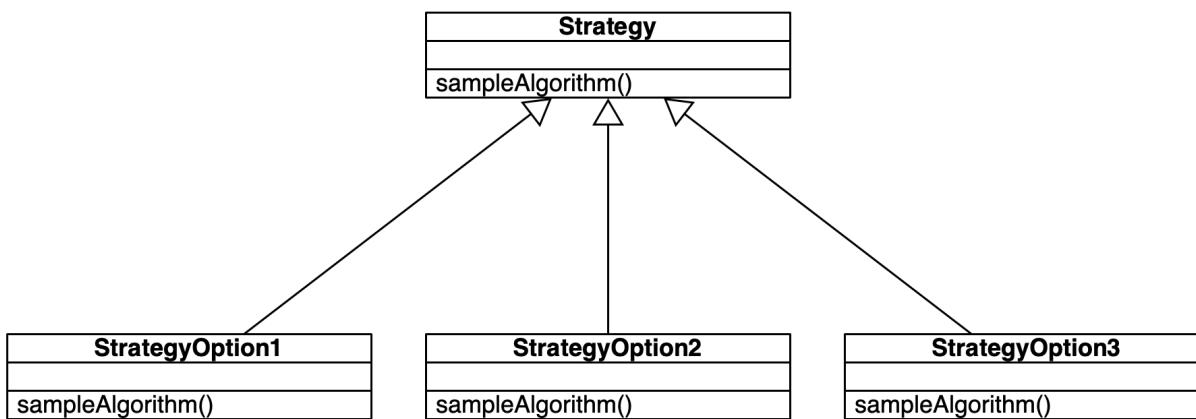
Observer pattern: Updates subscriber objects when a change is made to the publisher object's state. Шаблон проектирования наблюдателя требует объектной зависимости «один ко многим». Цель зависимости заключается в обновлении объектов-подписчиков при изменении объекта-издателя state. Примером может служить онлайн-форум для обсуждения университетских курсов. Есть один форум и много подписчиков. При обновлении форума подписчики уведомляются об этом.



State pattern: Allows an object to change its behavior based on internal state. Шаблон проектирования состояния позволяет объекту изменять свое поведение в зависимости от внутреннего изменения состояния. В результате может показаться, что объект меняет свой класс.



Strategy pattern: Allows us to individually encapsulate a set of interchangeable Algorithms Шаблон проектирования стратегии позволяет индивидуально инкапсулировать набор взаимозаменяемых алгоритмы. Это приводит к изменчивости алгоритма в зависимости от вызывающего клиента. Это аналогично перегрузке методов, которая позволяет классу иметь более одного метода с тем же именем. Разница между одноименными методами заключается в их списке аргументов. Шаблон проектирования стратегии отличается от примера с перегрузкой метода, когда каждый алгоритм индивидуально инкапсулирован.



Template method pattern: Involves creating an algorithm template with processing steps relegated to child classes Шаблон проектирования метода шаблона включает создание шаблона алгоритма с этапы обработки отнесены к дочерним классам. Цель состоит в том, чтобы дать дочерним классам возможность указывать свои собственные шаги, оставаясь при этом верным структуре алгоритма. Как следует из названия шаблона проектирования, мы создаем

шаблон, которому можно следовать подчиненные классы. Примером может служить шаблон рецепта, в котором пекари выполняют шаги, чтобы создать оболочку, а затем добавить свою собственную начинку для пирога в зависимости от своих предпочтений. В конце

Visitor pattern: Performs operations on an object without altering its structure

Шаблон проектирования посетителя позволяет нам выполнять операции над объектом, не изменяя его структуры. По сути, мы можем добавлять функциональные возможности к объекту, изменяя оригинал. структура объекта.

Creational Design Patterns

Abstract factory

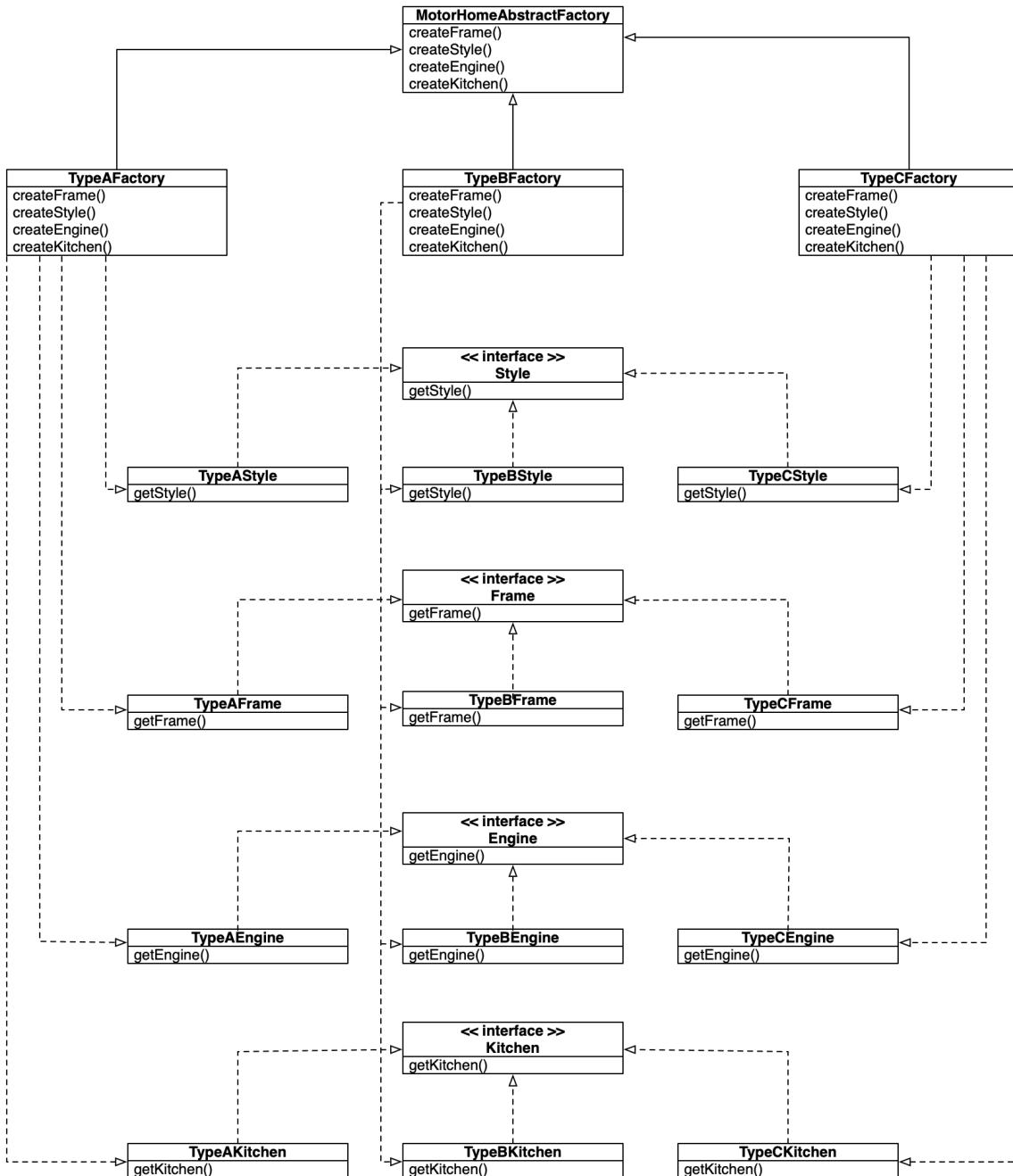
Термин абстрактный относится к чему-то, что не имеет определенного существования.

Grandmother constructor executed.

Mother constructor executed.

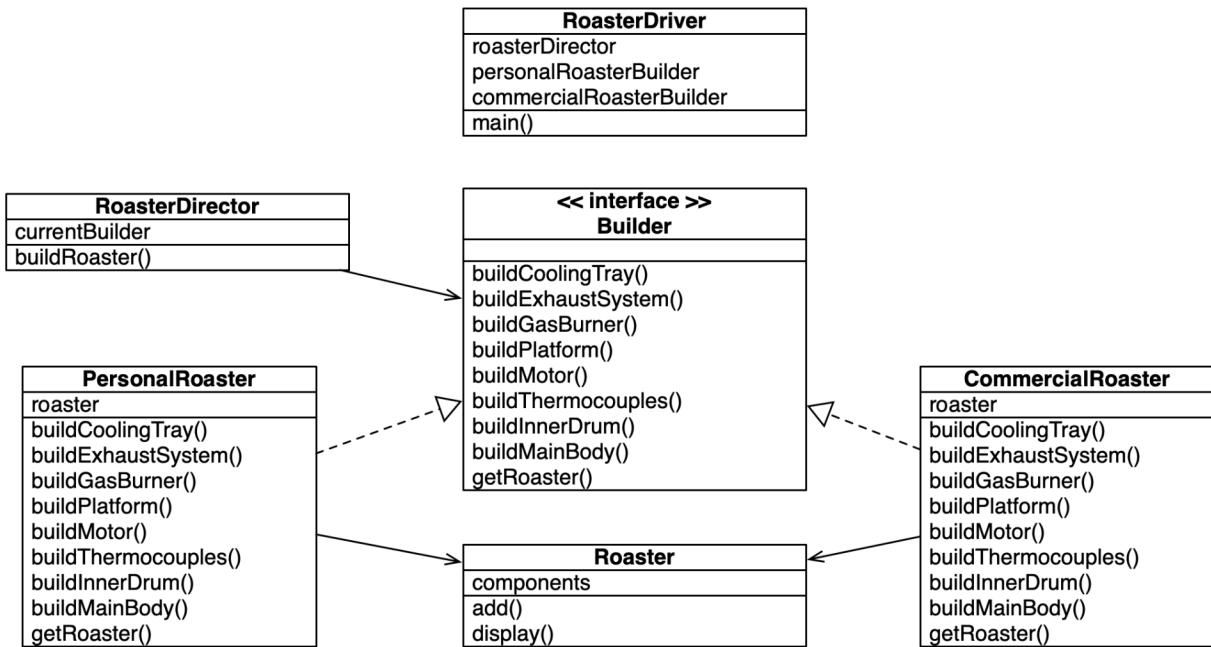
```
public class Daughter {  
  
    public static void main(String[] args) {  
        Mother mom = new Grandmother();  
    }  
}
```

'Grandmother' is abstract; cannot be instantiated



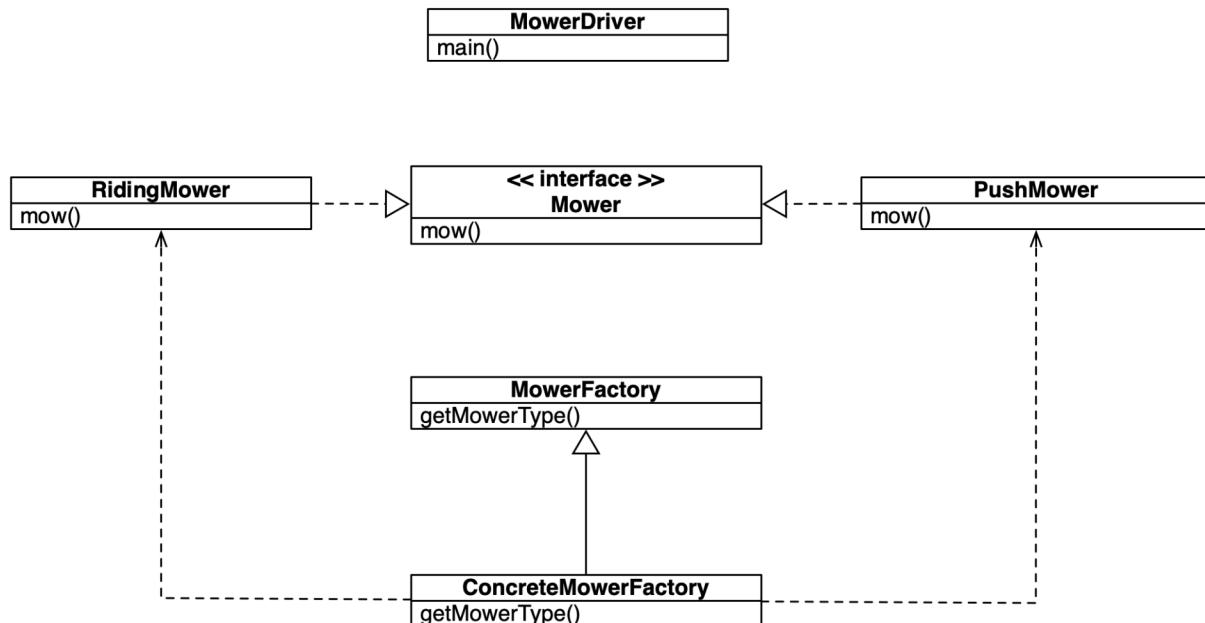
Builder

Шаблон проектирования построителя используется для создания разделения между созданием экземпляра объекта и представление. Цель состоит в том, чтобы разрешить различные представления с одним и тем же процессом создания объектов.



Factory method

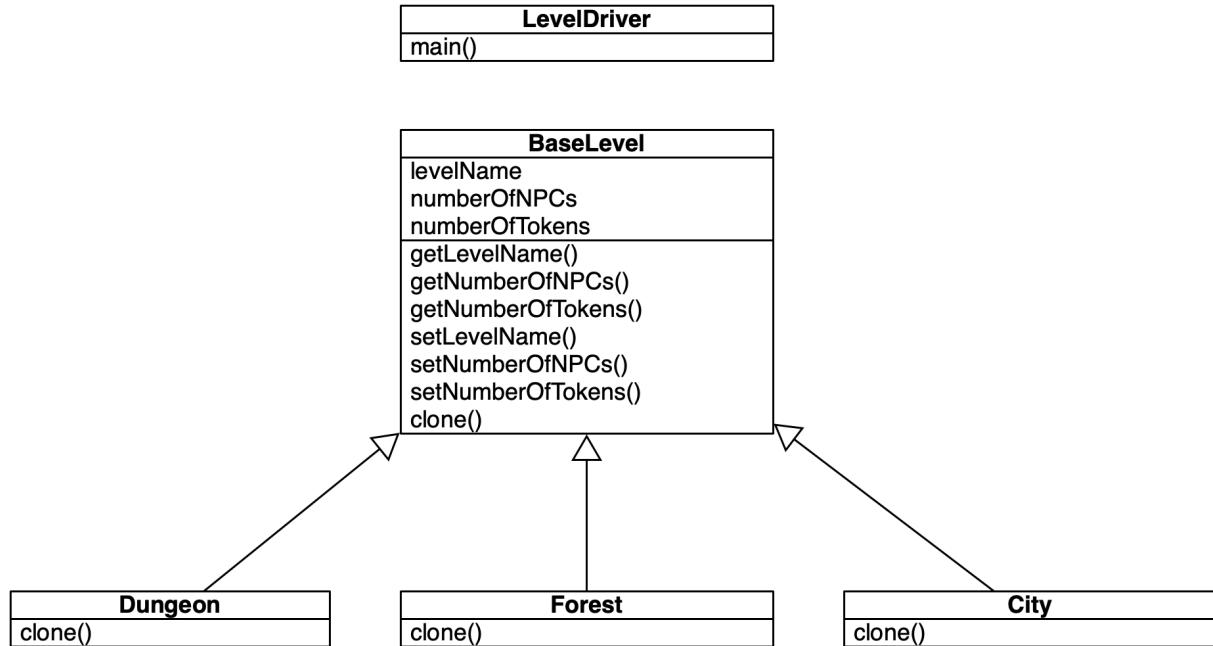
Шаблон проектирования фабричный метод позволяет подклассам определять, какой класс создавать. Это достигается за счет удаления деталей о том, какой класс создавать вне фреймворка. Вместо этого на подклассы возлагается ответственность за создание объектов. Этот шаблон проектирования полезно, когда фреймворк не знает, что должно быть создано.



Prototype

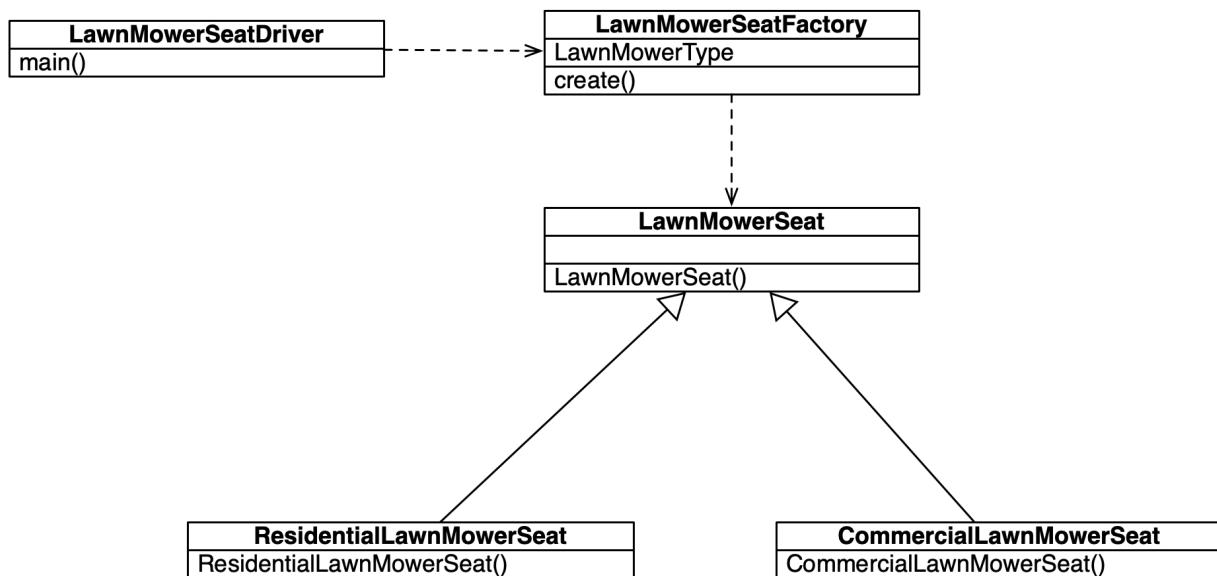
Шаблон разработки прототипа позволяет нам указать категорию объектов с помощью прототипа. Затем этот экземпляр копируется для создания новых объектов. Дизайн

прототипа идеально подходит для ситуаций, когда вы хотите, чтобы создание объекта не зависело от системы. Например, мы можем разрабатывать игру с несколькими уровнями. Каждый уровень основанный на базовом уровне и модифицированный впоследствии.



Simple factory

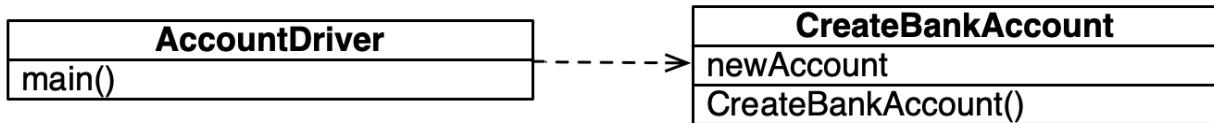
Простой фабричный шаблон проектирования используется для делегирования создания объекта определенному классу.



Singleton

Целью этого шаблона проектирования является обеспечение только один экземпляр класса, и он должен быть доступен извне. Это обычное дело для основанных на безопасности системы для реализации одноэлементного шаблона проектирования.

Примером может служить банковское дело. Система, которая создает новые номера счетов. Важно, чтобы эти номера счетов были генерируются только одной системой.

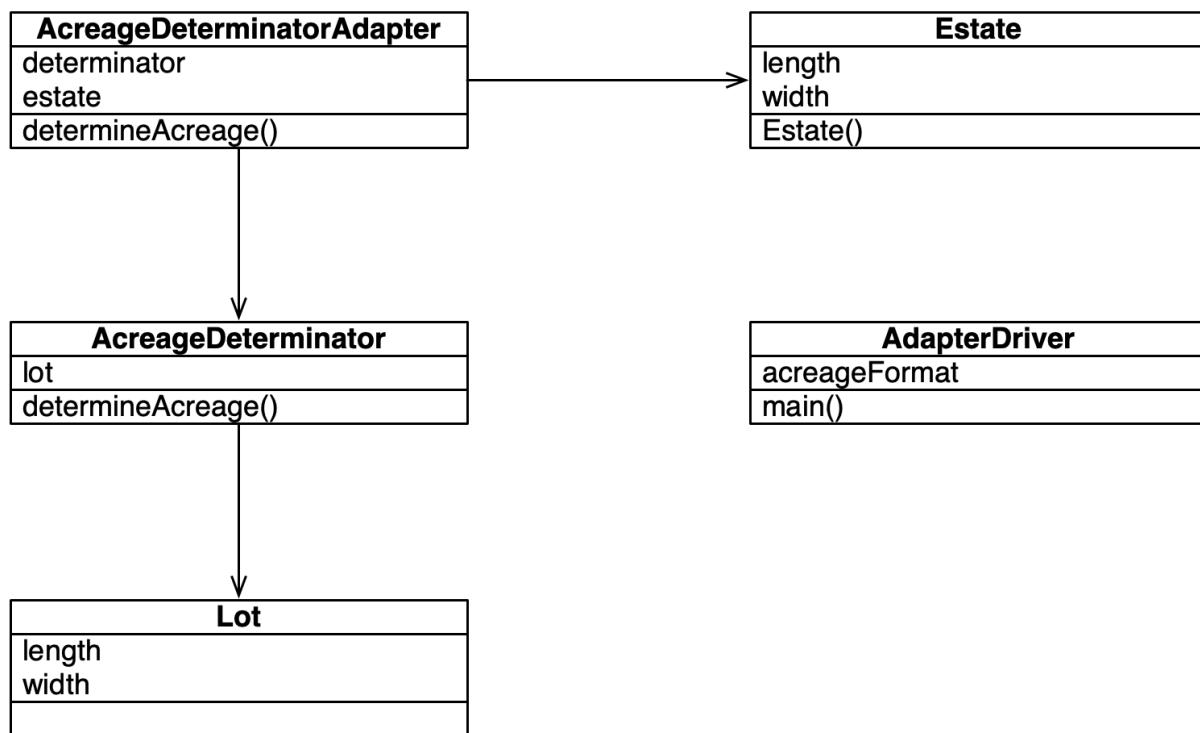


Structural Design Patterns

Structural object design patterns are used to describe how to create objects with new functionality

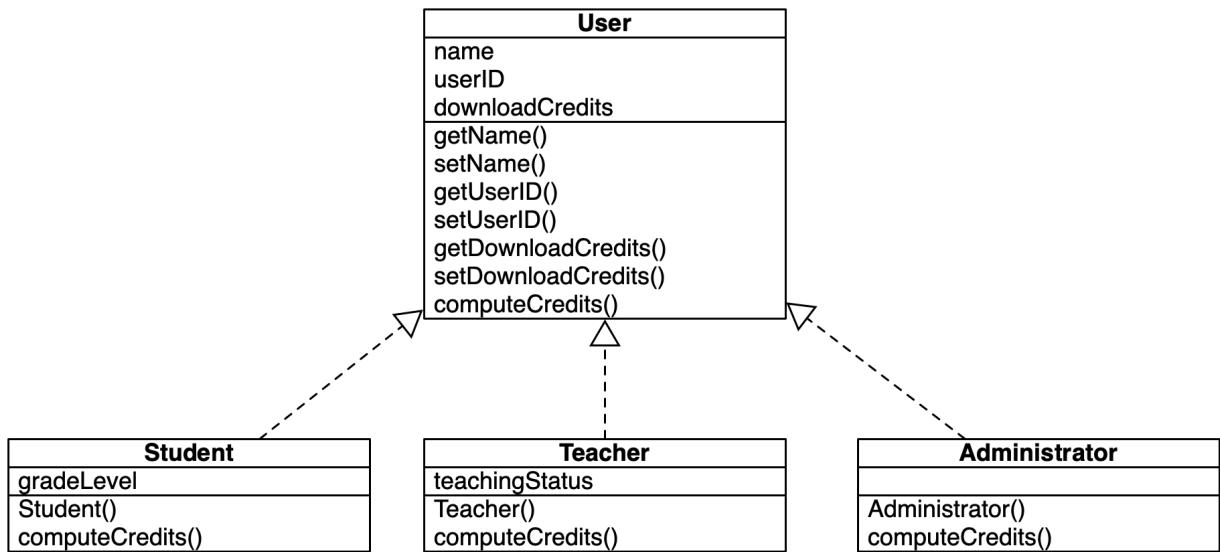
Adapter

Шаблон проектирования адаптера используется для преобразования интерфейса одного класса в другой интерфейс, что ожидается системой. Этот шаблон проектирования позволяет классам работать согласованно один с другим, независимо от совместимости их интерфейсов.



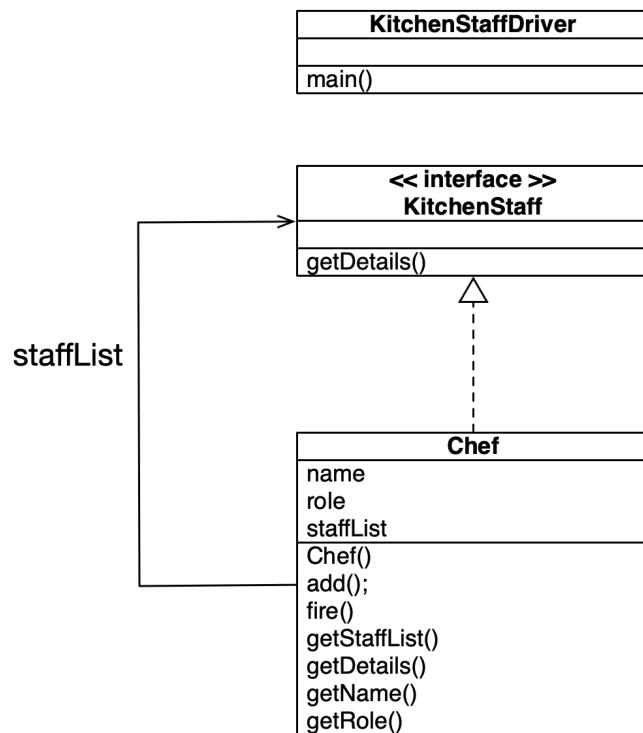
Bridge

Pattern включает в себя создание моста между абстракцией и реализацией.



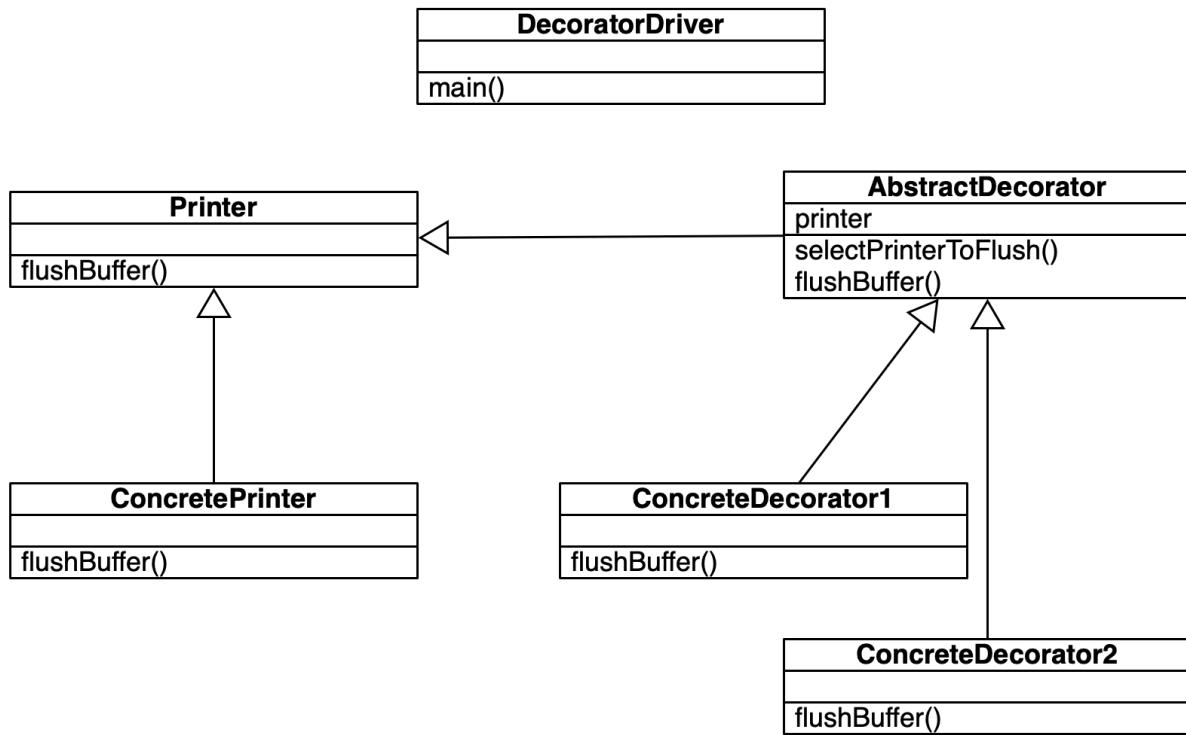
Composite

Шаблон составного проектирования позволяет нам создать древовидную структуру объектов.



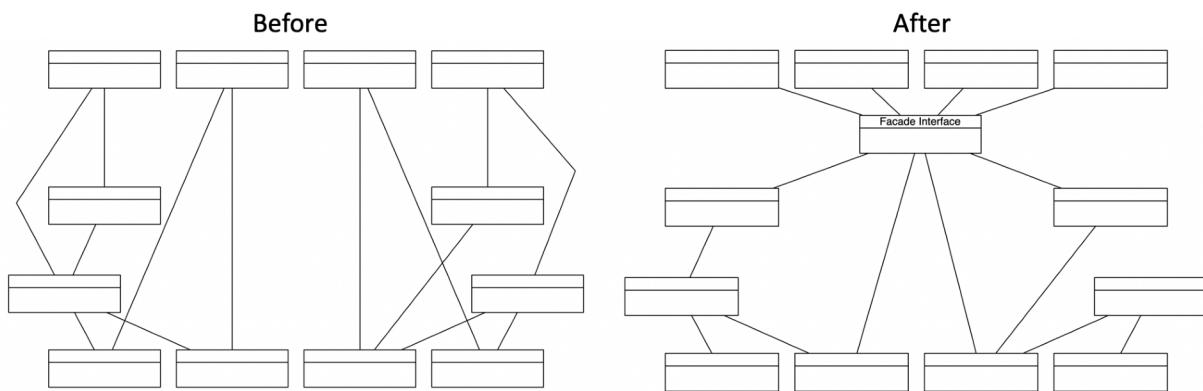
Decorator

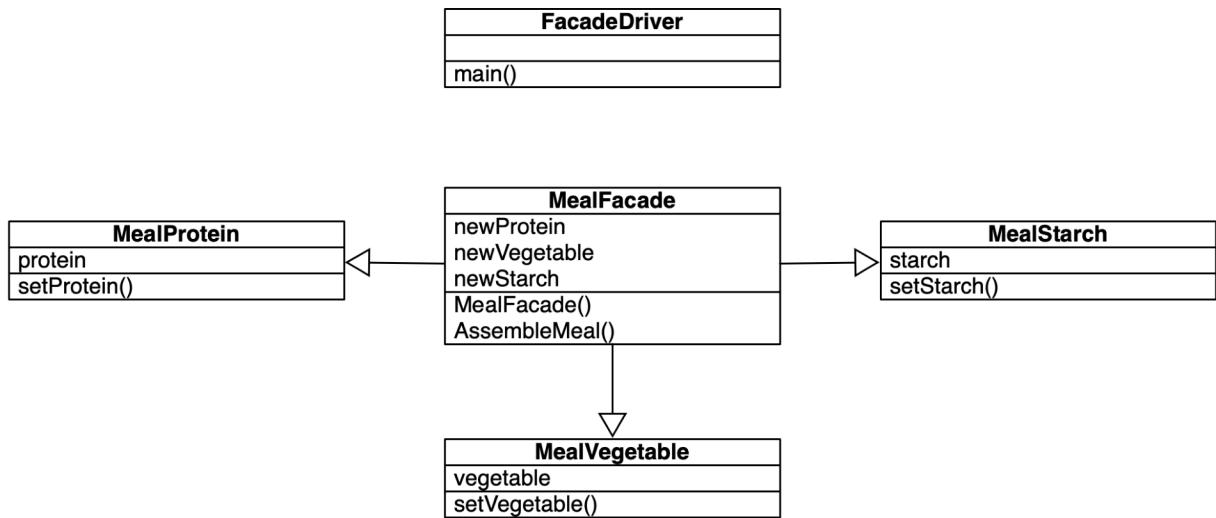
Шаблон проектирования декоратора позволяет нам назначать обязанности объекту без влияющие на класс.



Facade

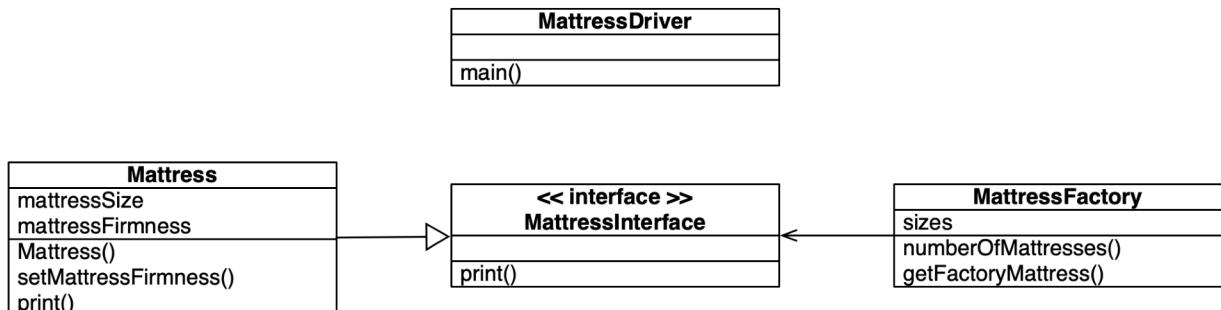
Шаблон проектирования фасада создает интерфейс, который служит интерфейсом для других интерфейсов внутри системы или подсистемы. Преимущества использования этого шаблона проектирования заключаются в том, что подсистемы менее сложны, компоненты надежности уменьшены, а связь между компонентами системы сведена к минимуму.





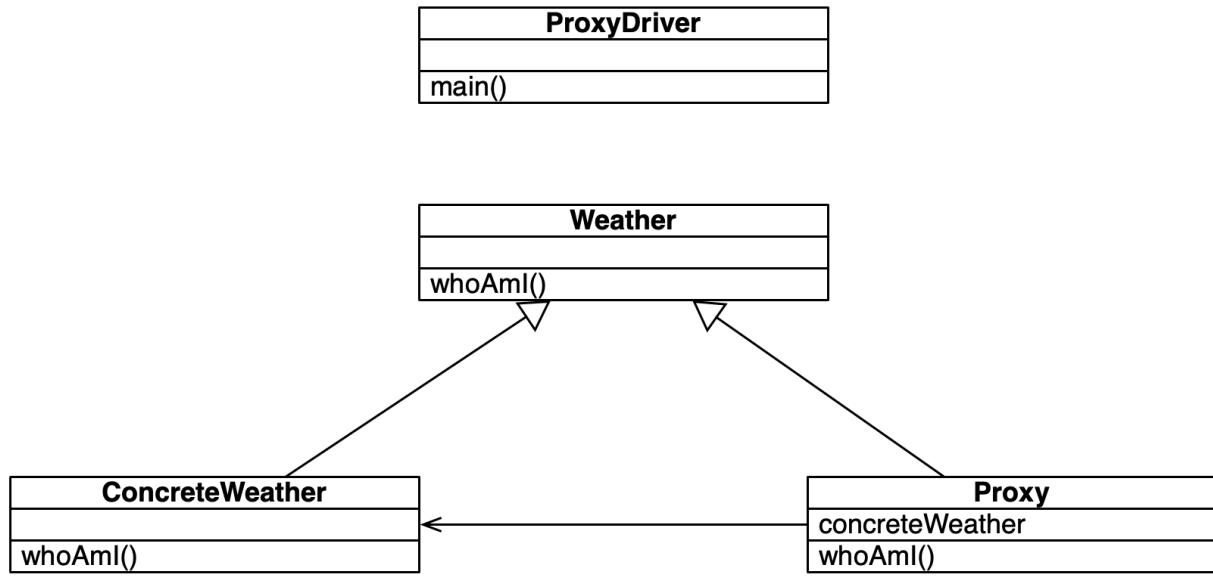
Flyweight

Легковесные шаблоны проектирования обеспечивают высокую эффективность и улучшенную обработку при работе с большим количеством однотипных объектов. Шаблон использует совместное использование составных частей идентичных объектов. Рассмотрите возможность программирования видеоигры с тысячами мозаичных объектов, которые используются для возведения архитектурных сооружений. Объекты плитки, скорее всего, будут иметь только цвет атрибут.



Proxy

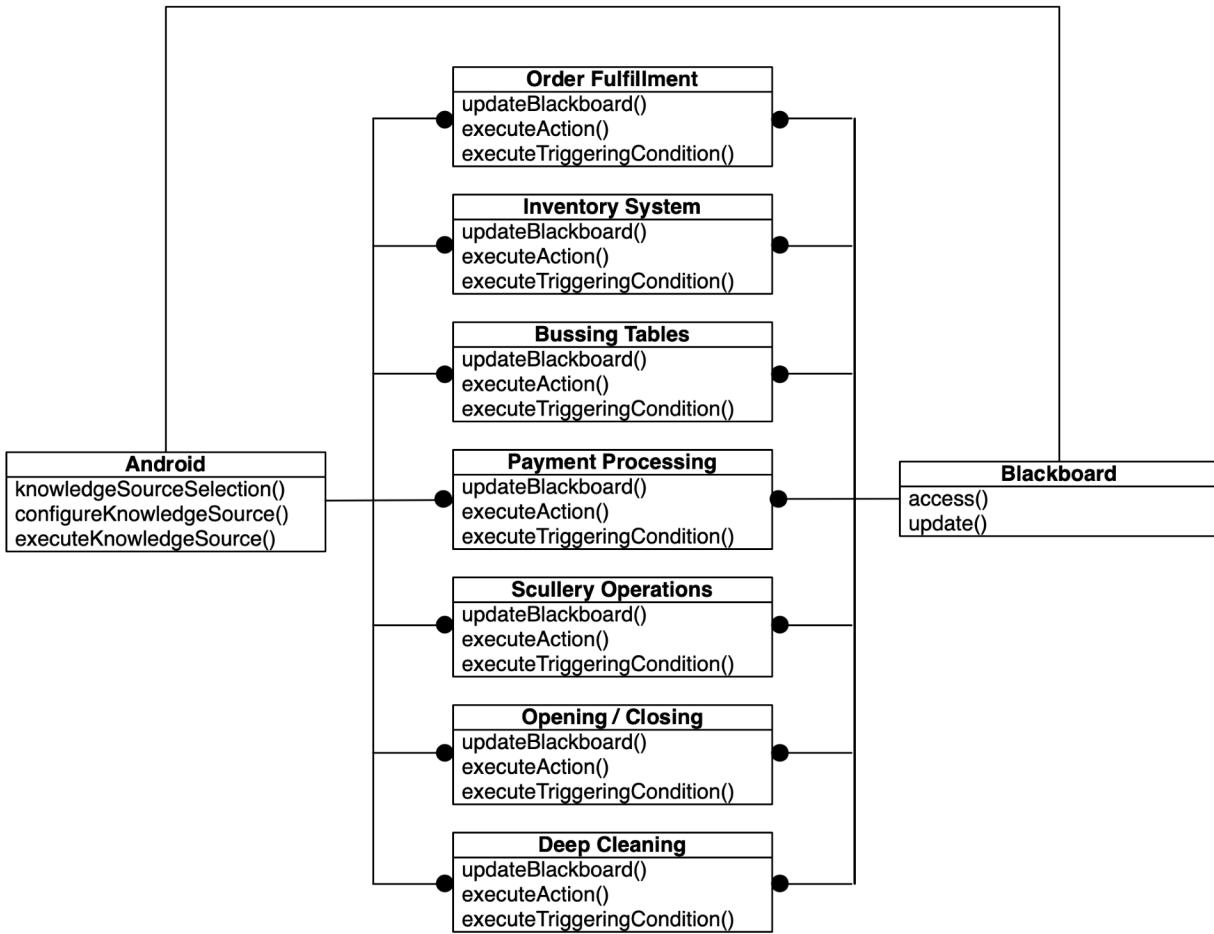
Прокси в контексте Java определяется как имеющий право представлять другой объект. Шаблон проектирования прокси соответствует своему названию в том смысле, что он устанавливает заполнитель, так что объект, отличный от самого себя, может управлять доступом.



New Design Patterns

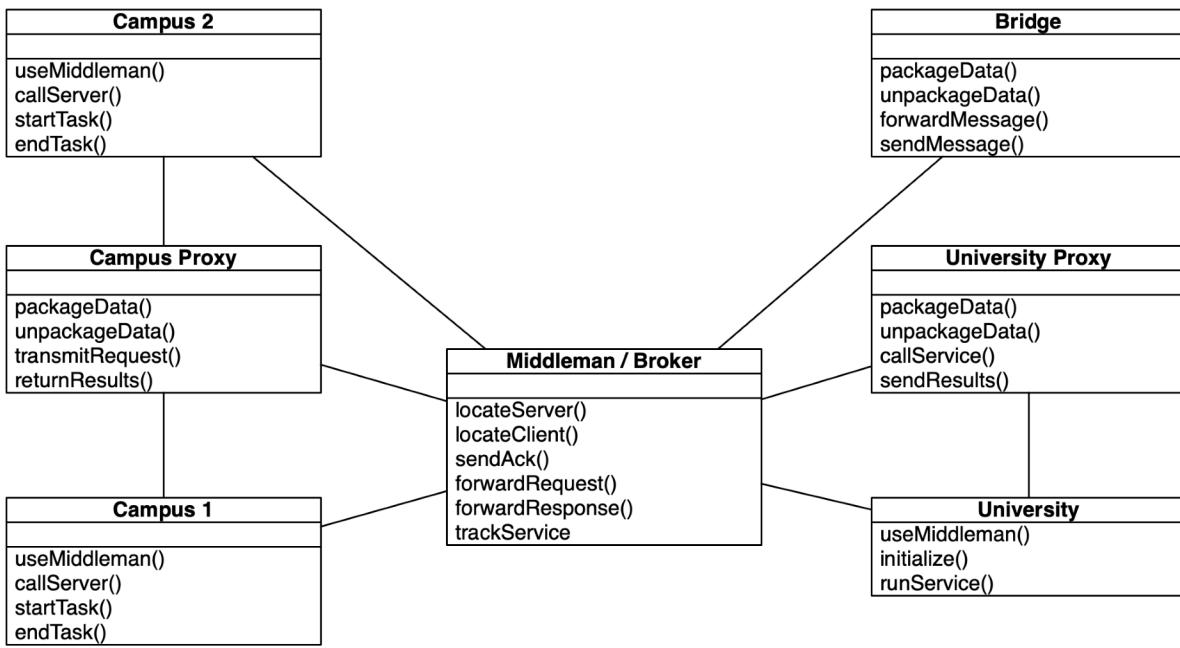
Blackboard pattern

Архитектурный паттерн «классная доска» используется для больших систем, не имеющих окончательного решения. Этот шаблон часто используется для разработки фреймворков для распознавания речи, компьютерных игры и динамические системы с искусственным интеллектом или машинным обучением.



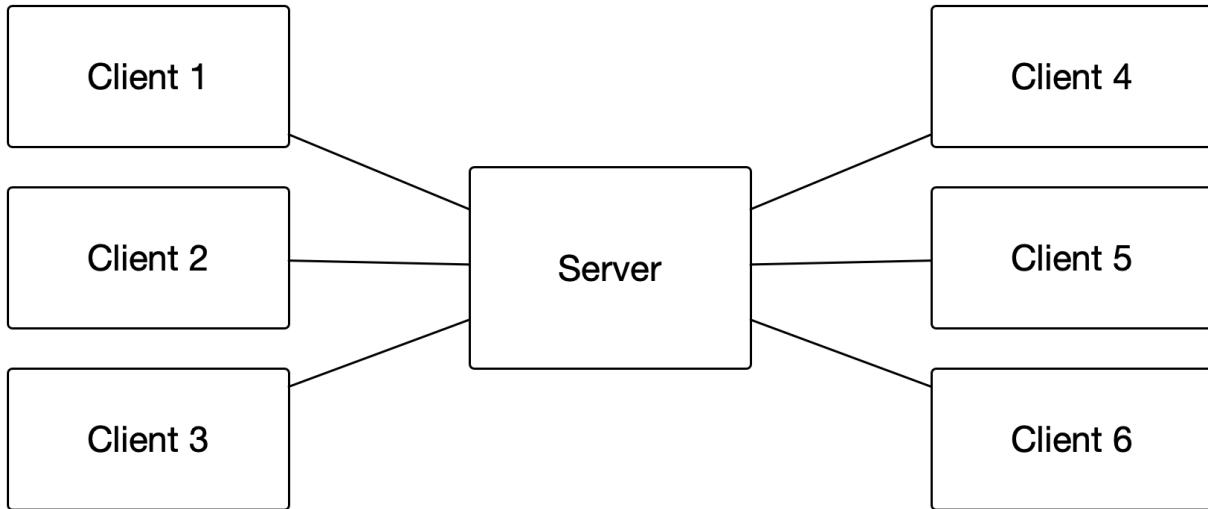
Broker pattern

Архитектурный шаблон брокера используется для проектирования распределенных систем. Эти системы использовать компонент посредника для координации и связи между составные части.



Client-server pattern

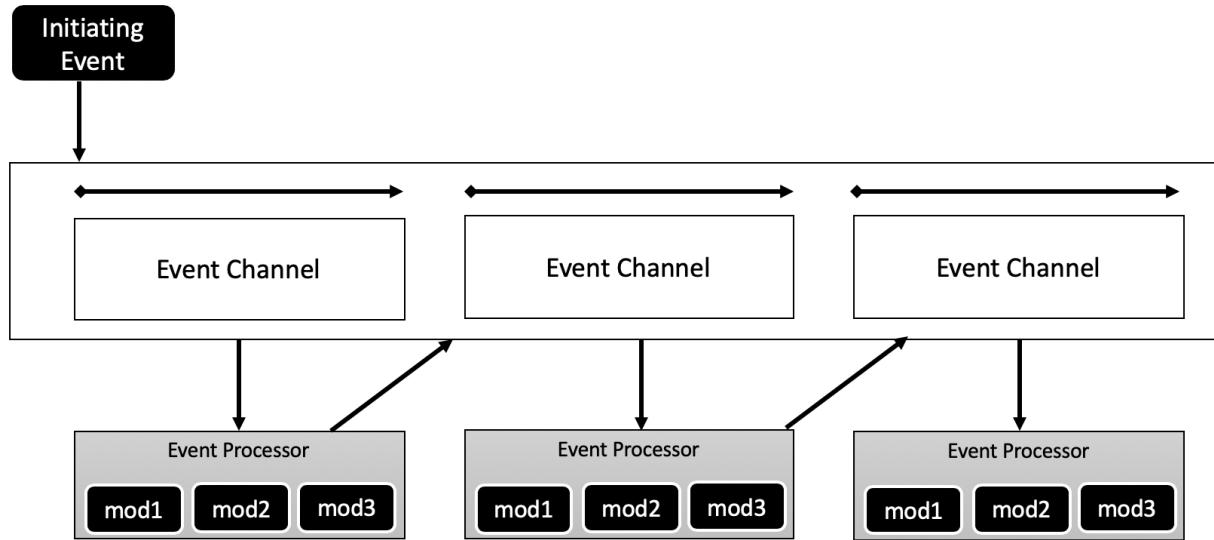
Архитектурный паттерн клиент-сервер является одним из наиболее широко известных архитектурных паттернов. Шаблоны, особенно с сетевыми архитектурами. Этот архитектурный образец состоит из двух типов компонентов — клиентские компоненты и серверные компоненты. Проще говоря, сервер слушает запросы от клиентов и предоставляет запрошенные услуги.



Event-driven pattern

Управляемый событиями архитектурный шаблон предназначен для легко адаптируемых распределенных систем. Это используется для реализации приложений, которые включают передачу событий в децентрализованной системе. В этом шаблоне у событий есть издатели и потребители. Использование этого шаблона позволяет более

эффективная разработка больших распределенных систем. Этот шаблон можно использовать для веб-систем, бизнес-процессов, игр и практически любого приложение, о котором вы можете думать, имеет события.



Extract-transform-load pattern

Архитектурный шаблон извлечения-преобразования-загрузки, как следует из названия, имеет следующий вид:

три этапа:

1. Извлечение данных из внешних источников
2. Преобразование извлеченных данных по мере необходимости
3. Загрузка недавно преобразованных данных в репозиторий, специфичный для текущей системы.

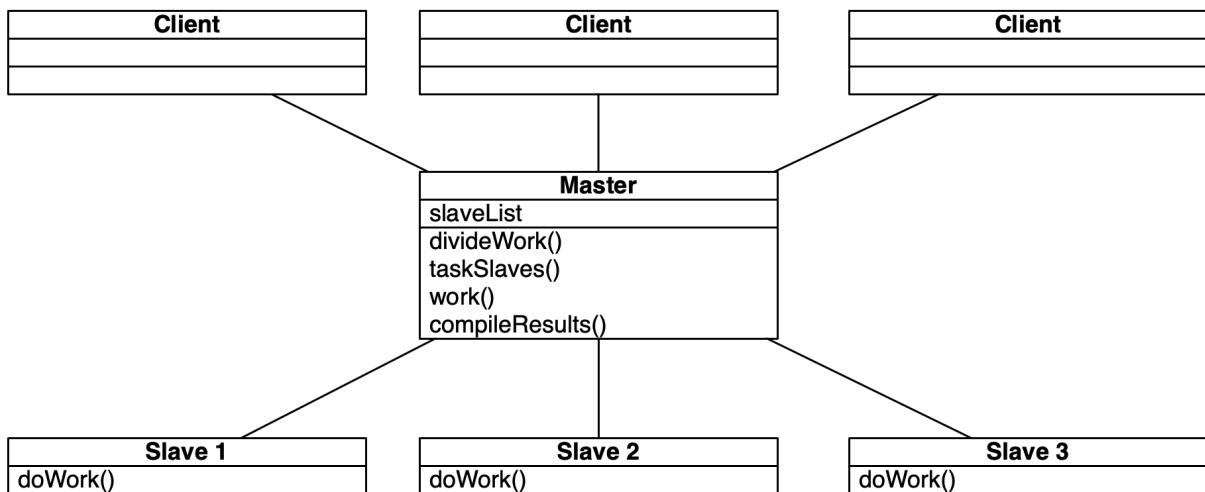
Шаблон извлечения-преобразования-загрузки чаще всего используется в бизнес-аналитике. В хранении данных, системах управления знаниями и отношениями с клиентами.

Layered pattern

Многоуровневый архитектурный шаблон составляет систему, состоящую из подзадач. А уровни системы имеют определенный набор обязанностей, специфичных для реализации. Это широко используемый шаблон, который можно найти в большинстве надежных систем, особенно в тех, воспользоваться преимуществами распределенных вычислений, таких как облачные вычисления или облачное хранилище.

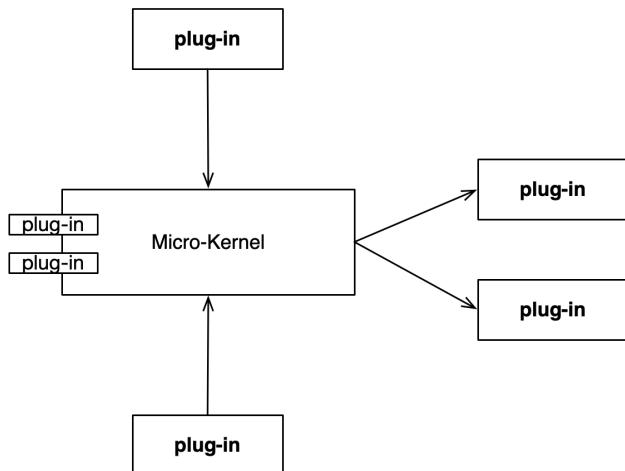
Master-slave pattern

Архитектурный шаблон ведущий-ведомый используется для повышения надежности системы и производительность путем разделения работы между ведущим и ведомым компонентами. Каждый компонент имеет определенные обязанности. Все подчиненные компоненты имеют идентичную или, по крайней мере, похожую работу, и эта работа должна быть определена до выполнения.



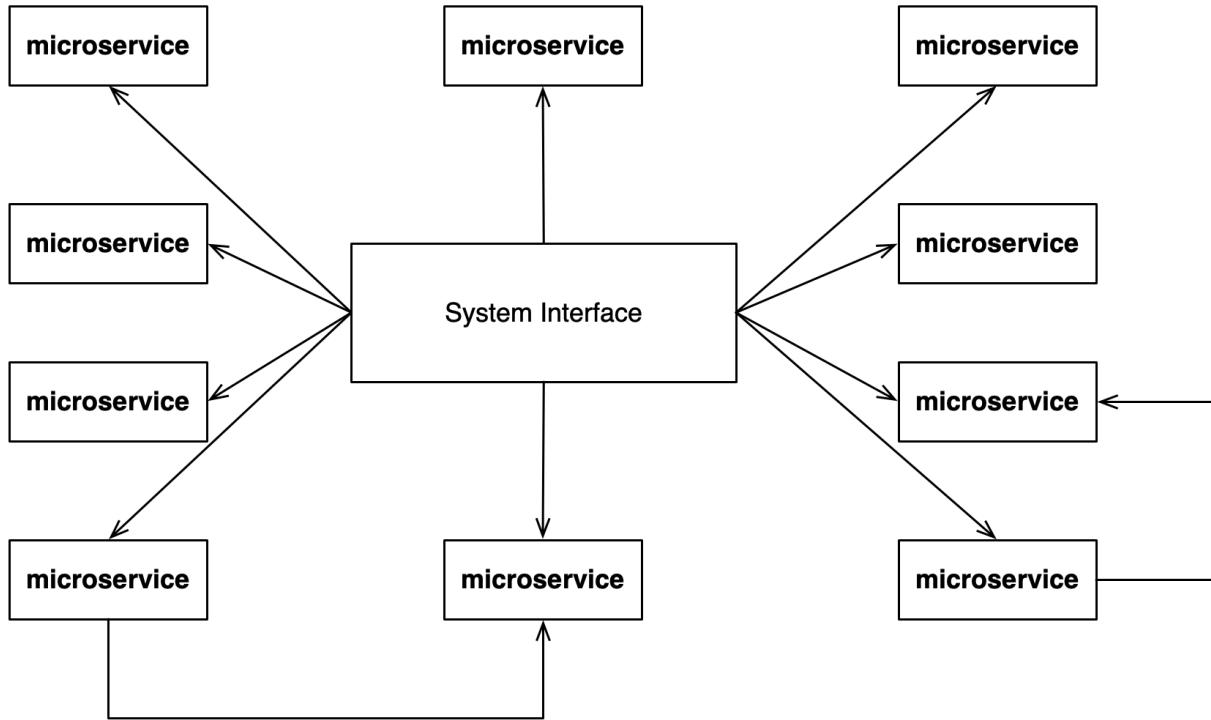
Microkernel pattern

Архитектурный шаблон микроядра также называют архитектурным шаблоном подключаемого модуля. Обычно мы используем этот шаблон, когда создаем системы со взаимозаменяемыми компонентами. Показан на следующей диаграмме в виде плагина:



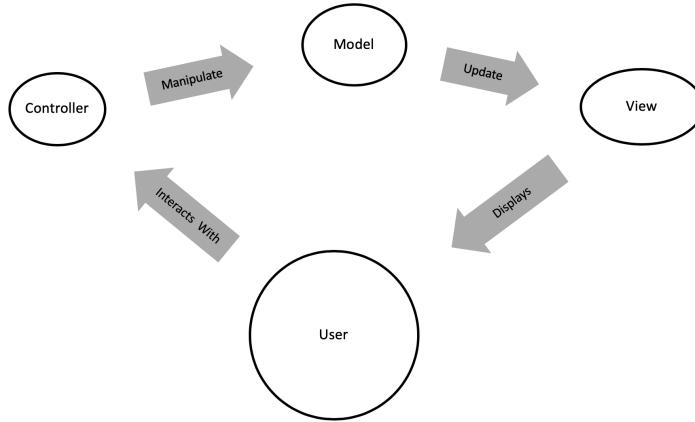
Microservices pattern

Архитектурный шаблон микросервисов используется для разбиения системы на несколько более мелких сервисов или микросервисы, которые имеют ограниченные взаимозависимости.



Model-view-controller pattern

Архитектурный шаблон модель-представление-контроллер является одним из наиболее часто используемых шаблонов.



Naked objects pattern

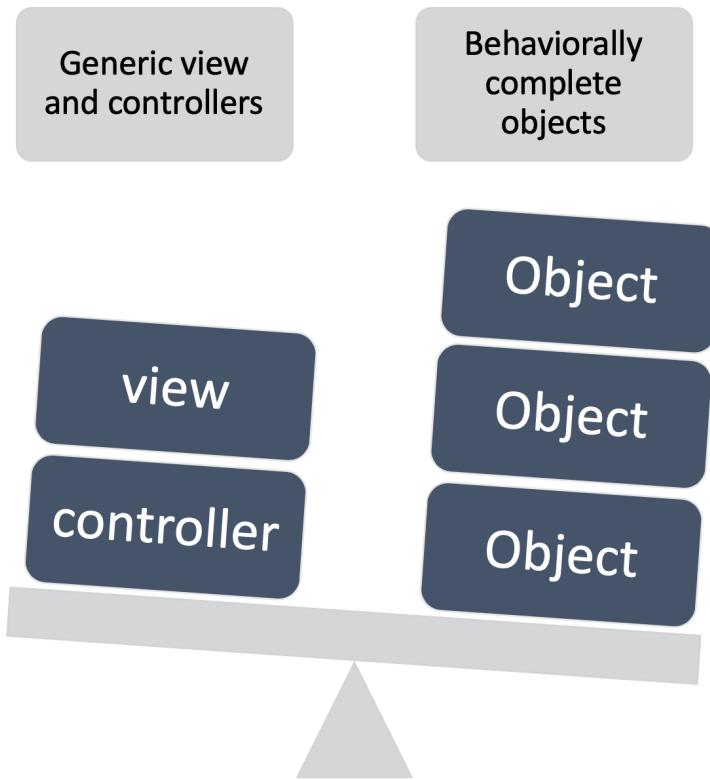
Архитектурный паттерн «голые объекты» предписывает инкапсуляцию объектов предметной области данных. Кроме того, необходимо создать пользовательский интерфейс для следующих действий:

Создание объектов

Поиск объекта

Извлечение данных объекта

Вызов метода



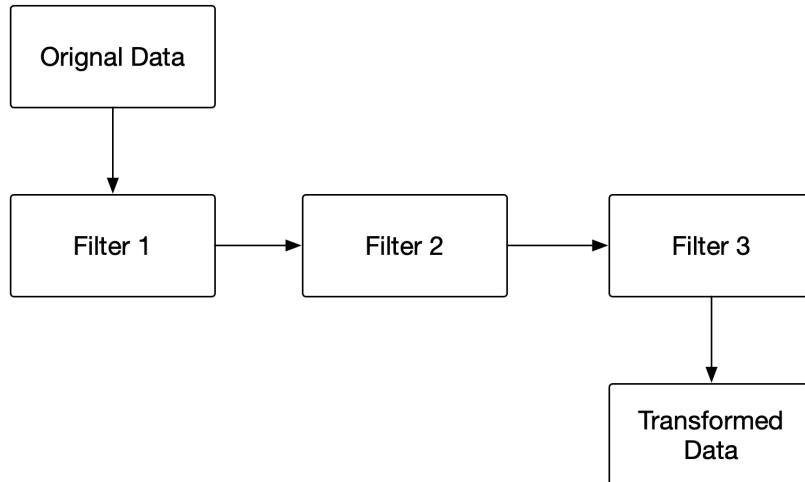
Peer-to-peer (P2P) pattern

Архитектурный шаблон одноранговой сети (P2P) состоит из ряда узлов, каждый из которых имеет одинаковый набор функций для выполнения. В этом шаблоне нет центрального контроллера и все узлы создаются одинаково. Равноправные узлы действуют как получатели и распространители данных.

It's highly vulnerable to Denial-of-Service (DoS) attacks

Pipe-filter pattern

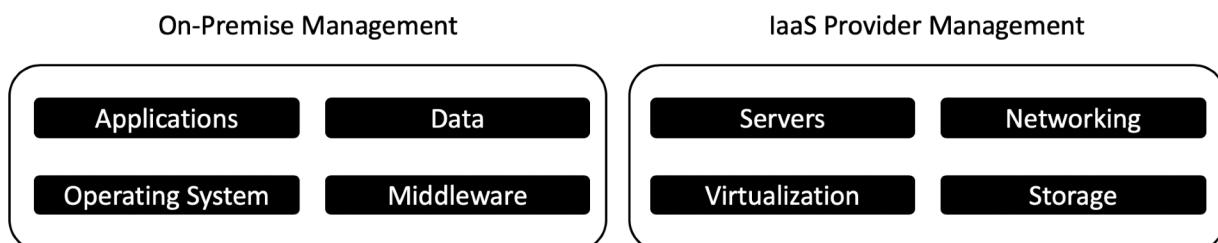
Архитектурный паттерн «труба-фильтр» или «труба и фильтр» — это надежная архитектура, которая может содержать любое количество фильтров. Шаблон начинается с получения данных из нескольких источников и затем пропускает их через последовательные фильтры для преобразования данных из одного формата в еще один. Фильтры соединены трубами.



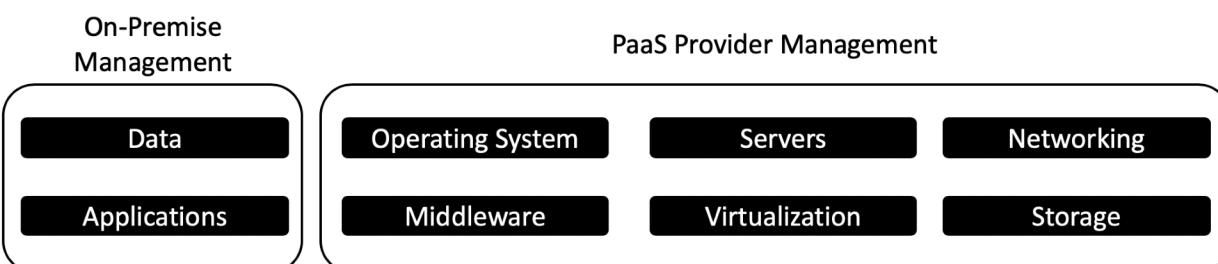
Serverless pattern

Реализация бессерверных архитектурных шаблонов становится все более распространенной из-за появление инфраструктуры как услуги (IaaS), платформы как услуги (PaaS), программного обеспечения как Сервис (SaaS), Серверная часть как услуга (BaaS), Мобильная внутренняя часть как услуга (MBaaS) и Облачные предложения «функции как услуга» (FaaS). Бессерверные архитектурные шаблоны просто шаблоны, которые не включают выделенные локальные серверы.

IaaS

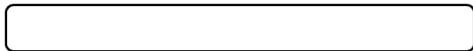


PaaS

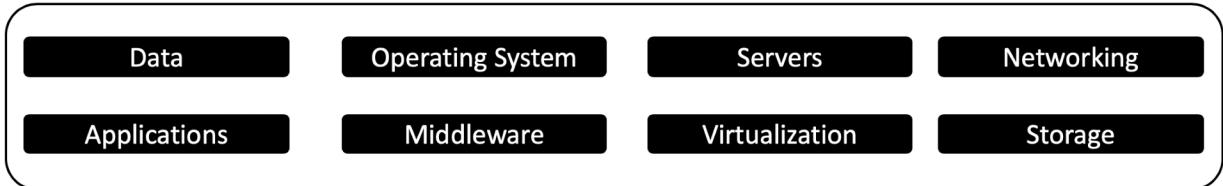


SaaS

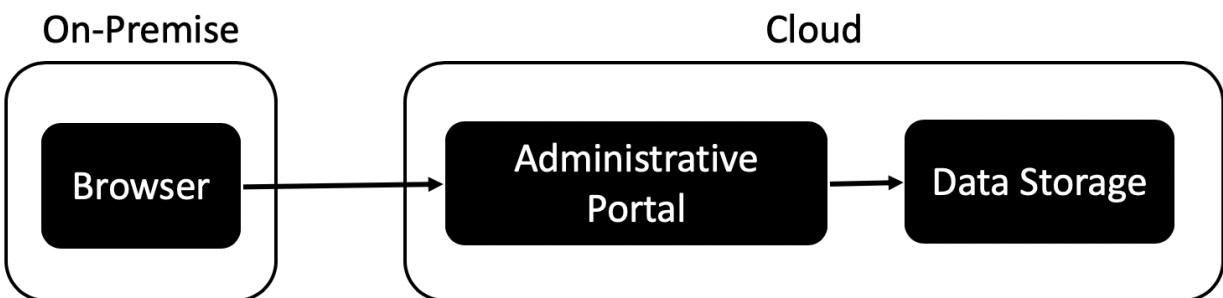
On-Premise Management



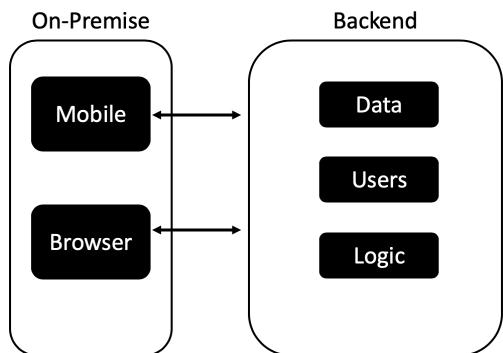
SaaS Provider Management



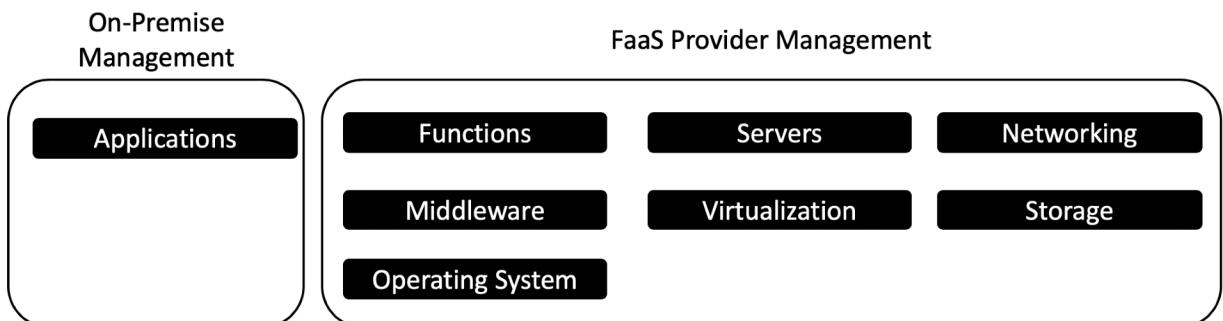
BaaS



MBaaS



FaaS



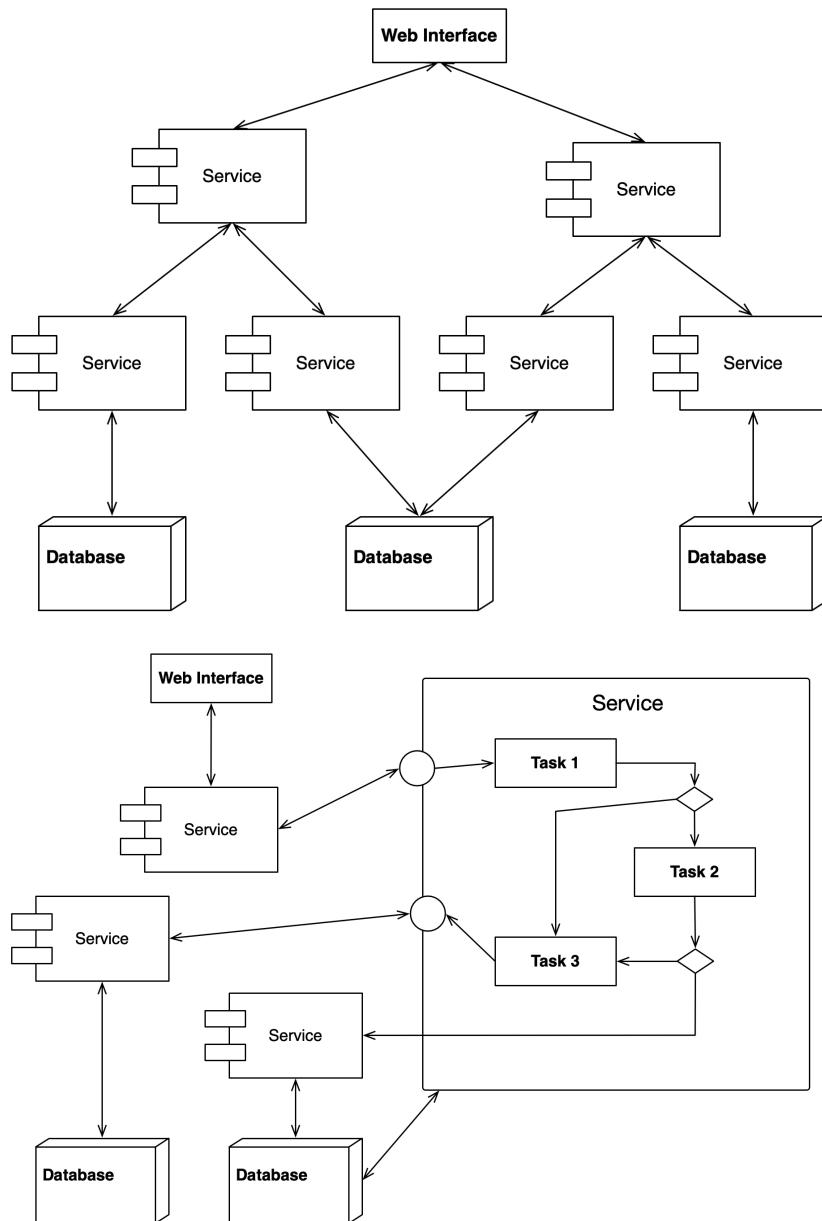
Service-oriented pattern

Сервис-ориентированный архитектурный шаблон, также называемый сервис-ориентированным. Архитектура (SOA) устанавливает интероперабельные службы с помощью методологий и правил. Веб-службы обычно разрабатываются по шаблону SOA. Эти интероперабельные сервисы состоят из организованных развертываемых сервисов со следующими характеристиками:

Работает независимо от других сервисов

Решает конкретную вычислительную задачу

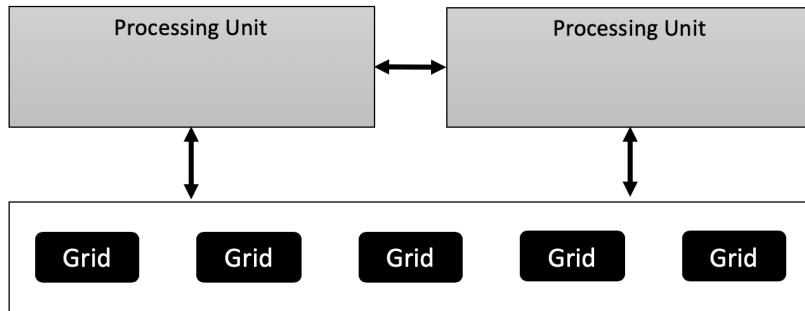
Имеет доступ к другим службам



Space-based pattern

Архитектурный паттерн разработан таким образом, чтобы избежать функционального коллапса при высокой нагрузке. Его цель загрузить и максимизировать масштабирование. Ключом к успеху этого шаблона является использование памяти.

Этот комплексный подход включает в себя следующее:
Устранение ограничений центральной базы данных
Реализация data grid
Репликация data grid в памяти
Сохранение данных приложения в реплицированной памяти



Functional Design Patterns

Understanding the execute around pattern

Шаблон функционального проектирования «выполнение вокруг» используется, когда процессы имеют предварительную и постобработку. Это позволяет нам сосредоточиться на основной функции, а не на обработке, которая происходит до или после. Код пре- и постобработки может существовать один раз, вместо того, чтобы быть частью каждого основного процесса.

```
FunctionalGift myObject = (int number)->System.out.println  
(number + " squared is " + (number*number));
```

Understanding the lambda design pattern

Все про лямбду

Understanding the loan design pattern

Шаблон функционального проектирования кредита можно использовать для создания приложений с учетом ресурсов. С этот шаблон, мы не создаем код, который должен управлять ресурсами, такими как память; скорее, мы хотим контролировать, как используются ресурсы. Ключевое значение имеет обеспечение того, чтобы мусор сборка происходит на наших условиях, а не в ожидании сборки мусора Java по умолчанию системы, чтобы начать действовать.

Understanding the MapReduce design pattern

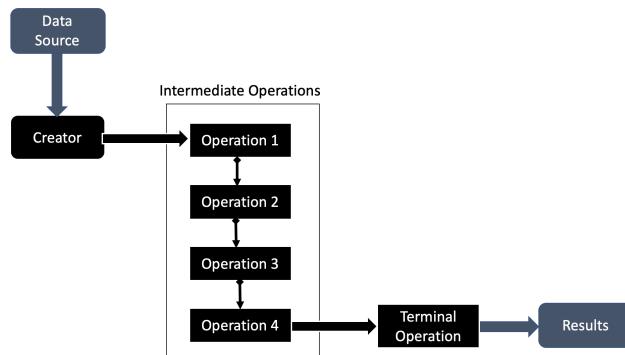
Шаблон функционального проектирования MapReduce используется для крупномасштабного параллельного программирования. Google разработал этот шаблон функционального проектирования, чтобы брать большие задачи и разбивать их на меньшие задачи. Затем эти более мелкие задачи выполняются параллельно и создают консолидированный результат. Цель шаблона функционального проектирования MapReduce заключалась в повышении производительности. при обработке больших наборов данных, также называемых большими данными.

Understanding the memoization design pattern

Шаблон функционального проектирования мемоизации сохраняет результаты ключевых функций, чтобы общая эффективность обработки повышается. Когда есть определенные процессы, которые имеют один и тот же вывод каждый раз, когда они вызываются, это дешевле, из-за времени обработки точки зрения, чтобы кэшировать результаты, а не пересчитывать их каждый раз.

Understanding the streams design pattern

Шаблон функционального проектирования потоков представляет собой конвейерную функциональность, используемую для преобразования данных. В этом смысле преобразование данных отличается от их изменения. Данные преобразуются в потоке, но не мутируют. Чтобы получить преобразованные данные, необходимо сделать вызов к работе терминала. Как только поток закрыт, к нему больше нельзя получить доступ, и преобразованные данные.



Understanding the tail call design pattern

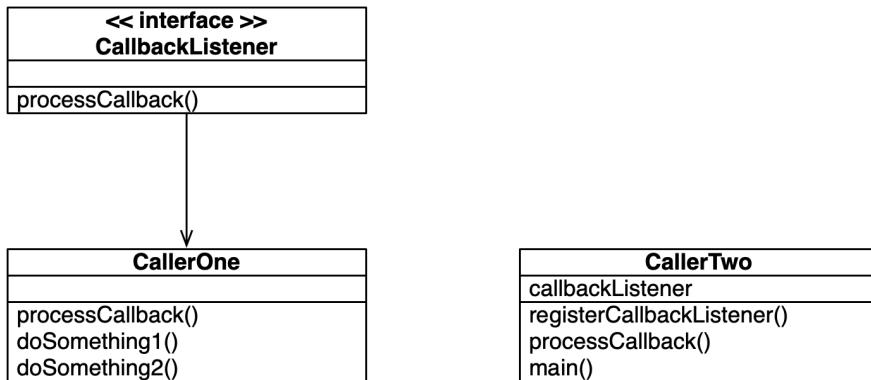
Шаблон функционального проектирования хвостового вызова представляет собой подпрограмму или хвостовую рекурсивную функцию, т. е. выполняется в конце процедуры. Этот шаблон проектирования также называют Tail Call. Оптимизация (TCO). Концепция этого шаблона проста. Хвост вызов последний вызов, выполняемый методом.

Reactive Design Patterns

Reactive Design Patterns

Asynchronous communication design pattern

Просто асинхронность



Autoscaling design pattern

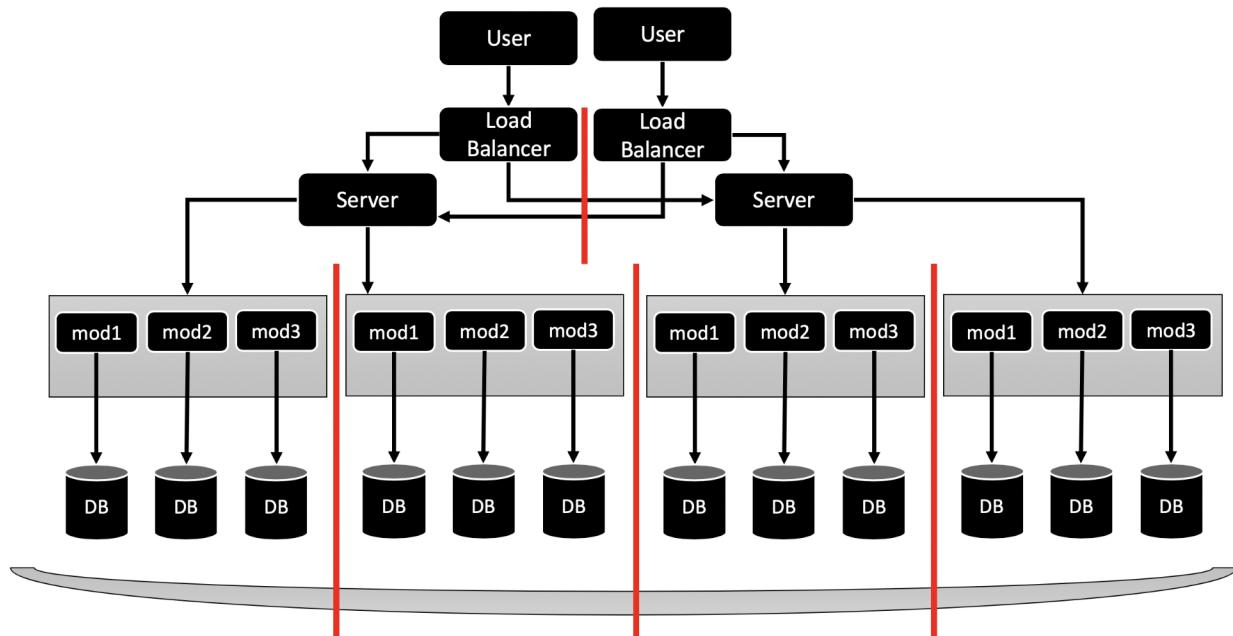
Шаблон проектирования автоматического масштабирования обычно относится к возможности автоматического масштабирования обработки и емкости хранения, как за счет увеличения, так и за счет уменьшения активов. Это один из величайших преимуществ использования облачной инфраструктуры как услуги (IaaS). Концепция просто то, когда вам нужна дополнительная емкость, ваша система автоматически масштабируется.

Bounded queue design pattern

Ограниченные очереди — это очереди с фиксированным числом элементов. Для реализации ограниченного очереди в Java, нам нужно использовать пакет `java.util.concurrent`. Если бы мы использовали `java.util` наши очереди не будут ограничены.

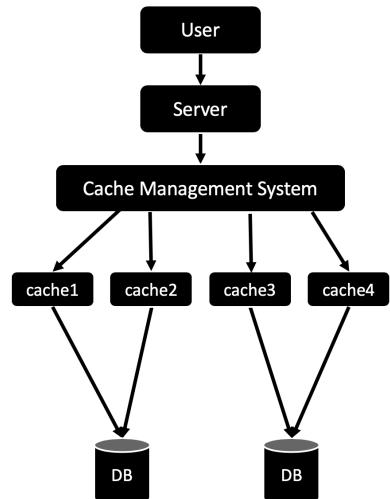
Bulkhead design pattern

Переборки на кораблях помогают изолировать части корабля, чтобы повреждения, такие как затопление, пожар или взрыв воздействует только на одну секцию корабля и не распространяется на другие секции. Переборки также обеспечивают устойчивость всего корабля.



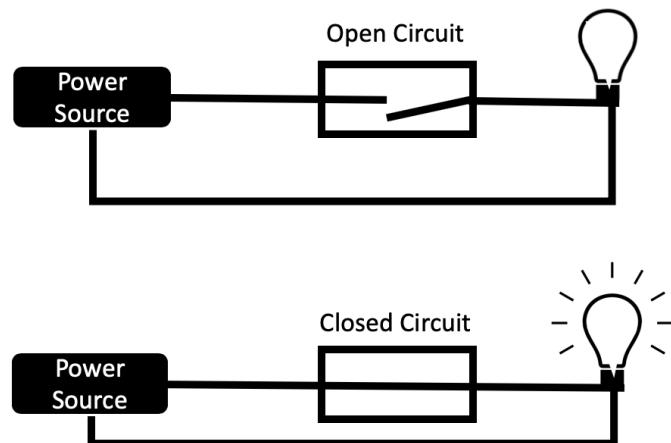
Caching design pattern

Концепция кэширования заключается во временном хранении данных, что ускоряет доступ к ним. Кэширование используется, чтобы помочь сэкономить на использовании ресурсов.



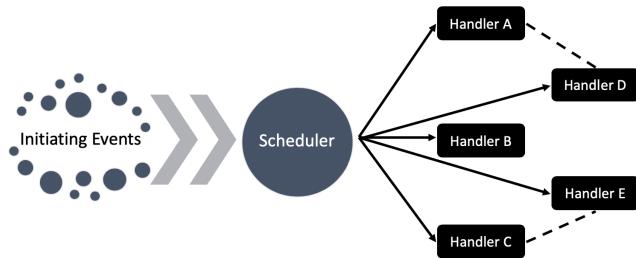
Circuit-breaker design pattern

Схема проектирования автоматического выключателя основана на той же концепции, что и электрическая цепь. Когда цепь разомкнута, поток электричества затруднен и не может протекать между двумя точками — источником питания и лампочкой в нашем примере.



Event-driven communication design pattern

Шаблон проектирования управляемой событиями коммуникации основан на инициировании таких событий, как запрос или запрос узла. Это событие инициирует связь между несколькими компонентами. Коммуникациями между компонентами системы можно управлять как сообщениями.



Fail-fast design pattern

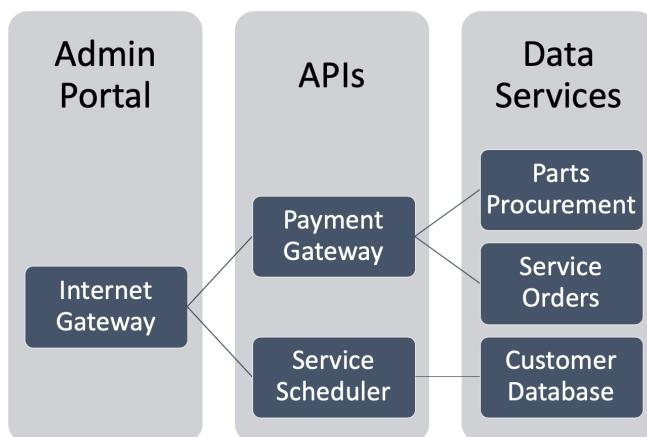
Шаблон отказоустойчивого реактивного проектирования предусматривает, что если что-то пойдет не так, пусть это произойдет как можно быстрее.

Failure-handling design pattern

Когда мы рассматриваем упругую характеристику реактивных систем, систем и компонентов неудачи вызывают крайнюю озабоченность. Важно справляться с этими сбоями таким образом, чтобы мы определить, что предполагает, что мы должны использовать целенаправленный подход к обработке сбоев. Этот Шаблон проектирования состоит из двух основных компонентов:

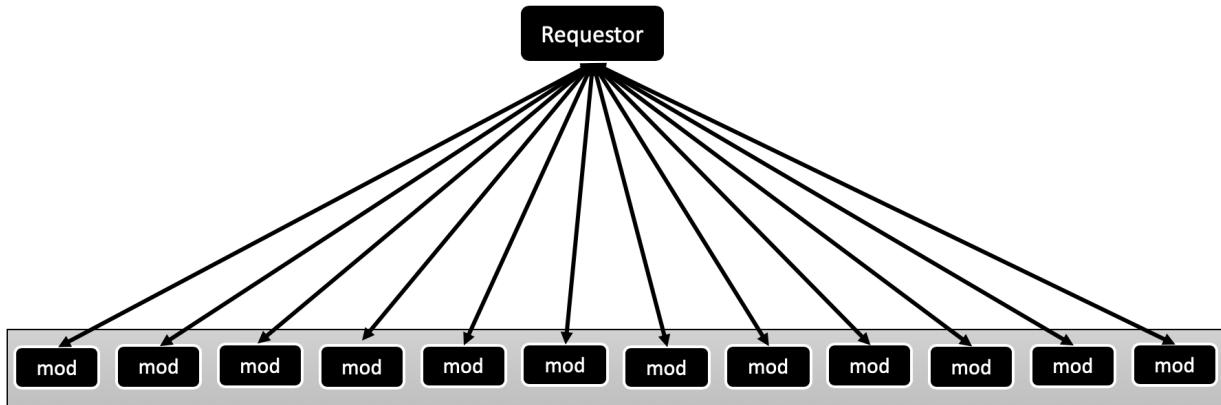
Изоляция отказа

Контролируемый отказ



Fan-out and quickest-reply design pattern

Разветвленный шаблон проектирования и реактивный дизайн с самым быстрым ответом делает упор на быструю обработку. Финансовое программное обеспечение, предоставляющее биржевую информацию в режиме реального или близкого к реальному времени, использует этот шаблон дизайна. Концепция состоит в том, чтобы обеспечить достаточное количество экземпляров обработки, чтобы запросы могли быть обрабатываться без ожидания в очереди.



Idempotency design pattern

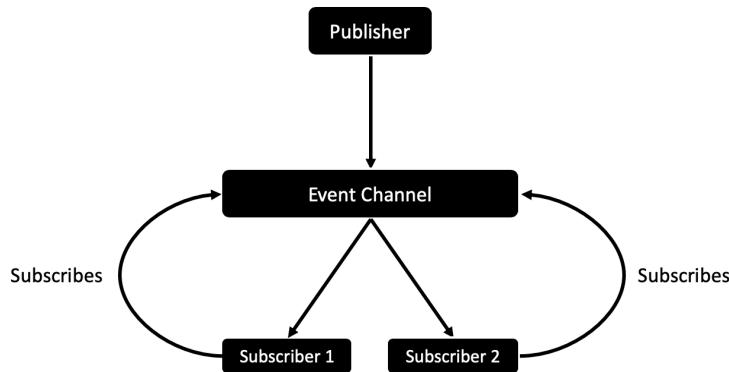
В информатике термин идемпотент означает элемент, значение которого остается неизменным, без изменений после повторного расчета.

Monitoring design pattern

Шаблон проектирования мониторинга не требует пояснений.

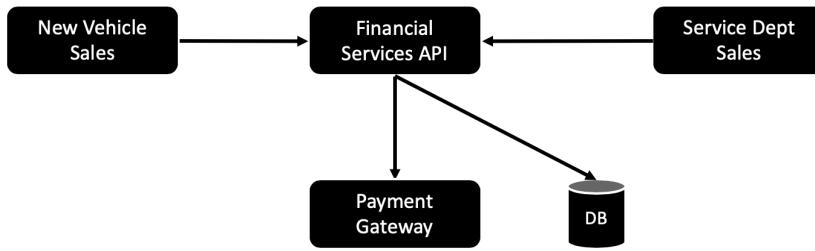
Publisher-subscriber design pattern

Шаблон проектирования издатель-подписчик устанавливает две сущности: издатель, который отправляет сообщения на основе событий и подписчик, который подписывается на эти события.



Self-containment design pattern

Системы, использующие самодостаточный реактивный шаблон проектирования, не зависят от несистемных составные части. Этот шаблон проектирования также можно применять на уровне компонентов системы.



Stateless design pattern

Когда мы реализуем шаблон проектирования без сохранения состояния, мы создаем классы и объекты, которые не сохраняют изменения состояния. В этом подходе каждое использование объекта, например, использует объект в своей органической форме.

