Структурный паттерн – Мост

# Определение

Мост (Bridge) – структурный паттерн проектирования, который позволяет отделить абстракцию от реализации таким образом, чтобы и абстракцию, и реализацию можно было изменять независимо друг от друга.

Даже если мы отделим абстракцию от конкретных реализаций, то у нас все равно все наследуемые классы будут жестко привязаны к интерфейсу, определяемому в базовом абстрактном классе. Для преодоления жестких связей и служит паттерн Мост.

*«Абстракция»* (или *«интерфейс»*) — это образный слой управления чем-либо. Он не делает работу самостоятельно, а делегирует её слою *«реализации»* (иногда называемому *«платформой»*).

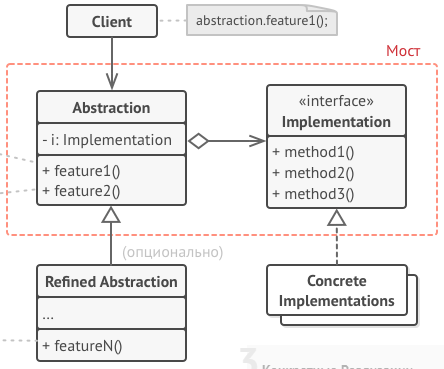
# Назначение и цель

1. Когда надо избежать постоянной привязки абстракции к реализации
2. Когда наряду с реализацией надо изменять и абстракцию независимо друг от друга. То есть изменения в абстракции не должно привести к изменениям в реализации

# Области применения

Если говорить о реальных программах, то абстракцией может выступать графический интерфейс программы, а реализацией — API к которому интерфейс обращается по реакции на действия пользователя.

# UML-диаграмма



**Abstraction**: определяет базовый интерфейс и хранит ссылку на объект **Implementor**. Выполнение операций в **Abstraction** делегируется методам объекта **Implementor**

**RefinedAbstraction**: уточненная абстракция, наследуется от **Abstraction** и расширяет унаследованный интерфейс

**Implementor**: определяет базовый интерфейс для конкретных реализаций. Как правило, **Implementor** определяет только примитивные операции. Более сложные операции, которые базируются на примитивных, определяются в **Abstraction**

**ConcreteImplementors**: конкретные реализации, которые унаследованы от **Implementor**

**Client**: использует объекты **Abstraction**

# Особенности

* Позволяет строить платформо-независимые программы.
* Скрывает лишние или опасные детали реализации от клиентского кода.
* Реализует принцип открытости/закрытости.
* Усложняет код программы за счёт дополнительных классов.

# Список источников

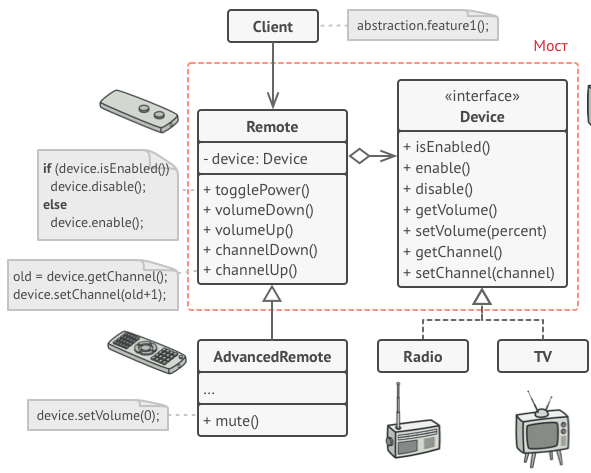
1. <https://metanit.com/sharp/patterns/4.6.php>
2. <https://refactoring.guru/ru/design-patterns/bridge>
3. <https://ru.wikipedia.org/wiki/Мост_(шаблон_проектирования)>

# Пример – Пульты и устройства

Класс пульта имеет ссылку на объект прибора, которым он управляет. Пульты работают с приборами через общий интерфейс. Это даёт возможность связать пульты с различными приборами.

Сами пульты можно развивать независимо от приборов. Для этого достаточно создать новый подкласс абстракции. Вы можете создать простой как простой пульт с двумя кнопками, так и более сложный пульт с тач-интерфейсом.

Клиентскому коду остаётся выбрать версию абстракции и реализации, с которым он хочет работать.



Код примера взят с <https://refactoring.guru/ru/design-patterns/bridge>

// Класс пультов имеет ссылку на устройство, которым

// управляет. Методы этого класса делегируют работу методам

// связанного устройства.

**class** **Remote** **is**

**protected** **field** device: Device

**constructor** Remote(device: Device) **is**

**this**.device = device

**method** togglePower() **is**

**if** (device.isEnabled()) **then**

device.disable()

**else**

device.enable()

**method** volumeDown() **is**

device.setVolume(device.getVolume() - 10)

**method** volumeUp() **is**

device.setVolume(device.getVolume() + 10)

**method** channelDown() **is**

device.setChannel(device.getChannel() - 1)

**method** channelUp() **is**

device.setChannel(device.getChannel() + 1)

// Вы можете расширять класс пультов не трогая

// код устройств.

**class** **AdvancedRemote** **extends** Remote **is**

**method** mute() **is**

device.setVolume(0)

// Все устройства имеют общий интерфейс. Поэтому с ними

// может работать любой пульт.

**interface** **Device** **is**

**method** isEnabled()

**method** enable()

**method** disable()

**method** getVolume()

**method** setVolume(percent)

**method** getChannel()

**method** setChannel(channel)

// Но каждое устройство имеет особую реализацию.

**class** **Tv** **implements** Device **is**

*// ...*

**class** **Radio** **implements** Device **is**

*// ...*

// Где-то в клиентском коде.

tv = **new** Tv()

remote = **new** Remote(tv)

remote.power()

radio = **new** Radio()

remote = **new** AdvancedRemote(radio)