Лабораторная работа №5: мост  
  
Суть паттерна.

**Мост** — это структурный паттерн проектирования, который разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга.

Приведем пример из жизни, где для реализации подходит этот паттерн.  
У нас есть класс «Геометрические фигуры», он имеет подклассы «Треугольник» и «Овал», мы хотим классифицировать их по цвету и по форме: зеленый треугольник, желтый овал и т.д.

**ЦВЕТ**

**ЦВЕТ**

Фигура

Фигура

Желтый

овал

Зеленый

овал

Зеленый

треугольник

Желтый

треугольник

Но тогда, если мы захотим добавить одну фигуру, то количество наших подклассов увеличится на 2. А если потом мы захотим добавить цвет, то необходимо будет ввести 3 класса на каждый вид фигуры. Следовательно, при добавлении новых фигур и цветов, количество комбинаций будет расти в геометрической прогрессии.

Для решения проблемы мы перестанем пытаться связать две независимые плоскости – по форме и цвету. Вместо этого мы выделаем одну из этих плоскостей в отдельную иерархию и будем ссылаться на объект внутри этой иерархии, вместо того чтобы хранить внутри одного класса его состояние и подведение.

**СОДЕРЖИТ**

Цвет

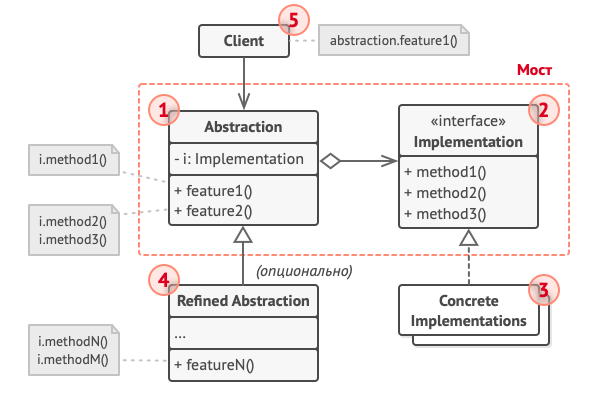
Фигура

Зеленый

Желтый

Овал

Треугольник



1. **Абстракция** содержит управляющую логику. Код абстракции делегирует реальную работу связанному объекту реализации.
2. **Реализация** задаёт общий интерфейс для всех реализаций. Все методы, которые здесь описаны, будут доступны из класса абстракции и его подклассов.

Интерфейсы абстракции и реализации могут как совпадать, так и быть совершенно разными. Но обычно в реализации живут базовые операции, на которых строятся сложные операции абстракции.

1. **Конкретные реализации** содержат платформо-зависимый код.
2. **Расширенные абстракции** содержат различные вариации управляющей логики. Как и родитель, работает с реализациями только через общий интерфейс реализации.
3. **Клиент** работает только с объектами абстракции. Не считая начального связывания абстракции с одной из реализаций, клиентский код не имеет прямого доступа к объектам реализации.

Обоснование выбора паттерна

В моем коде будем разбираться с пультами и приборами. Т.е. допустим есть монолитный код пультов и устройством, где используется наследование. Если мы захоти создать или изменить пульт, то придется подстраивать и приборы под данные изменения, из-за чего могут возникнуть новые комбинации пультов и устройств.

Разделю данный код на две части, где устройства выступают в качестве реализации, а пульты, которые ими управляют, в качестве абстракции. Благодаря тому, что класс пульта будет иметь ссылку на объект прибора, которым он управляет, плюс ко всему пульты работают с приборами через общий интерфейс, мы сможет связать пульты с различными приборами и у нас появится возможность развивать устройства и пульты независимо друг от друга.

ПОЧЕМУ НЕ ДРУГИЕ ПАТТЕРНЫ:

**1. Адаптер** — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе. У нас цель не объединение объектов и их интерфейсов, а избежание роста количества комбинаций пультов и устройств в геометрической прогрессии при добавлении новых параметров и элементов.

**2. Компоновщик** — это структурный паттерн проектирования, который позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект. Наша задача не предполагает древовидную структуру в принципе, нет наследования от чего-то одного, что может быть верхушкой иерархии.

**3. Декоратор** — это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки». Допустим у нас есть необходимость в динамическом добавлении новой функциональности наших пультов…или устройств. Но данный паттерн не предполагает возможность независимого изменения абстракции и реализации.

**4. Фасад** — это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку. У нас нет необходимости в упрощенном интерфейсе, наша главная задача – возможность независимого изменения абстракции и реализации.

**5. Легковес** — это структурный паттерн проектирования, который позволяет вместить большее количество объектов в отведённую оперативную память. Легковес экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте. У нас нет необходимости в экономии памяти и опять-таки наша главная задача – возможность независимого изменения абстракции и реализации.

**6. Заместитель** — это структурный паттерн проектирования, который позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу. У нас нет необходимости в создании в создании таковых объектов-заменителей с общим интерфейсом.

Словесная реализация кода

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 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, и связать их между собой.