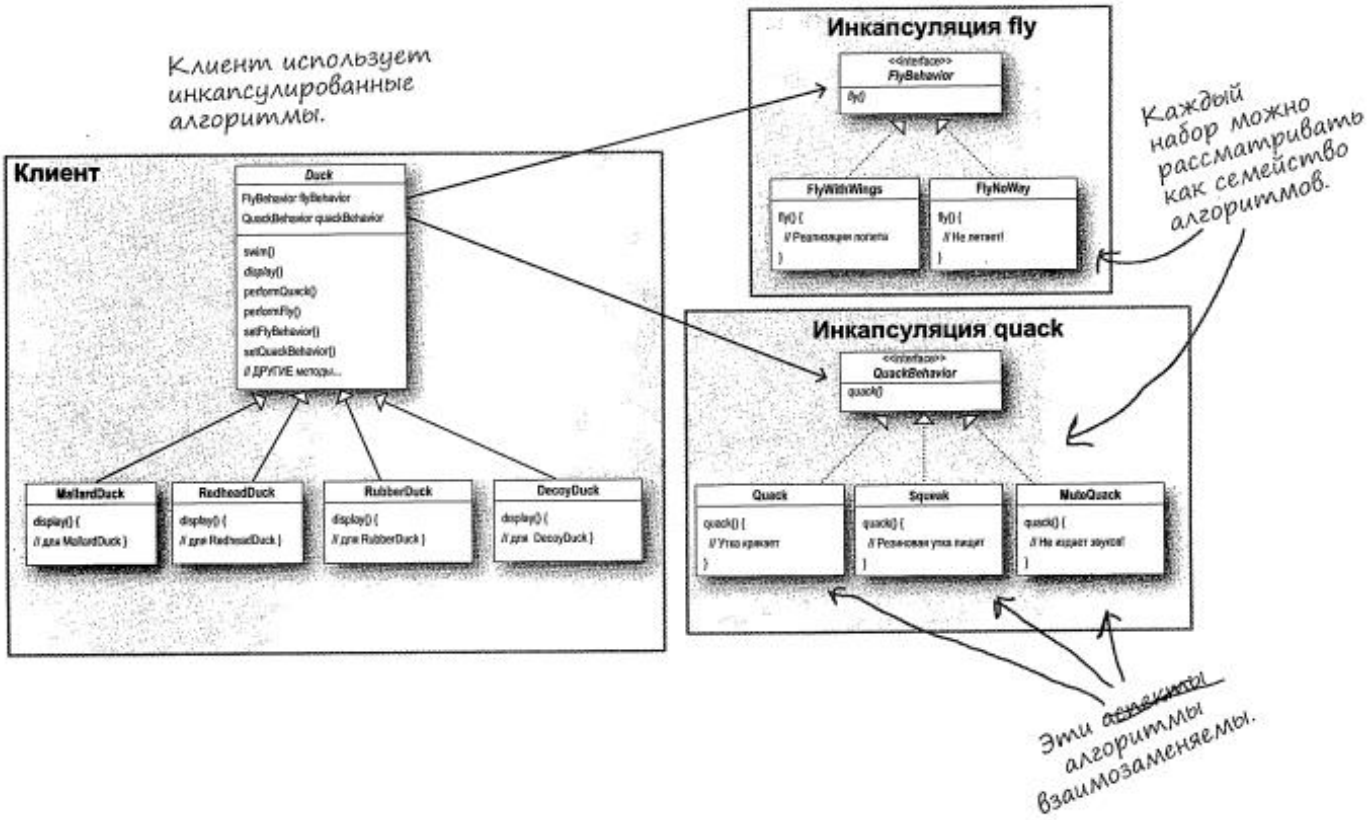
Паттерн Стратегия определяет семейство алгоритмов, инкапсулирует каждый из них и обеспечивает их взаимозаменяемость. Он позволяет модифицировать алгоритмы независимо от их использования на стороне клиента.

Принципы

Инкапсулируйте то, что изменяется.

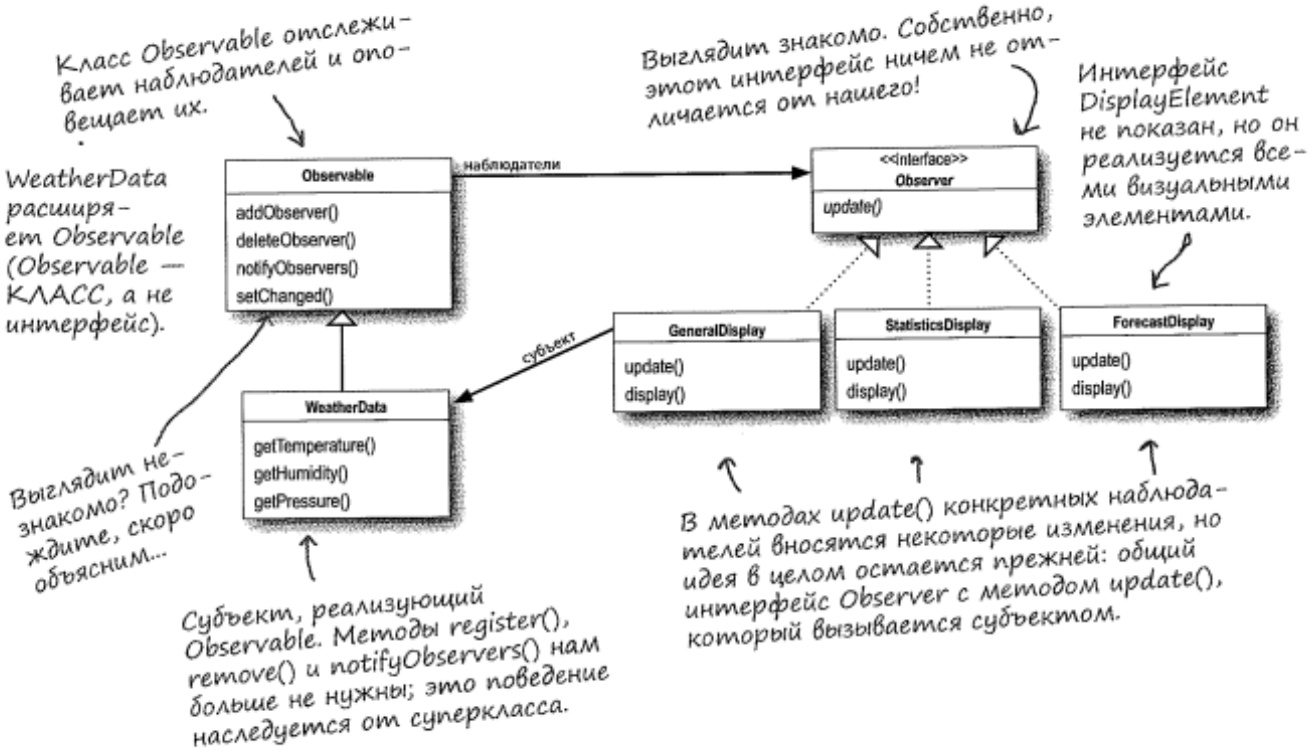
Отдавайте предпочтение композиции перед наследованием.

Программируйте на уровне интерфейсов, а не реализации.



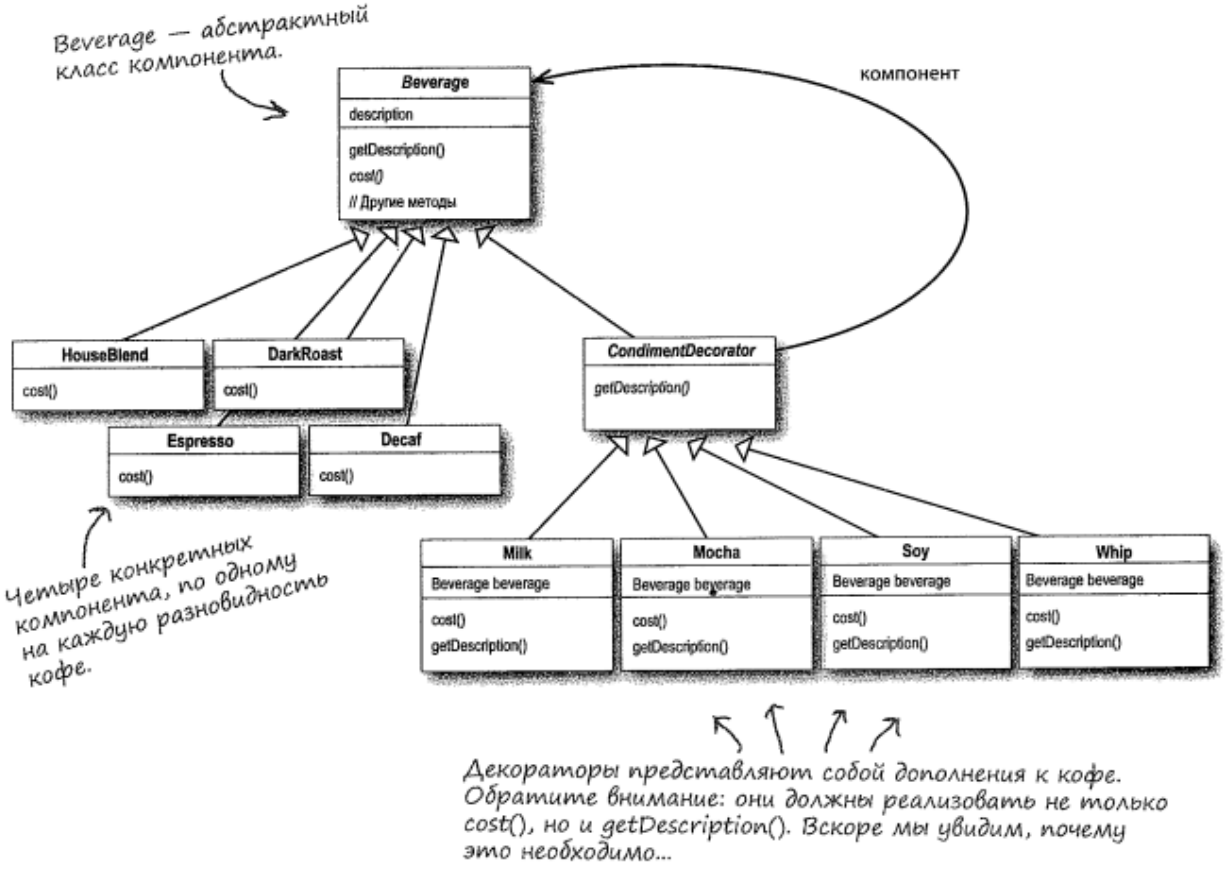
Паттерн Наблюдатель определяет отношение «один-ко-многим» между объектами таким образом, что при изменении состояния одного объекта происходит автоматическое оповещение и обновление всех зависимых объектов.

Стремитесь к слабой связанности взаимодействующих объектов.



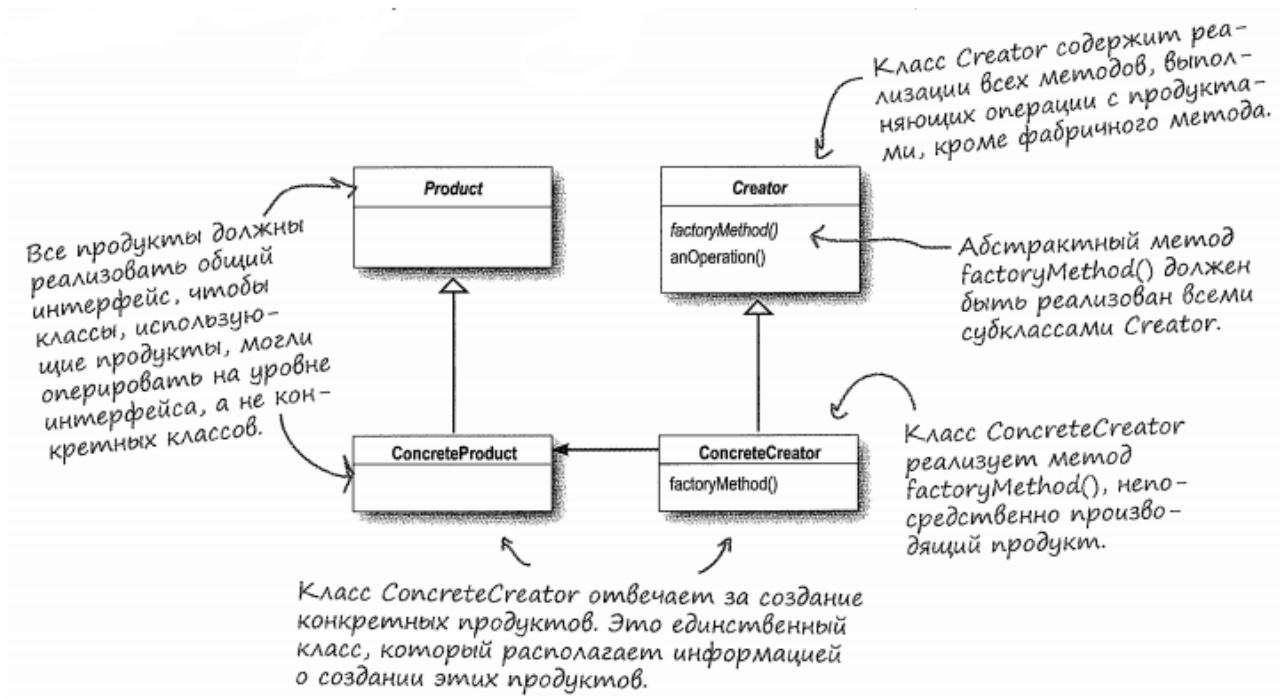
Паттерн Декоратор динамически наделяет объект новыми возможностями и является гибкой альтернативой субклассированию в области расширения функциональности.

Классы должны быть открыты для расширения, но закрыты для изменения.

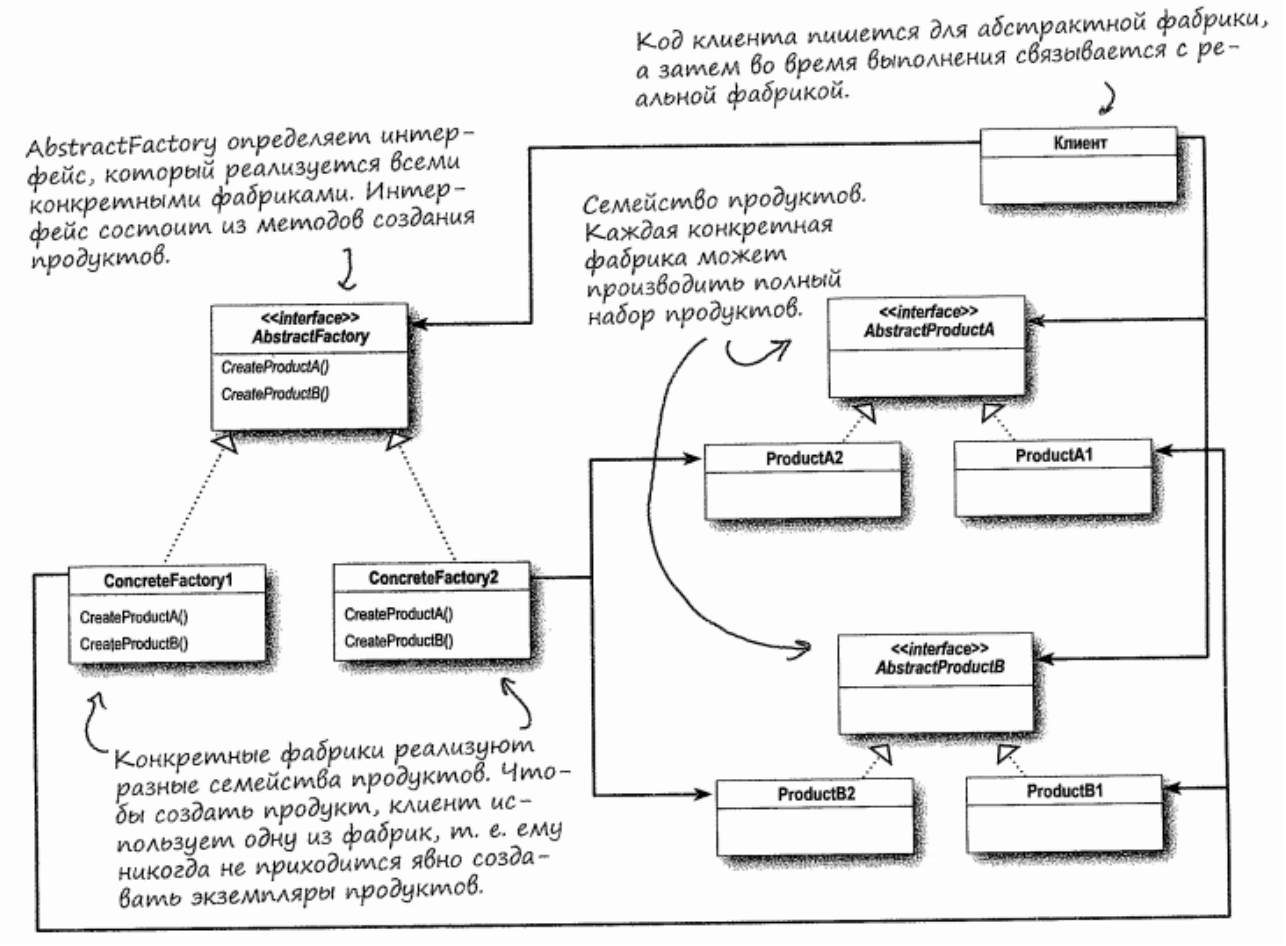


Паттерн Фабричный Метод определяет интерфейс создания объекта, но позволяет субклассам выбрать класс создаваемого экземпляра. Таким образом, Фабричный Метод делегирует операцию создания экземпляра субклассам.

Код должен зависеть от абстракций, а не от конкретных классов.



Паттерн Абстрактная Фабрика предоставляет интерфейс создания семейств взаимосвязанных или взаимозависимых объектов без указания их конкретных классов.



Паттерн Одиночка гарантирует, что класс имеет только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру.

Классическая реализация паттерна Одиночка

Класс MyClass переименован в Singleton.

Статическая переменная для хранения единственного экземпляра.

```
public class Singleton {
    private static Singleton uniqueInstance;

    // other useful instance variables here

    private Singleton() {}

    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }

    // Другие методы
}
```

Приватный конструктор; только Singleton может создавать экземпляры этого класса!

Метод getInstance() создает и возвращает экземпляр.

Как и всякий другой класс, Singleton содержит другие переменные и методы экземпляров.

Будьте осторожны!

Если вы бегло просматриваете книгу, не торопитесь использовать этот код. Как будет показано позднее в этой главе, он нуждается в доработке.

Код под увеличительным стеклом

uniqueInstance содержит ЕДИНСТВЕННЫЙ экземпляр; не забудьте, что это статическая переменная.

Если uniqueInstance содержит null, значит, экземпляр еще не создан...

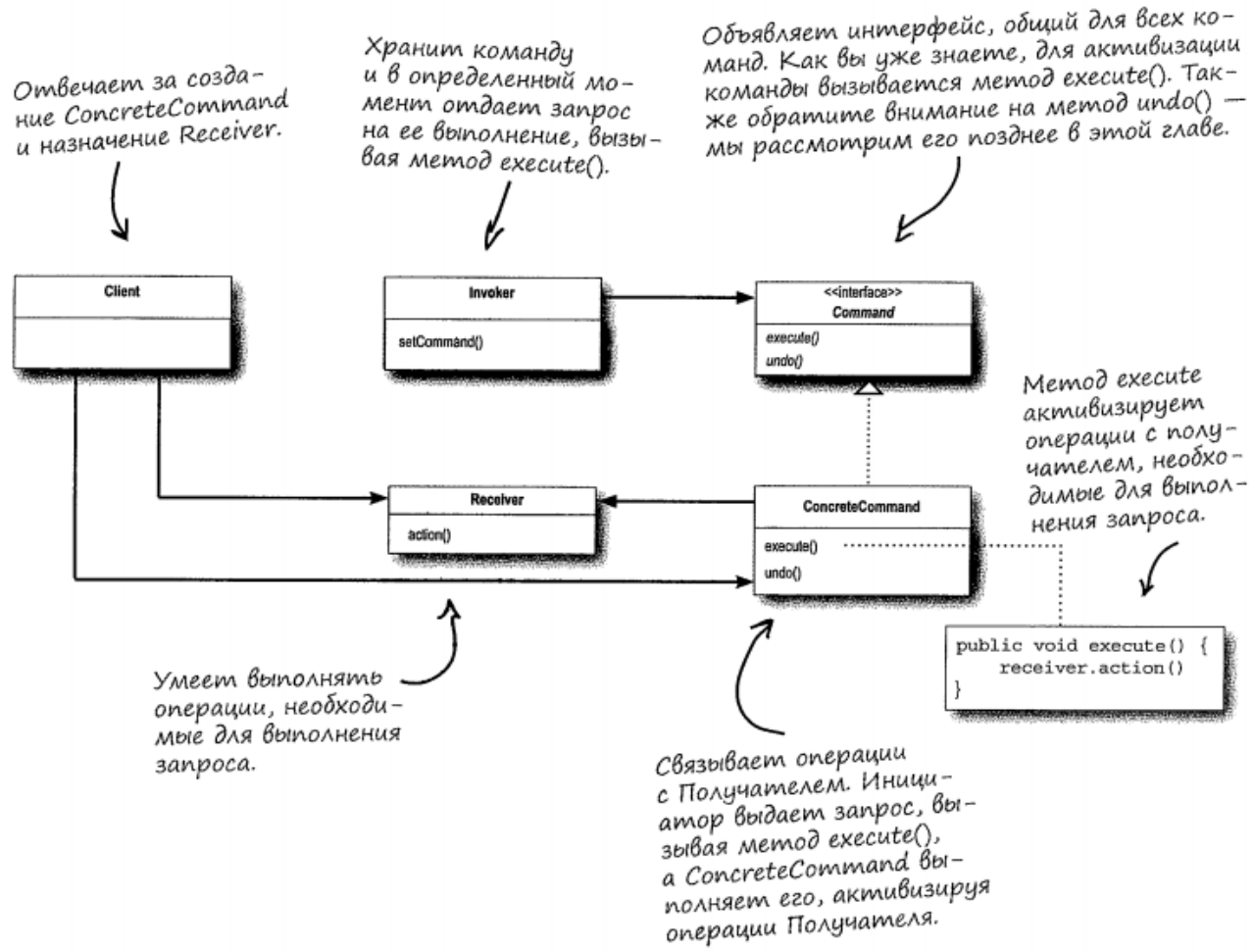
...тогда мы создаем экземпляр Singleton приватным конструктором и присваиваем его uniqueInstance.

```
if (uniqueInstance == null) {
    uniqueInstance = new MyClass();
}
return uniqueInstance;
```

Если uniqueInstance уже содержит значение, сразу переходим к команде return.

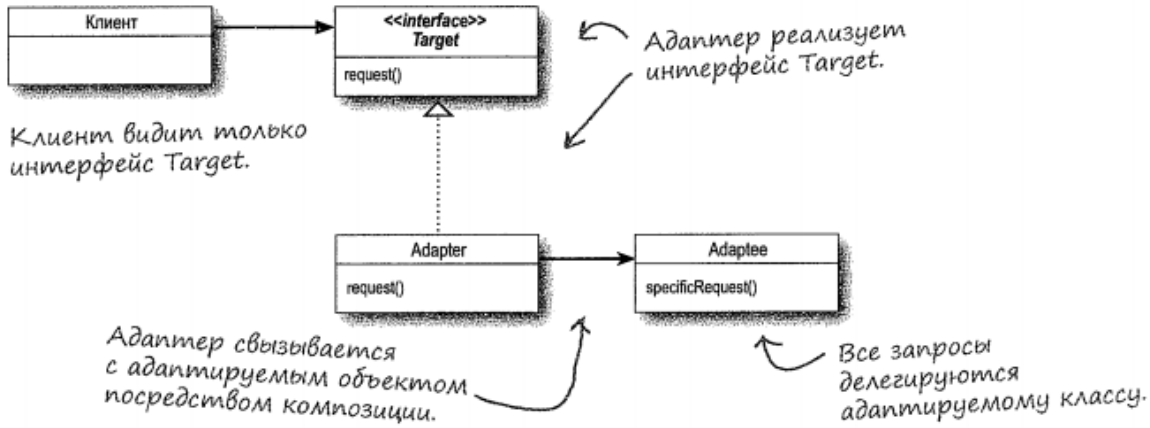
К моменту выполнения этой команды экземпляр уже создан — возвращаем его.

Паттерн Команда инкапсулирует запрос в виде объекта, делая возможной параметризацию клиентских объектов с другими запросами, организацию очереди или регистрацию запросов, а также поддержку отмены операций.

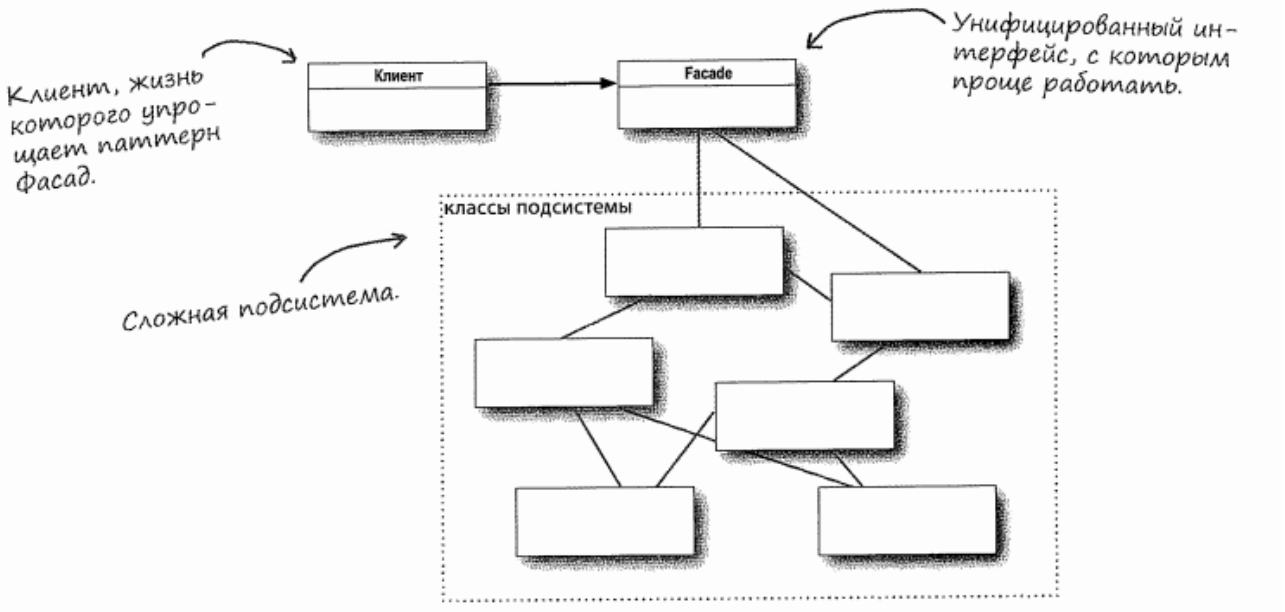


Паттерн Адаптер преобразует интерфейс класса к другому интерфейсу, на который рассчитан клиент. Адаптер обеспечивает совместную работу классов, невозможную в обычных условиях из-за несовместимости интерфейсов.

Взаимодействуйте только с «друзьями».

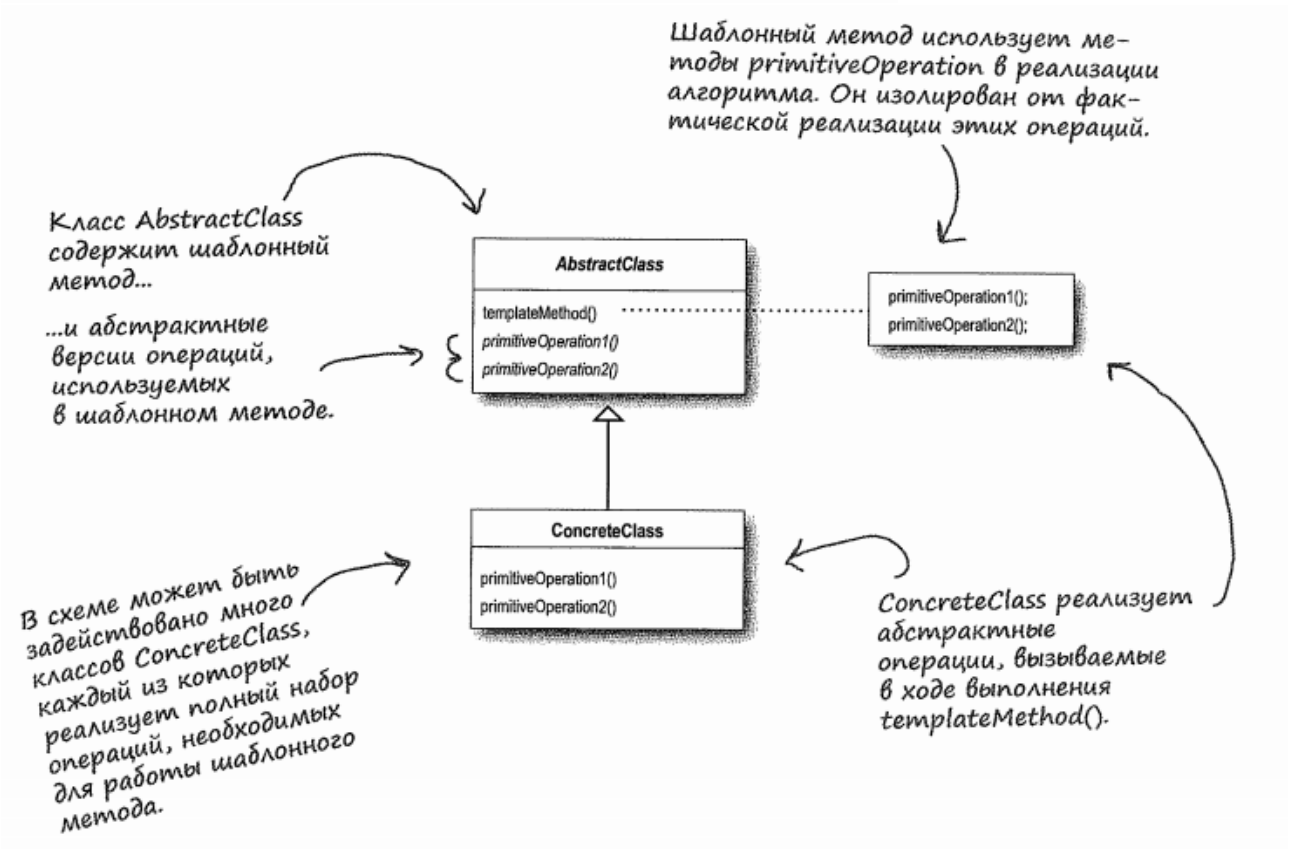


Паттерн Фасад предоставляет унифицированный интерфейс к группе интерфейсов подсистемы. Фасад определяет высокоуровневый интерфейс, упрощающий работу с подсистемой.



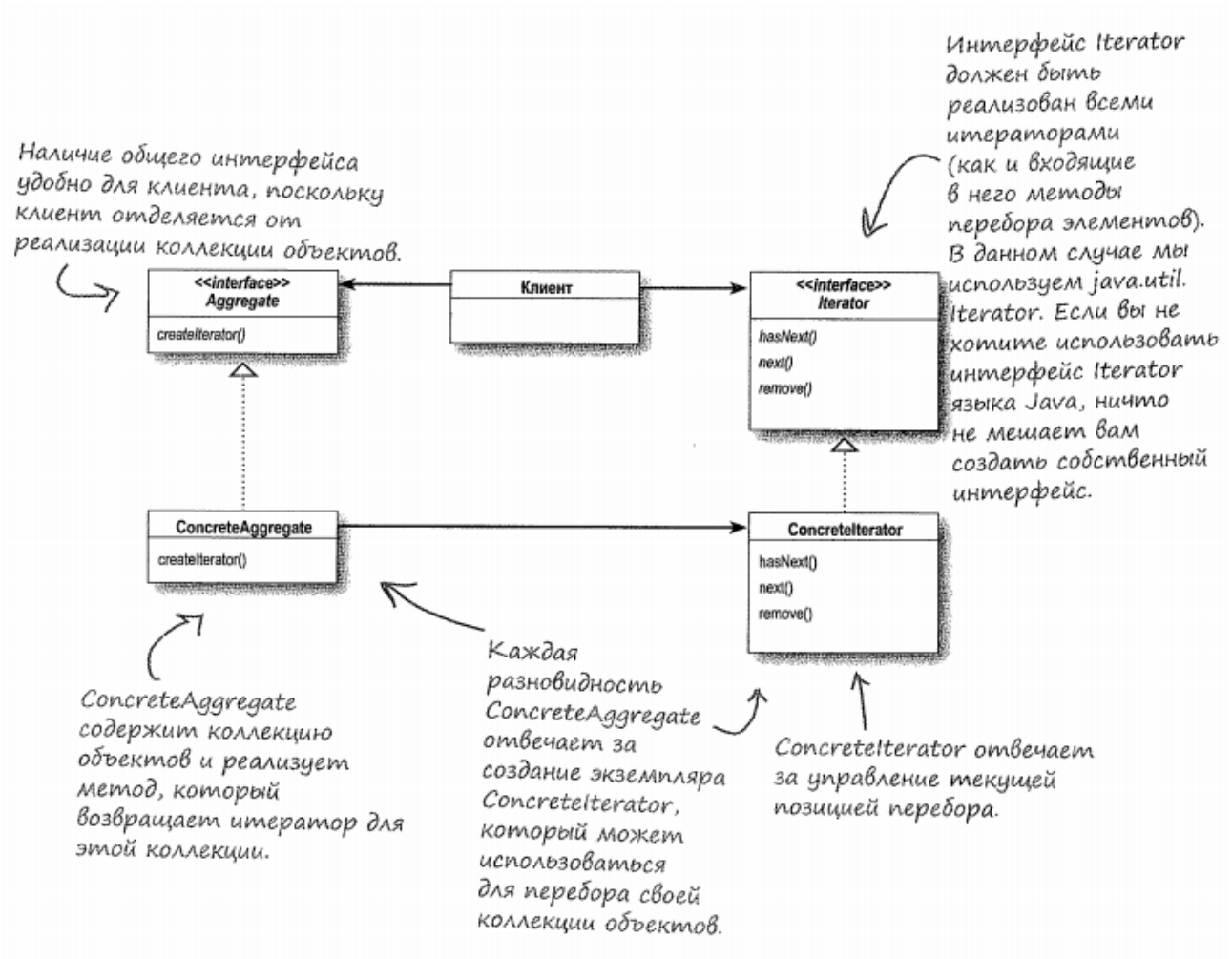
Паттерн Шаблонный Метод задает «скелет» алгоритма в методе, оставляя определение реализации некоторых шагов субклассам. Субклассы могут переопределять некоторые части алгоритма без изменения его структуры.

Не вызывайте нас — мы вас сами вызовем.

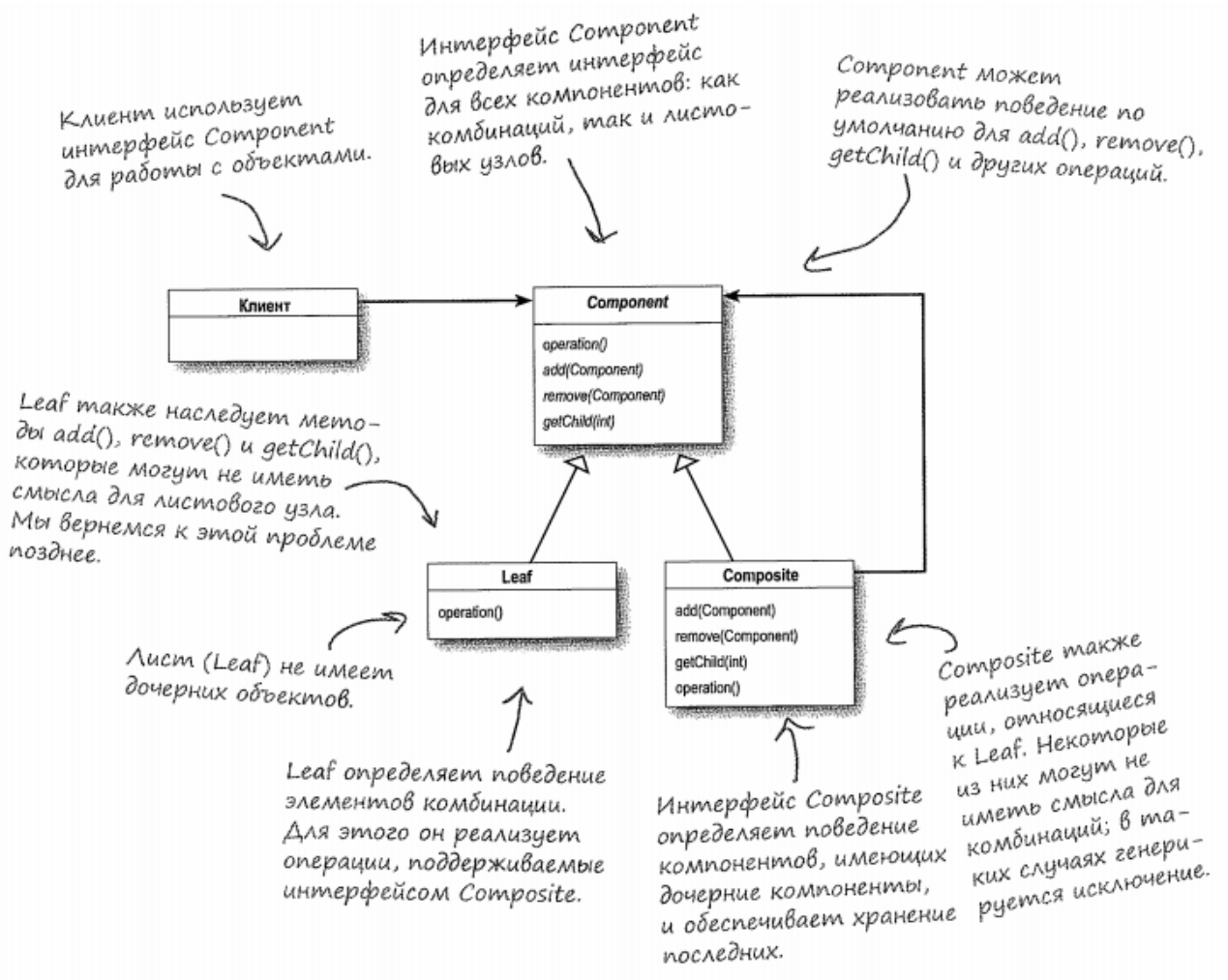


Паттерн Итератор предоставляет механизм последовательного перебора элементов коллекции без раскрытия ее внутреннего представления.

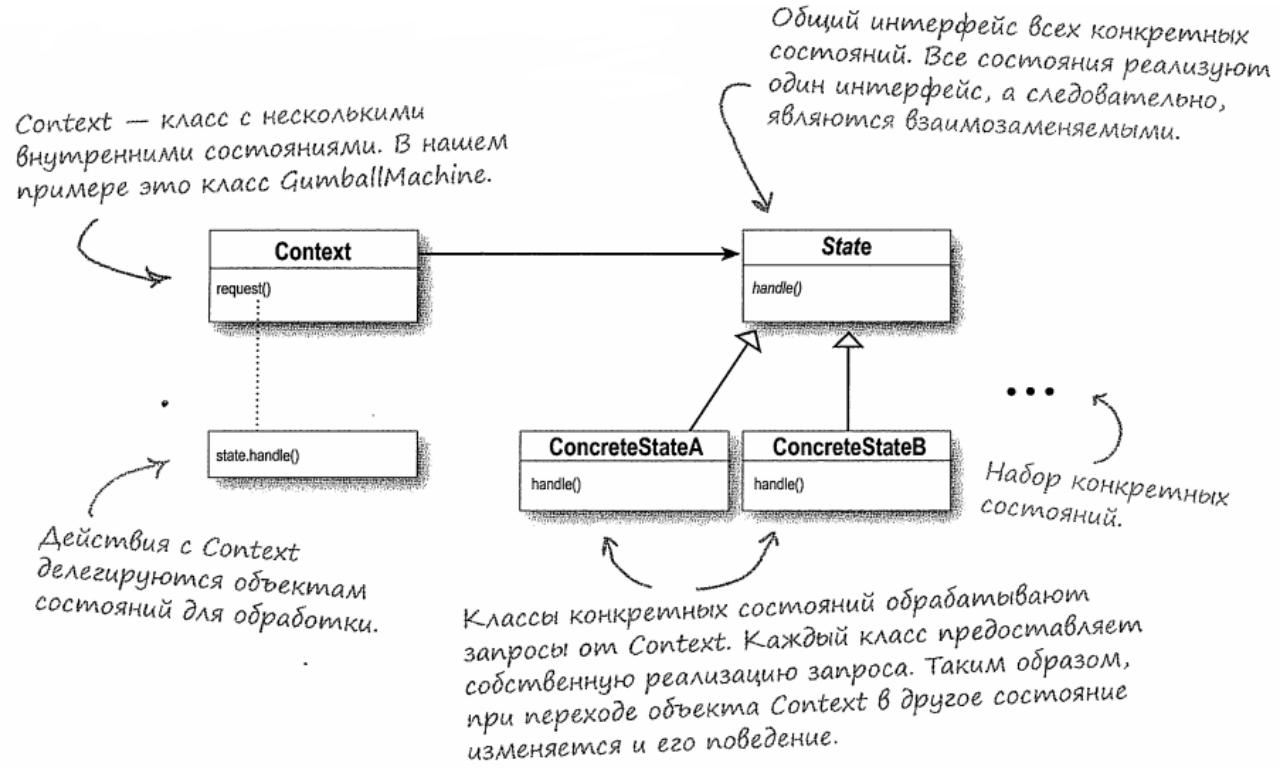
Класс должен иметь только одну причину для изменений.



Паттерн Компоновщик объединяет объекты в древовидные структуры для представления иерархий «часть/целое». Компоновщик позволяет клиенту выполнять однородные операции с отдельными объектами и их совокупностями.



Паттерн Состояние управляет изменением поведения объекта при изменении его внутреннего состояния. Внешне это выглядит так, словно объект меняет свой класс.

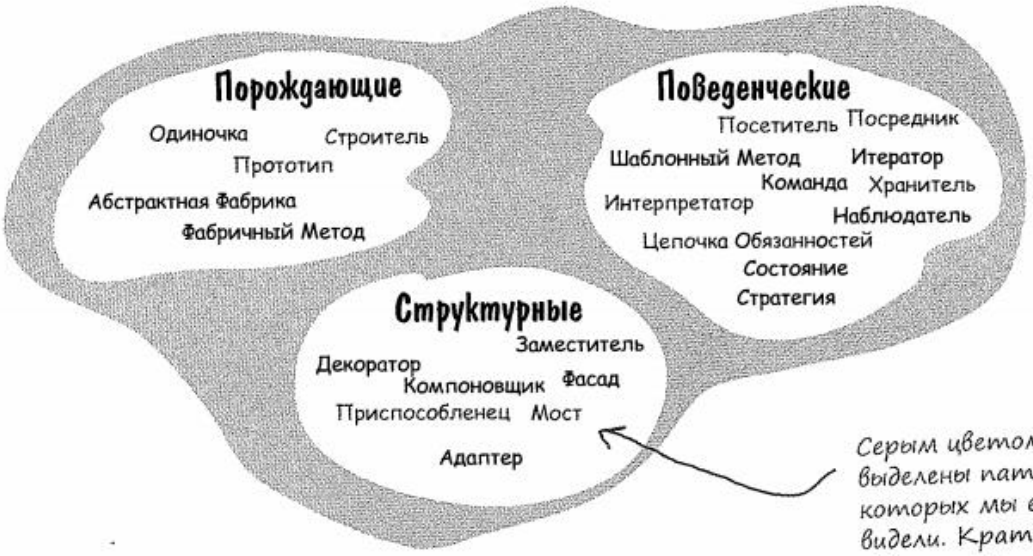


Паттерн – решение задачи в контексте.

Антипаттерн описывает ПЛОХОЕ решение задачи.

Порождающие паттерны связаны с созданием экземпляров объектов; все они обеспечивают средства логической изоляции клиента от создаваемых объектов.

Паттерны, относящиеся к **поведенческой категории,** относятся к взаимодействиям и распределению обязанностей между классами и объектами.



Серым цветом выделены паттерны, которых мы еще не видели. Краткий обзор этих паттернов приведен в приложении.

Структурные паттерны объединяют классы или объекты в более крупные структуры.

Паттерны также часто классифицируются по другому атрибуту: в зависимости от того, относится паттерн к классам или объектам.

Паттерны классов описывают определение отношений между классами посредством наследования. Отношения в паттернах классов определяются на стадии компиляции.

Паттерны объектов описывают отношения между объектами, прежде всего относящиеся к композиции. Отношения в паттернах объектов обычно определяются на стадии выполнения, а следовательно, обладают большей динамичностью и гибкостью.

Обратите внимание: паттернов объектов намного больше, чем паттернов классов!