Паттерны проектирования

(Design patterns)

Игорь Сухинский,
Екатерина Неповинных
ВМИ-311

Паттерн проектирования

Паттерн проектирования (design pattern) — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно паттерн не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях.

Зачем их использовать

- Каждый паттерн описывает решение целого класса проблем
- Каждый паттерн имеет известное имя
  - облегчается взаимодействие между разработчиками
  - правильно сформулированный паттерн проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова
- Паттерны проектирования не зависят от языка программирования, в отличие от идиом

История развития

- в 1970-е годы архитектор Кристофер Александр составил набор шаблонов проектирования;
- в 1987 К.Бэк и В.Каннингем разработали шаблоны применительно к разработке ПО для разработки графических оболочек на языке Smalltalk;
- в 1988 году Эрих Гамма начал писать докторскую диссертацию об общей переносимости этой методики на разработку программ;
- в 1994 году Эрих Гамма в сотрудничестве с Р.Хелмом, Р.Джонсоном и Д.Влиссидсом(Gang of Four,GoF, банда четырех) публикует книгу Design Patterns — Elements of Reusable Object-Oriented Software, где описаны 23 шаблона проектирования.

Классификация

1. Порождающие паттерны (Creational)
2. Структурные паттерны (Structual)
3. Поведенческие паттерны (Behavioral)

Порождающие паттерны

(Creational)

Порождающие паттерны (Creational)

- абстрагируют процесс инстанцирования
- позволяют сделать систему независимой от способа создания, композиции и представления объектов
- Примеры паттернов
  - Абстрактная фабрика (Abstract Factory)
  - Одиночка (Singleton)

Задача 1

Спроектировать игру про подземелье с монстрами двух видов, которые могут атаковать противника с дальнего и ближнего расстояния.

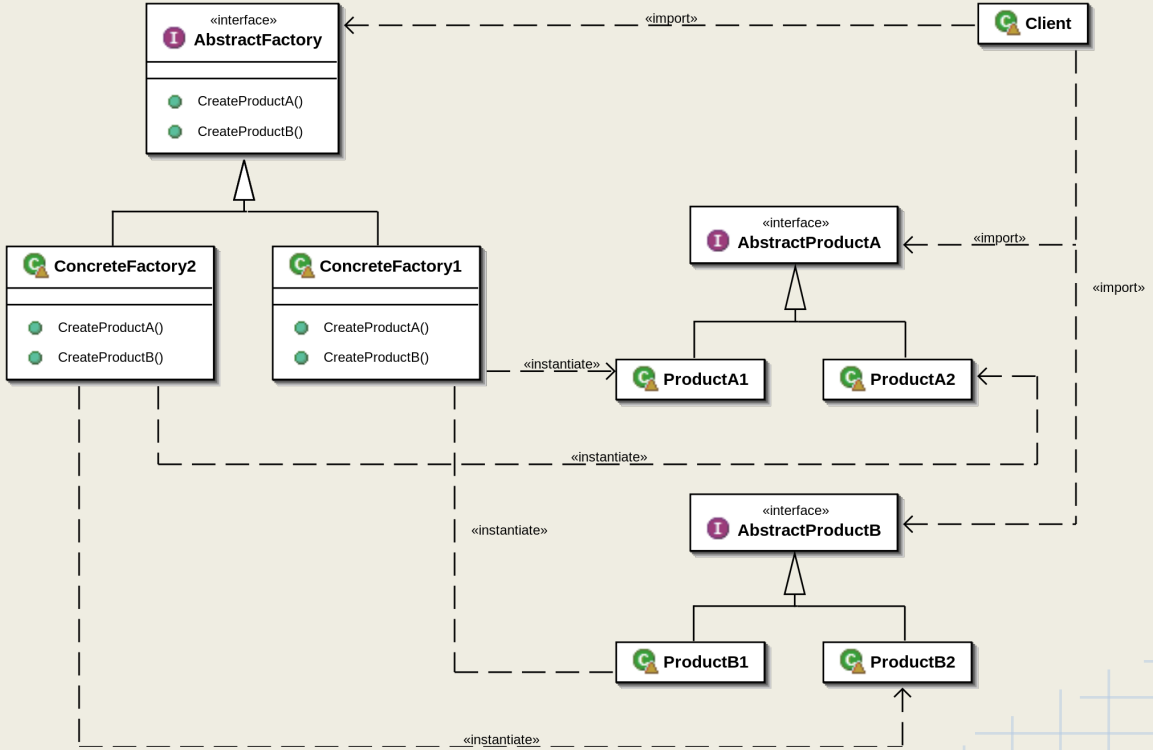
Абстрактная фабрика (Abstract factory)

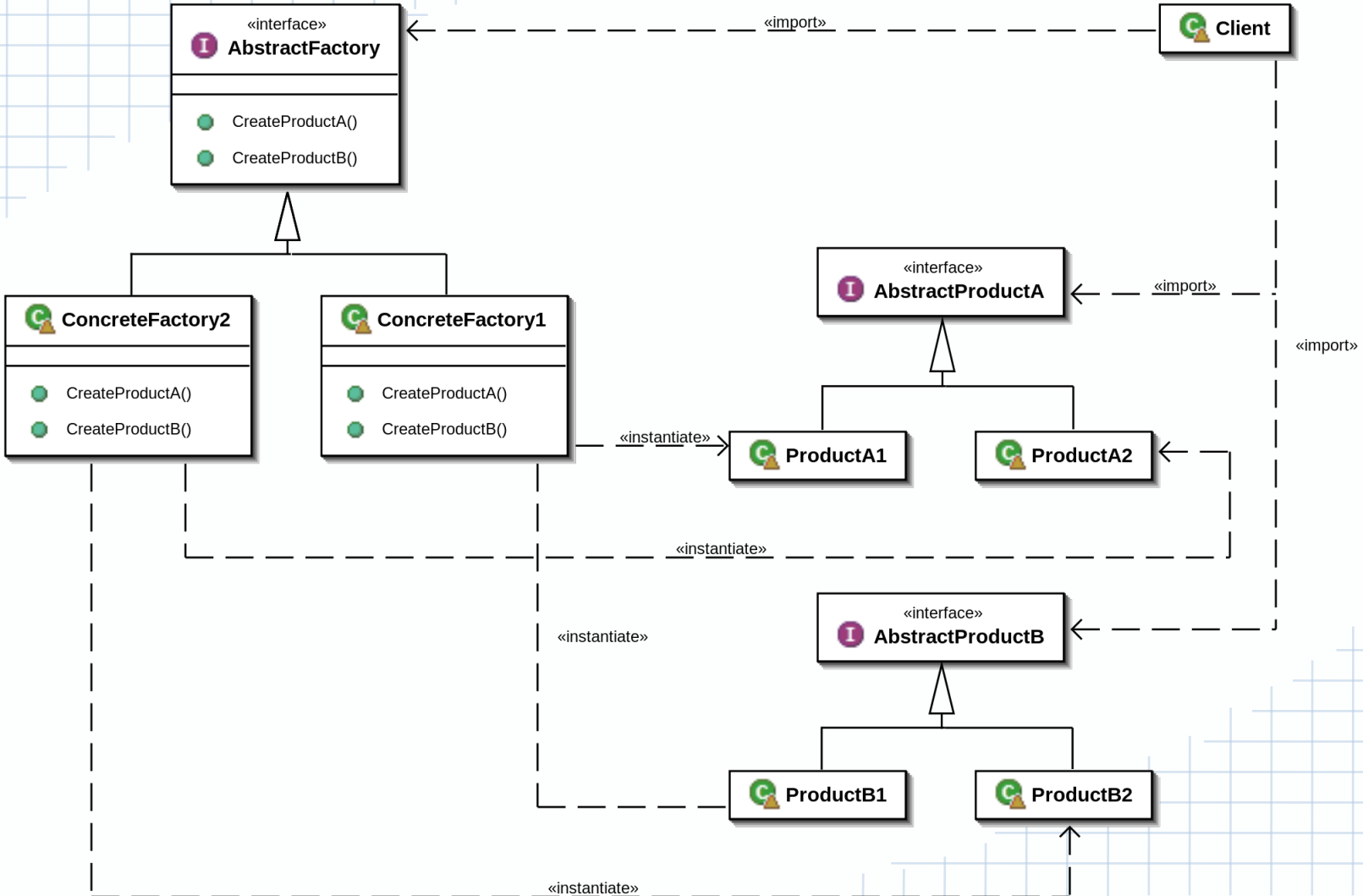
- Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов;
- Позволяет изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы.

VS Наследование

- Нет возможности создавать произвольные объекты в рантайме
- Конструктор объекта не позволяет скрыть объект за интерфейсом
- Следовательно, затрудняется расширение и тестирование

Абстрактная фабрика (Структура)




«interface»
AbstractFactory
CreateProductA()
CreateProductB()
«import»
Client
«import»
ConcreteFactory2
CreateProductA()
CreateProductB()
ConcreteFactory1
CreateProductA()
CreateProductB()
«interface»
AbstractProductA
«import»
«import»
«instantiate»
ProductA1
ProductA2
«instantiate»
«instantiate»
«interface»
AbstractProductB
ProductB1
ProductB2
«instantiate»

Абстрактная фабрика

Применяется, когда:

- программа должна быть независимой от процесса и типов создаваемых новых объектов;
- необходимо создать семейства или группы взаимосвязанных объектов исключая возможность одновременного использования объектов из разных этих наборов в одном контексте.

Задача 2

Задача: Имеется веб-приложение. Необходимо разработать систему логирования, к которой возможен доступ из любой части программы, при этом лог должен быть единственным.

Одиночка (Singleton)

Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа

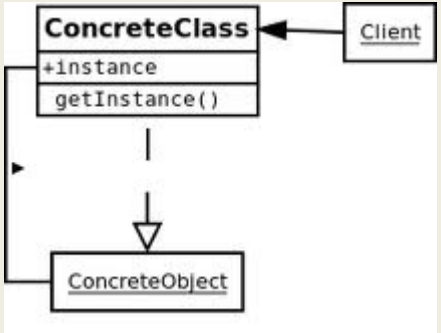
Применяется, когда:

- Должен быть ровно один экземпляр некоторого класса, легко доступный всем клиентам
- Единственный экземпляр должен расширяться путем порождения подклассов, и клиентам нужно иметь возможность работать с расширенным экземпляром без модификации своего кода

Почему не статический класс/поле/метод?

- Синглтон -- полноценный объект: можно передавать в методы, можно скрыть за интерфейсом, можно задействовать в иерархии наследования.

Одиночка (Структура)



Клиенты получают доступ к экземпляру класса Singleton только через его операцию Instance

Структурные паттерны

(Structual)

Структурные паттерны (Structual)

- Определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу;
- используют наследование для составления композиций из интерфейсов и реализаций;
- особенно полезны эти шаблоны, когда нужно организовать совместную работу нескольких независимо разработанных библиотек.

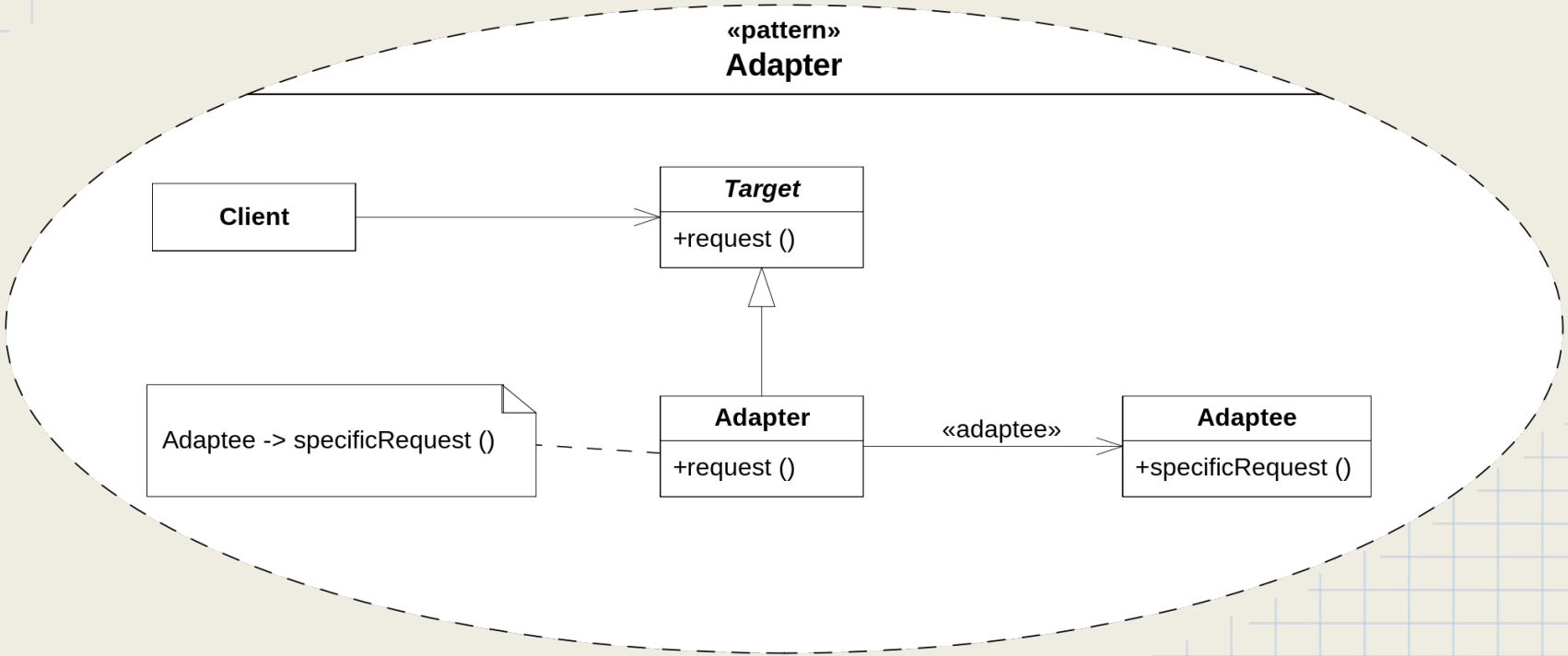
Задача 3

Необходимо обеспечить возможность работы с вектором как со стеком: реализовать push, pop, empty.

Адаптер (Adapter)

- Преобразует интерфейс одного класса в интерфейс другого, который ожидают клиенты
- Обеспечивает совместную работу классов с несовместимыми интерфейсами, которая без него была бы невозможна
- Альтернативное название – Wrapper (Обертка)

Адаптер (Структура)



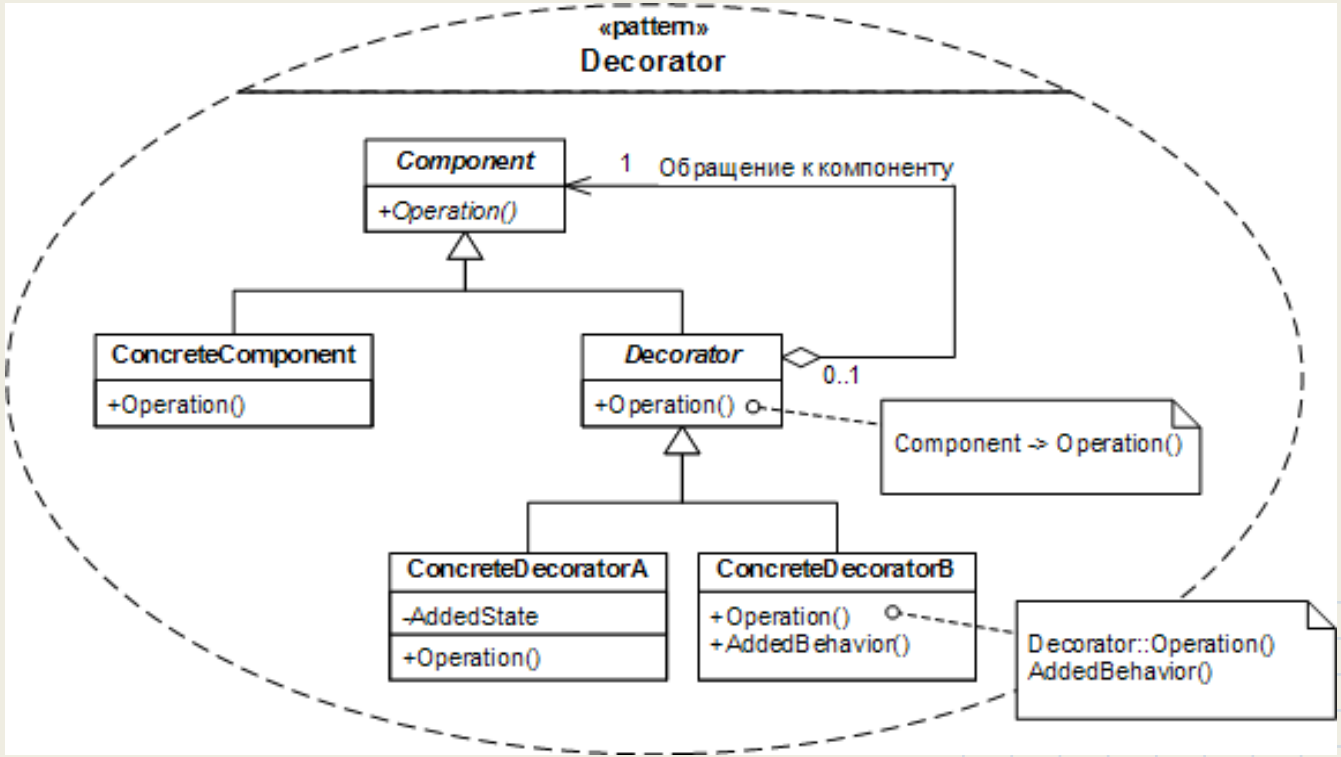
Задача 4

Спроектировать оконный менеджер: добавить отрисовку простого окна, отрисовку окна с вертикальным скролл-баром, горизонтальным скролл-баром, вертикальным и горизонтальным скролл-баром. Система должна поддерживать добавление функциональности к уже существующим окнам во время исполнения программы.

Декоратор (Decorator)

- Предназначен для динамического подключения дополнительного поведения к объекту;
- предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности;
- новую функциональность можно подключать до или после основной функциональности объекта ConcreteComponent.

Декоратор (Структура)



Поведенческие паттерны

(Behavioral)

Поведенческие паттерны (Behavioral)

- Определяют алгоритмы и способы реализации взаимодействия различных объектов и классов;
- используют наследование чтобы определить поведение для различных классов;
- Примеры:
  - Наблюдатель
  - Шаблонный метод

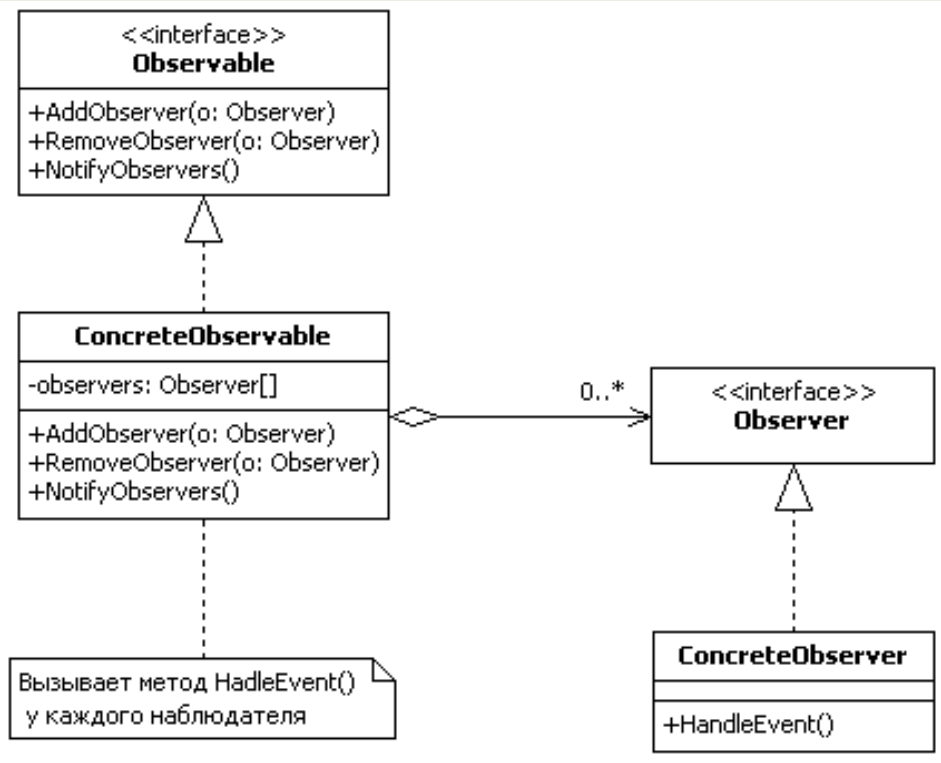
Задача 5

Спроектировать систему для разработки графического пользовательского интерфейса (GUI). Реализовать возможность генерации событий и реагирования на них.

Наблюдатель (Observer)

- Создает механизм у класса, который позволяет получать оповещения от других классов об изменении их состояния, тем самым наблюдая за ними;
- Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии;
- инкапсулирует главный (независимый) компонент в абстракцию Subject и изменяемые (зависимые) компоненты в иерархию Observer;
- определяет часть "View" в модели Model-View-Controller (MVC)

Наблюдатель (Структура)



Наблюдатель(Применение)

- Применяется в тех случаях, когда система обладает следующими свойствами:
  - существует, как минимум, один объект, рассылающий сообщения;
  - имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения;
  - нет надобности очень сильно связывать взаимодействующие объекты, что полезно для повторного использования.
- Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.

Задача 6

Спроектировать компьютерную игру, которая имеет игровой цикл, состоящий из трех фаз: обработка событий, логика и отрисовка. Система должна позволять изменять фазы игрового цикла, но не должна позволять изменять структуру игрового цикла.

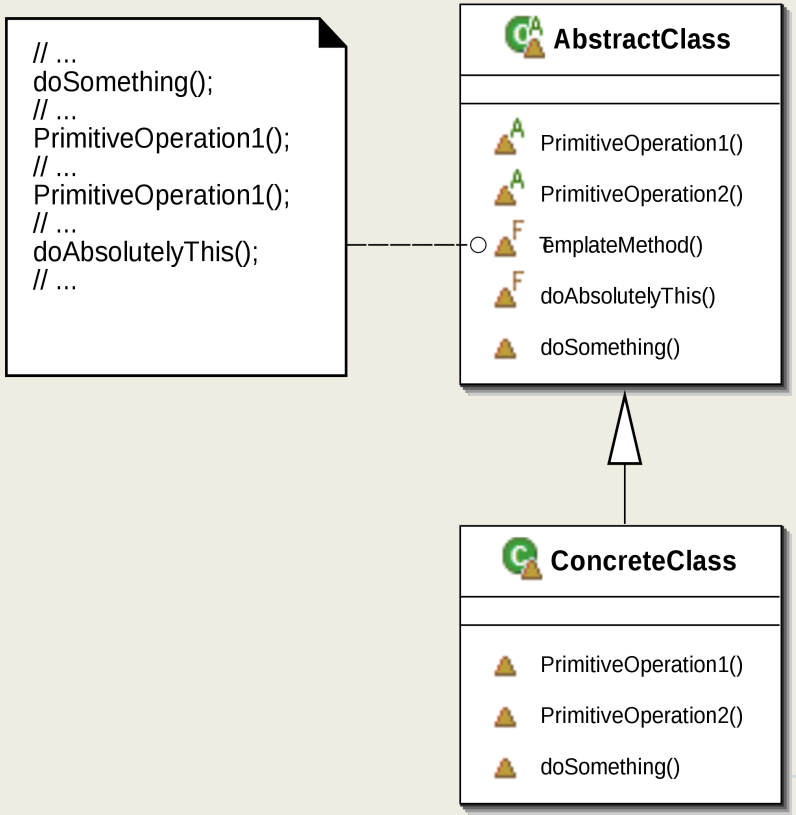
Шаблонный метод(Template method)

- Определяет основу алгоритма;
- позволяет наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.

Паттерн следует использовать, если необходимо:

- Однократное использование инвариантной части алгоритма, с оставлением изменяющейся части на усмотрение наследникам;
- Локализация и вычленение общего для нескольких классов кода для избегания дублирования;
- Разрешение расширения кода наследниками только в определенных местах.

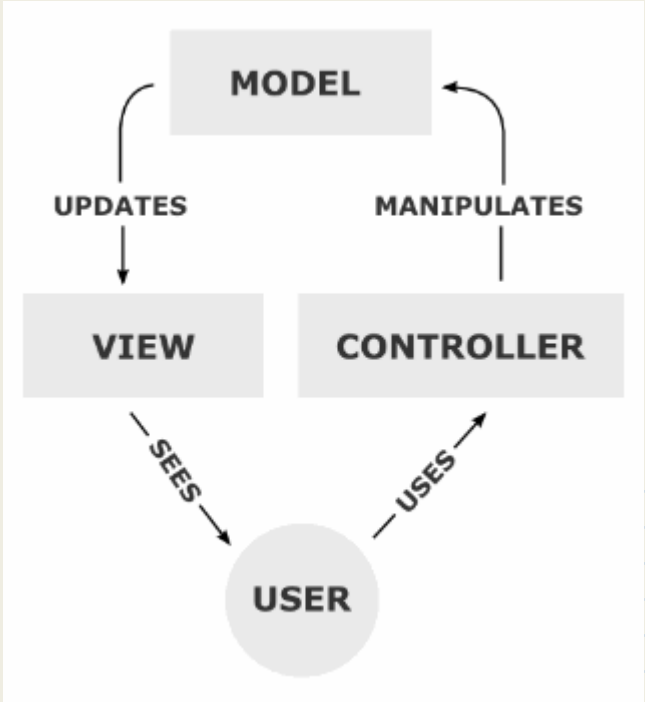
Шаблонный метод (Структура)



Не ООП единым

Концепция MVC выполняет следующие задачи:

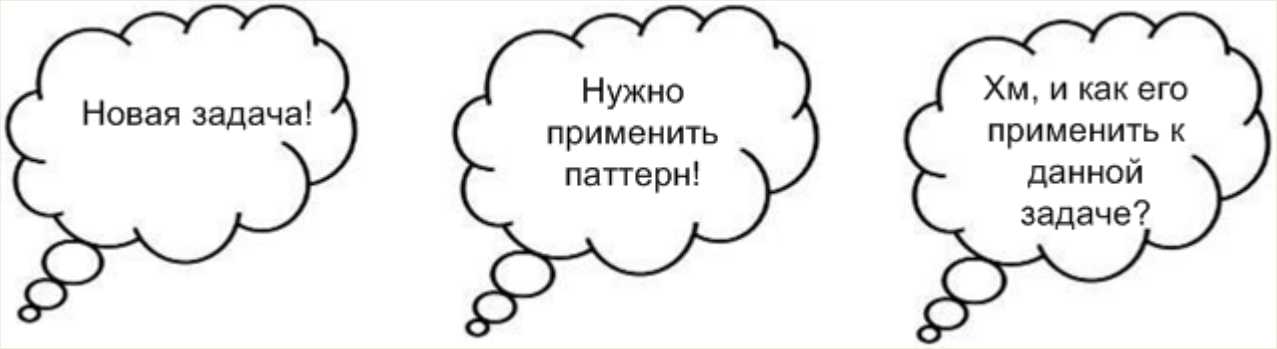
- К одной модели можно присоединить несколько представлений, при этом не затрагивая реализацию модели.
- Не затрагивая реализацию представлений, можно изменить реакции на действия пользователя , для этого достаточно использовать другой контроллер.
- Ряд разработчиков специализируется только в одной из областей. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (модели), вообще не будут осведомлены о том, какое представление будет использоваться.



Warning

Не нужно искать паттерны там, где их нет.

Плохо:



Лучше:

Литература:

- ru.wikipedia.org/wiki/Шаблон_проектирования
- Erich Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software, 1994
- Eric T Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, Head First Design Patterns, 2004