Паттерны проектирования

Паттерн — это в переводе с английского — это «узор». В программировании, его можно переводить как «образец» или «шаблон».

Паттерн описывает задачи, которые возникают в работе программиста раз за разом, чтобы не решать их каждый раз заново.

Паттерн проектирования — это описание взаимодействия объектов и классов адаптированных для решения общей задачи проектирования в конкретном контексте.

Идиомы — это паттерны, описывающие типичные решения на конкретном языке программирования, в то время, как паттерны проектирования от языка не зависят.

В общем случае, паттерн содержит:

1. **Имя** — уникальный идентификатор паттерна. Может использоваться как ссылка на типичное решение.
2. **Задача** — это ситуация, в которой можно применять паттерны. При формулировки задачи, важно описать контекст, в котором паттерн возникает.
3. **Решение** задачи проектирования определяет общие функции каждого элемента дизайна. Рассматривается именно общая ситуация.
4. **Результаты —** это следствия применения паттерна. В них описываются преимущества и недостатки паттерна, компромиссы и вариации.

Антипаттерн — это часто повторяемое плохое решение, которое не рекомендуется использовать.

Признаки плохого кода и дизайна

- Дублирование кода
- Большие методы
- Большие классы
- Чрезмерная зависимость
- Нарушения приватности
- Нарушение завещания
- Ленивый класс
- Чрезмерная сложность
- Длинные имена

Классификации паттернов

1. Архитектурные
2. Проектирования

3. Анализа
4. Тестирования
5. Реализации

Паттерны проектирования

1. Основные (фундаментальные) паттерны — это наиболее важные паттерны, используемые другими.
2. Порождающие паттерны — определяют способы создания объектов
3. Структурные паттерны — описывают сложные структуры
4. Поведенческие паттерны — способы взаимодействия между объектами
5. Паттерны параллельного программирования — координируют параллельные операции, решая проблемы борьбы за ресурсы и взаимоблокировок
6. Паттерны MVC – описывают Модель-Вид-Контроллер
7. Паттерны корпоративных систем — решения для построения больших корпоративных систем

Среди 23 паттернов Банды Четырёх описаны Порождающие, Структурные и Поведенческие.

Порождающие паттерны

Абстрагируют процесс инстанцирования. Они помогают сделать систему независимой от способа создания, композиции и представления объектов. Отвечают на вопрос «Кто и как создаёт объекты в системе?»

Абстрактная фабрика

Abstract Factory, также известен, как Toolkit/Инструментарий

Цель: предоставляет интерфейс для создания семейства взаимосвязанных и взаимозависимых объектов, не идентифицируя конкретныце классы.

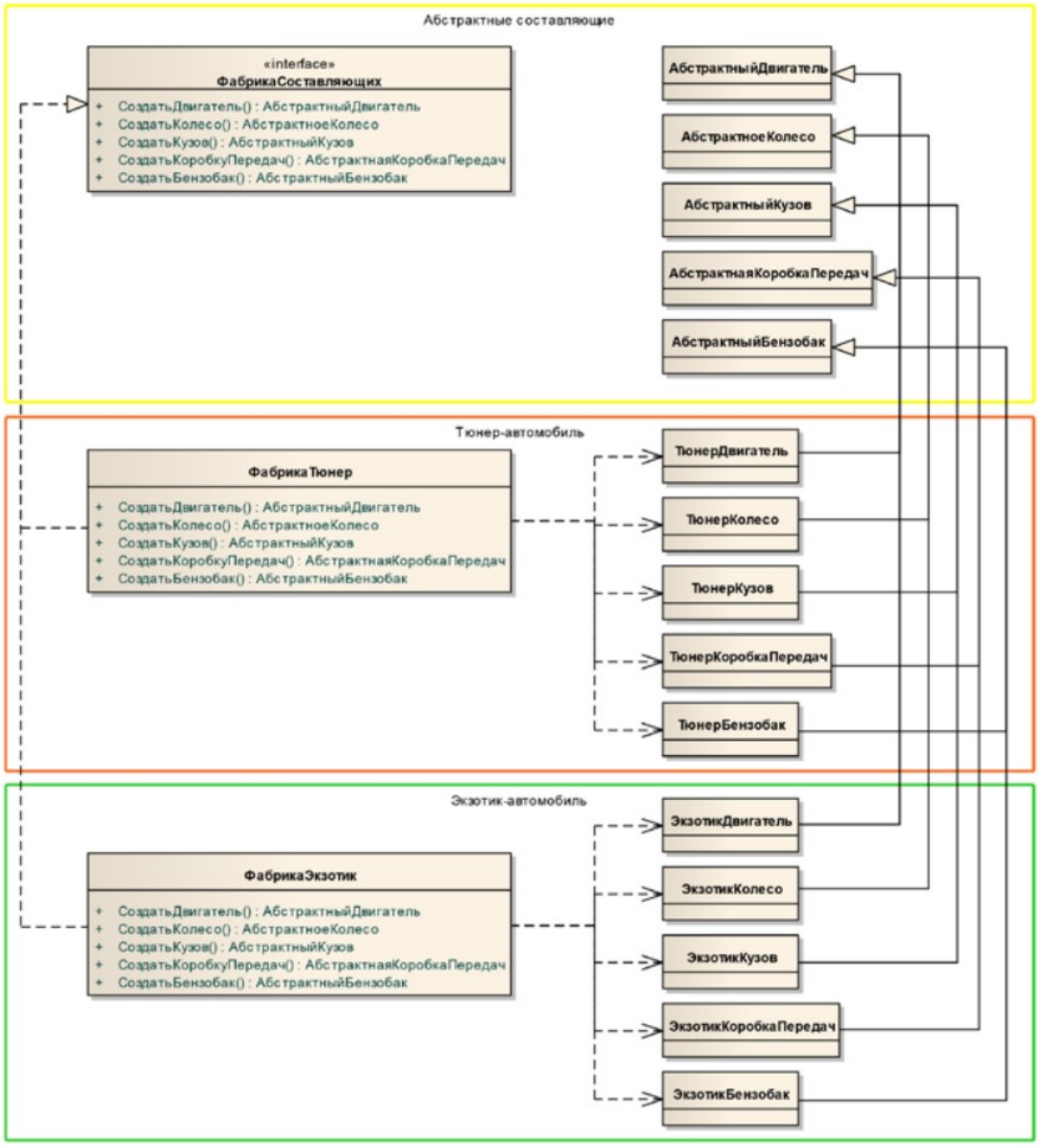
Используется, когда:

- Система не должна зависеть от того, как в ней создаются объекты

- Объекты в одном семействе используются вместе

- Система конфигурируется одним из семейств объектов

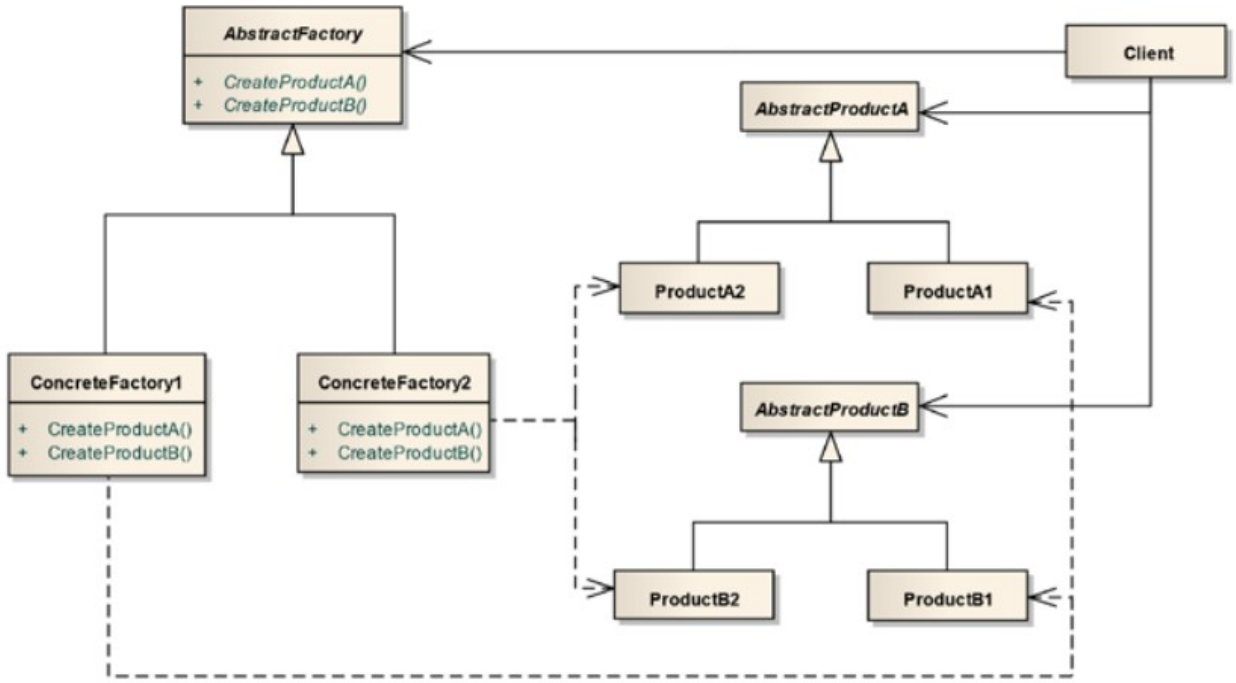
- Надо предоставить интерфейс библиотеки, не раскрывая её реализации



Существенным недостатком является то, что ФабрикаСоставляющих на этапе компиляции предусматривает создание фиксированного набора продуктов, следовательно — количество и тип продуктов жёстко определены на этапе компиляции, а при необходимости добавить новый продукт, необходимо изменить абстрактную фабрику.

Во избежание таких проблем можно рассмотреть использование паттерна прототип.

Саму фабрику можно создавать, как singleton.



Общая структура паттерна

Абстрактная фабрика, которая предоставляет общий интерфейс для создания семейства продуктов.

Конкретная фабрика, которая реализует интерфейс Абстрактной фабрики и создаёт семейство конкретных продуктов.

Абстрактный продукт — предоставляет интерфейс продукта, ссылку на который возвращают методы фабрик.

Конкретный продукт — реализует тип продукта.

Отношения:

Клиент знает только о существовании абстрактной фабрики и её абстрактных продуктах.

Для создания семейства конкретных продуктов клиент конфигурируется соответствующим экземпляром конкретной фабрики.

Методы конкретных фабрик создают экземпляры конкретных продуктов, возвращая ссылки на абстрактные продукты.

Результаты:

Изоляция конкретных классов продуктов.

Упрощает замену семейств продуктов.

Гарантия сочетаемости продуктов.

Недостаток паттерна: трудность поддержки нового вида продуктов — необходимо изменить всю иерархию фабрик и их код.

Builder

Строитель

Цель: отделяет процесс конструирования сложного объекта от его представления, так что в результате одного и того же процесса конструирования получаются разные представления.

Клиент может создавать сложные объекты, определяя не только его тип, но и содержимое, не зная при этом о деталях конструирования.

Паттерн используют, когда:

- Общий алгоритм построения сложного объекта не должен зависеть от специфики каждого шага

- В результате одного и того же процесса надо получить разные продукты

Причины возникновения

Допустим есть конвейер для производства автомобилей.

Шаги построения автомобиля:

1. Сборка кузова

2. Установка двигателя

3. Установка колёс

4. Покраска

5. Подготовка салона

Допустим на заводе производят автомобили «мини», спортивные и внедорожники.

В контексте ООП, это описывается так:

Класс Конвейер, который является прототипов реального конвейера.

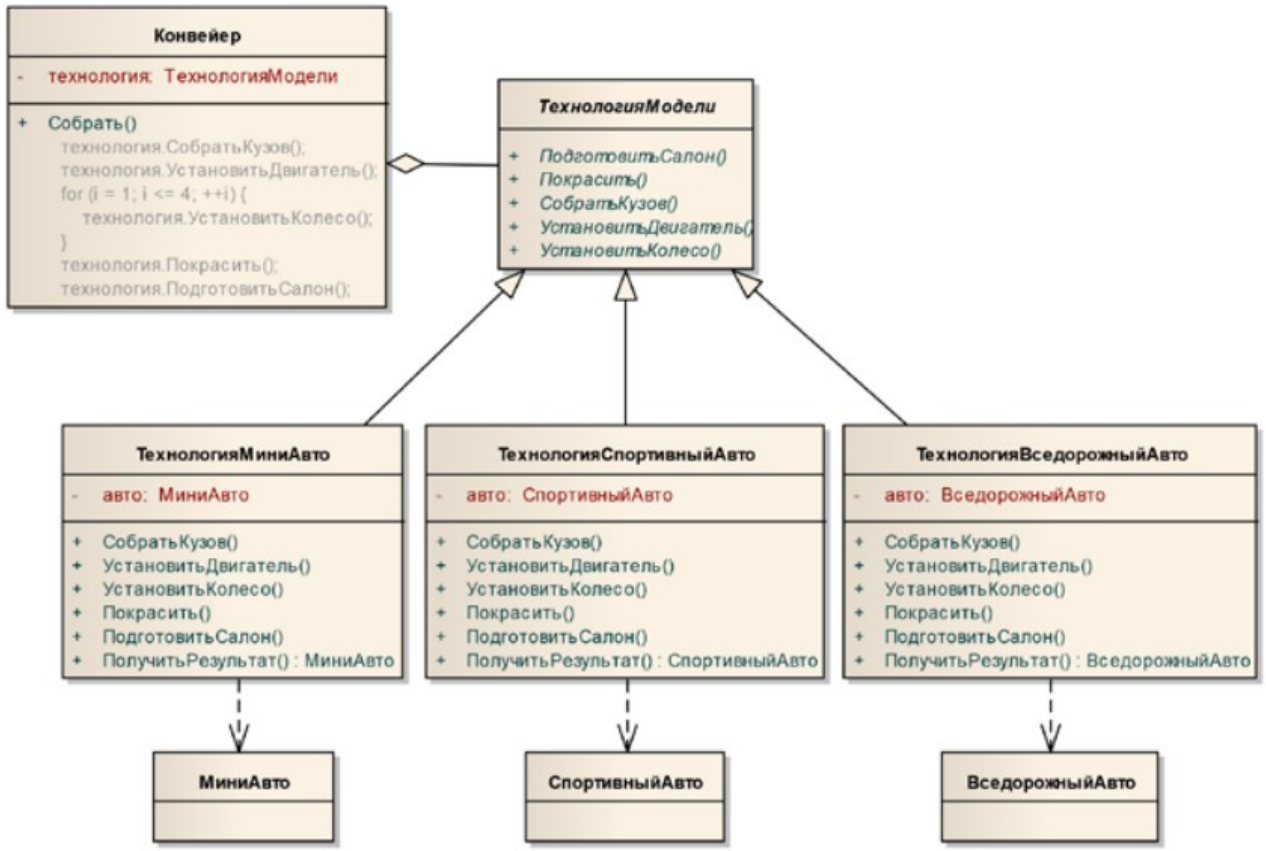
Метод Собрать исполняет процесс посредством выполнения этих шагов вне зависимости от технических деталей.

Ответственность за реализацию шагов несёт класс ТехнологияМодели.

Применяя различные подклассы мы получаем разные модели автомобилей. То есть, от него наследованы классы для каждого типа.

Для производства автомобиля, надо задать конкретную технологию и вызвать метод Собрать.

После завершения процесса сборка, автомобить можно получить с помощью метода «ПолучитьРезультат()».



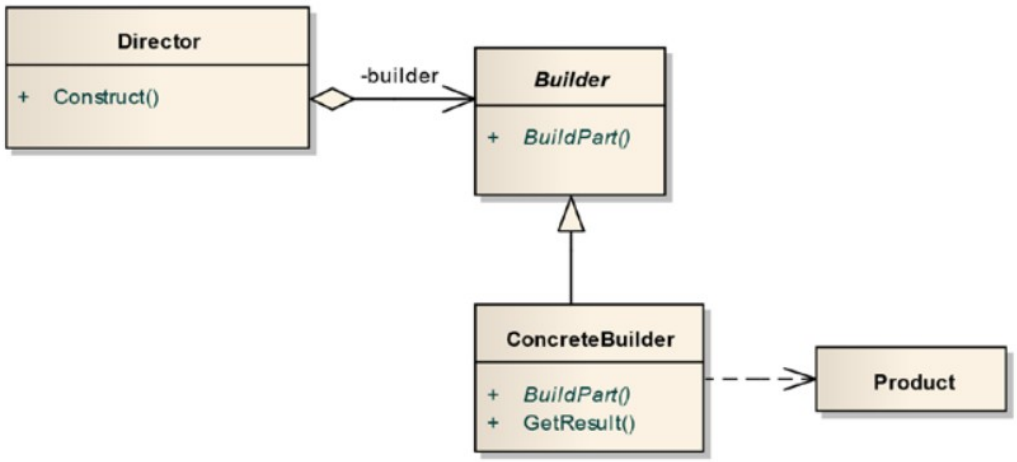
Преимущества модели:

1) конкретная технология конструирования строится по общему шаблону

2) общий алгоритм процесса не зависит от деталей конкретной технологии

3) есть возможность реализовать под общий алгоритм большое количество технологий

Структура паттерна



Builder/Строитель

Обеспечивает интерфейс для пошагового конструирования сложного объекта

ConcreteBuilder/Конкретный строитель

Реализует шаги построения сложного объекта

Создаёт результат построения

Определяет доступ к этому результату

Director/Распорядитель

Определяет общий алгоритм конструирования, используя для реализации шаги из Builder

Product/Продукт

Сложный объект, полученный в результате конструирования.

Отношения между участниками:

- Клиент конфигурирует Распорядителя.

- Распорядитель вызывает методы Строителя.

- Конкретный строитель создаёт продукт и следит за его конструированием.

- Конкретный строитель предоставляет интерфейс для доступа к продукту.

Результаты

Есть возможность изменять внутреннюю структуру объектов или создавать новые.

Разделение Распорядителя и Строителя повышает модульность ПО.

Пошаговое построение продукта позволяет обеспечить более точный контроль над процессом создания.