Фабричные шаблоны - Фабричный шаблон метода

Снеш Праджапати

2 октября 2016 г. CPOL

В этой статье мы разберемся с шаблоном фабричного метода. Эта статья является второй частью серии статей Factory Patterns и является продолжением части 1: Simple Factory Pattern.

Скачать исходный код - 9 КБ

Вступление

В этой статье мы разберемся с шаблоном фабричного метода. Эта статья является второй частью серии статей Factory Patterns и является продолжением Part1: Simple Factory Pattern . В первой части мы изучили Simple Factory Pattern. Если вы не прошли первую часть, я предлагаю вам сделать это, а затем продолжайте эту статью.

В первой части мы увидели, как Simple Factory Pattern помогает нам в создании объектов, и обнаружили некоторые проблемы с Simple Factory Pattern. В этой части мы узнаем, как Factory Method Pattern решает эти проблемы и в какой ситуации мы должны использовать Factory Method Pattern.

Контуры

Часть 1 - Простой заводской шаблон

Часть 2 - Шаблон заводского метода

- Что такое шаблон фабричного метода
- Понять определение на примере
- Когда использовать шаблон фабричного метода
- Преимущества паттерна фабричного метода

Часть 3 - Абстрактный узор фабрики

Что такое шаблон фабричного метода

Шаблон фабричного метода представлен в книге Gang Of Four (GoF) и относится к категории Creation Patterns. Ниже приводится определение Factory Method шаблон взят из той же книги:

«Определить интерфейс для создания объекта, но пусть подклассы решить , какой класс инстанцировать Фабричный метод позволяет класс отложить создание экземпляра на подклассы.»

Для того , чтобы быть ясно , об этом определение, мы поймем это определение, разбив его на части в разделах этой статьи.

Понять определение на примере

Мы продолжим с того же примера, который мы взяли в первой части, чтобы вы могли понять, когда одно и то же приложение разрастается, какие проблемы возникают и когда мы должны использовать шаблон фабричного метода (поскольку каждый шаблон имеет свое место, преимущества и ограничения).

Начнем с того же кода. Мы будем использовать тот же интерфейс под названием IFan, у которого было два метода, как показано ниже.

Копировать код

```
interface IFan
{
    void SwitchOn();
    void SwitchOff();
}
```

Изначально наша компания намеревалась создать только три типа вентиляторов: TableFan, CeilingFan и ExhaustFan. У всех фанатов есть методы SwitchOn () и SwitchOff (), и их реализация может отличаться. Давайте использовать те же классы и реализовать интерфейсы IFan.

Копировать код

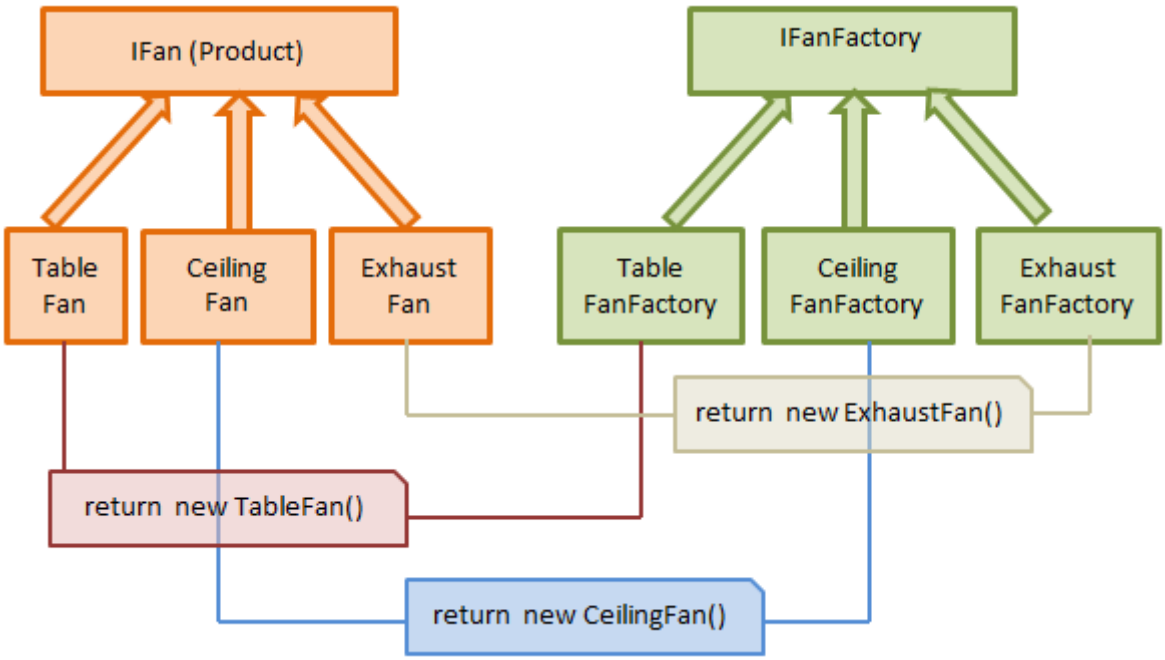
```
class TableFan : IFan {.... }

class CeilingFan : IFan {.... }

class ExhaustFan : IFan {..... }
```

Пока код, который мы здесь написали, совпадает с кодом, который мы написали в примере «Простой фабричный шаблон».

Когда мы реализуем шаблон фабричного метода, мы должны следовать его дизайну, как показано на диаграмме ниже:



Итак, далее мы проследим его дизайн и объясним стандартное определение шаблона фабричного метода.

Согласно определению ***«Определите интерфейс для создания объекта ...»,*** давайте определим интерфейс, который будет иметь метод для создания Поклонников. Реализация этого метода в подклассах будет иметь логику для создания конкретного объекта. **Примечание:**

мы используем интерфейс IFanFactory, но вместо интерфейса также можно использовать абстрактный класс. У абстрактного класса должен быть хотя бы один нереализованный метод abstarct, который будет реализован в подклассе для создания объекта.

Копировать код

```
interface IFanFactory
{
    IFan CreateFan();
}
```

Поскольку шаблоны проектирования - это просто особый способ проектирования и они не зависят от языка, мы также можем использовать другой подход для определения базового контакта для создания объекта, который будет заключаться в «Определении интерфейса для создания объекта». Например, в JavaScript у нас нет концепции абстрактного класса или интерфейса, но мы все же можем использовать шаблоны проектирования. Для получения дополнительной информации, пожалуйста, ознакомьтесь с этим сообщением в блоге Роба Додсона .

Продолжите определение дальше: ***....., позвольте подклассам решать, какой класс создать экземпляр ... ..*** : Давайте создадим подклассы, которые будут решать, какой конкретный класс будет использоваться для создания экземпляра.

Чтобы создать три типа объектов, а именно TableFan, CeilingFan и ExhaustFan, нам нужно создать три типа фабрик или подклассов IFanFactory. Эти три фабрики будут реализовывать интерфейс IFanFactory.

Копировать код

```
class TableFanFactory : IFanFactory {....}

class CeilingFanFactory : IFanFactory {....}

class ExhaustFanFactory : IFanFactory {....}
```

Снова продолжите определение дальше: ***... Заводской метод позволяет классу отложить создание экземпляра до подклассов.*** **"** : Вот `IFan CreateFan();` заводской метод. Внутри метода CreateFan (который является заводским методом) в конкретных Фабриках (TableFanFactory и т. Д.) Мы указываем, какой именно класс будет для создания объекта Fan. Таким образом, IFanFactory откладывает принятие решения для конкретного класса своим подклассам, а именно TableFanFactory, CeilingFanFactory и ExhaustFanFactory.

Теперь реализация клиента представлена ниже:

Копировать код

```
//The client code is as follows:
static void Main(string[] args)
{
     IFanFactory fanFactory = new PropellerFanFactory();

     // Creation of a Fan using Factory Method Pattern
     IFan fan = fanFactory.CreateFan();

     // Use created object
     fan.SwitchOn();

     Console.ReadLine();
 }
```

Когда использовать шаблон фабричного метода

В приведенном выше примере для создания трех типов объектов (TableFan, CeilingFan и ExhaustFan) мы создали три фабричных класса, но, изучая «Простой фабричный шаблон», мы создали только одну фабрику. Это потому, что здесь мы следуем правилам Factory Method Pattern. Теперь возникает вопрос, почему нам нужно использовать Factory Method Pattern, поскольку мы уже можем добиться того же с помощью «Simple Factory Pattern». *Предположим, компания собирается выпустить новый вентилятор под названием «PropellerFan». Это новое требование. Используя Factory Method Pattern, мы не добавляем новое условие в случае переключения, как мы это делали в «Простом заводском шаблоне». Просто мы создадим новый класс под названием «PropellerFan» и унаследуем этот класс с интерфейсом IFan, как мы сделали для других вентиляторов.*

Копировать код

```
class PropellerFan : IFan {..... }
```

Затем создаст новую фабрику под названием PropellerFanFactory и выведет ее из IFanFactory. Когда клиент хочет создать экземпляр класса PropellerFan. Просто создав момент PropellerFanFactory, клиенты получат экземпляр PropellerFan.

Преимущества паттерна фабричного метода

- В приведенном выше разделе мы видели, что шаблон фабричного метода следует принципу открытия и закрытия. Когда появилось новое требование, мы не внесли изменений в существующий код, но нам нужно было создать дополнительную Factory.
- Писать модульные тесты проще с помощью Factory Method Pattern по сравнению с Simple Factory Pattern, так как case switch (или длинные блоки if else) не используется.
- Для поддержки дополнительных продуктов мы не изменяем существующий код, а просто добавляем один новый класс Factory, поэтому нет необходимости повторно запускать существующие старые модульные тесты.
- Клиент вызывает CreateFan (фабричный метод), не зная, как и какой фактический тип объекта был создан.
- Если мы используем абстрактный класс, такой как BaseFanFactory (вместо IFanFactory), мы можем обеспечить реализацию общих методов в абстрактном классе BaseFanFactory, объявляя только метод CreateFan как абстрактный. Исходя из требований, у нас могут быть более абстрактные методы в BaseFanFactory.

Заключение

В этой статье мы познакомились с шаблоном фабричного метода и его использованием. Мы поняли контекст Factory Method Pattern и чем он отличается от Simple Factory Pattern. Но он может создавать только один вид продуктов (которые все реализуют IFan в нашем примере), поэтому в следующей статье мы увидим, как мы можем создать набор связанных продуктов с помощью абстрактного фабричного шаблона. Спасибо за прочтение. Мы будем рады вашим комментариям и предложениям по улучшению.

использованная литература

1. Обсуждение мотивации Simple Factory
2. Блог Кори Бродерика
3. Блог Coing Geek
4. Статья о заводских выкройках

Лицензия

Делиться

об авторе

Снеш Праджапати
Разработчик программного обеспечения
Индия

Я разработчик программного обеспечения, работающий над технологиями Microsoft. Меня интересует изучение и распространение новейших технологий.

Комментарии и обсуждения

Вы должны войти в систему, чтобы использовать эту доску сообщений.

Search Comments

Первая Предыдущая Следующая

Мой голос 5
iSahilSharma **4 марта 18, 4:56**

Мульти простая фабрика
zg l **2-апр-17 4:21**

Фантастическая и хорошая работа
Рамачандран Муруган **13-янв-17 8:33**

Re: Фантастическая и хорошая работа
Снеш Праджапати 13 января 17, 15:12

Немного другая реализация ...
vladi7 **7-ноя-16 22:11**

Очень понятная, легко читаемая статья в хорошем стиле.
Участник 8999948 **3-окт-16 23:58**

Re: Очень понятная, легко читаемая статья в хорошем стиле.
Снеш Праджапати 5 октября **2016 г.** 7:13

Отсутствует большой кусок ...
PureNsanity 3 октября 2016 г. 6:58

Re: Отсутствует большой кусок ...
Snesh Prajapati 3-Oct-16 16:46

Re: Отсутствует большой кусок ...
katibotar@hotmail.com 15-Jul-20 12:58

Re: Отсутствует большой кусок ...
PureNsanity 27-Jul-20 8:55

Шаблон заводского метода
DeyChandan 3-Oct-16 0:59

Re: Шаблон заводского метода
Snesh Prajapati 3-Oct-16 16:26

пожалуйста, предоставьте некоторую информацию об абстрактной фабрике
Tridip Bhattacharjee 2-Oct-16 21:53

Re: пожалуйста, предоставьте некоторую информацию об абстрактной фабрике
Snesh Prajapati 3-Oct-16 16:25

Дженерики
Afterlife99 2-Oct-16 17:05

Re: Дженерики
Snesh Prajapati 2-Oct-16 17:43

Re: Дженерики
PureNsanity 3-Oct-16 6:40

Re: Дженерики
dmjm-h 3-Oct-16 11:17

Спасибо за оба ваших сообщения. Я думал, что, может быть, я что-то упустил, но нет, шаблон просто не имеет смысла в том виде, в котором он представлен.

Войти · Просмотр темы

Re: Дженерики
Snesh Prajapati 3-Oct-16 16:56

Обновить 1

Общие Новости Предложение Вопрос Ошибка Ответ Шутка Похвала Rant Администратор

Используйте Ctrl + Left / Right для переключения сообщений, Ctrl + Up / Down для переключения цепочек, Ctrl + Shift + Left / Right для переключения страниц.