**Паттерн «Фабричный метод»**

**Проблема**

Представим, что нам нужно написать программу управления грузовыми перевозками. Сперва мы хотим перевозить товары только на автомобилях. Поэтому и весь код написан только для работы с классом “Грузовик”.

В какой-то момент мы решил добавить в программу морскую логистику. Но есть проблема: большая часть существующего кода жёстко привязана к классам “Грузовиков”, поэтому для добавления логики придётся переписать весь код. Более того, если мы решим потом добавить перевозку с помощью самолётов, то опять придется все переписать.

**Решение**

Паттерн “Фабричный метод” предлагает создавать объекты не напрямую, а через вызов особого фабричного метода. Теперь мы можем переопределить фабричный метод в подклассе, чтобы изменить тип создаваемого продукта.

Например, классы “Дорожная Логистика” и “Морская Логистика” будут наследоваться от класса “Логистика” и будут переопределять его метод createTransport(). Фабричный метод createTransport() в “Дорожной Логистике” будет возвращать объект “Грузовик”, а в “Морской Логистике” -- “Судно”

Чтобы эта система заработала, все возвращаемые объекты должны иметь общий интерфейс. Таком образом, подклассы смогут производить объекты различных классов, следующих одному и тому же интерфейсу.

Например, классы “Грузовик” и “Судно” реализуют интерфейс “Транспорт” с методом “доставить”. Каждый из этих классов реализует метод по-своему: грузовики везут грузы по земле, а суда — по морю.

Для клиента фабричного метода нет разницы между этими объектами, так как он будет трактовать их как некий абстрактный “Транспорт”. Для него будет важно, чтобы объект имел метод доставить, а как конкретно он работает — не важно.

**Ещё пример**

Допустим, мы создаем программу для сферы строительства. Мы хотим строить многоэтажные каменные дома и деревянные дома. Для строительства определенного типа дома, нам нужен свой подрядчик. Поэтому у нас будет интерфейс “House” c методом “починить” и две его реализации “BrickHouse” и “WoodHouse”, где мы по своему реализуем функцию “repair”. Также у нас будет класс “Builder”, в котором будет абстрактная функция “build”. От класса “Builder” мы наследуем два подкласса “BrickHouseBuilder” и “WoodHouseBuilder”, которые также по своему реализуют функцию “build”.

**Что делает?**

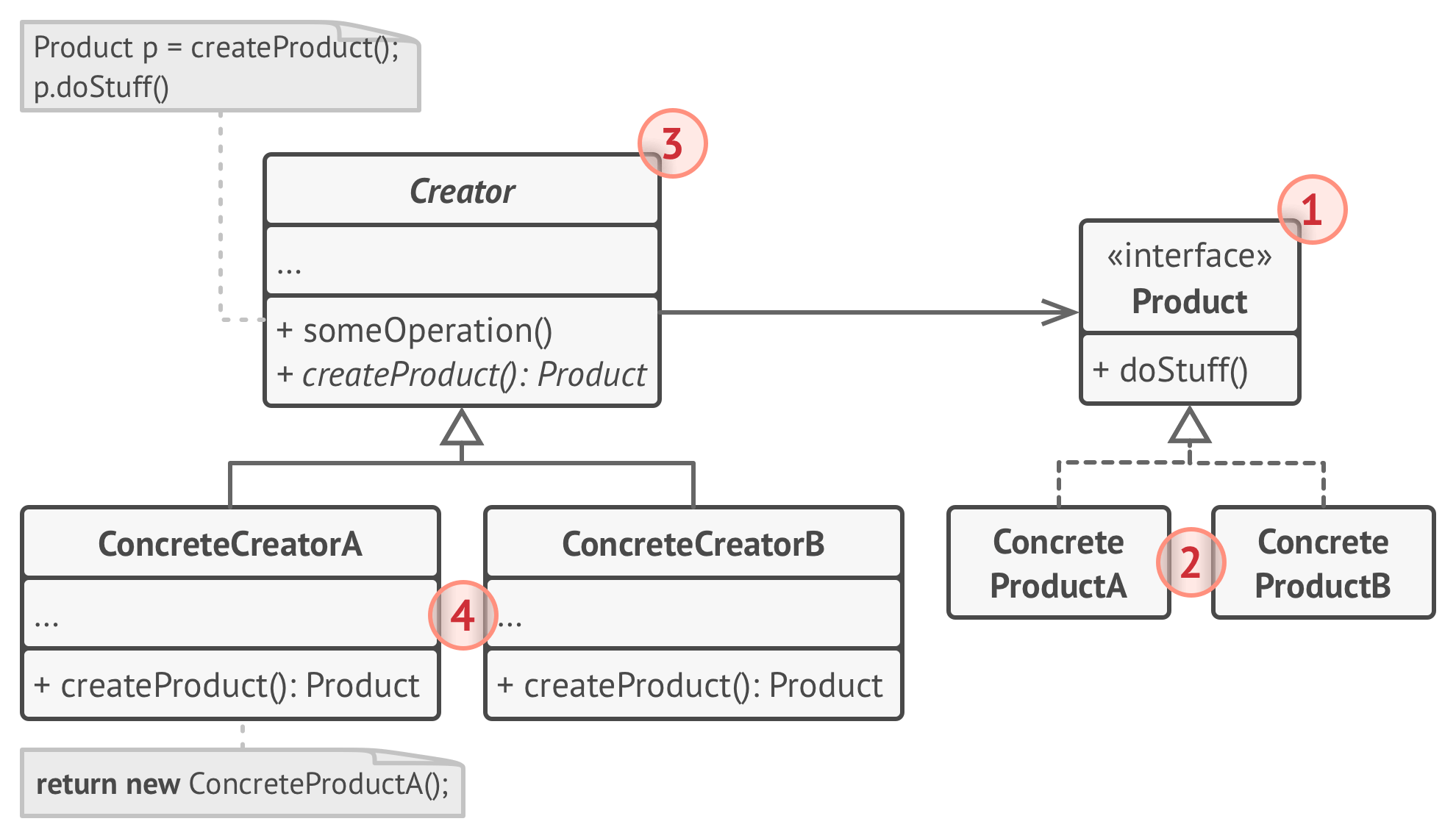
Фабричный метод — это порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе (родительском классе), позволяя подклассам изменять тип создаваемых объектов.

**Применимость**

Таким образом, используйте паттерн фабричный метод, когда:

* Классу заранее неизвестны типы и зависимости объектов, с которыми должен работать ваш код.
  + Фабричный метод отделяет код производства продуктов от остального кода, который эти продукты использует. Благодаря этому, код производства можно расширять, не трогая основной.
* Класс спроектирован так, чтобы объекты, которые он создает, специфицировались подклассами;
* Класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и вы планируете локализовать знание о том, какой класс принимает эти обязанности на себя

**Структура**

****

1. Интерфейс “Продукт” определяет общий интерфейс объектов, которые может произвести создатель и его подклассы.
2. Конкретные продукты содержат код различных продуктов. Продукты будут отличаться реализацией, но интерфейс у них будет общий.
3. Создатель объявляет фабричный метод, который должен возвращать новые объекты продуктов. Важно, чтобы тип результата совпадал с общим интерфейсом продуктов.

Зачастую фабричный метод объявляют абстрактным, чтобы заставить все подклассы реализовать его по-своему. Но он может возвращать и некий стандартный продукт.

Важно понимать, что создание продуктов не является единственной функцией создателя. Обычно он содержит и другой полезный код работы с продуктом. Например, в нашем примере класс “Логистика” не только имеет фабричный метод, но и функцию planDelivery(). Аналогия: большая софтверная компания может иметь центр подготовки программистов, но основная задача компании — создавать программные продукты, а не готовить программистов.

1. Конкретные создатели по-своему реализуют фабричный метод, производя те или иные конкретные продукты.

Фабричный метод не обязан всё время создавать новые объекты. Его можно переписать так, чтобы возвращать существующие объекты из какого-то хранилища или кэша.

**Реализация**

**interface** Product {}

**class** ConcreteProductA : Product {}

**class** ConcreteProductB : Product {}

**abstract class** Creator {

**abstract fun** factoryMethod(): Product

}

**class** ConcreteCreatorA : Creator() {

**override fun** factoryMethod(): Product {

**return** ConcreteProductA()

}

}

**class** ConcreteCreatorB : Creator() {

**override fun** factoryMethod(): Product {

**return** ConcreteProductB()

}

}

**Пример на Kotlin**

**interface** Transport {

**fun** delivery()

}

**class** Truck: Transport {

**override fun** delivery() {

*println*(**"Грузовик доставил котиков по суше"**)

}

}

**class** Ship: Transport {

**override fun** delivery() {

*println*(**"Судно доставило котиков по воде"**)

}

}

**abstract class** Logistics {

**fun** planDelivery(number: Int) {

*println*(**"Нужно доставить $**number **котиков"**)

}

**abstract fun** createTransport(): Transport

}

**class** TruckLogistics: Logistics() {

**override fun** createTransport(): Transport {

**return** Truck()

}

}

**class** ShipLogistics: Logistics() {

**override fun** createTransport(): Transport {

**return** Ship()

}

}

**fun** main() {

**val** config = *readLine*() *// Ship*

**val** logistics = **when** (config) {

**"Truck"** -> TruckLogistics()

**"Ship"** -> ShipLogistics()

**else** -> **throw** IllegalArgumentException(**"Наташа, мы все уронили"**)

}

**val** numberOfCats = *readLine*()?.*toInt*() ?: 0 *// 69*

logistics.planDelivery(numberOfCats) *// Нужно доставить 69 котиков*

**val** transport = logistics.createTransport()

transport.delivery() *// Судно доставило котиков по воде*

}

**Связь с другим паттернами**

* Многие архитектуры начинаются с применения Фабричного метода (более простого и расширяемого через подклассы) и эволюционируют в сторону Абстрактной фабрики, Прототипа или Строителя (более гибких, но и более сложных).
* Классы Абстрактной фабрики чаще всего реализуются с помощью Фабричного метода, хотя они могут быть построены и на основе Прототипа.
* Фабричный метод можно использовать вместе с Итератором, чтобы подклассы коллекций могли создавать подходящие им итераторы.
* Фабричный метод можно рассматривать как частный случай Шаблонного метода. Кроме того, Фабричный метод нередко бывает частью большого класса с Шаблонными методами.

**Вывод**

1. Избавляет класс от привязки к конкретным классам продуктов.
2. Выделяет код производства продуктов в одно место, упрощая поддержку кода.
3. Упрощает добавление новых продуктов в программу.
4. Используется при реализации Абстрактной фабрики.