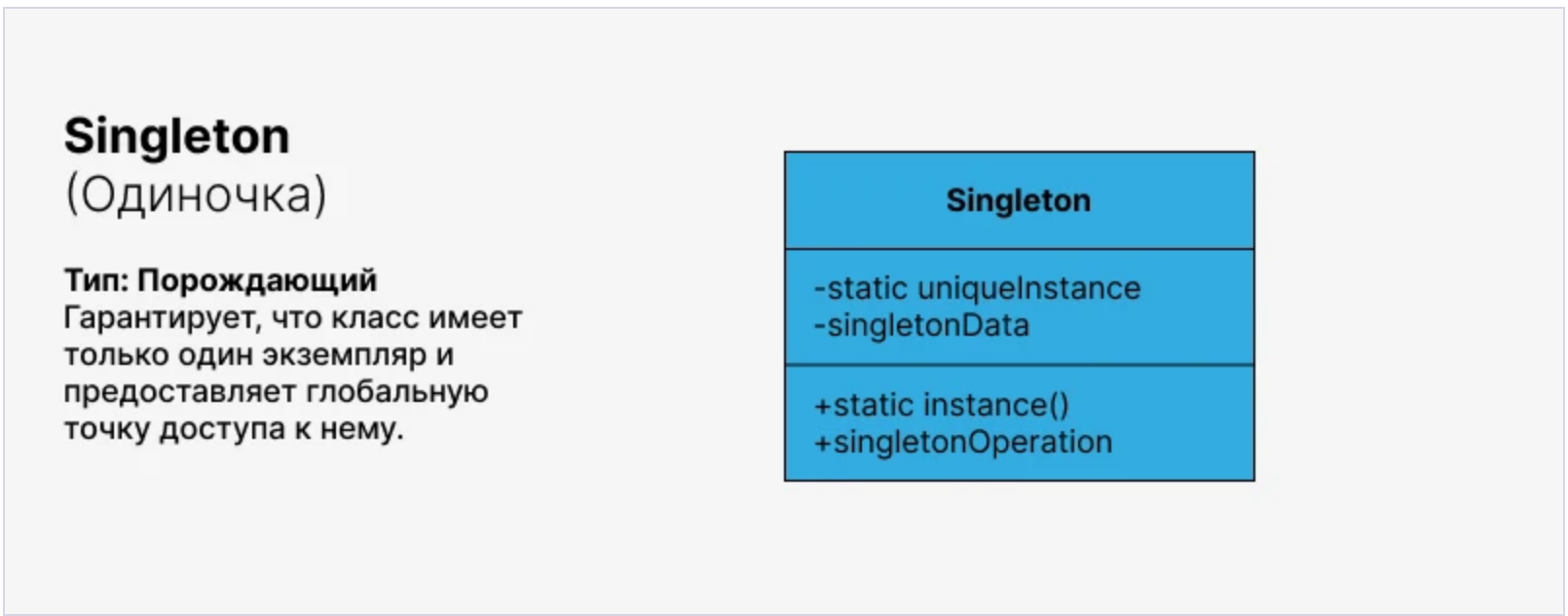
Порождающие паттерны

JSP & Servlets
16 уровень, 2 лекция

ОТКРЫТА

3.1 Singleton

Одиночка (Singleton) — порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.



Очень часто начинающие программисты любят собрать утилитные методы в некоторый статический класс – класс, содержащий только статические методы. У такого подхода есть ряд минусов – например, нельзя передать ссылку на объект такого класса, такие методы тяжело тестировать и тому подобное.

В качестве альтернативы предложили решение класс-синглетон: класс, у которого может быть только один объект. При попытке создания этого объекта он создается только в том случае, если еще не существует, в противном случае возвращается ссылка на уже существующий экземпляр.

Существенно то, что можно пользоваться именно экземпляром класса, так как при этом во многих случаях становится доступной более широкая функциональность. Например, этот класс может реализовывать некоторые интерфейсы и его объект можно передать в другие методы, как имплементацию интерфейса. Что нельзя сделать с набором статических методов.

Плюсы:

- Методы привязаны к объекту, а не к статическому классу – можно передавать объект по ссылке.
- Методы объекта значительно легче тестировать и мокировать.
- Объект создается только по необходимости: отложенная инициализация объекта.
- Ускорение начального запуска программы, если есть множество одиночек, которые не нужны для запуска.
- Одиночку можно в дальнейшем превратить в шаблон-стратегию или несколько таких объектов.

Минусы:

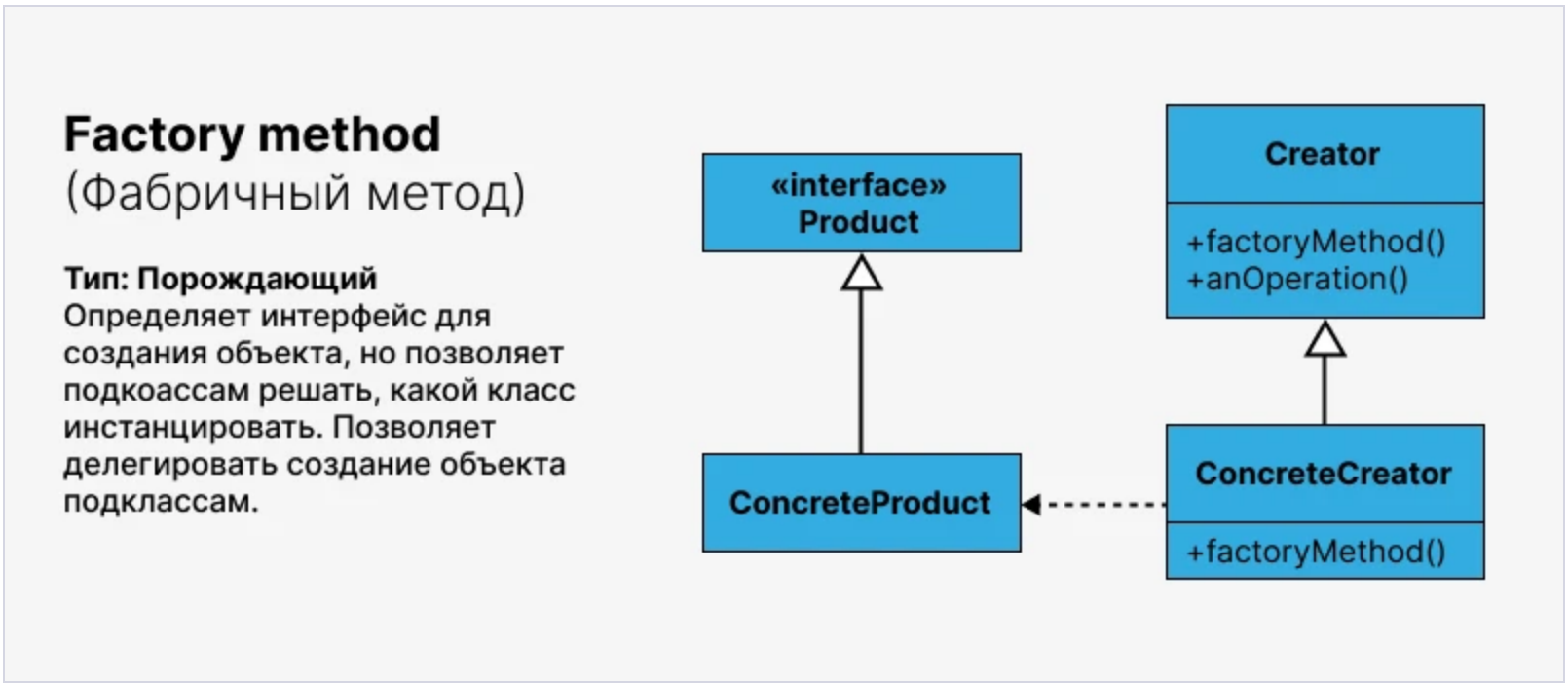
- Усложняется контроль над межпоточными гонками и задержками.
- Многопоточного “одиночку” сложно писать “из головы”: доступ к давно построенному одиночке в идеале не должен открывать мьютекс. Лучше проверенные решения.
- Конфликт двух потоков за недостроенного одиночку приведет к задержке.
- Если объект создается долго, задержка может мешать пользователю или нарушать реальное время. В таком случае его создание лучше перенести в стадию инициализации программы.
- Требуются особые функции для модульного тестирования — например, чтобы перевести библиотеку в “не-одинокий” режим и полностью изолировать тесты друг от друга.

- Требуется особая тактика тестирования готовой программы, ведь пропадает даже понятие “простейшая запускаемость”, ведь запускаемость зависит от конфигурации.

3.2 Factory [Method]

Фабричный метод (Factory Method) — порождающий шаблон проектирования, предоставляющий подклассам (классам-наследникам) интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать.

Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не конкретные классы, а манипулировать абстрактными объектами на более высоком уровне.



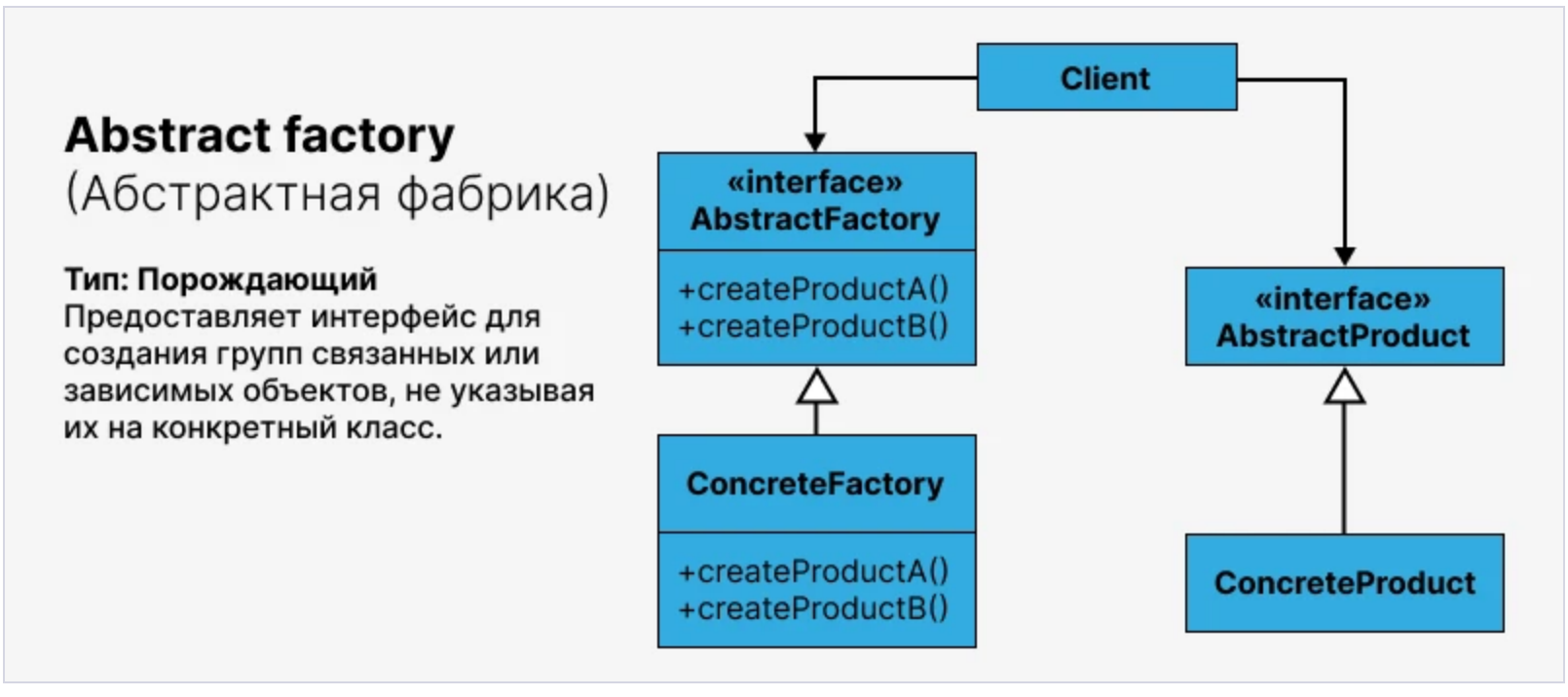
Этот паттерн определяет интерфейс для создания объекта, но оставляет подклассам решение о том, на основании какого класса создавать объект. Фабричный метод позволяет классу делегировать создание подклассов. Используется, когда:

- классу заранее неизвестно, объекты каких подклассов ему нужно создавать.
- класс спроектирован так, чтобы объекты, которые он создает, специфицировались подклассами.
- класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется определить, какой класс принимает эти обязанности на себя.

3.3 Abstract Factory

Абстрактная фабрика (Abstract factory) — порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся классы, реализующие этот интерфейс.



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

Сильные стороны:

- изолирует конкретные классы;
- упрощает замену семейств продуктов;
- гарантирует сочетаемость продуктов.

Допустим, твоя программа работает с файловой системой. Тогда для работы в Linux тебе нужны объекты LinuxFile, LinuxDirectory, LinuxFileSystem. А для работы в Windwos тебе нужны классы WindowsFile, WindowsDirectory, WindowsFileSystem.

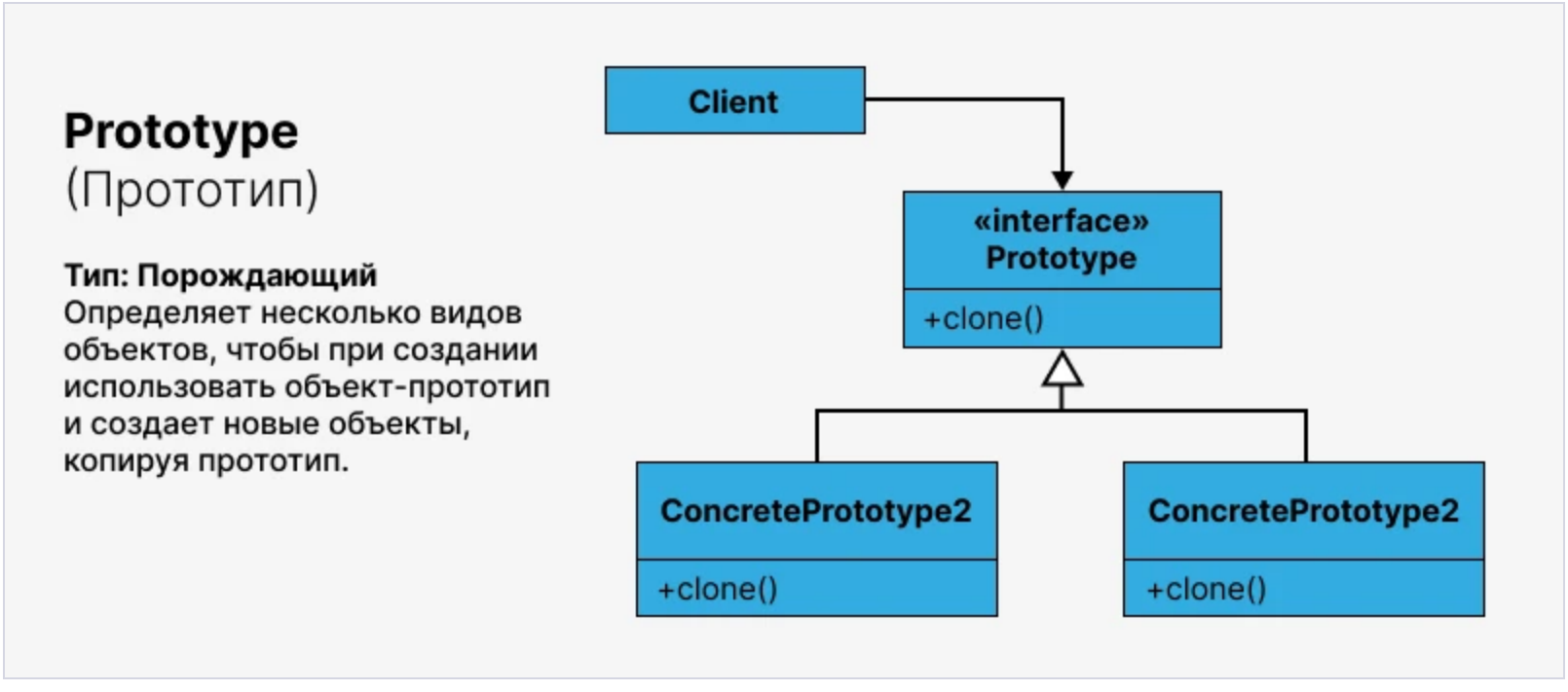
Класс Path, который создается через Path.of(), — это как раз тот случай. Path — это на самом деле не класс, а интерфейс и у него есть реализации WindowsPath и LinuxPath. А какой именно объект будет создан, скрыто от твоего кода и будет решаться во время работы программы.

3.4 Prototype

Прототип (Prototype) — порождающий шаблон проектирования.

Этот паттерн задает виды создаваемых объектов с помощью экземпляра-прототипа и создает новые объекты путем копирования этого прототипа. Он позволяет уйти от реализации и следовать принципу “программирование через интерфейсы”.

В качестве возвращающего типа указывается интерфейс/абстрактный класс на вершине иерархии, а классы-наследники могут подставить туда наследника, реализующего этот тип. Проще говоря, это паттерн создания объекта через клонирование другого объекта вместо создания через конструктор.



Паттерн используется чтобы:

- избежать дополнительных усилий по созданию объекта стандартным путем (имеется в виду использование конструктора, так как в этом случае также будут вызваны конструкторы всей иерархии предков объекта), когда это непозволительно дорого для приложения.
- избежать наследования создателя объекта (object creator) в клиентском приложении, как это делает паттерн abstract factory.

Используй этот шаблон проектирования, когда твоей программе безразлично, как именно в ней создаются, компонуются и представляются продукты:

- инстанцируемые классы определяются во время выполнения, например, с помощью динамической загрузки;
- ты хочешь избежать построения иерархий классов или фабрик, параллельных иерархии классов продуктов;
- экземпляры класса могут находиться в одном из нескольких различных состояний. Может оказаться удобнее установить соответствующее число прототипов и клонировать их, а не инстанцировать каждый раз класс вручную в подходящем состоянии.

‹ Предыдущая лекция

Следующая лекция ›

+12

Комментарии (2) популярные новые старые

JavaCoder

Введите текст комментария

Александр Уровень 85 EXPERT 31 октября 2022, 12:21

Последняя задача трешь

Ответить 0

Oleg Khilko Уровень 51 15 августа 2022, 19:20

Про фабрики писать не буду - сами поймете, лучше напишу про Singleton и Prototype.

Просто прочитайте и отложите где-то в голове, потом, когда поймете о чем я, вернитесь и поставьте лайк)

В Спринге все бины создаются по умолчанию как @Singleton, а чтобы получить возможность иметь более одного экземпляра класса - нужно написать @Prototype.

Ответить +7

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store