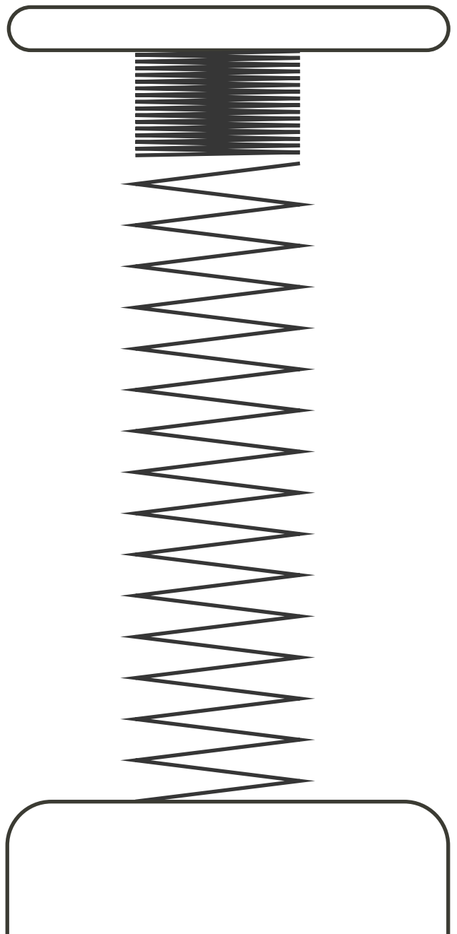
Паттерны и практики написания кода

IoC, DI, DI-Container. Часть 2

Конспект 9 | *(10)*

Dependency Injection

Dependency Injection ориентируется на создание слабой связанности.



- **Dependency Injection разрывает жёсткую связь между классом и его вспомогательными сервисами.** Благодаря ему идеально реализуется Low Coupling
- **DI улучшает тестируемость кода.** Все классы минимально знают друг о друге и тестировать их легко – все зависимости заменяемы
- **При переносе модуля в другие приложения уменьшается число классов адаптирующих код.** Это редкий случай, но слабая связанность решает и этот вопрос
- **DI позволяет проще переносить классы в другие приложения, находящиеся на верхних уровнях, а нижние — менять на другие реализации.** Такая модульность или слоистость приложения позволяет делать в системе намного больше смелых изменений, чем в закрепощённом проекте

Суть Dependency Injection во внедрении одних объектов в другие. Мартин Фаулер предлагает три основных способа, как это сделать, однако в интернете можно найти ещё один. В реальных проектах используются первые два, третий и четвёртый посмотрим справочно.

внедрение через конструктор

Constructor Injection

Используется в 99% случаев

1

ПЛЮСЫ

\+ Этот подход легко реализовать, нет никаких подводных камней

\+ Все зависимости обязательны

! Low Coupling говорит, что связь между классами должна быть минимальна. Если вы будете прокидывать пять и более объектов, то, что-то идёт не так, и сам по себе класс будет закрепощён

МИНУСЫ

× Нет возможности не внедрять в классы зависимости. Все они обязательные

— Нельзя добавить динамичности, когда сначала пробрасывается один набор классов, а затем другой

× Если аргументов слишком много, то реализация будет выглядеть некрасиво. Это будет неправильно с точки зрения Low Coupling

внедрение через сеттер-методы

Setter Injection

Такой вариант редко используется, но его предпочитают, когда нужна опциональность

2

ПЛЮСЫ

+ Есть возможность выбора, внедрять класс или нет
+ Внедрять классы можно динамически

МИНУСЫ

× Можно забыть внедрить зависимость и не получить чего-то в конечном результате

— Надо не забывать делать проверки на null и обрабатывать ситуации, если вдруг зависимость не была прокинута изначально

× Для каждой зависимости необходимо описывать свой сеттер. Код немного увеличивается в размерах

3 внедрение через интерфейс

Interface Injection

В этом подходе прокидывание зависимости происходит один к одному. Система сама догадывается о том, какую сущность необходимо прокинуть, и для этого ничего дополнительного делать не нужно

Здесь каждый интерфейс должен реализовываться одним классом. Иначе невозможно будет определить, какой класс необходимо использовать для внедрения

Плюсы и минусы этого подхода точно такие же, как у Injection через конструктор

4 внедрение через свойство

Property Injection

Пользоваться мы им не рекомендуем

Property injection похож на подход с сеттер-методами. Разница в том, что достаточно сделать свойства класса публичными и добавить в коде проверки на null. Здесь всё очень минимально: определили свойства, прокинули зависимости и сделали нужные проверки

В отличии от сеттер-методов property injection лишает вас возможности писать бизнес-логику и проверки. При наступлении необходимости перехода на сеттер-методы, вам придётся много переписывать

разница между DIP и DI

Задача Dependency Injection заключается в том, чтобы предоставлять программному компоненту внешнюю зависимость. То есть он задаёт способ, которым будут доставляться объекты в конкретный instance

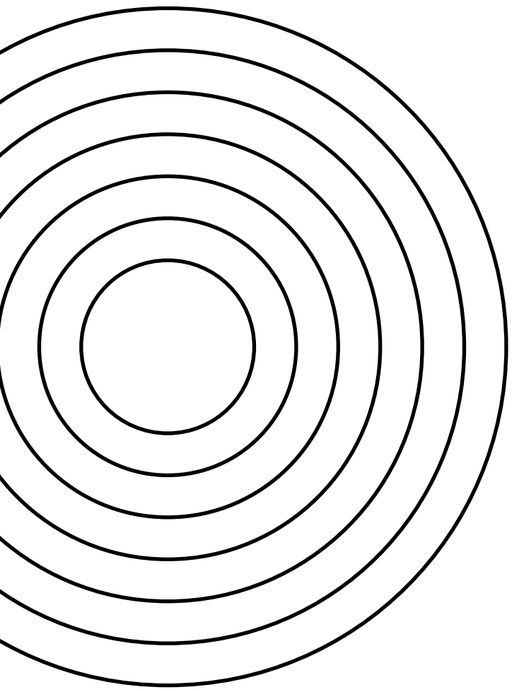
Инверсия про то, как относиться к работе с кодом, а внедрение – про способы:

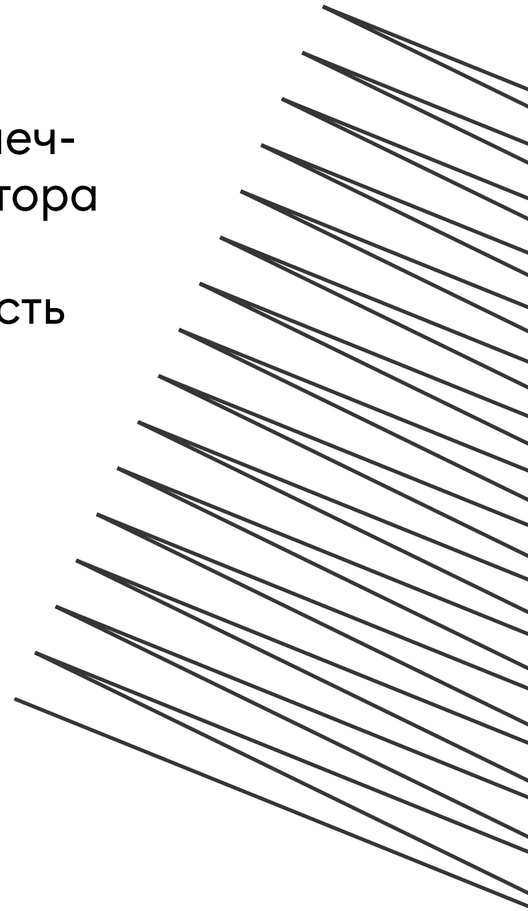
- Dependency Inversion Principle задаёт общий принцип инвертирования зависимости
- Dependency Injection рассказывает о конкретных способах внедрения зависимости

разница между Service Locator и DI

Различия между Service Locator и Dependency Injection более явные, проявляются в том, как они работают с зависимостями:

- В случае Service Locator конечные классы зависят от локатора
- В Dependency Injection зависимость внедряется на верхних уровнях
- В Service Locator конечные классы знают про существование Service Locator и зависят от него
- В Dependency Injection зависимость внедряется на верхних уровнях или в отдельной абстракции, а конечные классы ничего не знают о DI, и это очень весомый плюс






avito.tech