----------------------------------------------------------------------------------------------------------------------------

Cписок літератури:

<https://zlob.in/2013/12/chto-takoe-dependency-injection-vnedrenie-zavisimostej/> - “DI” описання з прикладами.

<http://blog.openlucius.com/en/blog/dependency-injection-drupal-8-introduction> - “DI” in Drupal.

<http://symfony.com/doc/current/components/dependency_injection/types.html#constructor-injection> - Реалізація DI в [Symfony](http://symfony.com/what-is-symfony), з прикладами коду yaml, php, xml.

<http://www.procata.com/talks/phptek-may2007-dependency.pdf> - презентація Джефа Мора.

<http://www.martinfowler.com/articles/injection.html> - введення МартінаФаулера.

GitHub репозиторій з ісходніками

<https://www.youtube.com/watch?v=FAgpDPqZyoQ> - відео-демо Введення залежності методом - конструктор.

----------------------------------------------------------------------------------------------------------------------------  
**Тема воркшопу: Dependency Injection**

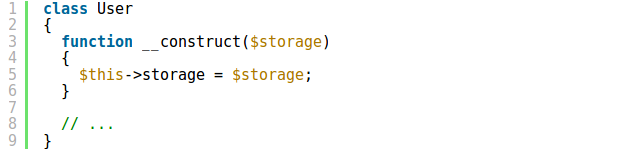
**Загальна інформація :**

Це паттерн який потрібен для реалізації слабо звязаної архітектури. Допомагає отримати більш ефективний і тестуємий код. Ми зможемо завантажувати тільки те що нам потрібно, тим самим знизимо навантаження на проект.

DI можна реалізувати такими методами:

* Через конструктор (Constructor injection)
* Через метод класса (Setter injection)
* Через властивість (Property injection)

1) Найбільш поширений спосіб передать залежність за допомогою конструктора класу у вигляді аргументу (Constructor injection)

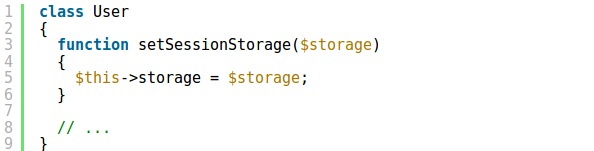


у додаванні через конструктор є декілька переваг

* ін'єкція в конструкторі може гарантувати, що клас не може буде побудований без необхідної залежності.
* Конструктор викликається один раз, коли створюється об’єкт, так що ми можемо бути впевнені, що залежність не буде мінятися за весь час роботи об’єкта.

Ці переваги означають що ін’єкція в конструкторі не дуже підходить для необов’язкових залежностей, крім того цей спосіб більш складніше використовувати з ієрархією класів. Якщо клас використовує конструктор для ін’єкції, то його розширення та переоприділення конструктора може бути проблематичним

2) Другий можливий спосіб це: додавання залежності в клас через метод сетер. (Setter injection)



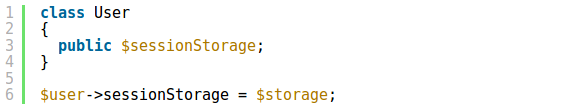
Переваги цього методу:

* Сетер ін’єкція працюватиме і з необов’язковими залежностями. Якщо вам непотрібна залежність, то просто не викликаєте сетер.
* Ви можете викликати сетер багато разів. Це особливо корисно якщо метод додає залежність в колекцію(масив), після чого ви можете мати перемінну кількість залежностей

Недостатки даного методу:

* Сетер може бути викликаний більше ніж раз, так що ви не можете бути впевнені, що залежність не замінюється протягом всього терміну служби об'єкта (за винятком додаткового написаного методу виклику, щоб перевірити, чи він вже був викликаний).
* Ви не можете бути впевнені, що сетер буде викликатися і тому вам потрібно додавати перевірки, щоб додавалися інші необхідні залежності.

3) Третій спосіб: введення залежності у властивість (Property Injection)



Тут в основному тільки недостатки використання ін’єкції Property, це аналогічна сетеру ін’єкція але з додатковими важливими проблемами:

* Ви не можете контролювати, коли залежність встановлена на все, вона може бути змінений в будь-який момент часу життя об'єкта.
* ви не можете вказувати тип аргументу методу так як ми це робили в сетері, тому ви не можете бути впевнені що залежність додається, або ж можна дописати код в класі для перевірку екземпляра класу перед його використанням

Але корисно знати що це може бути зроблено з контейнером служби, особливо якщо ви працюєте з кодом який вами не контролюється, напрклад бібліотека третьої сторони, яка використовує загальнодоступні властивості для його залежностей.

Як правило, введення через конструктор – це кращий спосіб для підключення основних залежностей, а от введення через setter – це кращий спосіб для додавання додаткових залежностей, нариклад таких як кеш.

**DI приклад**

Приклад (DI) Введення залежності методом - конструктор.

->(DI) - Dependency injection

У нас э два класи:

include 'db.php'; - клас який працюэ з базою данних, і виконує якісь запроси.

include 'user.php'; - клас який буде відповідати за роботу з "користувачів".

Класс "db" має один метод query($query), в який ми будемо передавати строкойвий запрос, і він буде нам його вертати.

Класс "user" маэ один метод, який видаэ інформацію про всіх "користувачів", в методі ми створюємо екземпляр класу "db", звертаэмось до методу "query" і передаємо запрос "SELECT \* FROM users"

В файлі index.php підключаємо файли:

include 'db.php';

include 'user.php',

І створюємо екземпляр класу "$user" і викликаємо в ньому метод "getAllUser".

В резельтаті що ми отримаємо?

Класс "User" залежить від классу "Db", таким чином якщо ми будемо робити якісь зміни в классі "Db" - це може негативно впливати на роботу классу "User".

Така реалізація може викликати проблеми в плані розширення архітектури (щерез таке звязування классів).

Для того щоб позбавитись від токої залежності ми будемо використовувати (DI).

Нам потрібно буде зробити так, щоб обект внідрявся із ВНЕ(ззовні), це нам дасть більш гибку архітектуру.

->(ДЕМОНСТРАЦІЯ РЕЗУЛЬТАТУ)

Для цього ми зробими інєкцію залежностей і протестуємо наші зміни.

Створимо в класі "user" змінну з "protected $db", і функцію-конструктор, яка буде викликатись при створенні і приймати значення (яке ми їй передамо).

також перепишемо ф-ю "getAllUsers()":

-> return $this->db->query('SELECT \* FROM users');

-ми звертаэмось до нашої змінної db, яка зберігає в собі екземпляр класу "db" і викликаємо метод query.

Таким чином ми забезпечили інєкцію залежностей, і наш класс приходить із ВНЕ(ззовні)

Тепер зробимо зміни в index.php

Створимо екземпляр классу "$db" -> "$db" = newDb(); і передамо його туди де сворюється класс $user в якості аргументу

Висновок:

З допомогою DI можна створити гибку архітектуру, не звязаний між собою код, який облегшить розробку і підтримку в майбутньому.