Projektowanie obiektowe oprogramowania
Wykład 10 – Inversion of Control
Wiktor Zychla 2012

1 Dependency Injection

Dependency Injection = zestaw technik pozwalających tworzyć struktury klas o luźniejszym powiązaniu.

Trzy kluczowe skojarzenia:

1. **Późne wiązanie** – możliwość modyfikacji kodu bez rekompilacji, wyłącznie przez rekonfigurację, „*programming against interfaces*" (DIP)
2. **Ułatwienie tworzenia testów jednostkowych** – zastąpienie podsystemów przez ich stuby/fake'i
3. ***Uniwersalna fabryka***– tworzenie instancji dowolnych typów według zadanych wcześniej reguł

2 … więc przypomnijmy sobie przykład dla Dependency Inversion Principle

Zalety:

1. **Rozszerzalność** (OCP) – teoretycznie możliwe rozszerzenia o konteksty nie znane w czasie planowania
2. **Równoległa implementacja** – dobrze zdefiniowany kontrakt zależności pozwala rozwijać oba podsystemy niezależnie
3. **Konserwowalność (maintainability)** – dobrze zdefiniowana odpowiedzialność to zawsze łatwiejsza konserwacja
4. **Łatwość testowania** - obie klasy mogą być testowane niezależnie; ta z wstrzykiwaną zależnością może być testowana przez wstrzyknięcie stuba/fake'a
5. **Późne wiązanie** – możliwość określenia konkretnej klasy bez rekompilacji

3 Twarde zależności vs miękkie zależności

Jeszcze inne spojrzenie na modularność:

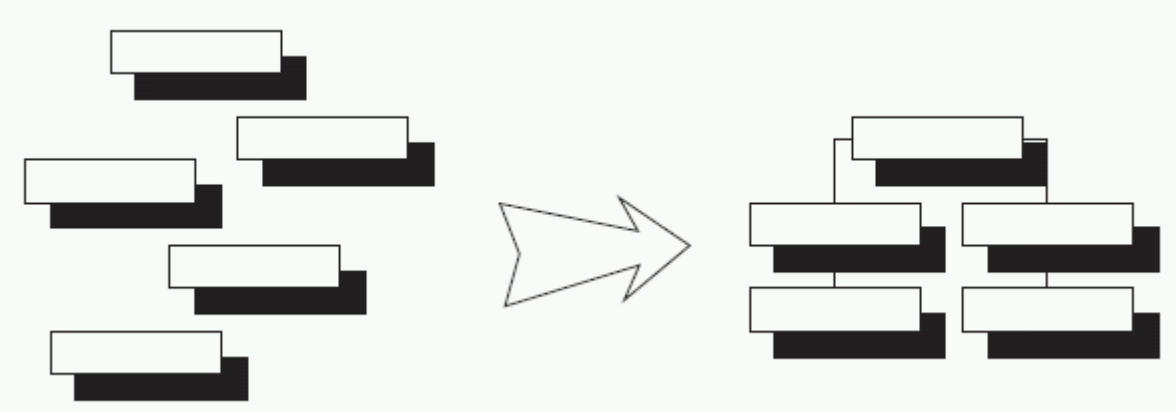
1. **Sztywna zależność** (stable dependency) – klasyczna modularność; zależne moduły już istnieją, są stabilne, znane i przewidywalne (np. biblioteka standardowa)

2. **Miękka zależność** (volatile dependency) – modularność dla której zachodzi któryś z powodów wprowadzenia spoiny:
    a. Konkretne środowisko może być konfigurowane dopiero w miejscu wdrożenia (późne wiązanie)
    b. Moduły powinny być rozwijane równolegle
3. **Spoina** (seam) – miejsce, w którym decydujemy się na zależność od interfejsu zamiast od konkretnej klasy

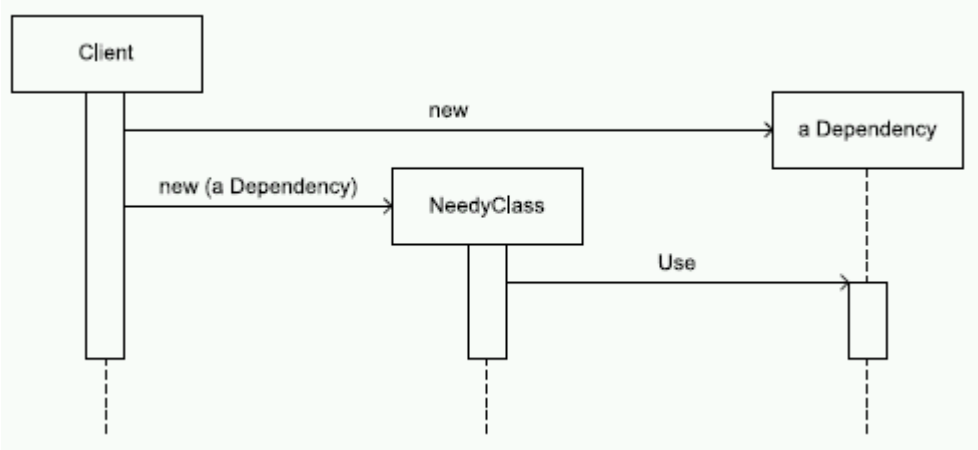
Uwaga. O ile zastosowanie technik DI powoduje wprowadzenie miękkich zależności w miejscach spoin, o tyle zwykle zależności do samych ram DI mają charakter sztywny.

4 Kluczowe funkcjonalności ram DI na przykładzie Unity

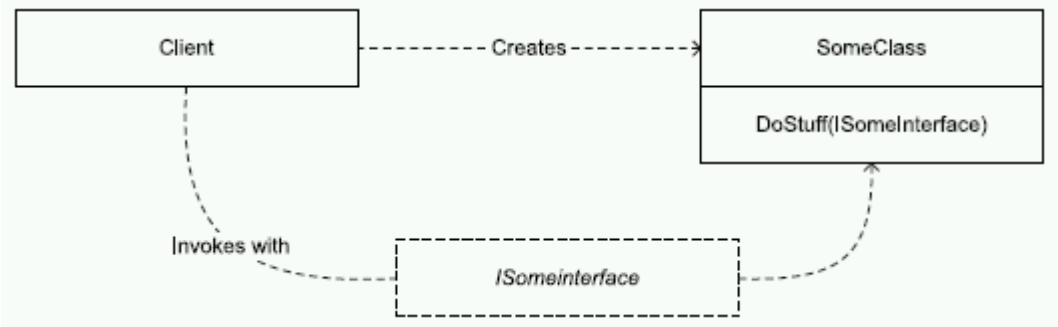
4.1 Składanie obiektów



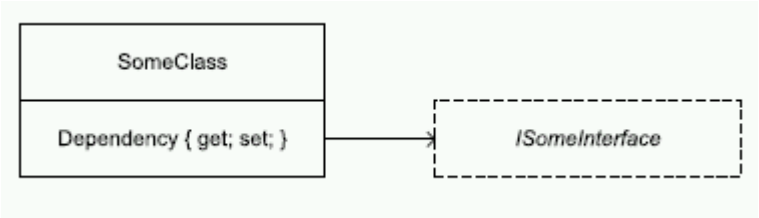
1. Kontener
2. Rozwiązywanie zależności sztywnych
3. Rozwiazywanie zależności miękkich
4. Rozwiązywanie instancji
5. Rozwiązywanie grafu zależności
6. Wstrzykiwanie przez konstruktor – najdłuższy lub wskazany ([InjectionConstructor]) – zapewnienie że zależność jest zawsze dostępna



7. Wstrzykiwanie przez metodę – atrybut [InjectionMethod] – zapewnienie że w różnych kontekstach (metodach) wstrzykiwane mogą być inne zależności

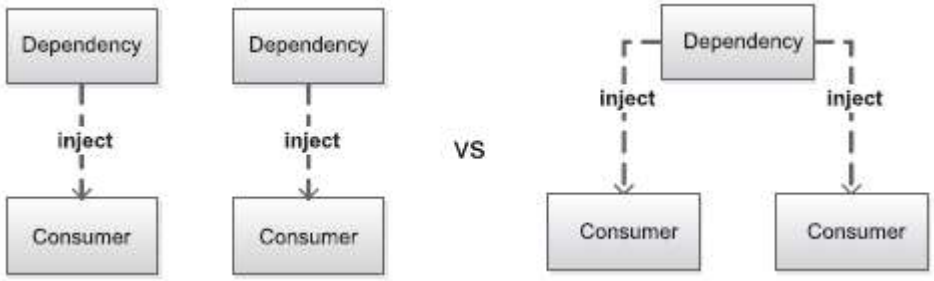


8. Wstrzykiwanie przez właściwości ([Dependency]) – zapewnienie że domyślna zależność jest dostępna, ale może być zmodyfikowana



9. „Budowanie" obiektu wyprodukowanego na zewnątrz
10. Rejestracja metody fabrykującej – zapewnienie możliwości tworzenia zależności przez dowolną metodę fabrykującą

4.2 Zarządzanie czasem życia obiektów

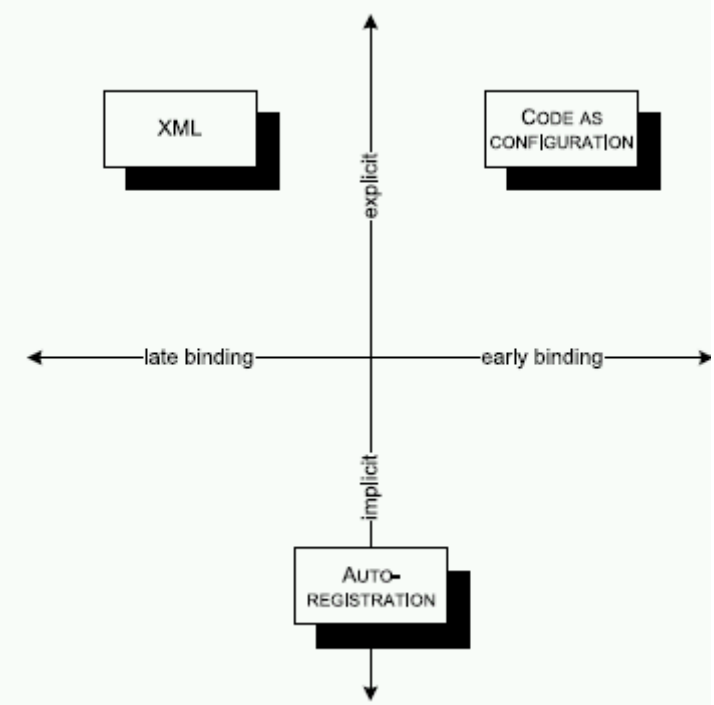


http://msdn.microsoft.com/en-us/library/ff660872(PandP.20).aspx

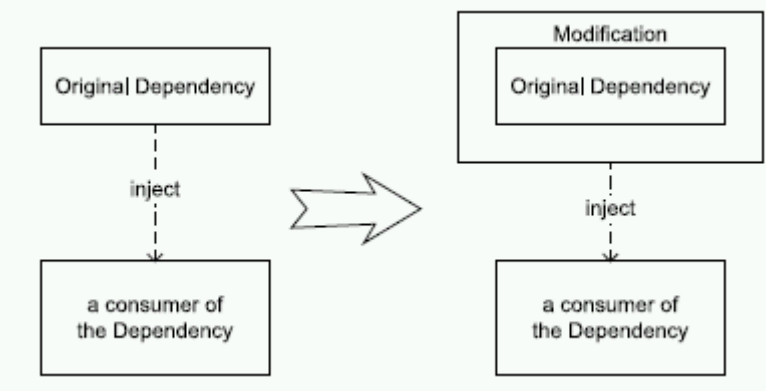
1. Transient – ulotne
2. ContainerControlled – singletony

3. Hierarchical – singletony, ale inne w dziedziczonych kontenerach
4. PerThread – inny obiekt per wątek
5. Custom (przykład)

4.3 Konfiguracja kontenera



4.4 Przechwytywanie żądań (proxies)

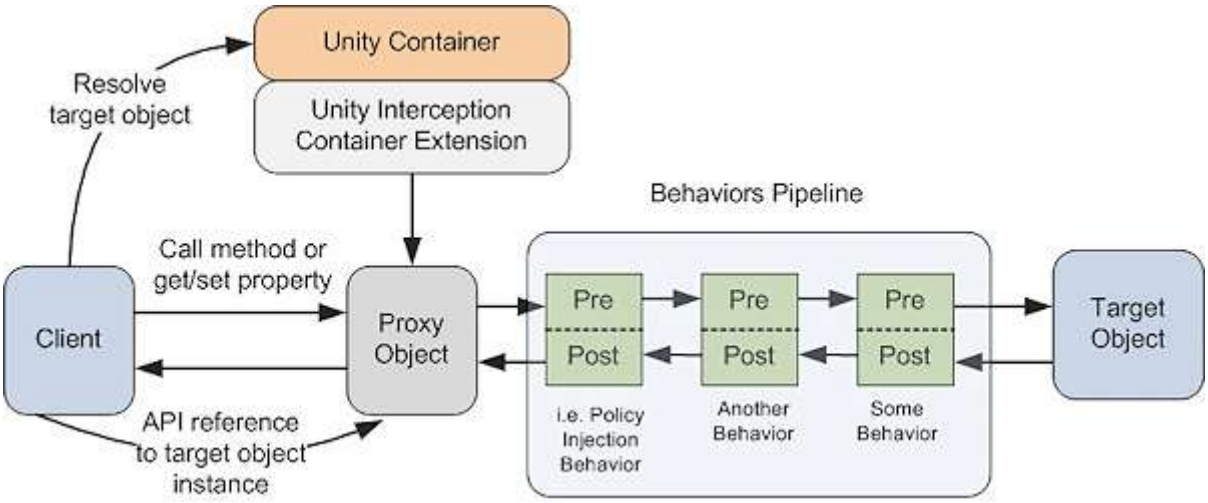


Typowe zagadnienia przekrojowe (cross-cutting concerns):

1. Audytowanie
2. Logowanie
3. Monitorowanie wydajności
4. Bezpieczeństwo
5. Cache'owanie
6. Obsługa błędów

Frameworki DI często obsługują podzbiory tak zdefiniowanego AOP. Przykład w Unity:

1. InterfaceInterceptor – tworzy proxy opakowujące dla rozwikływanego interfejsu, pozwala przechwycić tylko metody interfejsu
2. VirtualMethodInterceptor – tworzy proxy opakowujące dla rozwikływanej klasy, pozwala przechwycić tylko metody wirtualne



5 ServiceLocator

Service Locator = schowanie singletona kontenera DI za fasadą, pozwalającą z dowolnego miejsca aplikacji na rozwiązanie zależności do usługi. Pozwala znacznie zredukować jawne zależności miedzy klasami.

Uwaga! Broń obosieczna – to że zależności nie widać (nie są jawne) nie znaczy że ich nie ma!
Uwaga! Niektórzy uważają SL za antywzorzec, ale można mieć własne zdanie na ten temat (wydaje się, że uważanie SL za antywzorzec jest obecnie „trendy").

Przykład w Unity.

6 Przykład z życia wzięty – nieduży system o architekturze zbudowanej dookoła IoC

7 Literatura

1. Dhanji R. Prasanna – Dependency Injection (2009, Java)
2. Mark Seemann – Dependency Injection in .NET (2012, C#) (źródło ilustracji i planu prezentacji)