Inverse of Control, Dependency Injection, Microsoft Unity

Inversion of Control

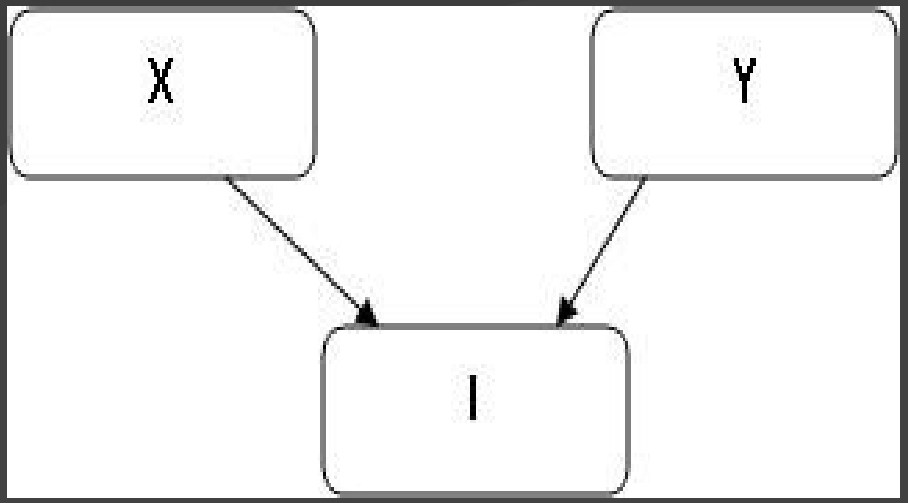
IoC jest abstrakcyjnym pojęciem opisującym aspekt konstrukcyjny architektury oprogramowania projektu, w którym przepływ sterowania jest odwrócony w stosunku do programowania proceduralnego.

Przykład

W klasycznym podejściu proceduralnym Klasa X Potrzebuje klasy Y więc tworzy jej instację oraz używa jej metod

```
public class TextEditor
{
    private SpellChecker checker;
    public TextEditor()
    {
        checker = new SpellChecker();
    }
}
```

Przykład



W tym przypadku klasa Y implementuje interfejs I a klasa X używa I nie wiedząc nic o klasie która ten interfes implementuje (czyli Y).

```
public class TextEditor
{
    private ISpellChecker checker;
    public TextEditor(ISpellChecker checker)
    {
        this.checker = checker;
    }
}
```

Dependency Injection

Wstrzykiwanie zależności jest przykładem odebrania części kontroli od klasy do której wstrzykujemy zależność.

Managing Dependencies Between Components

Aplikacje bazowane na bibliotece Prism są aplikacjami modułowymi które zawierają wiele niezależnych typów oraz serwisów. Potrzebują one sposobu aby oddziaływać oraz komunikować się między sobą aby zapewnić funkcjonalność biznesową. Prism używa do tego kontenera DI.

Funkcje kontenera

Kontener DI tworzy instacje klas oraz zarządza ich czasem życia bazując na jego konfiguracji. W trakcie tworzenia kontener wstrzykuje zależności do klas.

Zalety używania kontenerów

- Komponent (moduł) nie musi martwić się o tworzenie oraz zarządzanie zależnościami.
- Możliwość zmiany zależności bez zmian w komponencie
- Łatwiejsze testowanie.
- Prostrze utrzymywanie projektu i dodawanie nowych funkcjonalności

Zalety w aplikacji opartej na Prism

- Rejestrowanie i wstrzykiwanie modeli i modeli widoków
- Wstrzykiwanie serwisów konstrukcyjnych jak region manager i event aggregator

Wybór kontenera

Prism zawiera implementacje kontenerów Unity oraz MEF. Można używać innych przez interfejs IServiceLocator lub nie. My pokażemy użycie Unity.

Kiedy nie używać kontenera?

- Tworzenie batdzo dużej ilość obiektów. (Wielokąty przy renderowaniu grafiki 3D).
- Wiele głębokich zależności może znacznie spowolnić program.
- Jeśli komponent nie ma żadnych zależności.
- Jeśli komponent ma stałe, niezmienne zależności

Singleton czy instancja

- Jeśli komponent zawiera współdzielony stan lub jest serwisem globalnym (np logger) to singleton
- Jeśli potrzebujemy nowej instancji za każdym razem kiedy ją wstrzykujemy (np model widoku dla modelu)

Plik konfiguracyjny czy kod

- Centralne zarządzanie konfiguracją.
- Rejestracja w zależności od pewnych warunków