Projektowanie obiektowe oprogramowania
Zestaw 12

Inversion of Control (3)

2012-05-02

Liczba punktów do zdobycia: **6/74**
Zestaw ważny do: 2012-05-22

Uwaga! Kontynuacja pracy nad silnikiem Inversion of Control na identycznych zasadach. W szczególności obowiązkową częścią każdego zadania są testy jednostkowe, nawet jeśli nie wspomina się o tym w treści zadań.

1. (**2p**) (**Wstrzykiwanie właściwości**)

   Silnik IoC rozbudować o możliwość wstrzykiwania właściwości (propercji).

   Ściślej - kontener w trakcie rozwikływania zależności obiektu dodatkowo analizuje właściwosci i te, które są opatrzone atrybutem `[DependencyProperty]` i mają publiczny akcesor `set` próbuje rozwikływać i wstrzykiwać tak samo, jak robi to z parametrami konstruktora.

   Kolejność wstrzykiwania właściwości nie ma znaczenia.

```
public class A {

  public A( B b ) { }

  [DependencyProperty]
  public C TheC { get; set; }
}

public class B {
}

public class C {
}

SimpleContainer c = new SimpleContainer();
A a = c.Resolve<A>();

// tworzy nową instancję A z B wstrzykniętym przez konstruktor i C wstrzykniętym przez właściwość.
```

2. (**2p**) (**Uzupełnianie zależności istniejącego obiektu**)

   Silnik IoC rozbudować o możliwość uzupełaniania zależności istniejących obiektów, zbudowanych poza kontenerem.

```
public class SimpleContainer
{
    ...
    public void BuildUp<T>( T Instance );
}

SimpleContainer c = new SimpleContainer();
```

```
A theA = ....; // obiekt theA SKĄDŚ pochodzi, ale nie z kontenera
c.BuildUp( theA );

// wstrzykuje do theA zależności przez właściwości, tu: TheC
```

3. (**2p**) (**Lokalizator usług**)

   Dodajemy do silnika funkcjonalność lokalizatora usług (*Service Locator*). Typowo lokalizator jest singletonem, z zadaną logiką rozwikływania kontenera (w szczególności kontener też może być singletonem), a klient używa lokatora do rozwikłania zależności do usług (implementacji).

   Lokalizator pozwala zmniejszyć zależności między komponentami - zamiast wymagać zależności do usług (nawet wstrzykiwane!), obiekt może wyłącznie znać odwołanie do lokalizatora, a w przypadku gdy lokalizator jest singletonem obiekt w ogóle nie musi wymagać żadnych zależności.

```
public delegate SimpleContainer ContainerProviderDelegate();

public class ServiceLocator {

    public static void SetContainerProvider( ContainerProviderDelegate ContainerProvider ) {
        ...
    }

    public static ServiceLocator Current
    {
        get {
            ... singleton ...
        }
    }

    public T GetInstance<T>();
}

    // Przykład 1, rejestrowanie lokatora

    SimpleContainer c = new SimpleContainer();
    ContainerProviderDelegate containerProvider = () => c;

    ServiceLocator.SetContainerProvider( containerProvider );

    /* zapamiętano funkcję rozwikłującą referencję do kontenera - to zawsze będzie TEN sam kontener */

    // Przykład 2, rejestrowanie lokatora

    ContainerProviderDelegate containerProvider = () => new SimpleContainer();
    ServiceLocator.SetContainerProvider( containerProvider );

    /* zapamiętano funkcję rozwikłującą referencję do kontenera - to zawsze będzie NOWY kontener */

    // Przykład 3, kod kliencki

    class Foo

        void Bar() {

            IService service = ServiceLocator.Current.GetInstance<IService>();

            // GetInstance używa zarejestrowanej metody do rozwikływania kontenera
            // do pobrania kontenera i rozwikłania zależności
            // Tym razem jednak IService nie musiało być wstrzykiwane
        }
    }
```

   Zwrócić uwagę na pewien szczegół implementacyjny - klient lokalizatora może zechcieć otrzymać referencję do kontenera, którego wewnętrznie używa lokalizator.

```
SimpleContainer container = ServiceLocator.Current.GetInstance<SimpleContainer>();
```

Taka potrzeba ma swoje uzasadnienie - zwyczajowo interfejs kontenera jest szerszy niż interfejs lokalizatora (np. kontener potrafi uzupełniać zależności w istniejących obiektach, patrz poprzednie zadanie).

Poprawnie obsłużyć ten przypadek.

Wiktor Zychla