Projektowanie obiektowe oprogramowania
Zestaw 9

Inversion of Control

2013-05-07

Liczba punktów do zdobycia: **5/62**
Zestaw ważny do: 2013-05-28

Uwaga! Programujemy w parach lub indywidualnie. W przypadku pary - obie osoby otrzymują komplet punktów.

1. (**5p**) (**Rdzeń silnika DI**) Należy przygotować implementację rdzenia prostego silnika Dependency Injection/Inversion of Control.

```
public class SimpleContainer
{
  public void RegisterType<T>( bool Singleton ) where T : class;
  public void RegisterType<From, To>( bool Singleton ) where To : From;

  public T Resolve<T>();
}
```

Podstawową metodą dla klienta kontenera jest metoda `Resolve`, której klient używa do tworzenia instancji obiektów:

```
SimpleContainer c = new SimpleContainer();

Foo fs = c.Resolve<Foo>();
```

Zakłada się, że klasa, której instancję rozwikłuje kontener musi mieć konstruktor bezparametrowy. Kontener przy wywołaniu metody Resolve tworzy i zwraca nową instancję typu podanego jako parametr generyczny. Klient może doszczegółowić politykę tworzenia nowych instancji za pomocą metody `RegisterType`.

Jej pierwsze przeciążenie służy do poinformowania kontenera o konieczności zastosowania polityki singletonu do zarządzania czasem życia obiektów.

```
SimpleContainer c = new SimpleContainer();

c.RegisterType<Foo>( true );

Foo f1 = c.Resolve<Foo>();
Foo f2 = c.Resolve<Foo>();

// f1 == f2
```

Drugie przeciążenie służy do wyboru pożądanej implementacji klasy bazowej, abstrakcyjnej lub interfejsu:

```
SimpleContainer c = new SimpleContainer();

c.RegisterType<IFoo, Foo>( false );

IFoo f = c.Resolve<IFoo>();
// f ma typ Foo

c.RegisterType<IFoo, Bar>( false );

IFoo g = c.Resolve<IFoo>();
// g ma typ Bar
```

Uwaga. Jeżeli kontener nie ma zarejestrowanego typu interfejsu, a klient żąda obiektu typu interfejsu, powinien dowiedzieć się o tym jakimś rozsądnym wyjątkiem:

```
SimpleContainer c = new SimpleContainer();

IFoo f = c.Resolve<IFoo>();

// jakiś rozsądny wyjątek
```

Wiktor Zychla