Wzorce projektowe w języku C# oraz platformie .NET

Spis treści

1. Krótki zbiór użytecznej informacji
2. Najbardziej użyteczne wzorce z wewnątrz
3. Tworzenie web aplikacji z użyciem wzorców projektowych

Początek rozdziałów

## Rozdział 1: Krótki zbiór użytecznej informacji

W tym rozdziale:

* .NET oraz C#
* Pojęcia: interfejs, klasa, obiekt
* Wstęp do programowania obiektowego
* Wstęp do UML
* Wstęp do wzorcu projektowego
* Wstęp automatów skończonych
* Tablica wzorców, klasyfikacja, przeznaczenie, sposób wykorzystania w .NET
* Wzorce według częstotliwości użycia
* Wzorce według ważności

Большие программные системы достаточно сложно проектировать и сопровождать, поэтому обычно, для облегчения работы, большие системы принято разбивать на подсистемы. Главная задача, решаемая при проектировании – это сокращение количества связей отношений между классами или если сказать по-другому – уменьшение зависимостей классов друг от друга. Зависимость – это технический термин, который описывает количество имеющихся связей отношений. Если на диаграмме мы видим, что у некоторого класса имеется много связей отношений с другими классами - мы говорим, что такой класс сильно зависит от других классов. А если сказать правильнее – работа экземпляра такого класса будет сильно зависеть от работы экземпляров других классов. Зависимости классов образуют так называемые привязки. Привязка – это логический термин, который заставляет программиста задуматься над смыслом зависимости. С точки зрения ООП, привязки бывают двух типов: хорошие - бизнес привязки и плохие - технические привязки. Бизнес привязки выражают требования бизнеса, например, класс Customer связан связью отношения ассоциации с классом Order. Технические привязки выражают системные требования, например, класс Customer связан связью отношения ассоциации с классом DataSet. Важно заметить, что без технических привязок не обойтись, и техническая привязка может стать условно-хорошей тогда и только тогда, когда зависимые сущности находятся в разных слоях системы (например, класс Customer располагается в Business Layer, а DataSet располагается в слое Data Layer), в таком случае при анализе бизнес логики программной системы можно пренебречь связями, ведущими в нижележащий слой. Имеются привязки, которые не попадают под классификацию хороших и плохих привязок – это «вздорные» привязки, например, класс Order связан связью отношения ассоциации с классом Customer, или класс Customer связан связью отношения агрегации с классом Order. Понятно, что связь отношения ассоциации символизирует знание о чем-то или о ком-то и звучит «знаю о или использую это», а связь отношения агрегации символизирует составление из частей и звучит «состою из или включаю в себя». Поэтому приведенные выше примеры «вздорных» связей будут звучать так: Order знает о Customer (заказ знает о человеке), или Customer состоит из Order (человек состоит из заказа). Связь Order знает о Customer, можно заменить, например, на Order состоит из CustomersDetails. Так же класс Order

134

может использовать при построении заказа, например, калькулятор расчета скидок DiscountCalculator, такая связь выражается ассоциацией, а не агрегацией, так как не очень удачным решением будет создавать странную гибридную сущность - «заказо-скидко-калькулятор».

Понятно, что без привязок не обойтись, но нужно стараться минимизировать их количество и при этом сохранять логическую целостность смысла моделируемого процесса. Возможно ли выразить количественно силу зависимости? Конечно, сила зависимости выражается при помощи такого понятия как связанность (coupling). Связанность – есть мера зависимости. Связанность имеет более десятка метрик, позволяющих получить численное значение силы зависимости. Детальное рассмотрение метрик связанности не входит в контекст данной книги.

### Podrozdział 1.1: Wiedza wstępna

Żeby ładnie sobie radzić z pojęciem wzorcu projektowego oraz używać wzorce projektowe skutecznie, koniecznym jest rozumienie podstaw programowania obiektowego, języka programowania oraz używanego frameworku. Także dobrą zaletą jest umiejętność odczytu diagram UML oraz znajomość podstaw automatów skończonych.

{ PLACEHOLDER DLA OOP, UML, SM, DP basics }

## Basic Design Patterns and Groups

Design patterns fall into groups, based on the type and aims of the pattern. For example, some patterns provide presentation logic for displaying specific views that make up the user interface. Others control the way that the application behaves as the user interacts with it. There are also groups of patterns that specify techniques for persisting data, define best practices for data access, and indicate optimum approaches for creating instances of objects that the application uses. The following list shows some of the most common design patterns within these groups:

**Presentation Logic**

* + Model-View-Controller (MVC)
  + Model-View-Presenter (MVP)
  + Use Case Controller

**Host or Behavioral**

* + Command
  + Publish-Subscribe / Observer
  + Plug-in / Module / Intercepting Filter

**Structural**

* + Service Agent / Proxy / Broker
  + Provider / Adapter

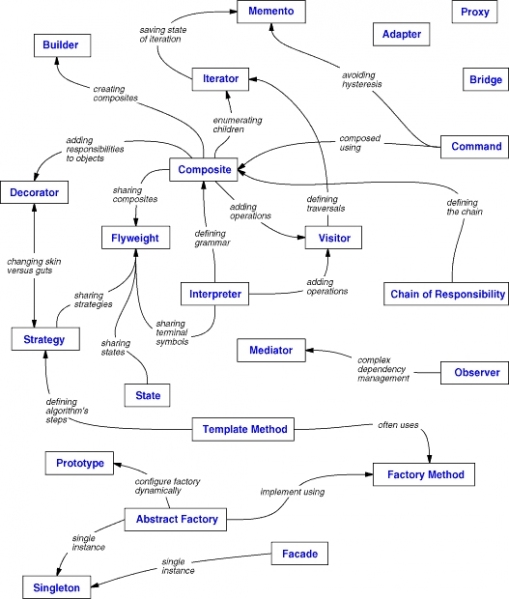
**Creational**

* + Factory / Builder / Injection
  + Singleton

**Persistence**

* + Repository





## Rozdział 3: Krótki zbiór użytecznej informacji

Użyta literatura

* Design Patterns via C# - Aleksandr Szewczuk, Dimitri Ochrimenko, Andrzej Kasjanow
* Katalog GoF – banda 4-ch