Die S.O.L.I.D-Prinzipien für C# Entwickler

DWX Developer Week 2016

Thomas Claudius Huber
@ThomasClaudiusH

BASEL ▪ BERN ▪ BRUGG ▪ DÜSSELDORF ▪ FRANKFURT A.M. ▪ FREIBURG I.BR. ▪ GENEVA
HAMBURG ▪ COPENHAGEN ▪ LAUSANNE ▪ MUNICH ▪ STUTTGART ▪ VIENNA ▪ ZURICH

trivadis
makes IT easier.

Thomas Claudius Huber

Principal Consultant @ Trivadis AG
Cloud Solutions

- Developer, Trainer, Architect
- Microsoft MVP for Windows Development

Spezialisiert in WPF, Angular 2, C#, .NET, Azure

What else:

- Pluralsight-Autor, Buchautor
- Fussball, Skaten, Fitness, Gitarre

MVP Microsoft® Most Valuable Professional

trivadis makes IT easier.

Agenda

1. Die S.O.L.I.D.-Prinzipien
2. Umsetzung in C#
3. Summary

1. Die S.O.L.I.D.-Prinzipien

S.O.L.I.D.-Prinzipien

- Fünf Prinzipien für wartbaren Code
- Nicht an eine Technologie gebunden

Single Responsibility	**O**pen / Closed	**L**iskov Substitution	**I**nterface Segregation	**D**ependency Inversion

Wo hat das Ganze seinen Ursprung

- S.O.L.I.D.-Acronym wurde von Michael Feathers eingeführt
  - Basierend auf den ersten 5 objektorientierten Prinzipien von Robert C. Martin
- Werden die fünf Prinzipien angewendet, soll der Code wartbarer und leicht erweiterbar sein

2. Umsetzung in C#

Ein Demo-Projekt

- Ein einfacher File-Reader
- Let’s «hack» it down

Demo

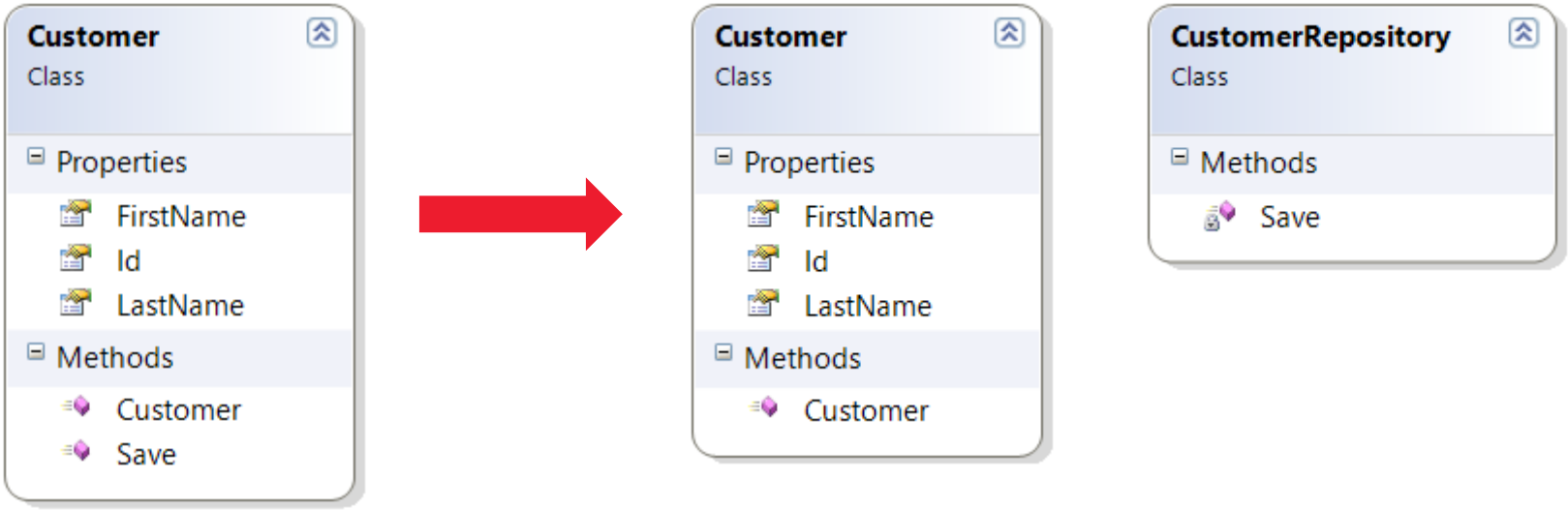
Single Responsibility Principle (SRP)

"There is one and only one reason to change a class"

Demo

Single Responsibility Principle (SRP)

Weitere Beispiele



Single Responsibility Principle (SRP)

- Nur eine Verantwortlichkeit
- Gilt für alle Objekte: Assemblies, Klassen, Methoden
- Sorgt für wartbare, kleine Klassen

Open/Closed Principle (OCP)

"Software entities (classes, modules, functions, etc.) should be **open for extension**, but **closed for modification**"

Open/Closed Principle (OCP)

Open for Extension

Die Klasse kann um neue Anforderungen erweitert werden

Closed for Modification

Bestehender Code wird nicht verändert

Open/Closed Principle (OCP)

- Änderungen durch Hinzufügen von neuem Code
- Änderungen NICHT durch Änderung von altem Code
- Der Schlüssel liegt in der Abstraktion

Demo

Open/Closed Principle (OCP)

- Bestehender Code wird nicht geändert
- Unit Tests bleiben grün
- Manchmal schwierig was virtual ist und was nicht

Liskov Substitution Principle (LSP)

"Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program"

Liskov Substitution Principle (LSP)

- Eine Erweiterung von OCP
- Sicherstellen, dass abgeleitete Klassen das Verhalten der Basisklasse nicht verändern

Demo

Interface Segregation Principle (ISP)

"Clients should not be forced to depend upon interfaces that they do not use"

Interface Segregation Principle (ISP)

- Grosse Interfaces = mehr Abhängigkeiten
- Interface Pollution (Clients müssen Member implementieren, welche sie nicht brauchen)
- Lösung: Keine „fat“ Interfaces, stattdessen viele kleine Interfaces

Demo

Dependency Inversion Principle (DIP)

"Depend upon Abstractions. Do not depend upon concretions"

Dependency Inversion Principle (DIP)

- Abstraktion durch Interfaces / abstrakte Basisklassen
- Code lässt sich ohne Implementierungen testen
  - Mocking von Interfaces
- Üblich wird zum Auflösen der Interfaces ein DI-Container genutzt

Demo

3. Summary

Anzeichen auf «stinkenden» Code

- If-Statement, welches Typen unterscheidet -> OCP
- Interface-Methoden, welche in vielen Fällen nicht implementiert werden müssen -> ISP
- FAT Interfaces -> ISP
- Methodennamen wie: UpdateAndSaveCustomer(), VerifyAndSaveData() -> SRP
- Klassennamen wie CustomerManager -> SRP

S.O.L.I.D.-Summary

- Hilft wartbaren Code zu schreiben
- Projekte bekommen eine gute Struktur

- Es sind lediglich Prinzipien
  - Nicht übertreiben. ☺

Fragen?

Thomas Claudius Huber
@thomasclaudiush
thomas.huber@trivadis.com

www.thomasclaudiushuber.com

trivadis
makes IT easier.