codelution

The SOLID Principles of Agile OO Design

codelution

Ziel

- Wartbarer Code
- Leicht erweiterbarer Code
- Weniger Bugs

codelution

SOLID

Single Responsibility Principle

Open Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

codelution

Single Responsibility

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

Gründe für SRP

- Bei Änderungen an einer Anforderung will man einfach die Klasse austauschen, das geht nicht wenn mehrere Concerns in einer Klasse vermischt sind
- Wenn Concerns vermischt: isoliertes Testen von Anforderungen nicht möglich

- Bsp:
  - Manager-Klassen
  - Controller-Klassen
  - Repository/ Adapter Klassen (Grenzfall)

- Refactoring: Extract Class

codelution

Single Responsibility

Beispiel

Single Responsibility

- != „A class has only one responsibility“

- → There can be only one requirement that, when changed, will cause a class to change...

codelution

Open Closed Principle

A class should be open for extensions but closed for modifications

Gründe für OCP

- Einmal getestete Klasse soll nicht immer wieder verändert werden müssen
- „Once a class is done, it is done“

Open Closed Principle

- Wie für Erweiterung öffnen aber gleichzeitig vor Modifikationen schützen??

- Statt Änderungen, neue Klasse erzeugen
- Häufig: Vererbung, Polymorphie nutzen

codelution

Open Closed Principle

Beispiel

codelution

Liskov Substitution Principle

Sei q(x) eine beweisbare Eigenschaft von Objekten x des Typs T, dann soll q(y) des Typs S wahr sein, wobei S ein Untertyp von T ist.

Liskov Substitution Principle

- Eine Unterklasse darf keine logischen Probleme erzeugen wenn Sie anstelle Ihrer Superklasse verwendet wird
- Eher kein „technisches Prinzip“
- Sehr leicht zu verletzen

Liskov Substitution Principle

- D.h. folgendes ist erlaubt:
  - Kovarianz der Methoden-Rückgabewerte in der Subklasse
  - Kontravarianz der Methoden-Parameter in der Subklasse
  - Nur Exceptions werfen die Ableitungen von Exceptions sind die die Superklasse sowieso schon wirft

  - Kontravarianz Parameter: die Subklasse darf in ihren Methoden nur Parameter akzeptieren die Basis-Typen derjenigen Typen sind, welche die Superklasse an dieser Stelle akzeptiert hat. Logisch: ansonsten könnte man B nicht anstelle von A verwenden. Alles was an die Methoden der Superklasse übergeben wurde muss auch an die Methoden der Subklasse übergeben werden können.
  - Kovarianz Rückgabewerte: der Rückgabewerte der Methode darf vom gleichen Typ derjenige der Superklassen-Methode oder einer Ableitung davon sein

Liskov Substitution Principle

- Einfacher ausgedrückt: „Man sollte anstelle einer Superklasse immer eine Subklasse verwenden können“
- Kovarianz und Kontravarianz Restriktionen in C#.NET per default (Compiler lässt nichts anderes zu)

- Sehr leicht zu verletzen

codelution

Liskov Substitution Principle

Beispiel

codelution

Interface Segregation Principle

Interface Segregation Principle

- Zusammenhang mit Single Responsibility
- Interfaces nach Verantwortlichkeiten trennen

➢Bei der Realisierung muss nichts unnötig implementiert werden

➢Übersichtlicher und sauberer bei der Verwendung

codelution

Dependency Inversion Principle

Dependency Inversion Principle

- Keine Abhängigkeiten zur Implementierung
- Abhängigkeiten nur zu Interfaces („*depend on abstractions*“)
- Tell don‘t ask

- != Dependency Injection
- Aber: DI wird in diesem Zusammenhang verwendet um die Abhängigkeiten aufzulösen

codelution

Fragen?

Stefan Dirschnabel

Software Developer / Trainer / Berater

stefan.dirschnabel@codelution.de

www.codelution.de