Clean Code & Design Pattern

Tag 2

Zeitplan

12:00

Start

Pause

13:15 & 14:45

16:00

Ende

Review Tag 3

Code auslagern

- Methode → Parameter einführen
- Klasse → Interface einführen
- Projekt (csproj) → Factory einführen
- Solution (sln) → nuget einführen

Interfaces

- Interfaces für Abstraktionen
- Extract and Inject
- ZIIP (Zero Impact Injection Pattern)

Design Pattern

- Factory (Method/Abstract) → On-Demand Erzeugung
- Singleton Pattern (Lazy<T>, Double Checked Locking)
- Service Locator Pattern

Clean Code Principles

- Broken Window Principle
- Version Control System
- Separation of Concerns

Wrapper & More

Wie dürfen wir es ihnen einpacken?

Agenda

Decorator Pattern
...und ein bisschen
Proxy Pattern

01

Adapter Pattern

02

Facade Pattern

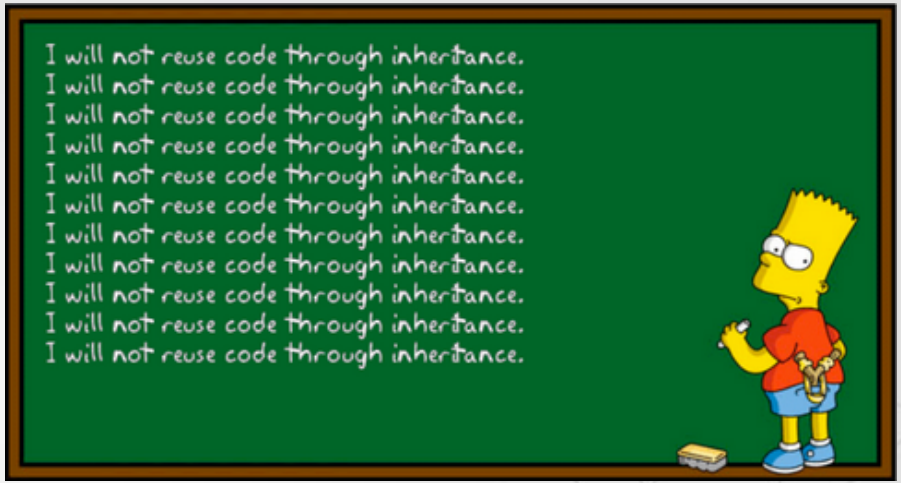
03

01

Decorator Pattern

Favour Composition over Inheritance

- Komposition statt Vererbung
- Entkoppeln der Klassen
- Verändern des Verhaltens zur Laufzeit



Decorator Pattern

- Flexible Alternative zur Unterklassenbildung
- Klasse um zusätzliche Funktionalitäten zu erweitern
- Zum Beispiel: Logging, abgerundete Fenster

- Dekorierer hat die gleiche Schnittstelle wie die zu dekorierende Klasse
- Transparent aus Sicht des Aufrufenden

Aufgabe

Logging implementieren

Projekt “PdfTools”

Die Strategies sollen die Parameter loggen

LoggingStrategyDecorator: ICommand
Einbau in der Factory

02

Adapter PAttern

Extract 3rd Party Dependency

- Projekt für die Contracts anlegen (Schnittstellen)
- Projekt für die konkrete Implementierung anlegen
- Contracts sind die gemeinsame Referenz

Adapter Pattern

- Auch “Hüllenklasse” oder “Wrapper”
- Übersetzung Schnittstelle (intern) nach Schnittstelle/Klasse (extern)
- Kompatibilität von heterogenen Bibliotheken

- Der Adapter implementiert die projektinterne Schnittstelle
- Der Adapter konvertiert die Methodenaufrufe auf die externe Schnittstelle

Demo

Abstraktion definieren
Nlog Log
Unabhaengigkeit

Projekt: „PdfTools“

Projekte für Contracts und Implementierung
IPtLogger (Trace(), Info(), Error())
NLogLoggerAdapter

Problem: Schnittstellenlose Klassen

- Statische Klassen (Console, Debug)
- Klassen ohne Schnittstelle (HttpClient)
- Externe Systeme (Datenbank, Dateisystem)

→ Interface + System Implementierung (Wrapper)

Aufgabe

Abstrahieren einer Schnittstellenlosen Klasse

Projekt: „PdfTools“

Console (WriteLine)

Interface, Adapter
Factory, Dependency Injection

03

Facade Pattern

Facade Pattern

- Vereinfachte Schnittstelle zu einem Subsystem
- Verbirgt die Details des Subsystems
- Verbergen der technisch orientierten Klassen des Subsystems
- Bereitstellen der verwendeten Funktionen

Aufgabe

Extract HttpClient

Projekt: “PdfTools”

Facade für den HttpClient erstellen
HttpClientFacade

Wrapper

- Erweitern der Funktionalität (Proxy, Decorator)
- Verbergen von Funktionalität (Façade)
- Anbinden von Funktionalität (Adapter)

- Separation of Concerns
- Testbarkeit
- Dynamisches Verhalten der Applikation

Moegliche Loesungen zum Dekorieren

Wie fügt man (optionales) “Logging” zu einer Klasse hinzu

- Decorator Pattern
- Null-Object Pattern
- Vererbung
- (Aspekt Orientierte Programmierung)

- Und sicherlich viele mehr…