Christian Metzler | ABAS Software AG

Advanced Patterns

Über mich

- Christian Metzler
- Studium der Informatik an der Uni Karlsruhe (2000-2008)
  - Webentwicklung während des Studiums (PHP)
  - Selbständige Webprojekte
- Softwareentwickler bei MetaSystems (2009-2013)
  - Delphi 2007/XE2
  - Framework zum Binden und Serialisieren von Objekten
  - Kamerasteuerung
- Softwareentwickler bei ABAS Software AG
  - JAVA
  - SSO und Identity Management

Agenda

- Einführung
- Behandelte Patterns
  - Strategie (Strategy)
  - Kommando (Command)
  - Beobachter (Observer)
  - Zustand (State)
  - Dekorierer (Decorator)
  - Erbauer (Builder)
- Zusammenfassung

Einführung

- Warum Design Patterns?
  - Viele Problemstellungen wurden schon von anderen gelöst
  - Einheitliche Sprache, die von vielen anderen Entwicklern verstanden wird
- Warum Advanced Patterns?
  - Wir fangen nicht bei „Null“ an
  - Factory oder Singleton werden oft behandelt
  - Gewisse Vorkenntnisse in Delphi sind erforderlich (Objekte, Interfaces)
- Aufbau
  - Erklärung des Patterns
  - Code Beispiel in Delphi
  - Vor-/Nachteile

Übersicht Design Patterns

- Design Patterns werden in verschiedene Kategorien unterteilt:

Erzeugungsmuster	Strukturmuster	Verhaltensmuster
- Abstrakte Fabrik - Singleton - Erbauer - Fabrikmethode - Prototyp	- Adapter - Brücke - Dekorierer - Fassade - Fliegengewicht - Kompositum - Stellvertreter	- Beobachter - Besucher - Iterator - Kommando - Memento - Nullobjekt - Schablonenmethode - Strategie - Zustand

Übersicht Design Patterns

- Im Vortrag behandelte Patterns:

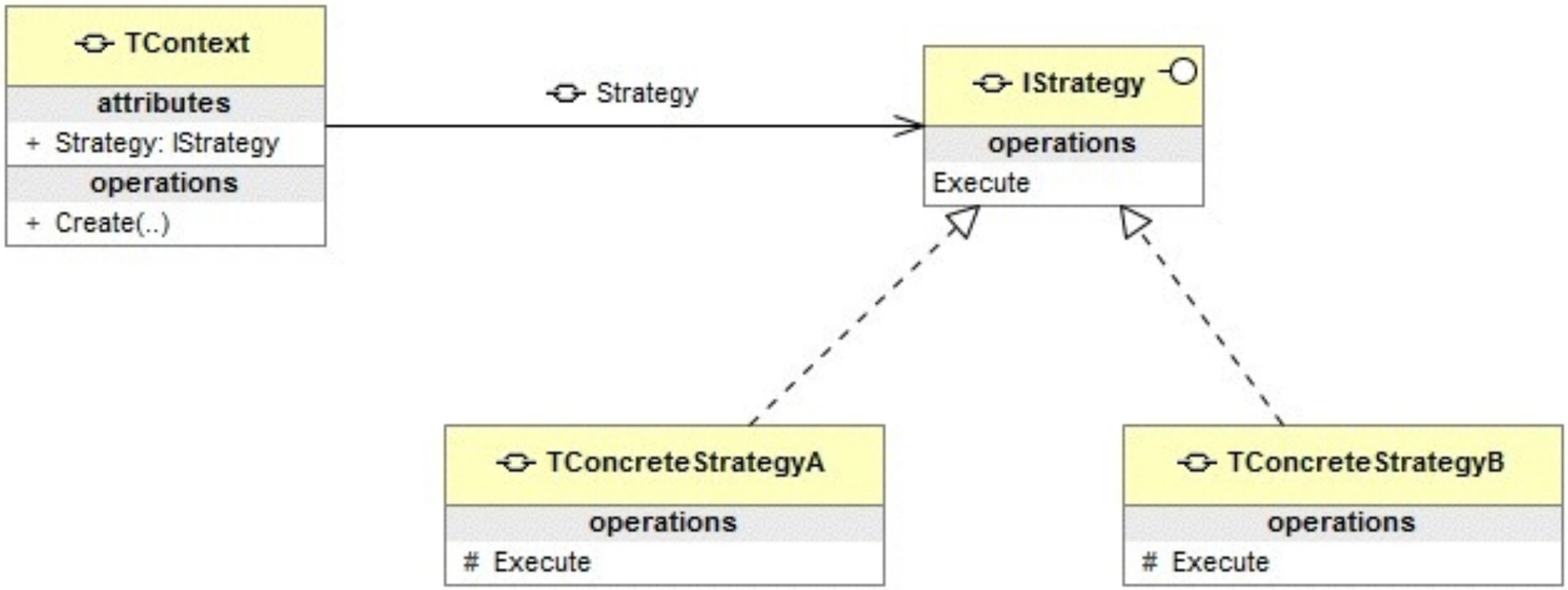
Erzeugungsmuster	Strukturmuster	Verhaltensmuster
- Abstrakte Fabrik - Singleton - Erbauer - Fabrikmethode - Prototyp	- Adapter - Brücke - Dekorierer - Fassade - Fliegengewicht - Kompositum - Stellvertreter	- Beobachter - Besucher - Iterator - Kommando - Memento - Nullobjekt - Schablonenmethode - Strategie - Zustand

Strategie

- Abstrakte Definition
  - Das Strategie Pattern definiert eine Familie von Algorithmen, kapselt diese und macht sie austauschbar. Das Strategie Pattern erlaubt Änderungen der Algorithmen unabhängig vom Client.
- Was bedeutet das?
  - Änderbares Verhalten wird ausgelagert
  - Verhalten kann durch hinzufügen von Strategien erweitert werden
  - Strategie kann auch von anderen Clients benutzt werden
- Prinzip: Trennung von Dingen, die sich ändern und von Dingen, die gleich bleiben

Strategie

- UML-Diagramm



Strategie

DEMO

Strategie

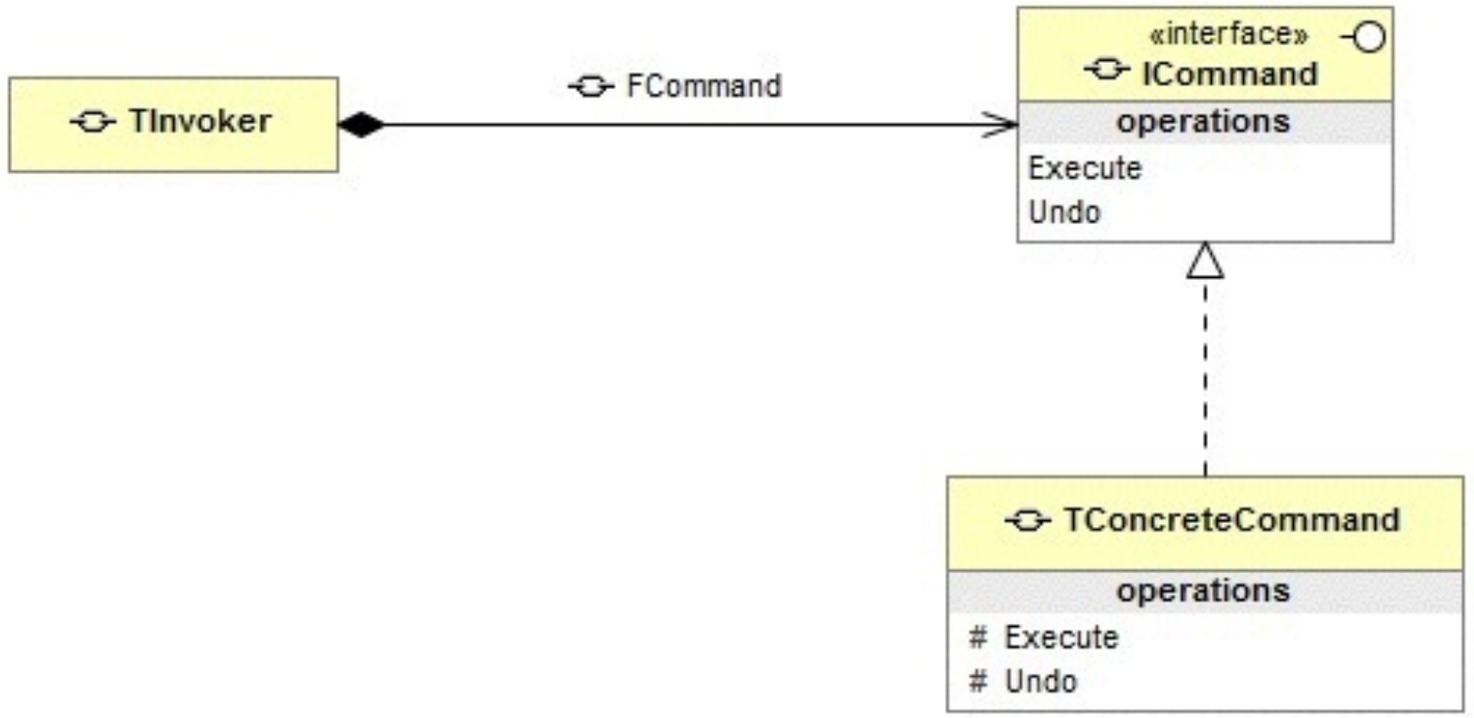
- Vorteile:
  - Wiederverwendbarkeit
  - Mehrfachverzweigungen können vermieden werden
  - Alternative zu Unterklassenbildung
- Nachteile
  - Zusätzliche Kommunikation zwischen Strategie und Kontext
  - Höhere Anzahl von Objekten
- Hilfreiche Patterns
  - Null-Objekt
  - Fabrik

Kommando

- Abstrakte Definition
  - Das Kommando Pattern kapselt einen Befehl in einem Objekt und ermöglicht es Befehle in eine Warteschlange einzureihen, sie zu loggen und sie rückgängig zu machen
- Was bedeutet das?
  - Aktionen beim Empfänger werden nicht direkt, sondern über ein Kommando aufgerufen
  - Ein Kommando Objekt kann Methoden zum Rückgängig machen, Loggen, Speichern, Laden enthalten

Kommando

- UML-Diagramm



Kommando

DEMO

Kommando

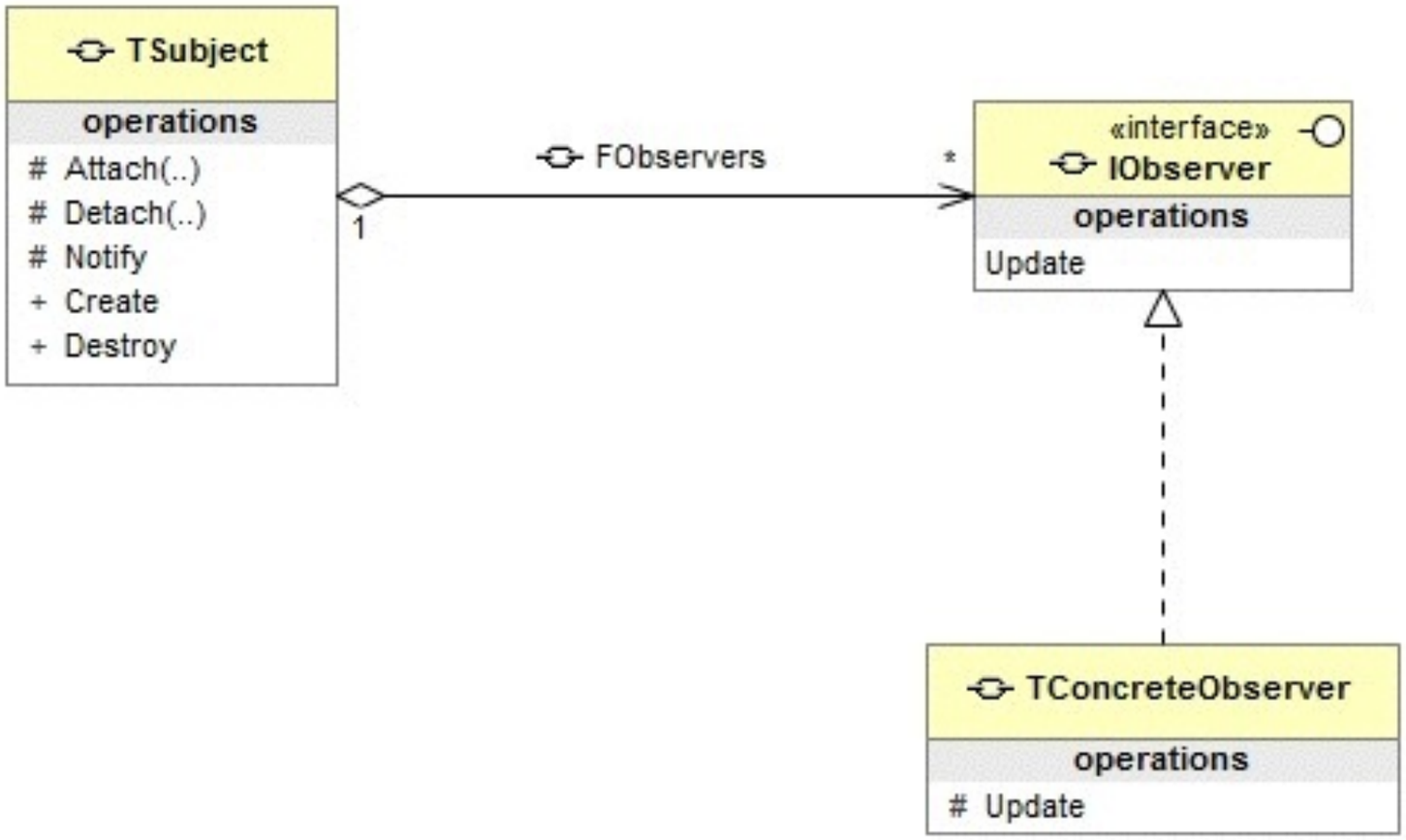
- Vorteile
  - Entkopplung von Auslöser und Ausführer
  - Befehlsobjekte bieten eigene Funktionalität
  - Kombination zu komplexen Befehlen möglich
- Nachteile
  - Anzahl der Klassen nimmt zu
- Hilfreiche Zusatzpatterns
  - Kompositum (für Makro-Kommandos)

Beobachter

- Abstrakte Definition
  - Das Beobachter Pattern definiert eine Eins-zu-Vielen Abhängigkeit zwischen Objekten, bei der alle Objekte benachrichtigt und aktualisiert werden, sobald sich ein Objekt ändert
- Was bedeutet das?
  - Unterscheidung zwischen Subjekt - dem Objekt, das sich ändert - und Beobachtern - den Objekten, die über die Änderung des Objekts benachrichtigt werden wollen
- Prinzip
  - Lose Kopplung zwischen interagierenden Objekten
  - Vergleichbar mit Zeitung und Abonnenten

Beobachter

- UML-Diagramm



Beobachter

DEMO

Beobachter

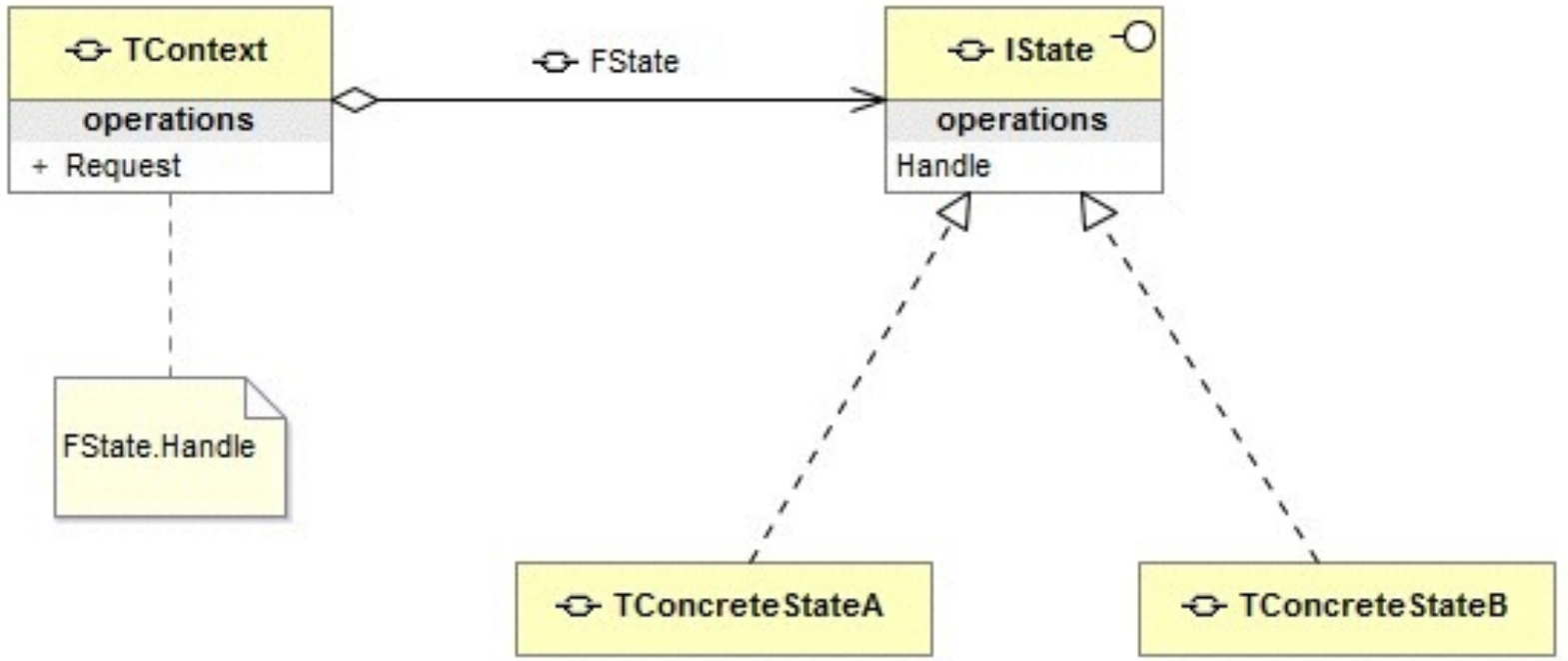
- Vorteile
  - Subjekte und Beobachter können unabhängig variiert werden
  - Minimale Schnittstelle des Beobachters
  - Subjekte benötigen kein zusätzliches Wissen über ihre Beobachter
- Nachteile
  - Bei hoher Beobachterzahl führt zu hohen Änderungskosten
  - Endlosschleifen können entstehen, wenn der Beobachter seinerseits mit einer Manipulation des Subjekts reagiert
- Achtung
  - In Delphi muss über das Reference Counting nachgedacht werden!

Zustand

- Abstrakte Definition
  - Das Zustandspattern erlaubt einem Objekt, sein Verhalten zu ändern, wenn der interne Zustand sich ändert.
- Was bedeutet das?
  - Zustände sind Objekte
  - Zustandsübergänge sind Methoden der Objekte
- Prinzip
  - Zustandsautomat

Zustand

- UML-Diagramm



Zustand

DEMO

Zustand

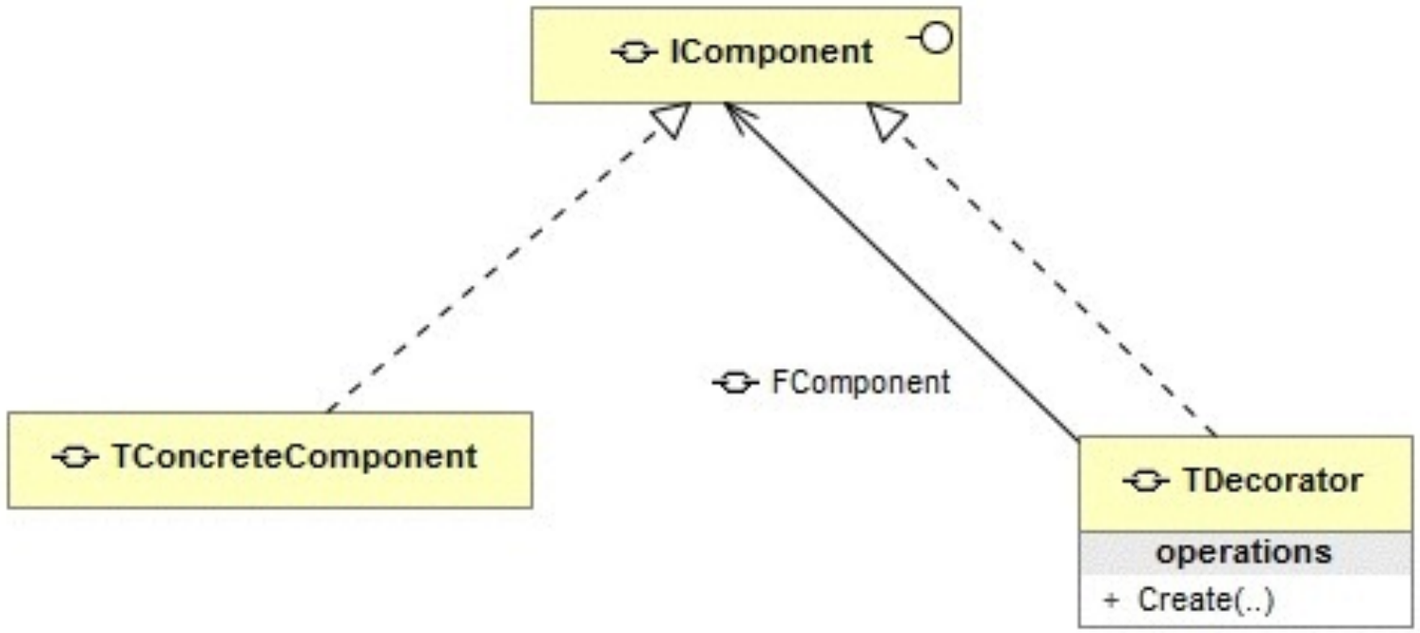
- Vorteile
  - Schwer zu lesende Bedingungsanweisungen können vermieden werden (case State of…..)
  - Einfaches Ergänzen neuer Zustände
- Nachteile
  - Bei einfachem Verhalten erheblicher Programmier Overhead
  - Wenn viele Übergänge nicht möglich sind ist es einfacher diese in einem „else“ Zweig zusammen zu fassen

Dekorierer

- Abstrakte Definition
  - Das Dekorierer Pattern fügt zusätzliche Aufgaben zu einem Objekt dynamisch hinzu. Dekorierer bieten eine flexible Alternative zur Unterklassenbildung, um Funktionalität zu erweitern
- Was bedeutet das?
  - Möglichkeit der flexiblen Zusammenstellung von Grundfunktionalität und Erweiterung dieser
  - Beliebige Verschachtelung möglich
- Prinzip: Klassen sollten offen für die Erweiterbarkeit sein, aber geschlossen für die Änderbarkeit

Dekorierer

- UML Diagramm



Dekorierer

DEMO

Dekorierer

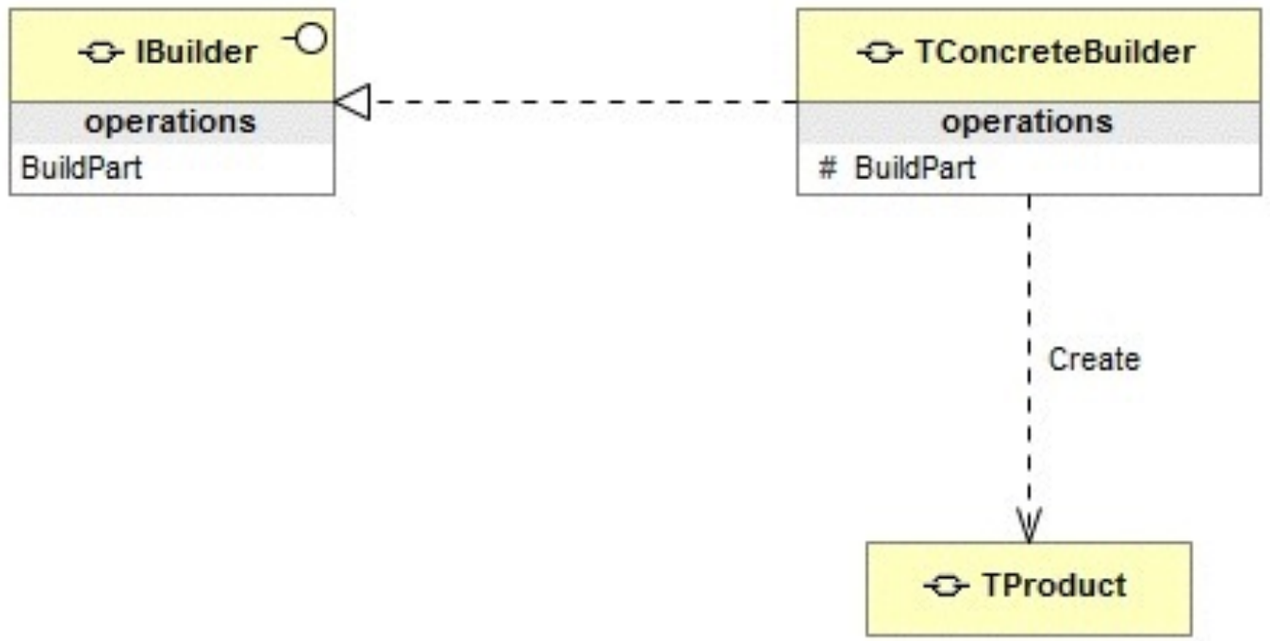
- Vorteile
  - Dekorierer können nacheinander geschaltet
  - Dekorierer können zur Laufzeit ausgetauscht werden
- Nachteile
  - Objektidentität kann nicht geprüft werden
  - Jede Nachricht muss an dekoriertes Objekt weiter gereicht werden
- Verwandte Patterns
  - Interceptor Pattern für „Cross-Cutting Concerns“ (Vortrag Stefan Glienke)

Erbauer

- Abstrakte Definition
  - Der Erbauer trennt die Konstruktion von komplexen Objekten von deren Repräsentation, wodurch die selben Konstruktionsprozesse wiederverwendet werden können.
- Was bedeutet das?
  - Es können komplexe Objekte mit unterschiedlichen Repräsentationen erzeugt werden.
  - Der Erbauer garantiert, dass am Ende ein komplett konfiguriertes Objekt erstellt wird

Erbauer

- UML Diagramm:



Erbauer

DEMO

Erbauer

- Vorteile
  - Implementierung der Konstruktion wird von der Repräsentation isoliert
  - Kann unveränderliche Objekte erzeugen
- Nachteile
  - Starke Kopplung zwischen Erbauer und dem konstruierten Objekt
- Verwandte Patterns
  - Fabrik für einfache Konstruktion von Objekten
- Hilfreiche Patterns
  - Fluent Interface

Zusammenfassung

- Patterns können gutes Software Design fördern
- Patterns sollten nicht als Allheilmittel gesehen werden
- Übertriebener Einsatz von Patterns erhöht die Komplexität
- Fallstricke in Delphi
  - Reference Counting bei Interfaces
    - Deaktivieren wenn Objektreferenzen eingesetzt werden
    - Schwache Referenzen bei gegenseitigen Beziehungen
  - Unterschiedliche Schnittstellen müssen teilweise in einer Unit untergebracht werden (Zyklische Abhängigkeit)

Zusammenfassung

- Beispiele und Folien zum Download
  - Github: https://github.com/coco1979ka/EKON17AdvancedPatterns

Literatur

- Design Patterns. Elements of Reusable Object-Oriented Software. (Gamma et al. - die „Gang of Four“)
- Head First: Design Patterns (Eric Freeman & Elisabeth Freeman)
- Internet:
  - http://www.philipphauer.de/study/se/design-pattern.php
  - http://en.wikipedia.org/wiki/Software_design_pattern

Vielen Dank für die Aufmerksamkeit