Design Patterns

Markus Rundel
Julian Haug
Markus Schnalke
Dimitar Dimitrov

27. Juni 2007

Einführung in Design Patterns
Markus Rundel

Observer-Pattern
Markus Schnalke

Composite-Pattern
Dimitar Dimitriv

Andere Patterns
Julian Haug

Fazit

Einführung

Markus Rundel

Was sind Patterns?

Was sind Design Patterns?

Definitionen

Christopher Alexander

Each pattern is a three-part-rule, which expresses a relation between a certain context, a problem and a solution

Martin Fowler

A pattern is an idea that has been useful in one practical context and will be probably useful in others.

Gang of Four (GoF)

Design Patterns sind Beschreibungen zusammenhängender Objekte und Klassen, die maßgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen.

Definitionen

Christopher Alexander

Each pattern is a three-part-rule, which expresses a relation between a certain context, a problem and a solution

Martin Fowler

A pattern is an idea that has been useful in one practical context and will be probably useful in others.

Gang of Four (GoF)

Design Patterns sind Beschreibungen zusammenhängender Objekte und Klassen, die maßgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen.

Definitionen

Christopher Alexander

Each pattern is a three-part-rule, which expresses a relation between a certain context, a problem and a solution

Martin Fowler

A pattern is an idea that has been useful in one practical context and will be probably useful in others.

Gang of Four (GoF)

Design Patterns sind Beschreibungen zusammenhängender Objekte und Klassen, die maßgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen.

Geschichte

1970er Jahre

Erstellung erster Entwurfsmuster von einem Architekten

1980er Jahre

Entwicklung von Entwurfsmuster für grafische Benutzerschnittstellen

1991

“Design Patterns - Elements of Reusable Object-Oriented Software”

Geschichte

1970er Jahre

Erstellung erster Entwurfsmuster von einem Architekten

1980er Jahre

Entwicklung von Entwurfsmuster für grafische Benutzerschnittstellen

1991

“Design Patterns - Elements of Reusable Object-Oriented Software”

Geschichte

1970er Jahre

Erstellung erster Entwurfsmuster von einem Architekten

1980er Jahre

Entwicklung von Entwurfsmuster für grafische Benutzerschnittstellen

1991

“Design Patterns - Elements of Reusable Object-Oriented Software”

Klassifizierung

Schema zum beschreiben von Design Patterns

- Mustername und Klassifikation
- Zweck (Wozu dient dieses Muster?)
- Synonyme
- Motivation
- Anwendbarkeit
- Struktur
- Beteiligte Klassen (Akteure)
- Zusammenspiel der involvierten Klassen
- Vor- und Nachteile
- Implementierung
- Beispielcode
- Praxiseinsatz
- Querverweise

Nutzen und Vorteile

Nutzen / Vorteile von Design Pattern

- Zeitersparnis
- Fehlerfreiheit
- Gemeinsame Kommunikationsgrundlage
- Sauberes OO-Design
- Lesbarkeit
- Geringerer Testaufwand
- Höhere Robustheit

Literatur

Design Patterns

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns Explained

A New Perspective on Object Oriented Design

Allan Shalloway, James R. Trott

Modern C++ Design

Generic Programming and Design Patterns applied

Andrei Alexandrescu

Literatur

Design Patterns

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns Explained

A New Perspective on Object Oriented Design

Allan Shalloway, James R. Trott

Modern C++ Design

Generic Programming and Design Patterns applied

Andrei Alexandrescu

Literatur

Design Patterns

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns Explained

A New Perspective on Object Oriented Design
Allan Shalloway, James R. Trott

Modern C++ Design

Generic Programming and Design Patterns applied
Andrei Alexandrescu

Beispiele

Im Script behandelte Design Pattern

- Singleton
- Strategie
- Fassade

Weitere Design-Patterns

- Iterator
- General Hierarchie
- Player Role Pattern
- Immutable
- Read-Only Interface

Zusammenfassung

- Design Patterns sind bewährte Lösungen zu bekannten, häufiger auftretenden Problemen in der Softwareentwicklung
- Ende der 80er Jahre wurden Entwurfsmusster erstmals in der Softwareentwicklung eingesetzt
- Die GoF entwickelte ein einheitliches Schema um die einzelnen Design Pattern beschreiben zu können

Observer-Pattern

Markus Schnalke

Vorgehensweise

Erarbeitung des Patterns

- Ansatz
- Erarbeitung einer Lösung
- Zusammenfassung

Das Pattern

- UML-Diagramme
- Beispiele
- Code

Zusammenfassung

Von der anderen Seite

Gründe

Design Pattern sind “Best Practices” (= Erfolgsrezepte), meist **nach Vorbildern in der Realität**

Meine Vorgehensweise

Ich will nun anhand eines Beispiels aus dem täglichen Leben zeigen, dass das Design Pattern “Observer” ein absolut natürliches Vorgehen ist, das bei ähnlichen Situationen in Programmen deshalb ebenso verwendet werden sollte.

Von der anderen Seite

Gründe

Design Pattern sind “Best Practices” (= Erfolgsrezepte), meist **nach Vorbildern in der Realität**

Meine Vorgehensweise

Ich will nun anhand eines Beispiels aus dem täglichen Leben zeigen, dass das Design Pattern “Observer” ein absolut natürliches Vorgehen ist, das bei ähnlichen Situationen in Programmen deshalb ebenso verwendet werden sollte.

Ein Beispiel

Die Situation

- Personen die verkaufen möchten
- Personen die kaufen möchten
- Sie wollen/sollen sich nicht kennen
- Möglichst effektives Vorgehen

Vorschläge?

-
-

Ein Beispiel

Die Situation

- Personen die verkaufen möchten
- Personen die kaufen möchten
- Sie wollen/sollen sich nicht kennen
- Möglichst effektives Vorgehen

Vorschläge?

-
-

Ansatz

Mögliche Vorgehen

- Personen direkt ansprechen
- Laut in die Menge rufen
- Den Freunden erzählen, die es dann weitererzählen

Probleme

- Man weiß nicht wer Interesse hat
- nicht mal wieviele
- Manche Interessenten sind vielleicht nur zu bestimmten Zeiten da
- (... oder sie sind taub)

Ansatz

Mögliche Vorgehen

- Personen direkt ansprechen
- Laut in die Menge rufen
- Den Freunden erzählen, die es dann weitererzählen

Probleme

- Man weiß nicht wer Interesse hat
- nicht mal wieviele
- Manche Interessenten sind vielleicht nur zu bestimmten Zeiten da
- (... oder sie sind taub)

Eine (gute) Lösung: Pinnwand

Funktionsweise

- Man kann Zettel anpinnen
- Hingehen und nach neuen Zetteln schauen
- Zettel lesen
- Zettel abnehmen
- Jeder der ein paar Fähigkeiten hat kann es

Probleme

- Man muss hingehen, nur um festzustellen, dass nichts Neues dabei ist
- Man kann wichtige Zettel verpassen
- Zettel sollten nicht weggenommen werden können

Eine (gute) Lösung: Pinnwand

Funktionsweise

- Man kann Zettel anpinnen
- Hingehen und nach neuen Zetteln schauen
- Zettel lesen
- Zettel abnehmen
- Jeder der ein paar Fähigkeiten hat kann es

Probleme

- Man muss hingehen, nur um festzustellen, dass nichts Neues dabei ist
- Man kann wichtige Zettel verpassen
- Zettel sollten nicht weggenommen werden können

Verbesserung: Pinnwand-Sekretärin

Lösung

- Durch Studiengebühren wird eine Sekretärin für die Pinnwand angestellt
- Pinnwand und Sekretärin sind fortan eine Einheit
- Man kann bei ihr einen Zettel in Auftrag geben (auch telefonisch)
- (Sie schreibt mit lesbarer Schrift)
- Sie verhindert, dass Zettel abgenommen werden

Bestehendes Problem

- Man läuft immer noch oft unnötig zur Pinnwand

Verbesserung: Pinnwand-Sekretärin

Lösung

- Durch Studiengebühren wird eine Sekretärin für die Pinnwand angestellt
- Pinnwand und Sekretärin sind fortan eine Einheit
- Man kann bei ihr einen Zettel in Auftrag geben (auch telefonisch)
- (Sie schreibt mit lesbarer Schrift)
- Sie verhindert, dass Zettel abgenommen werden

Bestehendes Problem

- Man läuft immer noch oft unnötig zur Pinnwand

2. Verbesserung: Pinnwand-Sekretärin mit Benachrichtigung

Lösung

- Durch Studiengebühren wird eine längere Arbeitszeit der Sekretärin finanziert
- Man kann sich bei der Sekretärin nun als “Interessierter” registrieren
- Die Sekretärin trägt die Telefonnummer in eine Liste ein
- Zukünftig ruft sie alle Personen der Liste an, wenn sie einen neuen Zettel anpinnt

Zusammenfassung des Beispiels

Pinnwand + Sekretärin + Benachrichtigung

- Man kann neue Zettel anpinnen lassen
- Man kann sich als Interessierter anmelden (und auch abmelden)
- Interessierte werden bei Änderungen der Pinnwand benachrichtigt
- Sie können dann zur Pinnwand gehen und sie sich anschauen

Eure Meinung?

- Ist diese Struktur zufriedenstellend?
- Erfüllt sie alle Anforderungen?
- Was fehlt?

Zusammenfassung des Beispiels

Pinnwand + Sekretärin + Benachrichtigung

- Man kann neue Zettel anpinnen lassen
- Man kann sich als Interessierter anmelden (und auch abmelden)
- Interessierte werden bei Änderungen der Pinnwand benachrichtigt
- Sie können dann zur Pinnwand gehen und sie sich anschauen

Eure Meinung?

- Ist diese Struktur zufriedenstellend?
- Erfüllt sie alle Anforderungen?
- Was fehlt?

Überleitung

Neue Namen

- Pinnwand-Sekretärin-Einheit $\longrightarrow$ “Subject”
- Die Zettel auf der Pinnwand $\longrightarrow$ “subjectState”
- Interessenten $\longrightarrow$ “Observers”

Schnittstellen

Die Fähigkeiten der Pinnwand/Sekretärin und Interessenten sind ihre “Interfaces”.
(vgl: taub, minimale Fähigkeiten, leserliche Schrift, ...)

Überleitung

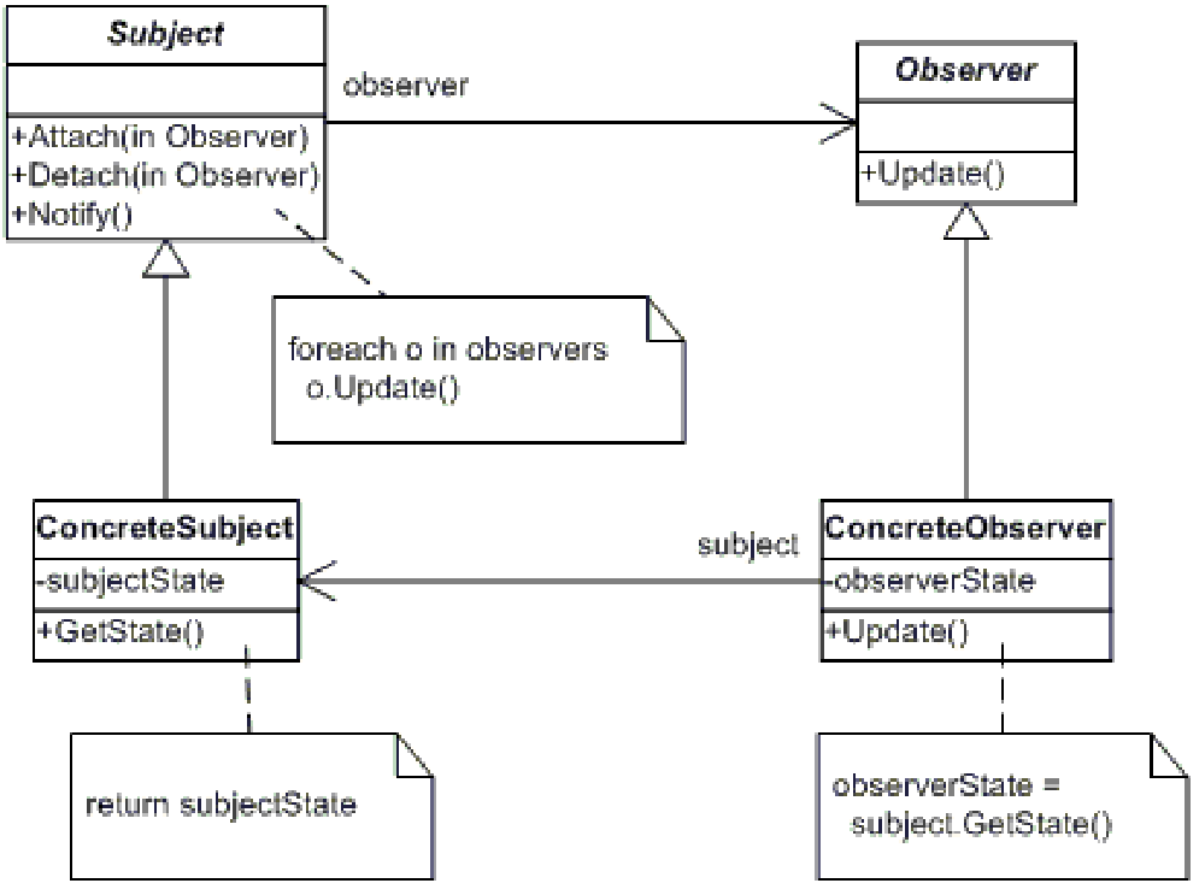
Neue Namen

- Pinnwand-Sekretärin-Einheit $\longrightarrow$ “Subject”
- Die Zettel auf der Pinnwand $\longrightarrow$ “subjectState”
- Interessenten $\longrightarrow$ “Observers”

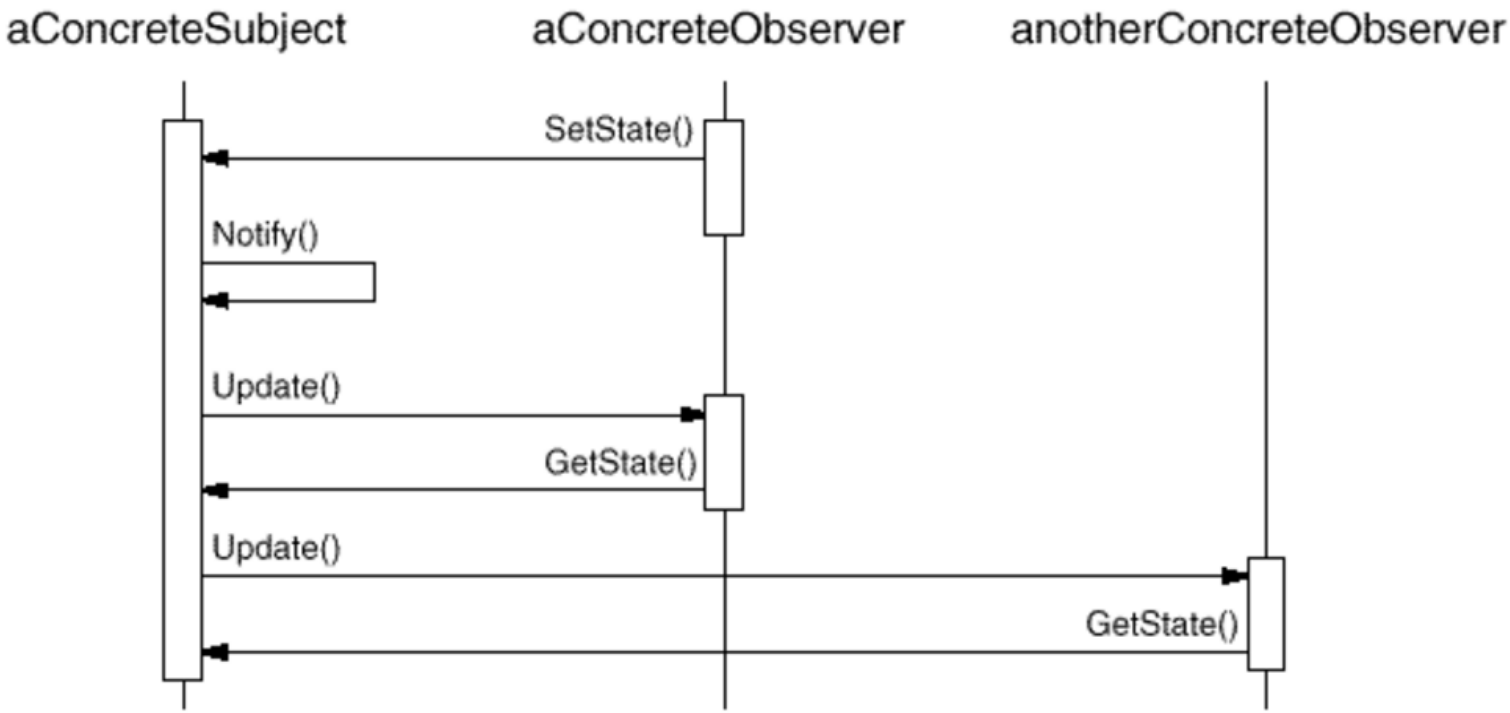
Schnittstellen

Die Fähigkeiten der Pinnwand/Sekretärin und Interessenten sind ihre “Interfaces”.
(vgl: taub, minimale Fähigkeiten, leserliche Schrift, ...)

Struktur-Diagramm des Observers



Interaktions-Diagramm des Observers



Der Observer in der Praxis

Beispiele

- Observer ist sehr verbreitet
- v.a. MVC (Model = Subject; View = Observer)
- Mailinglisten
- Ebay Such-Abo

Aber

- nicht Blog mit RSS-Feed!

Der Observer in der Praxis

Beispiele

- Observer ist sehr verbreitet
- v.a. MVC (Model = Subject; View = Observer)
- Mailinglisten
- Ebay Such-Abo

Aber

- nicht Blog mit RSS-Feed!

Code? — Nein, kein Code!

Denn

Implementierungen sind Schall und Rauch,
Konzepte dagegen bleiben bestehen!

Beispiel-Implementierung in der Ausarbeitung.

Zusammenfassung

- Menschen denken basierend auf der Realität
- deshalb Design Patterns auf Realität zurückführen
- Patterns anwenden weil man es in der Realität auch so machen würde

Composite-Pattern

Dimitar Dimitrov

Outline

Definition

Definition

- Organisiert Objekte in Baumstrukturen für die Repräsentation von Teil-Ganzes-Beziehungen
- Erlaubt den gleichförmigen Zugriff auf atomare Einzelobjekte wie auf zusammengesetzte Objekte

Beispiel Telefonnummer

0800-CAR-TALK
0800-227-8255

Motivation

Beispiel Grafikprogramm

- Einfache Objekte (Primitive) wie Linien und Texte sollen gruppierbar sein; der Benutzer möchte Gruppen genau wie Einzelobjekte behandeln
- Implementierungsidee: Klassen für Primitive + Klassen für Container
- Diese Unterscheidung macht den Programmcode sehr komplex

Lösung

- Eine abstrakte Oberklasse repräsentiert Primitive und Container
- Operationen von Primitiven werden von Containerobjekten an die enthaltenen Objekte delegiert

Motivation

Beispiel Grafikprogramm

- Einfache Objekte (Primitive) wie Linien und Texte sollen gruppierbar sein; der Benutzer möchte Gruppen genau wie Einzelobjekte behandeln
- Implementierungsidee: Klassen für Primitive + Klassen für Container
- Diese Unterscheidung macht den Programmcode sehr komplex

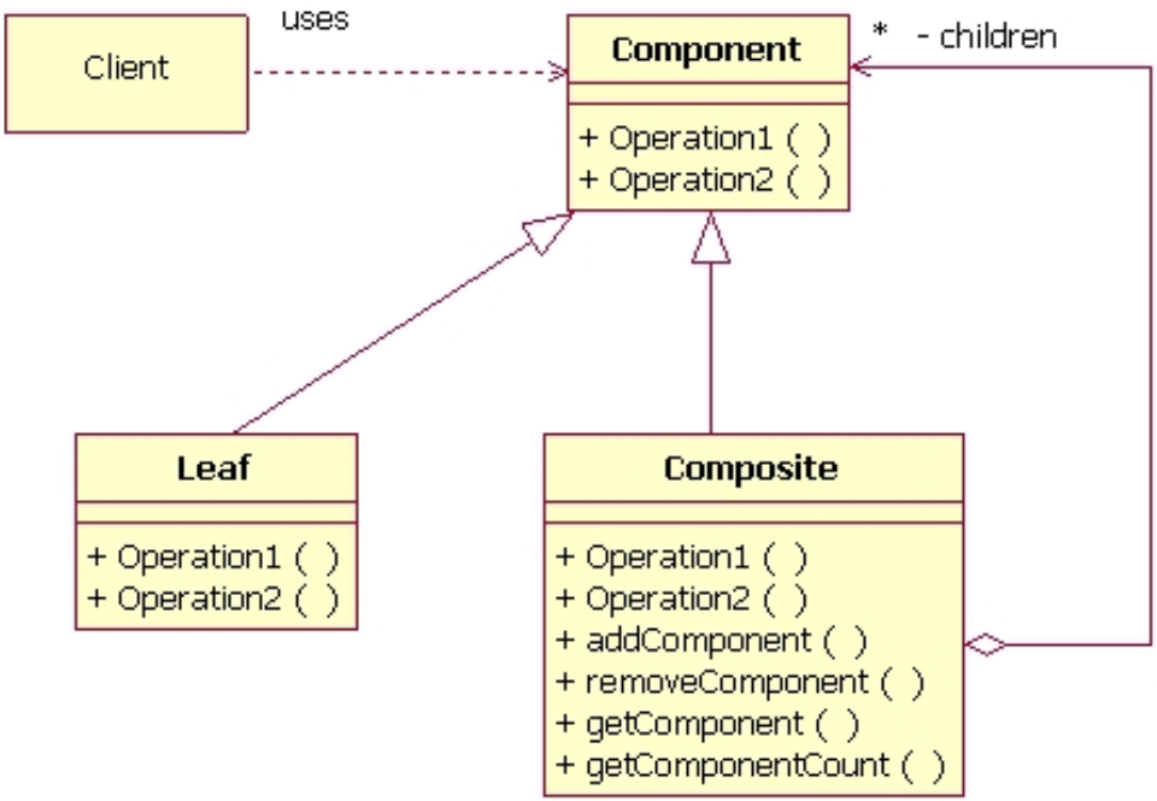
Lösung

- Eine abstrakte Oberklasse repräsentiert Primitive und Container
- Operationen von Primitiven werden von Containerobjekten an die enthaltenen Objekte delegiert

Wann verwenden?

- Wenn Teil-/Ganzes-Hierarchien von Objekten zu repräsentieren sind
- Wenn Anwendungsklassen den Unterschied zwischen atomaren und zusammengesetzten Objekten ignorieren sollen

UML-Diagramm des Composite-Pattern



Beteiligte

Component

- Definiert die gemeinsame Schnittstelle aller Objekte im Baum Implementiert Default-Verhalten, wo möglich und sinnvoll
- Definiert eine Schnittstelle für den Zugriff auf Kinder einer Komponente

Leaf

- Repräsentiert Blätter in der Baumhierarchie
- Definiert das Verhalten von atomaren Objekten

Composite

- Repräsentiert Wurzel- und innere Knoten in der Baumhierarchie
- Definiert das Verhalten dieser Knoten
- Implementiert Kind-bezogene Operationen

Client

- manipuliert Objekte mittels Component-Schnittstelle

class Main, interface IComponent

```
public class Main {
        public static void main(String[] args) {
                Composite england = new Composite("England");
                Leaf york = new Leaf("York");
                Leaf london = new Leaf("London");
                england.addComponent(york);
                england.addComponent(london);
                england.removeComponent(york);

                Composite france = new Composite("France");
                france.addComponent(new Leaf("Paris"));

                Composite europe = new Composite("Europe");
                europe.addComponent(england);
                europe.addComponent(france);

                System.out.println(europe.toString());
        }
}

interface IComponent {
        Collection getChildren();
        boolean addComponent(IComponent c);
        boolean removeComponent(IComponent c);
}
```

class Composite

```
class Composite implements IComponent {
        private String id;
        private List<IComponent> list = new ArrayList<IComponent> ();

        public Composite(String id) {
                this.id = id;
        }

        public String toString() {
                StringBuilder buf = new StringBuilder();
                buf.append(String.format("(%s:", id));

                for (IComponent child : list) {
                        buf.append(" " + child.toString());
                }
                buf.append(")");

                return buf.toString();
        }

        //public List<IComponent> getChildren()

        public Collection getChildren(){
                return list;
        }

        public boolean addComponent(IComponent c){
                return list.add(c);
        }

        public boolean removeComponent(IComponent c){
                return list.remove(c);
        }
}
```

class Leaf

```
class Leaf implements IComponent {
        private String id;

        public Leaf(String id) {
                this.id = id;
        }

        public String toString() {
                return this.id;
        }

        public Collection getChildren() {
                return null;
        }

        // false because failed to add
        public boolean addComponent(IComponent c) {
                return false;
        }

        // false because failed to find it for removal
        public boolean removeComponent(IComponent c) {
                return false;
        }
}
```

Zusammenfassung

- Definiert Klassenhierarchien bestehend aus atomaren Objekten (Primitiven) und zusammengesetzten Objekten Verbirgt den Unterschied zwischen diesen Objekten vor Anwendungsklassen
- Vereinfacht Anwendungsklassen (viele Fallunterscheidungen entfallen)
- Macht es einfach, neue Arten von Komponenten hinzuzufügen;
- Anwendungsklassen funktionieren ohne Änderung. Macht das Design vielleicht “allgemeiner” als gewünscht; falls nur bestimmte Klassen in ein bestimmtes Composite aufgenommen werden sollen, sind dafür Laufzeitüberprüfungen notwendig

Andere Patterns

Julian Haug

Andere Arten von Mustern

Andere Arten von Mustern

- Gang of Four motivieren viele Autoren zu weiteren Veröffentlichungen
- Problematik: ein Muster lässt sich nicht mehr ohne weiteres als Entwurfsmuster klassifizieren
- Es entstanden mehrere Arten von Mustern

Architekturmuster

- beschreiben typische Software-Architekturen
- bestimmen nicht ein konkretes Teilproblem, sondern den Grundaufbau der Anwendung
- Lässt sich in 4 verschiedene Kategorien einteilen
  - Mud-to-structure: hilft die Unmengen von Komponenten und Objekten eines Softwaresystems zu organisieren.
  - Verteilte Systeme: unterstützten die Verwendung verteilter Ressourcen und Dienste in Netzwerken
  - Interaktive Systeme: helfen Mensch-Computer-Interaktionen zu strukturieren
  - Adaptive Systeme: unterstützen besonders die Erweiterungs- und Anpassungsfähigkeit von Softwaresystemen.

Idom

Definition

- Beschreibt konkrete Implementierung eines Entwurfsmusters
- ist programmiersprachenspezifisch

Anwendung

- Konkrete Implementierung eines Entwurfsmusters.
- Programmierkonventionen (Codeformatierung, Namenskonventionen, Kommentar Formatierung, usw.)
- Typische Lösungsansätze für Probleme die durch die Programmiersprache nicht direkt unterstützt werden (Speicherverwaltung)

Idom

Definition

- Beschreibt konkrete Implementierung eines Entwurfsmusters
- ist programmiersprachenspezifisch

Anwendung

- Konkrete Implementierung eines Entwurfsmusters.
- Programmierkonventionen (Codeformatierung, Namenskonventionen, Kommentar Formatierung, usw.)
- Typische Lösungsansätze für Probleme die durch die Programmiersprache nicht direkt unterstützt werden (Speicherverwaltung)

Weitere Beispiele

Weitere Beispiele

- Analysemuster
  (beschreiben typische Fälle der Anforderungsanalyse)
- Kommunikationsmuster
  (beschreiben Kommunikationswege zwischen Personen einer Organisation)
- Organisationsmuster
  (beschreiben Elemente der Strukturen von Organisationen)

Anti-Patterns

Dokumentieren wiederkehrende Fehler bei der Software-Entwicklung um

- durch das Wissen ihrer Existenz diese zu vermeiden
- bereits manifestierte Anti-Pattern durch geschickte Maßnahmen zu beheben

Analog zu positiven Mustern gibt es auch hier eine weitere Unterscheidung

- Projektmanagement-Anti-Patterns
- Architektur- bzw. Design-Anti- Patterns
- Meta-Patterns
- Organisations-, Prozess- Anti-Patterns
- **Programmierungs-Anti-Patterns**

Anti-Patterns

Dokumentieren wiederkehrende Fehler bei der Software-Entwicklung um

- durch das Wissen ihrer Existenz diese zu vermeiden
- bereits manifestierte Anti-Pattern durch geschickte Maßnahmen zu beheben

Analog zu positiven Mustern gibt es auch hier eine weitere Unterscheidung

- Projektmanagement-Anti-Patterns
- Architektur- bzw. Design-Anti- Patterns
- Meta-Patterns
- Organisations-, Prozess- Anti-Patterns
- **Programmierungs-Anti-Patterns**

Programmierungs-Anti-Patterns

Zwiebel

- Neue Funktionalität wird um (oder über) die alte gelegt
- Häufig bei Erweiterungen
- Führt zu vielschichtigem Programmcode (Zwiebel)

Lavafluss

- in Anwendung häuft sich “toter Quelltext”
- Statt zu löschen wird um ihn “herum” programmiert

Programmierungs-Anti-Patterns

Zwiebel

- Neue Funktionalität wird um (oder über) die alte gelegt
- Häufig bei Erweiterungen
- Führt zu vielschichtigem Programmcode (Zwiebel)

Lavafluss

- in Anwendung häuft sich “toter Quelltext”
- Statt zu löschen wird um ihn “herum” programmiert

Tipps zur Anwendung

Auswahl eines Entwurfmusters

Problem: Viele (ähnliche) Muster kommen in Frage

- Musterkatalog
- Genaue Analyse des Problems

Verwendung des Musters

Problem: Verstehen wie das Muster unser Problem löst

- Strukturdiagramm (UML)
- Objekt/Klassenliste
- Implementierungsbeispiele

Wann sollte KEIN Entwurfsmuster benutzt werden

- Muster dürfen den Code nicht verkomplizieren
- Oft reicht eine Lösung die zwar weniger flexibel aber simpler ist

Tipps zur Anwendung

Auswahl eines Entwurfmusters

Problem: Viele (ähnliche) Muster kommen in Frage

- Musterkatalog
- Genaue Analyse des Problems

Verwendung des Musters

Problem: Verstehen wie das Muster unser Problem löst

- Strukturdiagramm (UML)
- Objekt/Klassenliste
- Implementierungsbeispiele

Wann sollte KEIN Entwurfsmuster benutzt werden

- Muster dürfen den Code nicht verkomplizieren
- Oft reicht eine Lösung die zwar weniger flexibel aber simpler ist

Tipps zur Anwendung

Auswahl eines Entwurfmusters

Problem: Viele (ähnliche) Muster kommen in Frage

- Musterkatalog
- Genaue Analyse des Problems

Verwendung des Musters

Problem: Verstehen wie das Muster unser Problem löst

- Strukturdiagramm (UML)
- Objekt/Klassenliste
- Implementierungsbeispiele

Wann sollte KEIN Entwurfsmuster benutzt werden

- Muster dürfen den Code nicht verkomplizieren
- Oft reicht eine Lösung die zwar weniger flexibel aber simpler ist

Zusammenfassung

- Es gibt verschiedene Arten von Mustern – für uns von direkter Bedeutung sind Entwurfs-, Architekturmuster und Idome
- Anti-Pattern zeigen welche Fehler man vermeiden sollte
- Bei Anwendung von Mustern das Problem genau analysieren, passendes Muster suchen und gegebenenfalls anwenden

Fazit

Fazit

1. understand the problem
2. understand the pattern
3. understand how the pattern solves the problem

Fazit

1. understand the problem
2. understand the pattern
3. understand how the pattern solves the problem

Fazit

1. understand the problem
2. understand the pattern
3. understand how the pattern solves the problem

Fragen?

?

Abschließend

Verwendete Software

- Debian GNU/Linux
- LaTeX-Beamer und `pdflatex`
- Vim
- qiv und ImageMagick
- Mercurial

Danke für eure Aufmerksamkeit