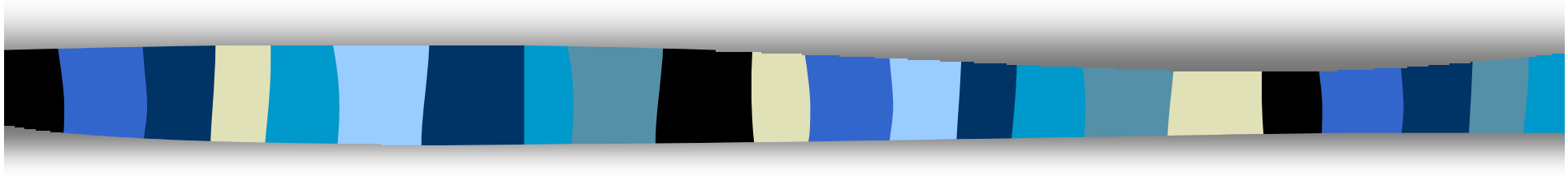
Was sind Pattern? Beispiel Singleton



Christina Bell

SE: OO Entwurf & Analyse

Gliederung

1. Was sind Pattern?

- Ausspruch
- Wozu benötigt man sie
- Beispiel
- Beschreibung
- Auswahl

2. Singleton

- Struktur abarbeiten

Was sind Pattern?

„Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so daß sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen“

Christoph Alexander

Wozu benötigt man Pattern?

- Vereinfachung des Entwurfs von wiederverwendbarer Software
- Lösen von spezifischen Entwurfsproblemen
- Objektorientierte Entwürfe flexibler und eleganter
- Vereinfachung der Wiederverwendung von erfolgreichen Entwürfen und Architekturen
- Wählen zwischen alternativen Entwürfen
- Verbesserung von Dokumentation und Wartung
- Helfen Entwickler Entwurf schneller „richtig“ zu machen

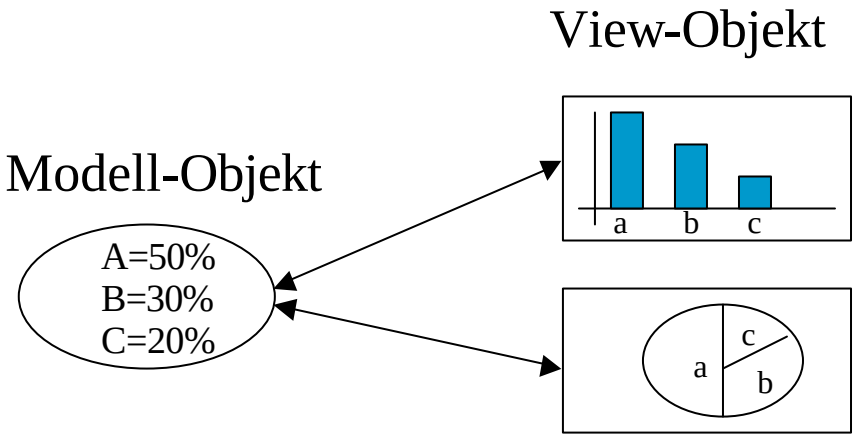
Was sind Pattern?

4 Grundelemente:

- Mustername
- Problemabschnitt
- Lösungsabschnitt
- Konsequenzabschnitt

Beispiel: MVC Pattern in Smalltalk

- Modell View Controller Paradigma
- Konstruktion von Benutzerschnittstellen
- 3 Klassen: Modell, View und Controller
- Möglichkeit Reaktion eines Views auf Eingabe zu ändern, mehrere Views an ein Modell



Beschreibung von Pattern

- Mustername und Klassifizierung
- Zweck
- Auch bekannt als
- Motivation
- Anwendbarkeit
- Struktur
- Teilnehmer
- Interaktion
- Konsequenz
- Implementierung
- Beispielcode
- Bekannte Verwendung
- Verwandte Muster

Auswahl von Entwurfsmustern

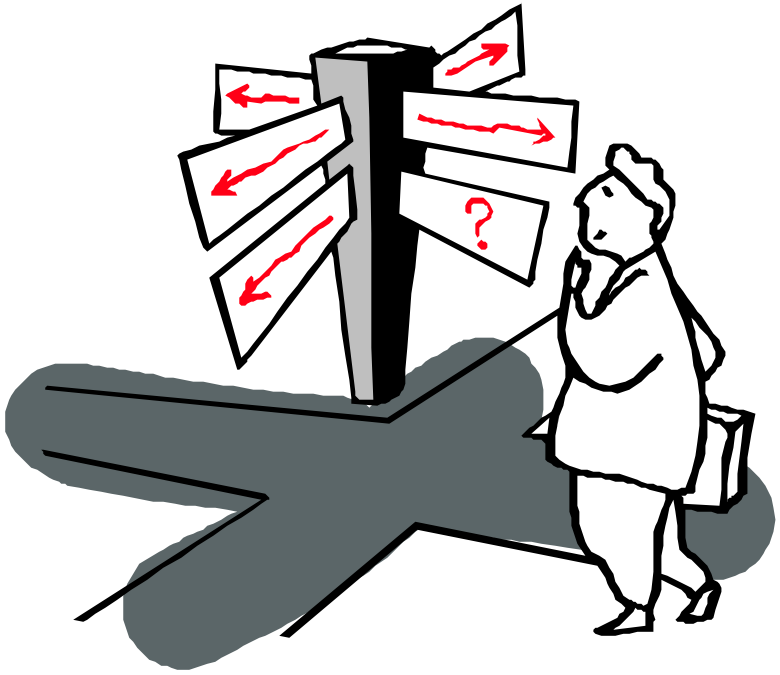
- Bedenken, wie lösen Pattern Entwurfsprobleme

  (Finden passender Objekte,
  Bestimmen von Granularitäten,
  spezifizieren von Objektschnittstellen,
  spezifizieren von Objektimplementationen,
  Anwendung von Wiederverwendungsmechanismen,
  Veränderungen von Entwürfen vorhersehen)

Auswahl von Entwurfsmustern

- Zweckabschnitte quer lesen
- Betrachten sie wie Muster miteinander in Beziehung stehen
- Untersuchen sie die Muster mit gleicher Aufgabe

Fragen zum ersten Abschnitt



Singleton

- Mustername und Klassifizierung
- Zweck
- Motivation
- Anwendbarkeit
- Struktur
- Teilnehmer
- Interaktion
- Konsequenz
- Implementierung
- Beispielcode
- Bekannte Verwendung
- Verwandte Muster

Mustername und Klassifizierung

- Singleton
- Objektbasiertes Erzeugungsmuster

Zweck

- Absicherung, dass eine Klasse genau ein Exemplar besitzt
- Bereitstellung eines globalen Zugriffspunktes

Motivation

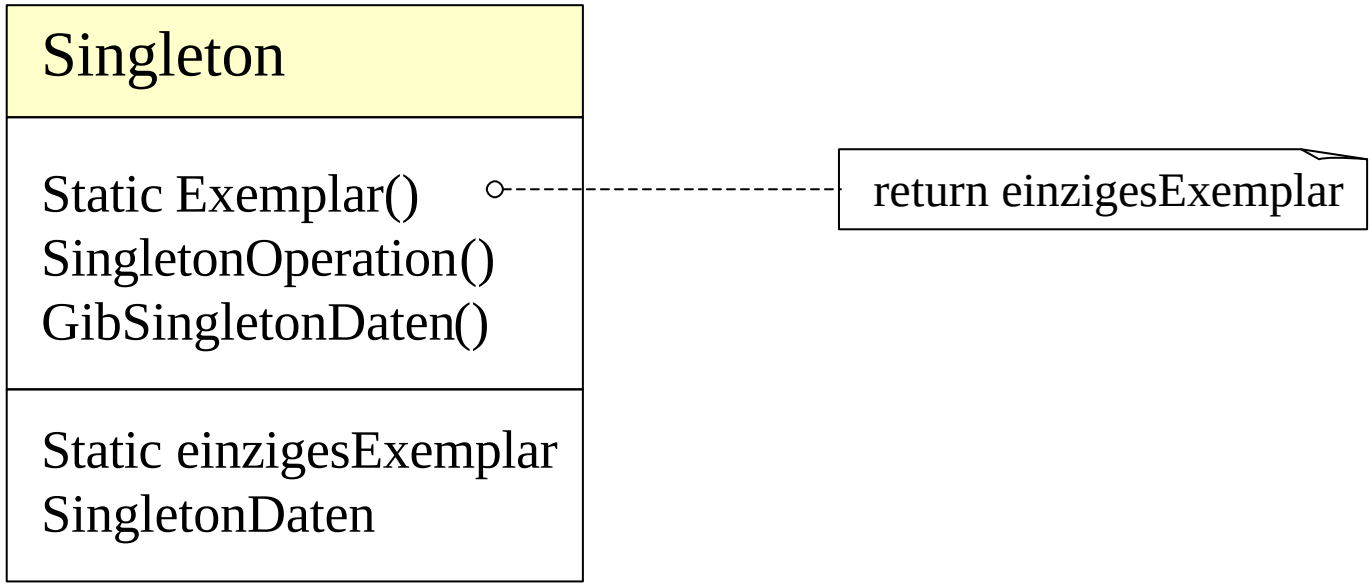
- Bei manchen Klassen wichtig, dass nur ein Exemplar existiert
  (Druckerspooler, Fensterverwaltung)
- Globale Variable stellt Zugriff sicher, verhindert aber nicht Erzeugen von mehreren Objekten

Anwendbarkeit

Verwenden Sie das Singleton, wenn

- es genau ein Exemplar einer Klasse geben und es für Klienten an einem wohldefinierten Punkt zugreifbar sein muss.
- das einzige Exemplar durch Bildung von Unterklassen erweiterbar sein soll und Klienten in der Lage sein sollen, das erweiterte Exemplar ohne Modifikation ihres Codes verwenden zu können.

Struktur


Singleton
Static Exemplar()
SingletonOperation()
GibSingletonDaten()
Static einzigesExemplar
SingletonDaten
return einzigesExemplar

Teilnehmer

- Singleton
  - Operation, die es Klienten ermöglicht auf sein einziges Exemplar zuzugreifen
  - Ist potentiell für Erzeugung seines einzigen Exemplars zuständig

Interaktionen

- Klienten greifen ausschließlich durch die Exemplar-Operation der Singletonklasse zu

Konsequenz

- Zugriffskontrolle auf das Exemplar
- Eingeschränkter Namensraum
- Verfeinerung von Operationen und Repräsentationen
- Variable Anzahl von Exemplaren

Implementierung

1. Garantie eines einzigen Exemplares

```
class Singleton {
public:
        static Singleton* Exemplar();
protected:
        Singleton();
Private:
        static Singleton* _exemplar;
};
```

Implementierung

```
Singleton* Singleton::_exemlpar = 0;

Singleton *Singleton::Exemplar() {
        if (_exemplar == 0) {
                _exempar =new singleton;
        }
        return _exemplar;
}
```

Implementierung

2. Ableiten der Singletonklasse

- Bestimmung des zu verwendenden Singletons in der Operation Exemplar der Singletonklasse
- Implementierung von Exemplar zu entfernen und in die Unterklasse einzufügen
- Registratur für Singletons

Beispielcode

```
Class LabyrinthFabrik {
public:
    static LabyrinthFabrik* Exemplar();
protected:
    LabyrinthFabrik();
private:
    static LabyrinthFabrik* _exemplar;
};
```

Beispielcode

```
LabyrinthFabrik* LabyrinthFabrik :: _exemplar=0;
LabyrinthFabrik* LabyrinthFabrik ::Exemplar() {
    if (_exemplar==0) {
        const char* labyrinthStil =getenv(„LABYRINTHSTIL“);

        if (strcmp(labyrinthStil,“mitbombe“)===){
            _exemplar = new LabyrinthMitBombeFabrik;
        }
        else if {...}
        ...
        else{
            _exemplar = new LabyrinthFabrik;
        }
    }
    return _exemplar;
}
```

Bekannte Verwendung

- Beziehung zwischen Klassen und Metaklassen
- InterViews, Klassenbibliothek zur Erstellung von Benutzerschnittstellen

Verwandte Muster

- Abstrakte-Fabrik-Muster
- Erbauermuster
- Prototypmuster

Fragen