Seminar Pattern und Antipattern

Das Visitor Pattern

Christian Stahl

stahl@informatik.hu-berlin.de

07.12.2001

1. Gliederung

1. Gliederung
2. Zweck
3. Beispiel ohne Muster
4. Motivation
5. Anwendbarkeit
6. Teilnehmer
7. Konsequenzen
8. Beispiel mit Muster
9. Implementation
10. Verwandte Muster
11. Bekannte Verwendungen

2. Zweck

- Kapselung einer Operation als Objekt
- Muster ermöglicht :
  - Definition einer neuen Operation ohne Veränderung der Klassen

3. Beispiel ohne Muster

- Umgeändertes Beispiel aus:
  „GoTo Java2“ von Guido Krüger

```
abstract class Mitarbeiter {
   public Mitarbeiter() {}
   public abstract double monatsBrutto();
   public abstract int jahresUrlaub();
}
class Arbeiter extends Mitarbeiter {
   double stundenlohn, anzahlstunden, gehalt;   int urlaub;
   public double monatsBrutto(){ return (gehalt = stundenlohn * anzahlstunden); }
   public int jahresUrlaub() { return (urlaub = 25);}
}
class Angestellter extends Mitarbeiter {
   double grundgehalt, zulage, gehalt;   int urlaub;
   public double monatsBrutto() {return (gehalt = grundgehalt + zulage);}
   public int jahresUrlaub() {return (urlaub = 28);}
}
```

4. Motivation

- abstrakte Klassenstruktur
- Operationen für viele der Knoten verschieden
- Verteilung der Operationen auf alle Klassen:
  - System ist schwer zu verstehen, zu verwalten, zu ändern
  - Hinzufügen einer neuen Operation bedeutet alle Klassen neu übersetzen

4. Motivation

- Besser:
- neue Operationen unabhängig vom Klassenbaum hinzufügen
- Klassenbaum unabhängig von den Operationen

4. Motivation

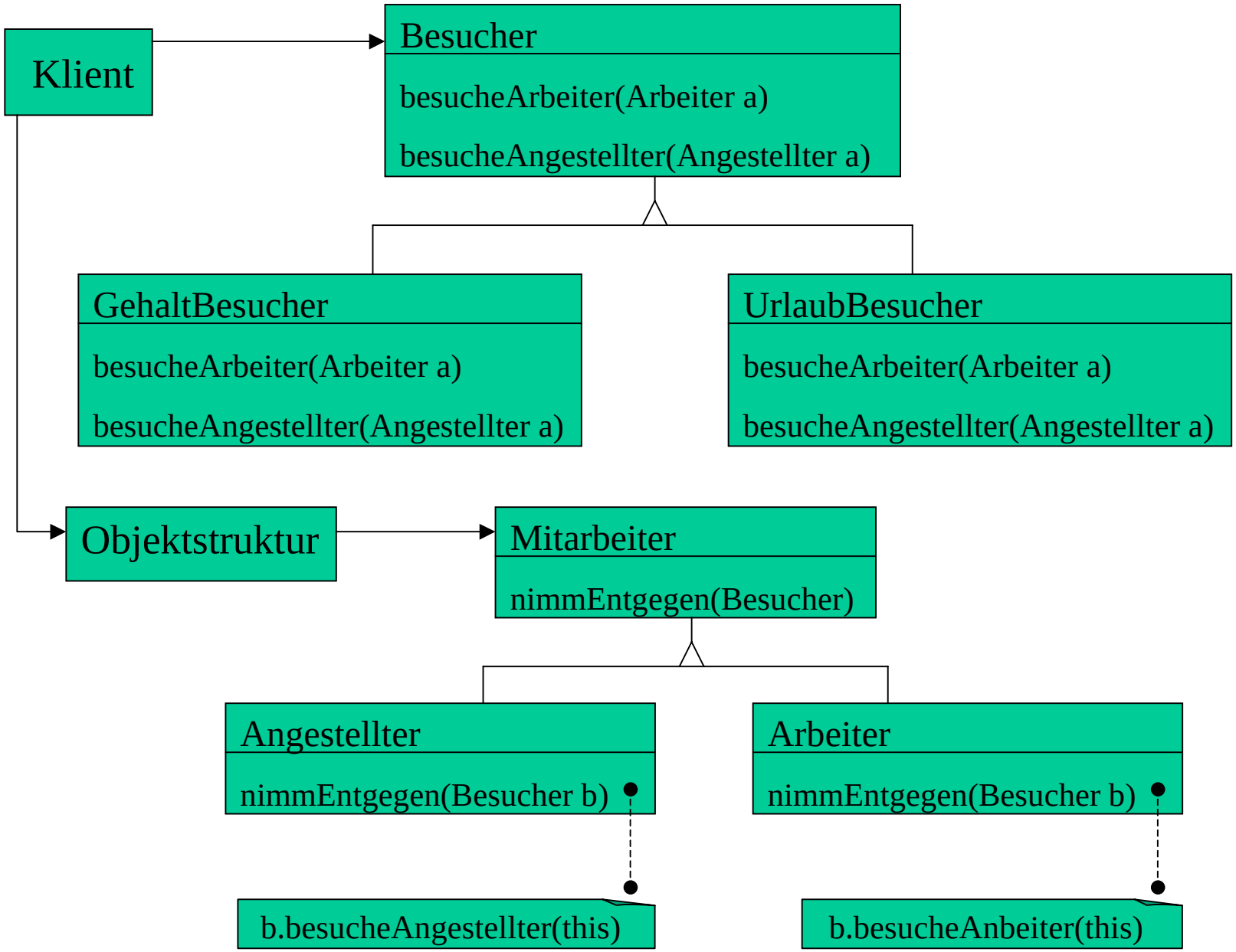
Lösung:

- verwandte Operationen in separate Objekte (=Besucher) zusammenfassen
- 2 Klassenhierarchien:
  - Knotenhierarchie für bearbeitete Elemente
  - Besucherhierarchie für die Operationen auf den Elementen
- neue Operation bildet eine neue Besucherklasse

4. Motivation

Ablauf:

- Besucher an Elemente des Klassenbaums übergeben
- Element ruft Besucher auf und übergibt sich selbst als Parameter
- Besucher führt Operationen für dieses Element aus


Klient
Besucher
besucheArbeiter(Arbeiter a)
besucheAngestellter(Angestellter a)
GehaltBesucher
besucheArbeiter(Arbeiter a)
besucheAngestellter(Angestellter a)
UrlaubBesucher
besucheArbeiter(Arbeiter a)
besucheAngestellter(Angestellter a)
Objektstruktur
Mitarbeiter
nimmEntgegen(Besucher)
Angestellter
nimmEntgegen(Besucher b)
Arbeiter
nimmEntgegen(Besucher b)
b.besucheAngestellter(this)
b.besucheAnbeiter(this)

5. Anwendbarkeit

- viele Klassen mit unterschiedlichen Schnittstellen und die Operationen hängen von den konkreten Klassen ab
- viele unterschiedliche und nicht miteinander verwandte Operationen
- mehrere Anwendungen nutzen die Objektstruktur
- Knotenhierarchie konstant aber Operationen werden geändert bzw. neu definiert

6. Teilnehmer

- Besucher:
  - deklariert eine Besuche-Operation für jede KonkretesElement-Klasse
  - übergibt ein KonkretesElement
- KonkreterBesucher:
  - implementiert jede von der Besucherklasse deklarierte Operation
  - liefert Kontext für Algorithmus und speichert dessen lokalen Zustand
  - speichert die während der Traversierung der Struktur angesammelten Ergebnisse

6. Teilnehmer

- Element:
  - definiert NimmEntgegen-Operation
- KonkretesElement:
  - definiert NimmEntgegen-Operation
- ObjektStruktur:
  - kann seine Elemente aufzählen
  - bietet möglicherweise abstrakte Schnittstelle, über die Besucher seine Elemente besucht
  - Kompositum oder Behälter (Liste, Menge)

7. Konsequenzen

Positiv:

- Hinzufügen neuer Operationen einfach
- Zusammenführen verwandter Operationen und Trennung von den anderen
- Klassenhierarchieübergreifende Besucher
- Ansammeln von Zustandsinformationen
  - bleiben in der Besucherklasse
  - sonst globale Variable oder während der Traversierung als Parameter weitergerecht

7. Konsequenzen

Negativ:

- Hinzufügen neuer KonkretesElement-Klassen (z.B. Beamte) ist schwer
- Aufbrechen der Kapselung
  - möglicher Zwang zu öffentlichen Operationen

8. Beispiel mit Muster

```
abstract class Mitarbeiter {
   public Mitarbeiter() {}
   public abstract void nimmEntgegen(Besucher b);
}
class Arbeiter extends Mitarbeiter {
   public void nimmEntgegen(Besucher b) { b.besucheArbeiter(this); }
}
class Angestellter extends Mitarbeiter {
   public void nimmEntgegen(Besucher b) { b.besucheAngestellter(this); }
}
```

```
abstract class Besucher {
   public abstract void besucheArbeiter(Arbeiter a);
   public abstract void besucheAngetellter(Angestellter a);
}
class GehaltBesucher extends Besucher {
   private double gehalt = 0.0;  /*...*/
   public void besucheArbeiter(Arbeiter a) {gehalt += stundenlohn*anzahlstunden;}
   public void besucheAngestellter(Angestellter a) {gehalt += grundgehalt + zulage; }
}
class UrlaubBesucher extends Besucher {
   private int urlaub; /*...*/
   public void besucheArbeiter(Arbeiter a) {urlaub = 25; /*...*/}
   public void besucheAngestellter(Angestellter a) {urlaub = 28; /*...*/}
}
```

9. Implementation

- 2 weitere Implementierungsaspekte:
  - Double-Dispatch
    - Operation hängt von der Art der Anfrage und den Typen zweier Empfänger ab
    - zentrale Überlegung des Musters:
    - nimmEntgegen() hängt vom Besucher und vom Element ab
    - Besucher können unterschiedliche Operationen für jede Elementklasse aufrufen
    - Bindung erfolgt zur Laufzeit
    - Elementschnittstelle effektiv durch Einführen einer neuen Besucherklasse erweitern

9. Implementation

- Zuständigkeit für die Traversierung der Objektstruktur
  - Objektstruktur zuständig für Iteration
    - Behälter iteriert über seine Elemente
  - Iterator verwenden
  - Besucher traversiert
    - Komplexe Traversierung, die von den Ergebnissen von Operationsaufrufen auf der Objektstruktur selbst abhängt(→ Code wird dupliziert)
- je Objektstruktur eine Besucherklasse (Oberklasse)

10. Verwandte Muster

- Kompositionsmuster
  - Visitor führt Operationen des Kompositums aus
- Interpretermuster
  - Visitor übernimmt die Übersetzung

11. Bekannte Verwendungen

- Mark Linton prägte den Begriff Visitor
  - Spezifikation für Fresco-Application-Toolkit
- Smalltalk-80 Übersetzer
  - Besucherklasse ProgrammNode-Enumerator für Quelltextanalyse
- IRIS-Inventor
  - Klassenbibliothek zur Entwicklung von graphischen 3D-Anwendungen

Vielen Dank für die Aufmerksamkeit!

Fragen?