Programmieren in Java

Vorlesung 07: Composite Classes

Prof. Dr. Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

SS 2017

Inhalt

Codequalität: Immutable Inputs

Vergleiche Vorlesung 04!

Immutable = unveränderlich

Codequalität: Immutable Inputs

Vergleiche Vorlesung 04!

Immutable = unveränderlich

Java unterscheidet

- primitive Datentypen: bool, byte, int, short, long, float, double, char werden direkt als 32 Bit Wort dargestellt (long, double: 64 Bit)
- Referenzdatentypen: Arrays, Objekttypen, (insbesondere Collections!) werden durch Referenz (Adresse eines Speicherbereichs) dargestellt

Parameterübergabe

Unterschied bei der Übergabe als Parameter

- Werte von primitiven Datentypen werden kopiert
  - Zuweisungen an Parameter sind nach außen nicht sichtbar
- Für Referenztypen wird die Adresse übergeben,
  - Änderungen am Parameter sind für den Aufrufer (global) sichtbar
  - Oft unerwartet, **muss** dokumentiert werden, falls dieses Verhalten gewünscht ist!

Parameterübergabe

Unterschied bei der Übergabe als Parameter

- Werte von primitiven Datentypen werden kopiert
  - Zuweisungen an Parameter sind nach außen nicht sichtbar
- Für Referenztypen wird die Adresse übergeben,
  - Änderungen am Parameter sind für den Aufrufer (global) sichtbar
  - Oft unerwartet, **muss** dokumentiert werden, falls dieses Verhalten gewünscht ist!

Konvention (ab sofort)

- Parameter von (getesteten) Methoden dürfen nicht verändert werden
- Zuweisungen an Parameter aller Art sind code smells und verboten
- Objekte werden nach der Initialisierung nicht mehr verändert, wann immer das möglich ist.

Beispiel

Blumenstrauss

Stichworte

Unmodifiable

Keine Kopie, aber Schreiboperationen sind ausgeschlossen

Beispiel: saferGetContent()

Shallow Copy

Kopiere nur die äußere Struktur, übernehme die Referenzen der Elemente

Beispiel: shallowCopyContent()

Deep Copy

Kopiere die komplette Struktur, keine Referenzen zum Original

Beispiel: deepCopyContent()

Aus der Praxis

Rational

```
1  public class Rational {
2      private long nominator;
3      private long denominator;
4      // ...
5      public Rational simplify() {
6          long a = this.nominator;
7          long b = this.denominator;
8          long c = 1;
9
10         while (b != 0) {
11             c = a % b;
12             a = b;
13             b = c;
14         }
15
16         this.nominator /= a;
17         this.denominator /= a;
18
19         return this;
20     }
21 }
```

Rational #2

```
1  public class Rational {
2      private long nominator;
3      private long denominator;
4      // ...
5      @Override
6      public boolean equals(Object r) {
7          if (r == this) {
8              return true;
9          }
10         if (!(r instanceof Rational)) {
11             return false;
12         }
13         Rational rational = (Rational) r;
14         return Objects.equals(this.toDouble(), rational.toDouble());
15
16     }
17 }
```

Interfaces und Unions

Die Tierhandlung

Aufgabe

Ein Tierhändler möchte seinen Katalog interaktiv gestalten. Er verkauft verschiedene Tiergattungen: Hunde, Katzen, Mäuse, Fische, Kanarienvögel usw. Jedes Tier kann: ein Geräusch machen, sich bewegen. Jedes Tier hat einen Mindestpreis.

Die Tierhandlung

Aufgabe

Ein Tierhändler möchte seinen Katalog interaktiv gestalten. Er verkauft verschiedene Tiergattungen: Hunde, Katzen, Mäuse, Fische, Kanarienvögel usw. Jedes Tier kann: ein Geräusch machen, sich bewegen. Jedes Tier hat einen Mindestpreis.

Modellierung

- Verwende ein Interface zur Modellierung des Konzepts Tiergattung
- Das Interface definiert die Signaturen der Operationen
- Verwende je eine Klasse zur Modellierung der einzelnen Tiergattungen

Modellierung

```
1 public interface Animal {
2    public String makeSound();
3    public String move();
4    public int getPrice();
5 }
```

- Jede Implementierung von Animal braucht diese drei Methoden
- Alle Methoden eines Interface müssen **public** sein

Beispiel: Dog implementiert Animal

```
1  public class Dog implements Animal {
2    // predefined constructor: new Dog()
3    public String makeSound() {
4      return ...
5    }
6    public String move() {
7      return ...
8    }
9    public int getPrice() {
10     return 150;
11   }
12 }
```

- Das **implements** Animal in der Klassendeklaration zeigt Java an, dass Dog das Interface Animal implementiert.

Beispiel: Cat, Mouse, Fish, Canary

```
1 public class Cat implements Animal { ... }
2 public class Mouse implements Animal { ... }
3 public class Fish implements Animal { ... }
4 public class Canary implements Animal { ... }
```

Verwendung von Animals

```
1  public class Animals {
2    public void testAnimals() {
3      Animal doggy = new Dog();
4      Animal kitty = new Cat();
5      Animal mousy = new Mouse();
6      Animal fishy = new Fish();
7      Animal canary = new Canary();
8
9      doggy.makeSound();
10     kitty.makeSound();
11     mousy.move();
12     fishy.move();
13     canary.getPrice();
14   }
15 }
```

- Animal ist ein Referenztyp (denn er steht für ein Objekt)
- Jedes Objekt einer implementierenden Klasse ist ein Animal
- Jede Interfacemethode kann auf Animal aufgerufen werden

Interfaces, Unions und Rekursion

Beispiel 1: Binärbaum

Aufgabe

Ein Binärbaum ist entweder ein Blattknoten mit einer Zahl oder ein innerer Knoten, der zwei Binärbäume enthält. Wir sind interessiert an der Anzahl der Knoten, der Höhe eines Baums und der Spiegelung eines Baums.

Beispiel 1: Binärbaum

Aufgabe

Ein Binärbaum ist entweder ein Blattknoten mit einer Zahl oder ein innerer Knoten, der zwei Binärbäume enthält. Wir sind interessiert an der Anzahl der Knoten, der Höhe eines Baums und der Spiegelung eines Baums.

Modellierung

- Interface BTree
- mit Operationen count(), height(), mirror()
- Je eine Klasse Leaf und Branch
- Nützlich: eine Klasse BTrees mit Fabrikmethoden zur Erzeugung von BTree Objekten

BTree Interface

```
1 public interface BTree {
2   public int count();
3   public int height();
4   public BTree mirror();
5 }
```

- Die Knotenklassen Leaf und Branch müssen das Interface BTree implementieren.

Beispiel 2: Interpretation

Aufgabe

Paul möchte einen Auswerter (Interpreter) für arithmetische Ausdrücke mit ganzen Zahlen schreiben. Ein Ausdruck ist entweder eine Variable oder eine Konstante oder eine einstellige Operation oder eine zweistellige Operation. Die Werte der Variablen sind durch einen Speicher (Abbildung von Variablennamen auf ganze Zahlen) gegeben.

Beispiel 2: Interpretation

Aufgabe

Paul möchte einen Auswerter (Interpreter) für arithmetische Ausdrücke mit ganzen Zahlen schreiben. Ein Ausdruck ist entweder eine Variable oder eine Konstante oder eine einstellige Operation oder eine zweistellige Operation. Die Werte der Variablen sind durch einen Speicher (Abbildung von Variablennamen auf ganze Zahlen) gegeben.

Modellierung

- Interface Expression, sonst analog zu BTree
- Variablennamen sollen durch strings modelliert werden.
- Modellierung des Speichers durch eine Abbildung (Map) von String nach Integer

Expression Interface

```
1 public interface Expression {
2   public int eval(State state);
3 }
```

Modellierung des Speichers

```
1   public class State extends HashMap<String,Integer> {}
```

- Definiere State als eine spezialisierte HashMap (Details später)
- Konvention: es darf nur auf definierte Variable zugegriffen werden

Zusammenfassung

Zusammenfassung

Verwende Interfaces

- zur Deklaration von gemeinsamen Operationen mehreren Klassen
- zur Modellierung von Alternativen
- zur Modellierung von Alternativen mit Rekursion

Diese Art der Modellierung heißt

Composite Pattern.

Siehe auch Google oder wikipedia:
https://en.wikipedia.org/wiki/Composite_pattern

Fragen