UNIKASSEL VERSITÄT

Research Group Software Engineering

Programmiermethodik

Übung 5

Sommersemester 2010

Fachgebiet Software Engineering

Andreas Scharf

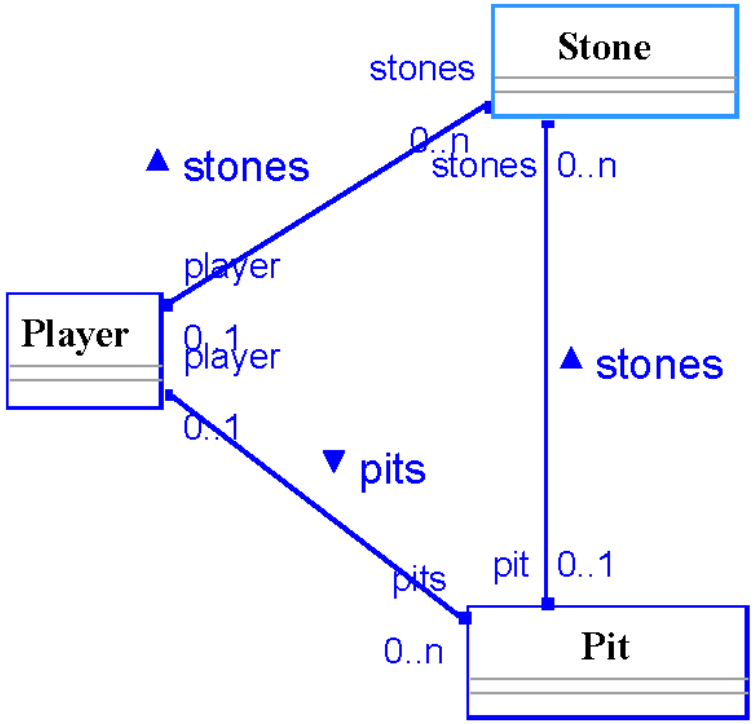
andreas.scharf@cs.uni-kassel.de

Agenda

- **Besprechung HA4**
- **Methodenentwurf**
  - Textueller Entwurf
  - Implementierung in Java
  - Zetteltest zur Verifikation
- **eDOBS**
- **Besprechung HA5**

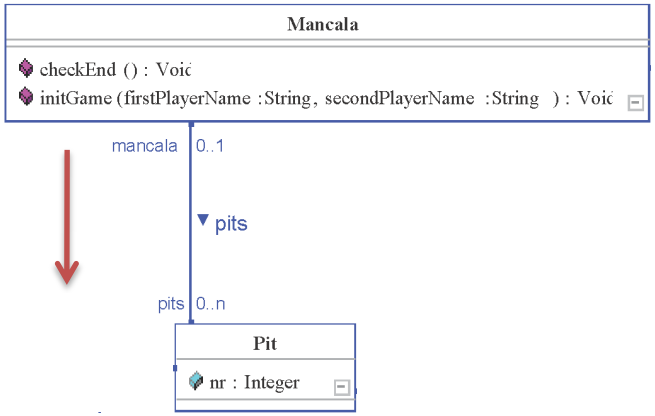
Besprechung HA4 I

- **Aufgabe 1: Implementierung Klassendiagramm**
  - Implementierung erfolgt strukturiert durch vorgegebenes Schema
    1. Klassen
    2. Attribute
    3. Methoden
    4. Assoziationen
  - Sichern von referentieller Integrität
- **Referentielle Integrität**
  - Immer bezüglich **einer** Assoziation



Besprechung HA4 II

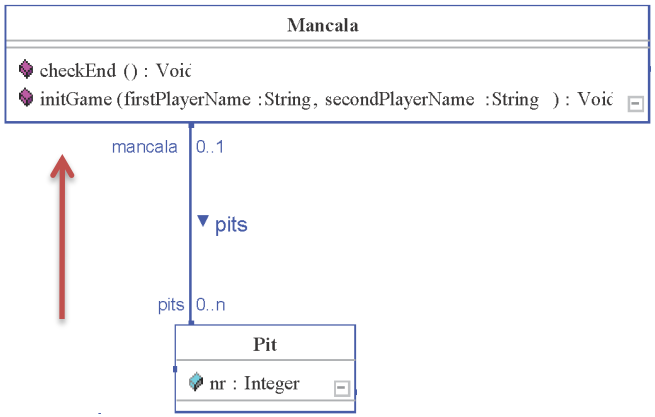
- **Aufgabe 2: JUnit Tests**
- **Sicherstellen, dass referentielle Integrität korrekt implementiert ist. Beispiel:**




```
@Test
public void testMancalaPitReferentialIntegrity()
{
   // Create start situation
   Mancala mancala = new Mancala();
   Pit pit = new Pit();

   // Add the pit by calling mancala.addToPit()
   mancala.addToPit(pit);

   // Check referential integrity
   assertTrue("Pit was not added to mancala",mancala.hasInPit(pit));
   assertSame("Mancala not found",mancala, pit.getMancala());

   // Remove the pit
   mancala.removeFromPit(pit);

   // Check referential integrity
   assertFalse("Pit is still contained", mancala.hasInPit(pit));
   assertNull("Mancala is not null", pit.getMancala());
}
```

Besprechung HA4 III

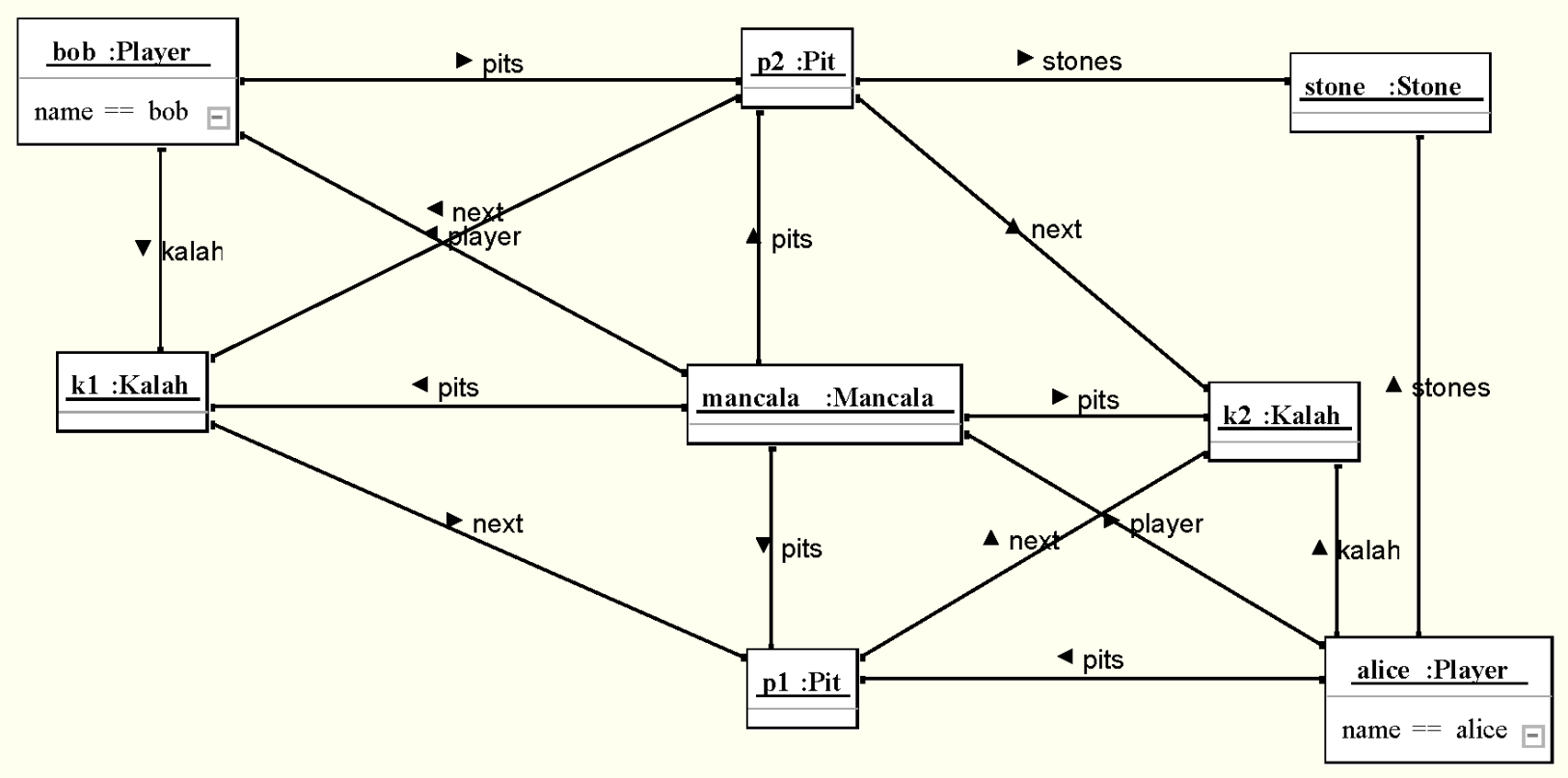
- **Rückrichtung:**




```
@Test
public void testPitMancalaReferentialIntegrity()
{
   // Create start situation
   Mancala mancala = new Mancala();
   Pit pit = new Pit();

   // Set mancala by calling  pit.setMancala()
   pit.setMancala(mancala);

   // Check referential integrity
   assertTrue("Pit was not added to mancala",mancala.hasInPit(pit));
   assertSame("Mancala not found",mancala, pit.getMancala());

   // Remove mancala
   pit.setMancala(null);

   // Check referential integrity
   assertFalse("Pit is still contained", mancala.hasInPit(pit));
   assertNull("Mancala is not null", pit.getMancala());
}
```

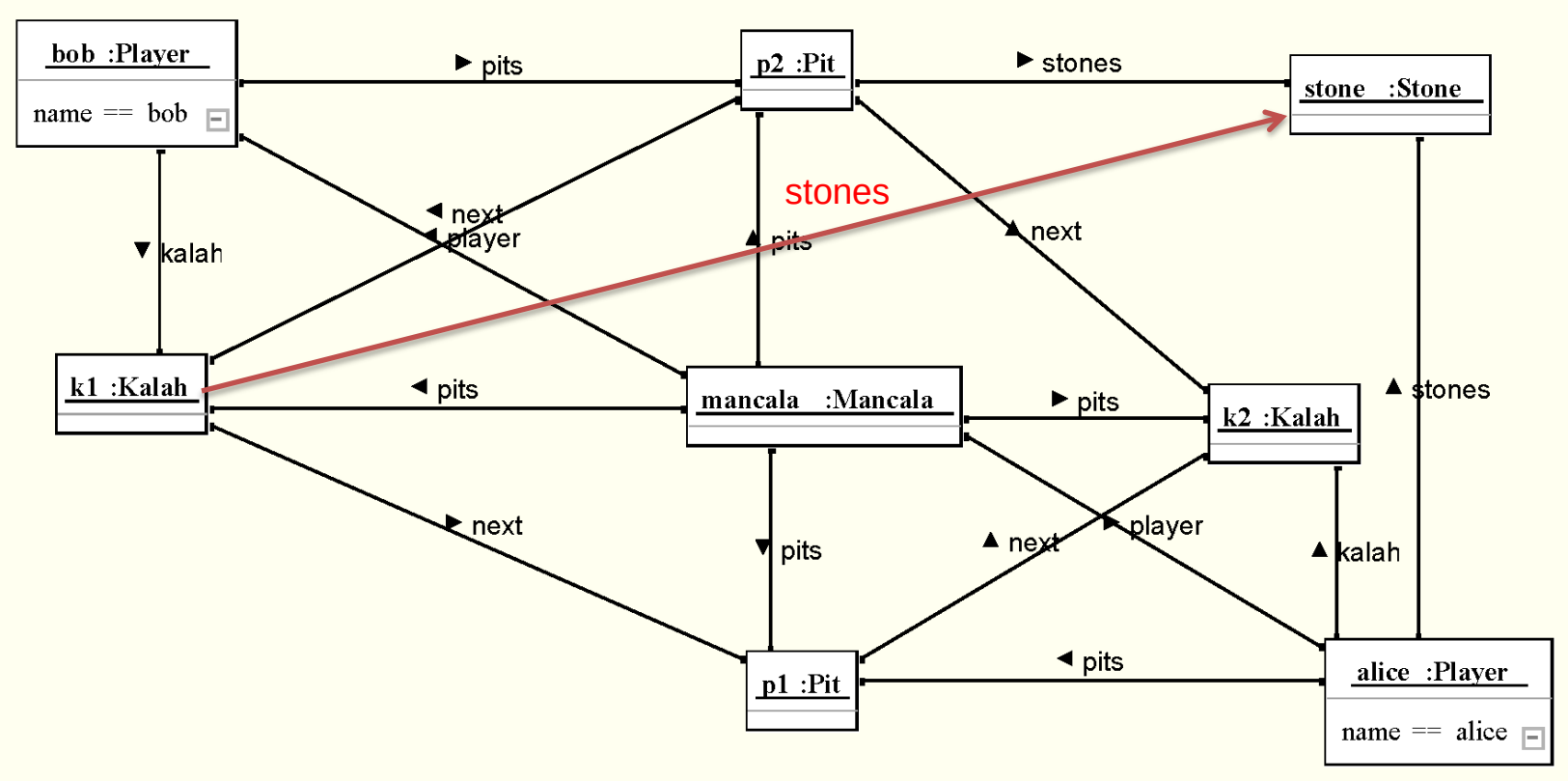
Besprechung HA4 IV

- **Zusatzaufgabe:**



- **Aufruf:** `k1.getStones().add(stone)`

Besprechung HA4 V



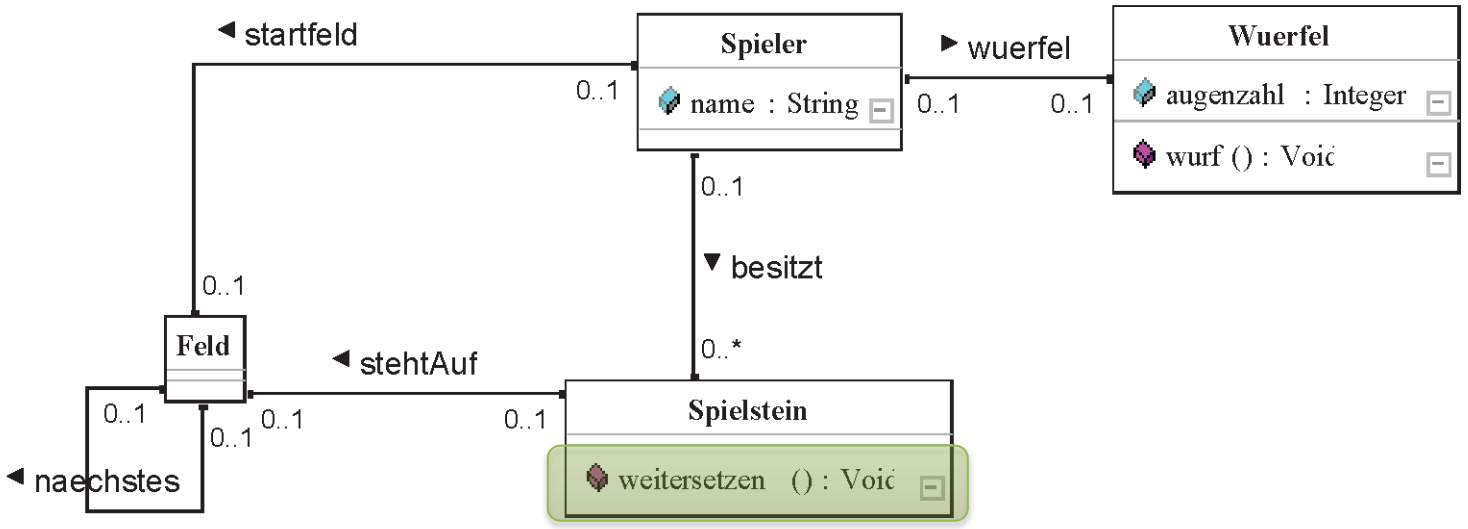
- **Probleme?**
  - Referentielle Integrität zerstört!

Methodenentwurf I

- **Nach Implementierung des Klassendiagramms erfolgt der Methodenentwurf**
- **Zunächst textuell:**
  - Schritte aufschreiben, die die Methode erledigen soll
  - Bedingungen: „Wenn noch nicht alle Credits, dann….“
  - Sprünge: „Gehe zu Schritt X“

Methodenentwurf II

- **Beispiel: „Mensch ärgere dich nicht“**



- **Textueller Methodenentwurf für: `weitersetzen():void` der Klasse `Spielstein`**

Methodenentwurf III – Praktische Übung

- **Anforderung:**
  - Die Spielfigur soll so viele Felder weitergesetzt werden, wie der Würfel anzeigt
- **Mögliche Lösung:**
  1. Überprüfe ob der Spieler den Würfel besitzt
  2. Merke dir die Augenzahl des Würfels in einer Variable `„nochGehen“`
  3. Setze den Spielstein ein Feld nach vorn.
  4. Dekrementiere `„nochGehen“`
  5. Wenn `„nochGehen“` größer als 0 gehe zu Schritt 3. Ansonsten fertig.

Methodenentwurf IV – Praktische Übung

- **Implementierung in Java**
- **Eclipse Projekt vom Blog herunterladen:**
  - http://seblog.cs.uni-kassel.de/wp-content/uploads/2010/05/PMSS2010LudoEclipseProject.zip
- **In Eclipse importieren**
- **JUnit Test schreiben, der:**
  - 4 Felder erzeugt und diese miteinander verbindet
  - 1 Spieler
  - 1 Spielfigur die dem Spieler zugeordnet ist und auf dem ersten Feld steht
  - 1 Würfel mit Augenzahl 2
  - Methode `weitersetzen()` auf dem Spielstein aufrufen
- **Methode `weitersetzen():void` der Klasse `Spielstein` nach textuellem Entwurf implementieren**

Methodenentwurf V

- **Mögliche Implementierung**

```
public void weitersetzen ()
    {
       Spieler spieler = getSpieler();
       Wuerfel wuerfel = spieler.getWuerfel();
       if(wuerfel != null)
       {
          int nochGehen = wuerfel.getAugenzahl();

          while(nochGehen > 0)
          {
             Feld aktF = getStehtAuf();
             Feld naechstes = aktF.getZu();
             setStehtAuf(naechstes);
             nochGehen--;
          }
       }
    }
```

- **Verifikation durch Zetteltest**

Methodenentwurf VI - Zetteltest

- **Zetteltest**
  - Hilft Methoden zu verstehen
  - Fehler aufzudecken
  - Ist „manuelles Debuggen“
- **Zetteltest Methode** `weitersetzen():void` **der Klasse** `Spielstein`

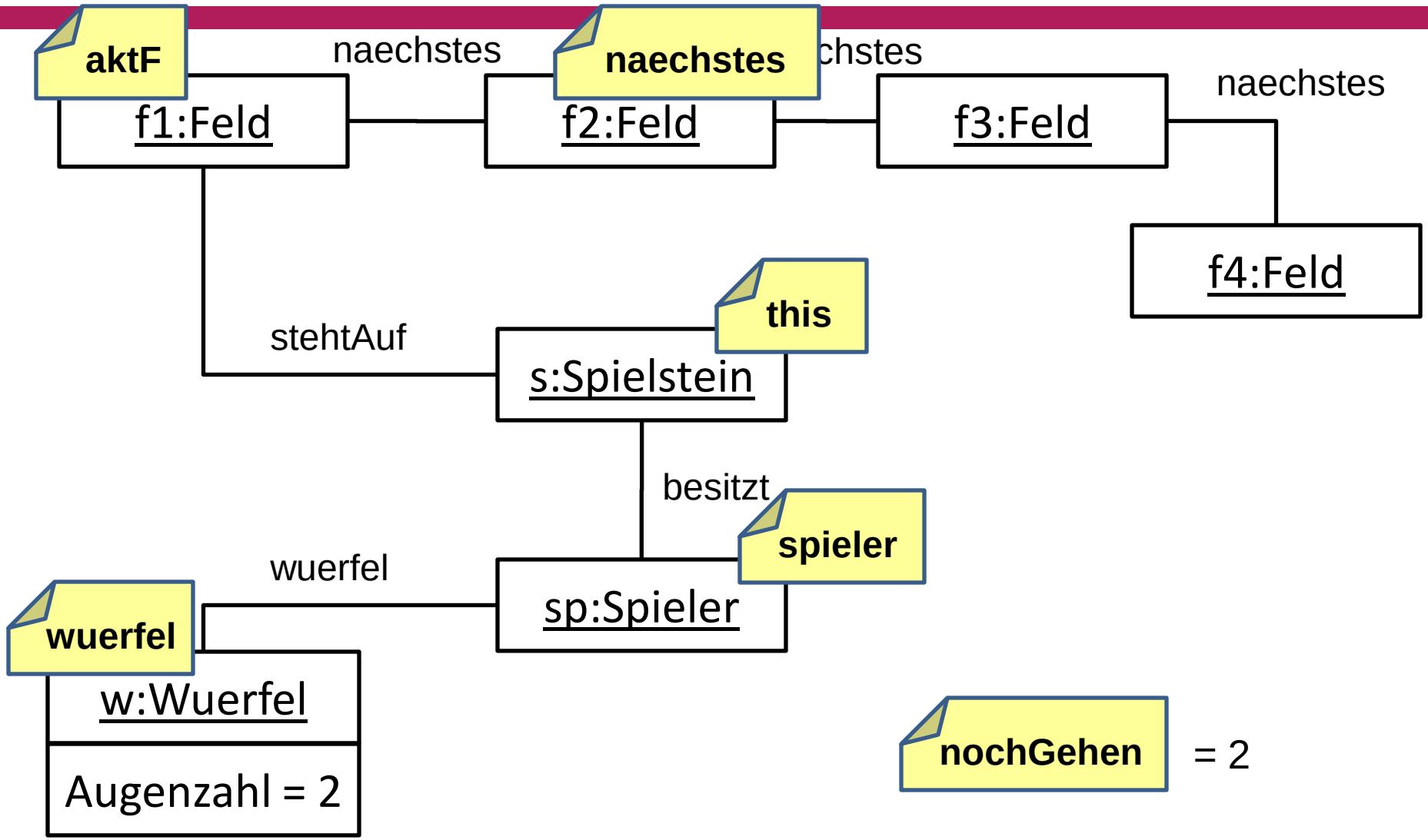
Methodenentwurf V

- **Mögliche Implementierung**

```
public void weitersetzen ()
    {
        Spieler spieler = getSpieler();
        Wuerfel wuerfel = spieler.getWuerfel();
        if(wuerfel != null)
        {
            int nochGehen = wuerfel.getAugenzahl();

            while(nochGehen > 0)
            {
                Feld aktF = getStehtAuf();
                Feld naechstes = aktF.getZu();
                setStehtAuf(naechstes);
                nochGehen--;
            }
        }
    }
```

- **Verifikation durch Zetteltest**

Methodenentwurf VII - Zetteltest



Methodenentwurf V

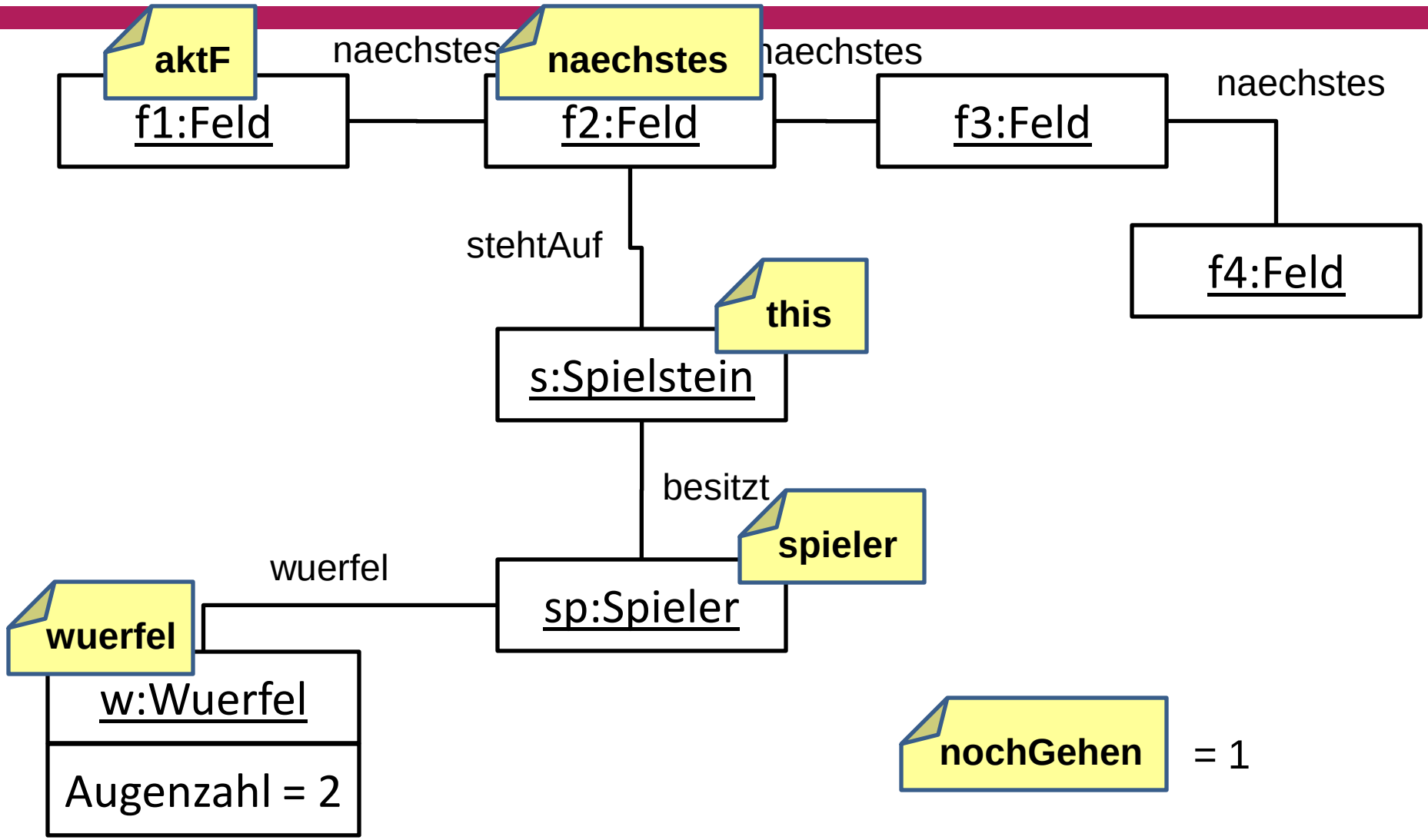
- **Mögliche Implementierung**

```
public void weitersetzen ()
    {
        Spieler spieler = getSpieler();
        Wuerfel wuerfel = spieler.getWuerfel();
        if(wuerfel != null)
        {
            int nochGehen = wuerfel.getAugenzahl();

            while(nochGehen > 0)
            {
                Feld aktF = getStehtAuf();
                Feld naechstes = aktF.getZu();
                setStehtAuf(naechstes);
                nochGehen--;
            }
        }
    }
```

- **Verifikation durch Zetteltest**

= 2

Methodenentwurf VIII - Zetteltest



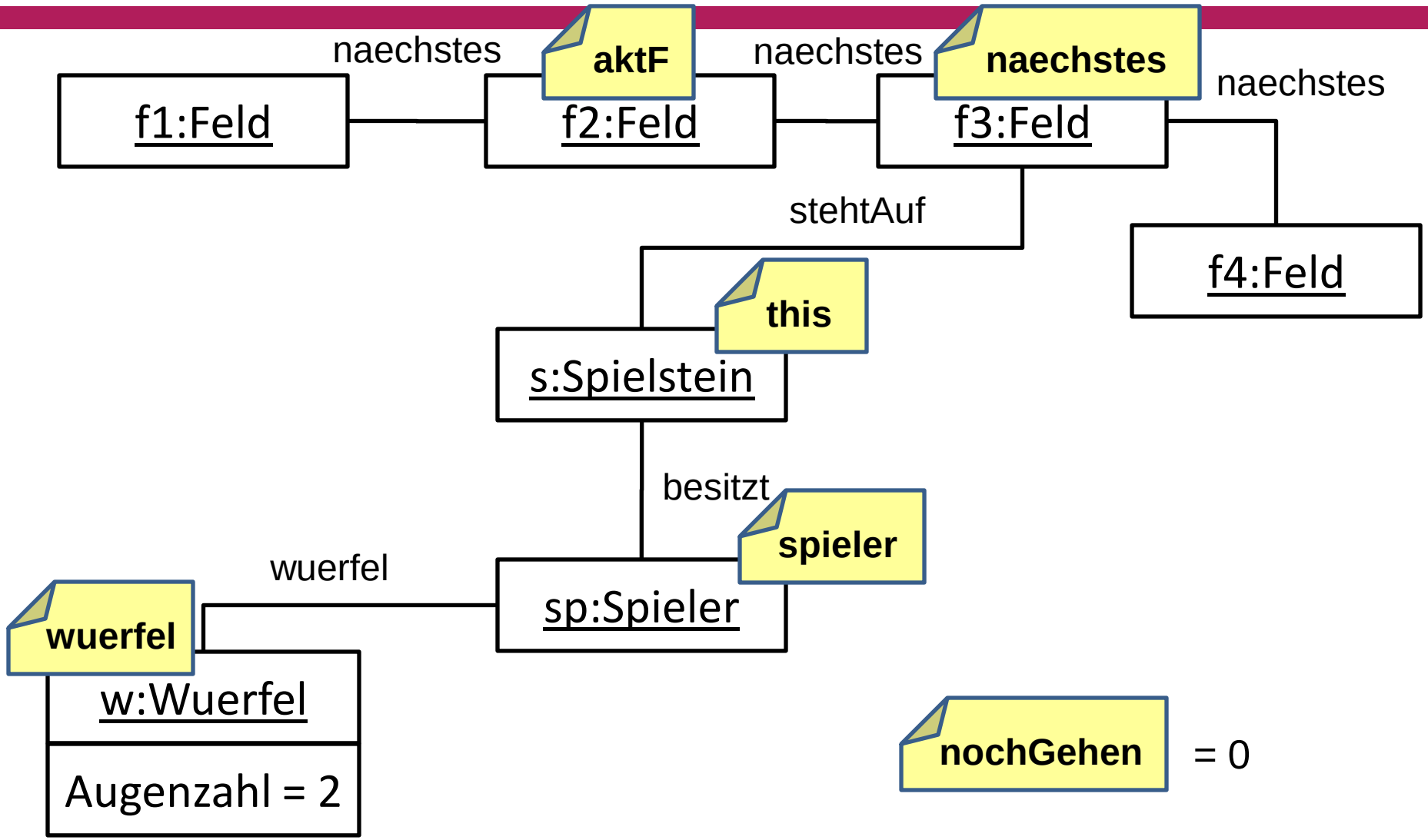
Methodenentwurf V

- **Mögliche Implementierung**

```
public void weitersetzen ()
    {
        Spieler spieler = getSpieler();
        Wuerfel wuerfel = spieler.getWuerfel();
        if(wuerfel != null)
        {
            int nochGehen = wuerfel.getAugenzahl();

            while(nochGehen > 0)
            {
                Feld aktF = getStehtAuf();
                Feld naechstes = aktF.getZu();
                setStehtAuf(naechstes);
                nochGehen--;
            }
        }
    }
```

- **Verifikation durch Zetteltest**

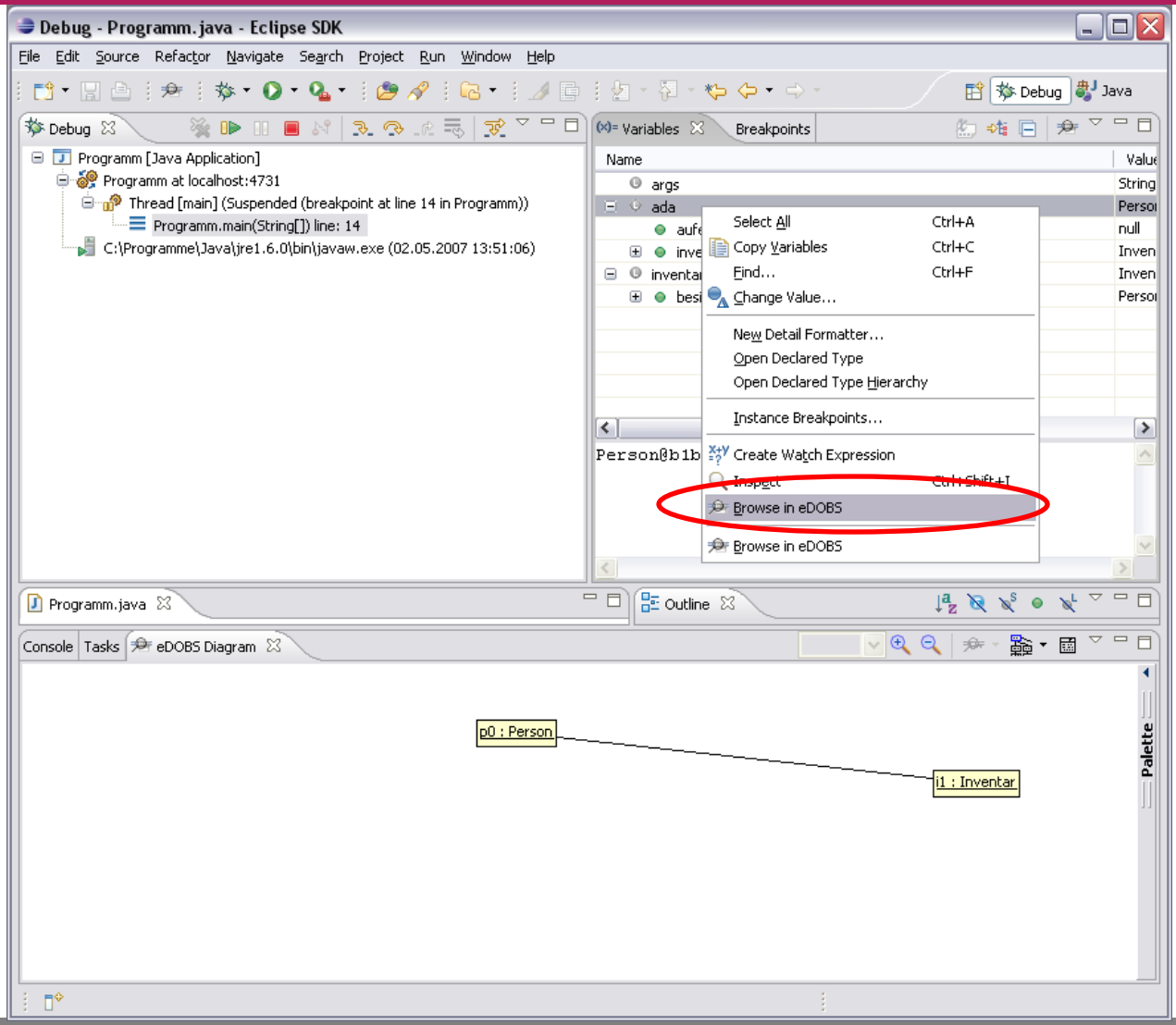
Methodenentwurf IX - Zetteltest



eDOBS I

- **eDOBS (eclipse Dynamic Object Browsing System)**
  - Update Site: http://www.se.eecs.uni-kassel.de/se/fileadmin/se/projects/eDOBS/update
- **„Variables-View als Objektdiagramm“**
- **Zeigt die Objekte im Speicher (Java Heap) grafisch an**
- **Klassisches Objektdiagramm**
  - Instanzen
  - Attributbelegungen
  - Links
- **Erlaubt Interaktion mit den Objekten**
  - Ändern von Attributwerten
  - Aufruf von Methoden
  - Erzeugen/Löschen von Objekten und Links

eDOBS II

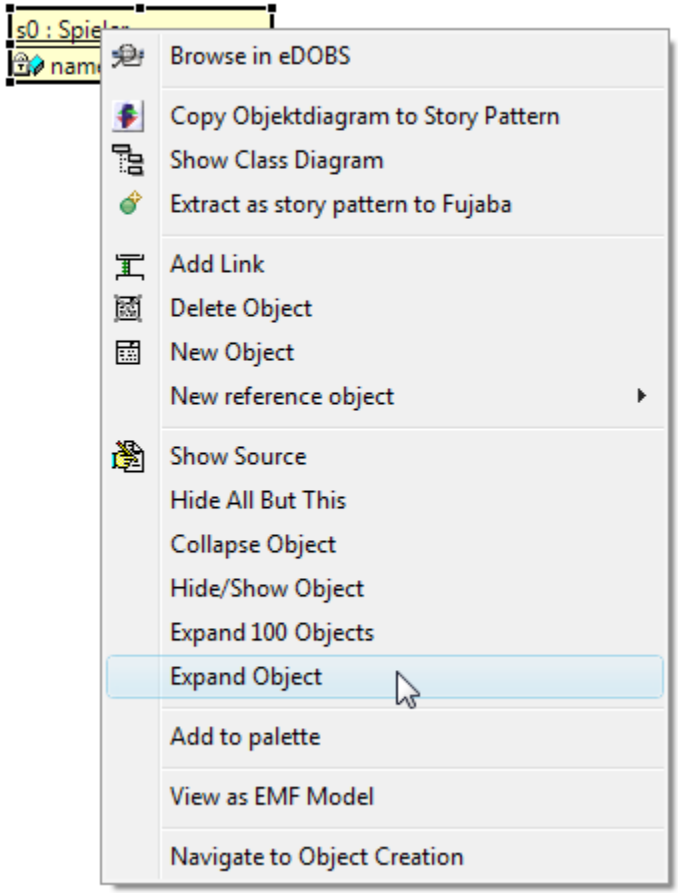

Debug - Programm.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Java
Debug
Programm [Java Application]
Programm at localhost:4731
Thread [main] (Suspended (breakpoint at line 14 in Programm))
Programm.main(String[]) line: 14
C:\Programme\Java\jre1.6.0\bin\javaw.exe (02.05.2007 13:51:06)
Variables
Breakpoints
Name
args
ada
Select All Ctrl+A
Copy Variables Ctrl+C
Find... Ctrl+F
Change Value...
New Detail Formatter...
Open Declared Type
Open Declared Type Hierarchy
Instance Breakpoints...
Create Watch Expression
Browse in eDOBS
Browse in eDOBS
Programm.java
Outline
Console
Tasks
eDOBS Diagram
p0 : Person
i1 : Inventar
Palette

eDOBS III

- **Breakpoint im Programm setzen**
- **Starte das Programm im Debug-Modus**
  - Debug as -> Java Application, NICHT Debug as eDOBS VM
- **Warte bis der Debugger auf dem Breakpoint hält**
- **Wähle eine Objekt-Variable im Variables-View (z.b. this), Rechtsklick, „Browse in eDOBS“**
- **Wechsle die Perspektive zu „eDOBS (Debug)“**

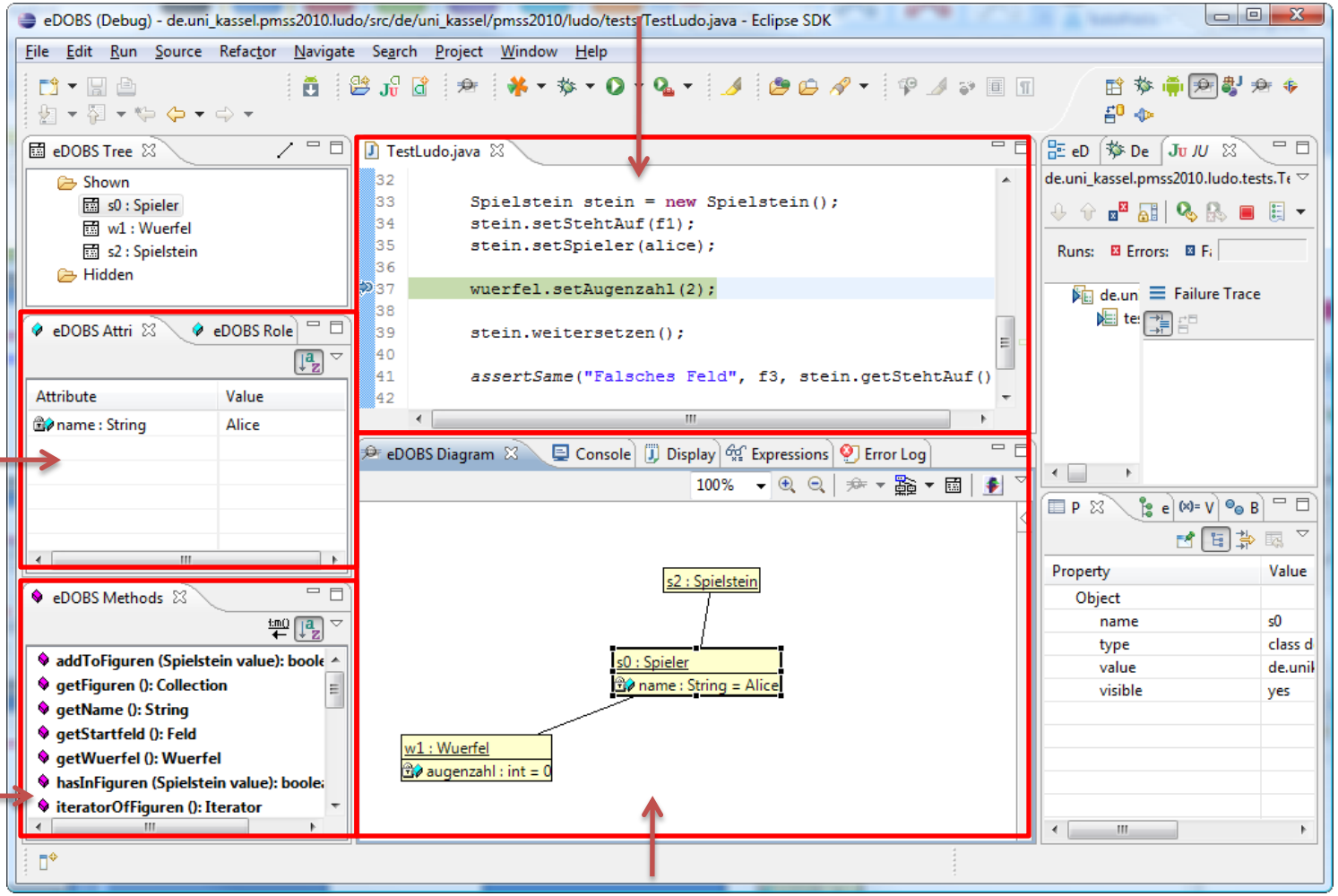
eDOBS IV

- **Objekte „expandieren“**
  - Zeige alle Nachbarn
  - Rechtsklick auf Objekt -> Expand



eDobs V

Quelltext



Attribute anzeigen/ ändern

Methoden aufrufen

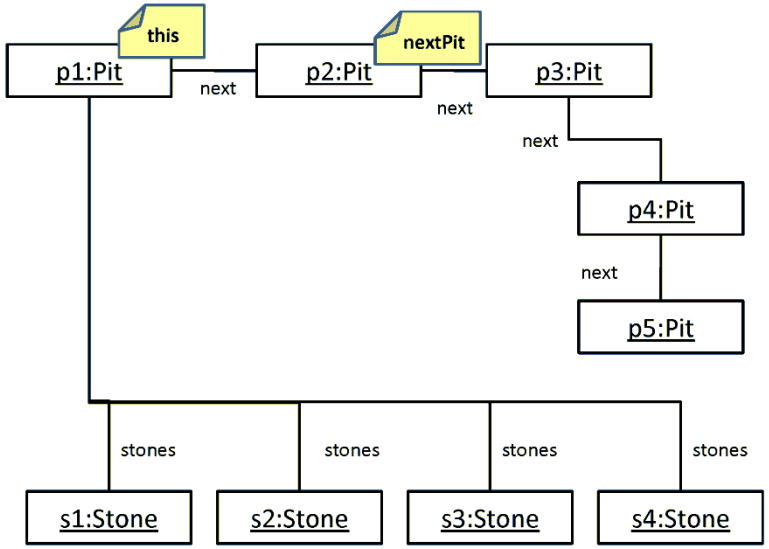
Objektdiagramm

Besprechung HA5 I

- **Abgabe in 2 Wochen (26.05.2010, 23.59 Uhr)**
- **Vorbereitung:**
  - eDOBS installieren
  - Mancala Projekt herunterladen (optional)
    - Enthält bereits Implementierung des Klassendiagramms und der Methode `Pit::moveStones():void`
- **Aufgabe 1**
  - `Mancala::initGame(…):void` implementieren
    - Zwei Spieler mit den als Parameter übergebenen Namen
    - Alle 12 Löcher
    - Vier Steine für jedes Loch
    - Zwei Kalahs

Besprechung HA5 II

- **Aufgabe 2**
  - eDOBS Screentshot von Objektsituation am ende von `Mancala::initGame(…):void`
- **Aufgabe 3**
  - Startsituation ist vorgegeben
  - Zetteltest zur Methode `Pit::moveStones():void`
  - eDOBS Screenshot



Besprechung HA5 III

- **Zusatzaufgabe**
  - JUnit Test schreiben der prüft, ob Regel „Steine klauen“ funktioniert
    - **Regel**: „Trifft einer der Spieler mit dem letzten Stein in der eigenen Hälfte auf eine leere Grube, darf er diese Steine und die des Gegners aus der gegenüberliegenden Grube nehmen und in seine Kalah legen. Auf der gegnerischen Seite muss mindestens ein Stein übrigbleiben, damit derjenige Spieler weiterspielen kann.“
  - Methode `Pit::moveStones():void` erweitern
  - Zwei eDOBS Screenshots:
    - Startsituation
    - Endsituation

Ende

Schönes WE!