UNIKASSEL
VERSITÄT

Research Group
Software Engineering

Programmiermethodik

Übung 3

Sommersemester 2011

Fachgebiet Software Engineering

Andreas Scharf

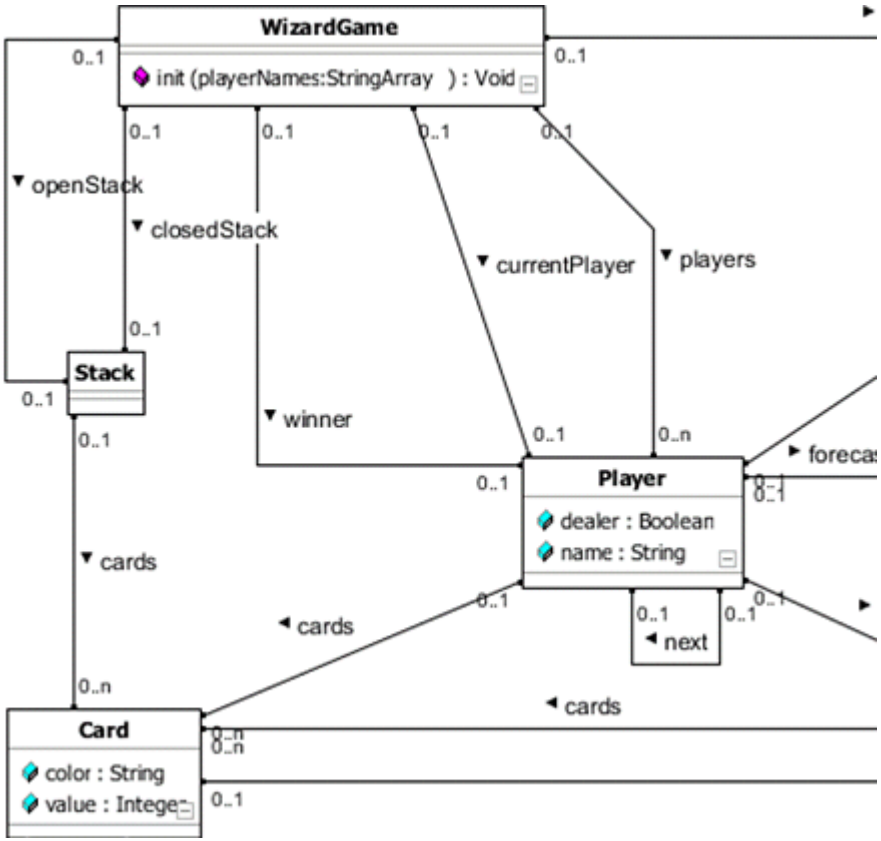
andreas.scharf@cs.uni-kassel.de

Agenda

- **Besprechung HA2**
- **Methodenentwurf**
  - Textueller Entwurf
  - Implementierung in Java
  - Zetteltest zur Verifikation
- **eDOBS**
- **Vorschau HA3**

Besprechung HA2 I

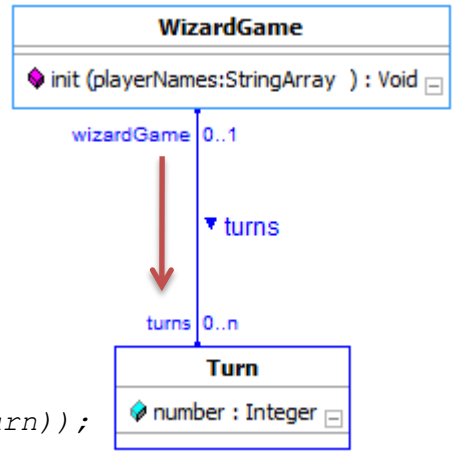
- **Aufgabe 1: Implementierung Klassendiagramm**
  - Implementierung erfolgt strukturiert durch vorgegebenes Schema
    1. Klassen
    2. Attribute
    3. Methoden
    4. Assoziationen
  - Sichern von referentieller Integrität
- **Referentielle Integrität**
  - Immer bezüglich **einer** Assoziation



Besprechung HA2 II

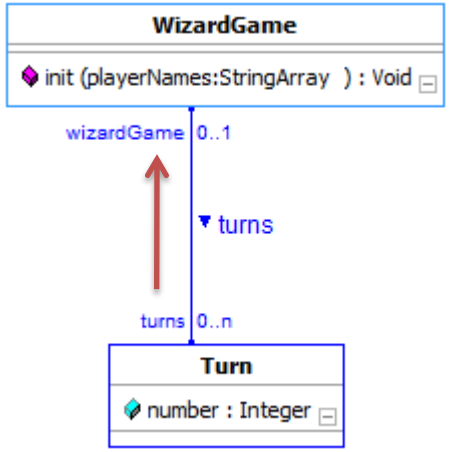
- **Zusatzaufgabe: JUnit Tests**
- **Sicherstellen, dass referentielle Integrität korrekt implementiert ist. Beispiel:**

```
@Test
public void testWizardGameTurnReferentialIntegrity()
{
   // Create start situation
   WizardGame wizardGame = new WizardGame();
   Turn turn = new Turn();

   // Add the turn to the game
   wizardGame.addToTurns(turn);

   // Check referential integrity
   assertTrue("Turn was not added to wizard game", wizardGame.hasInTurns(turn));
   assertSame("Wizard game not found", wizardGame, turn.getWizardGame());

   // Remove the turn
   wizardGame.removeFromTurns(turn);

   // Check referential integrity
   assertFalse("Turn is still contained", wizardGame.hasInTurns(turn));
   assertNull("Turn still references the wizard game", turn.getWizardGame());
}
```



Besprechung HA2 III

- **Rückrichtung:**



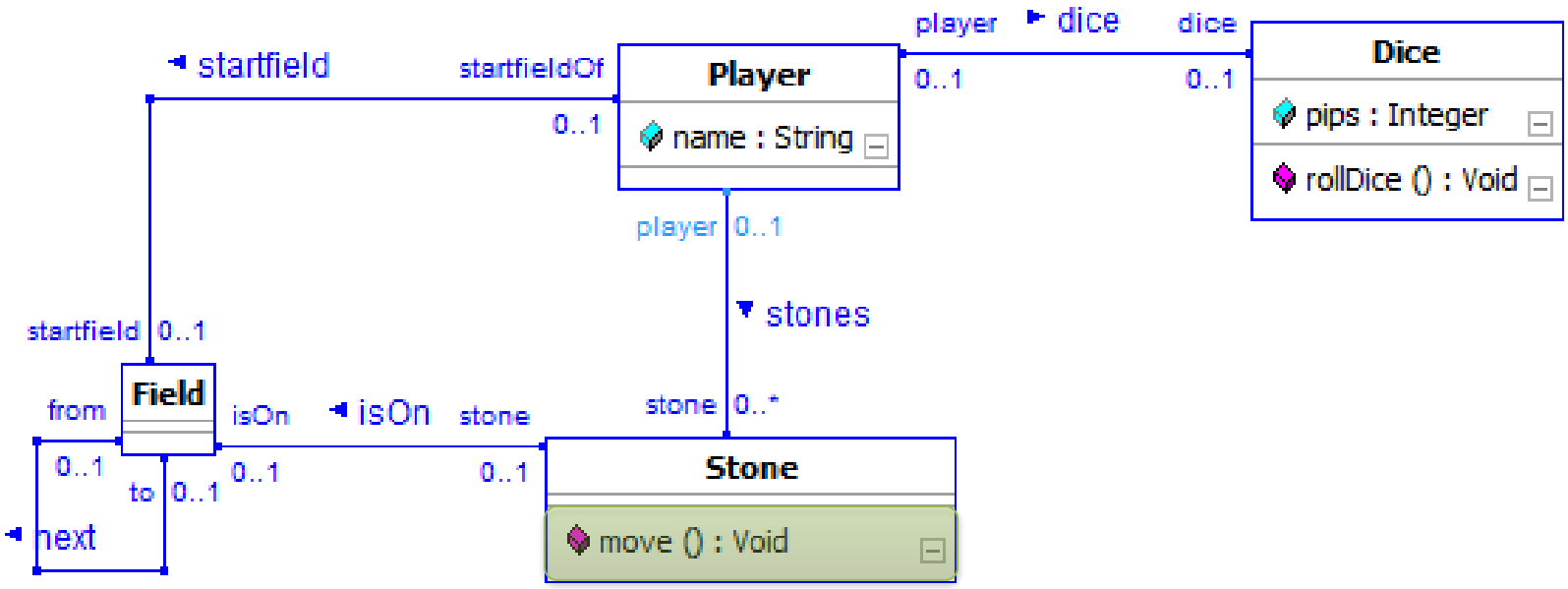

```
@Test
public void testTurnWizardGameReferentialIntegrity()
{
   // Create start situation
   WizardGame wizardGame = new WizardGame();
   Turn turn = new Turn();

   // Set wizard game
   turn.setWizardGame(wizardGame);

   // Check referential integrity
   assertTrue("Turn was not added to wizard game", wizardGame.hasInTurns(turn));
   assertSame("Wizard game not found", wizardGame, turn.getWizardGame());

   // Remove the game
   turn.setWizardGame(null);

   // Check referential integrity
   assertFalse("Turn is still contained", wizardGame.hasInTurns(turn));
   assertNull("Turn still references the wizard game", turn.getWizardGame());
}
```

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: `move():void` der Klasse `Stone`**

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 „`toMove`“
  3. Setze den Spielstein ein Feld nach vorn.
  4. Dekrementiere „`toMove`“
  5. Wenn „`toMove`“ 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/2011/05/PMSS11LudoEclipseProject.zip
- **In Eclipse importieren**
- **JUnit Test schreiben, der:**
  - 4 Felder erzeugt und diese miteinander verbindet
  - 1 Spieler erzeugt
  - 1 Spielfigur die dem Spieler zugeordnet ist und auf dem ersten Feld steht
  - 1 Würfel mit Augenzahl 2 erzeugt und ihm dem Spieler zuordnet
  - Methode `move()` auf dem Spielstein aufruft
  - Testet, ob der Stein nun auf dem dritten Feld steht
- **Methode `move():void` der Klasse `Stone` nach textuellem Entwurf implementieren**

Methodenentwurf V

- **Mögliche Implementierung**

```
public void move()
    {
        Player player = getPlayer();
        Dice dice = player.getDice();
        if (dice != null)
        {
            int toMove = dice.getPips();

            while (toMove > 0)
            {
                Field curF = getIsOn();
                Field toField = curF.getTo();
                setIsOn(toField);
                toMove--;
            }
        }
    }
```

- **Verifikation durch Zetteltest**

Methodenentwurf VI - Zetteltest

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

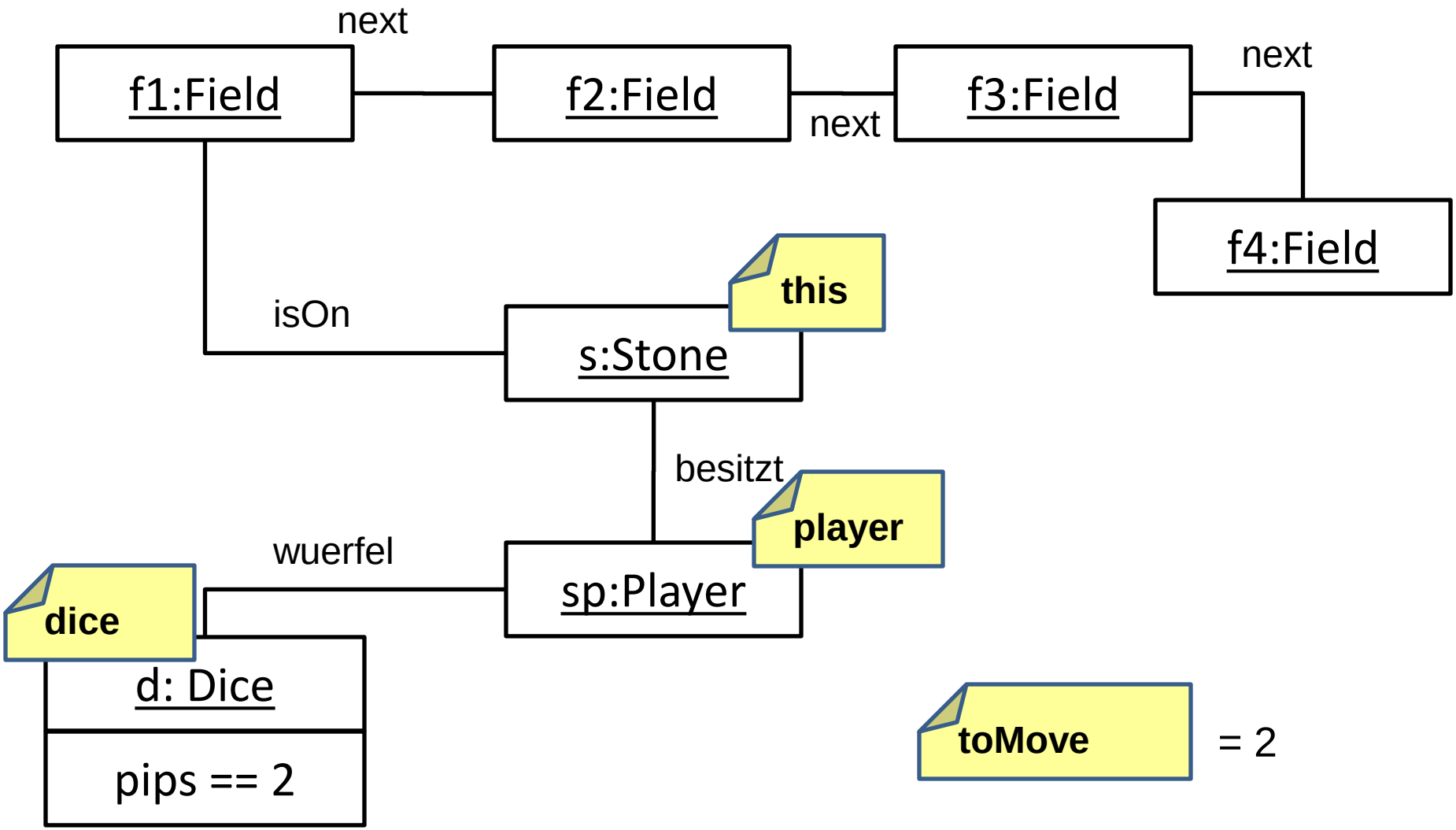
Methodenentwurf VIII

- **Mögliche Implementierung**

```
public void move()
    {
        Player player = getPlayer();
        Dice dice = player.getDice();
        if (dice != null)
        {
            int toMove = dice.getPips();

            while (toMove > 0)
            {
                Field curF = getIsOn();
                Field toField = curF.getTo();
                setIsOn(toField);
                toMove--;
            }
        }
    }
```

- **Verifikation durch Zetteltest**

Methodenentwurf IX - Zetteltest



Methodenentwurf X

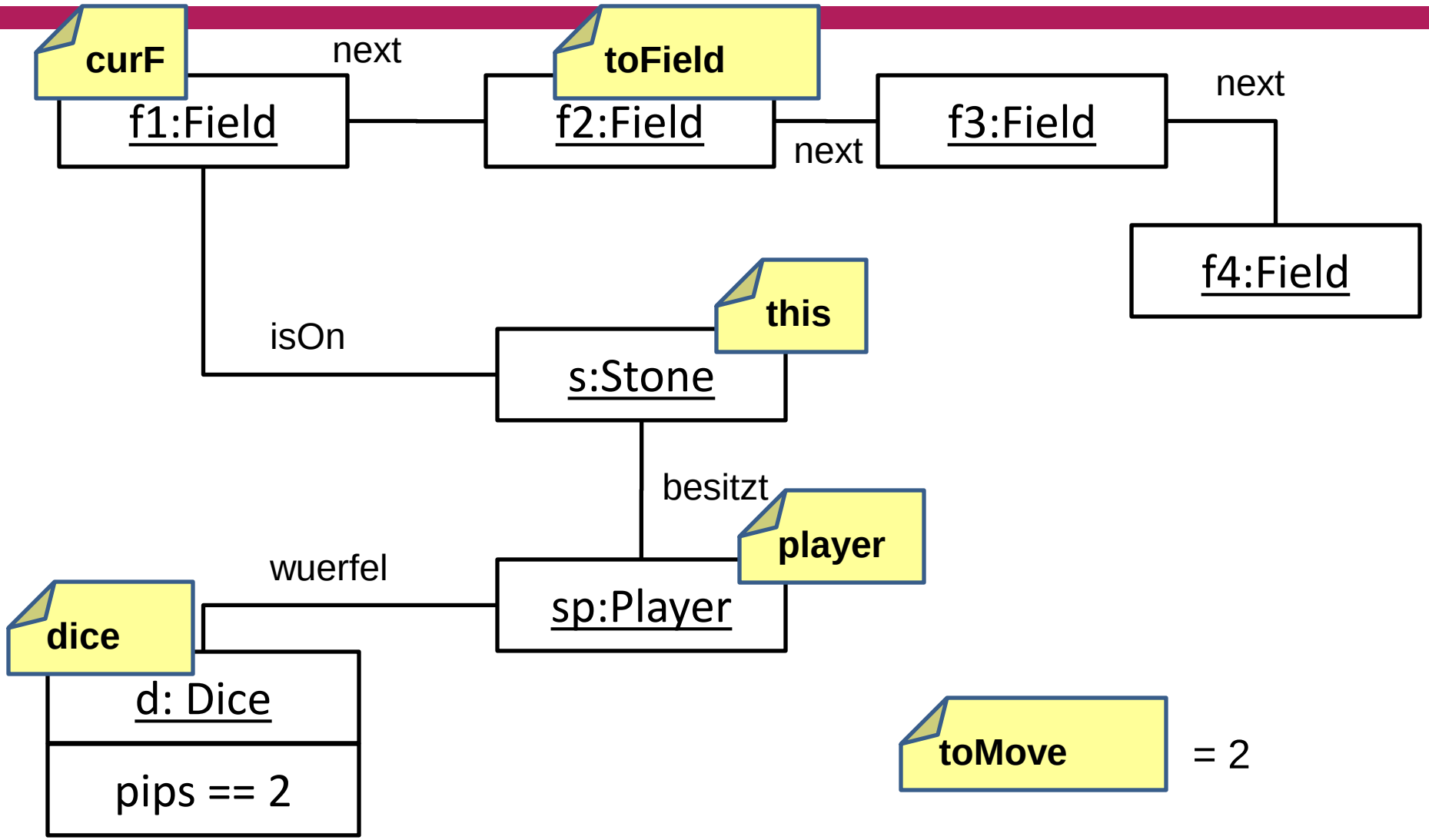
- **Mögliche Implementierung**

```
public void move()
    {
        Player player = getPlayer();
        Dice dice = player.getDice();
        if (dice != null)
        {
            int toMove = dice.getPips();

            while (toMove > 0)
            {
                Field curF = getIsOn();
                Field toField = curF.getTo();
                setIsOn(toField);
                toMove--;
            }
        }
    }
```

- **Verifikation durch Zetteltest**

= 2

Methodenentwurf XI - Zetteltest



Methodenentwurf XIII

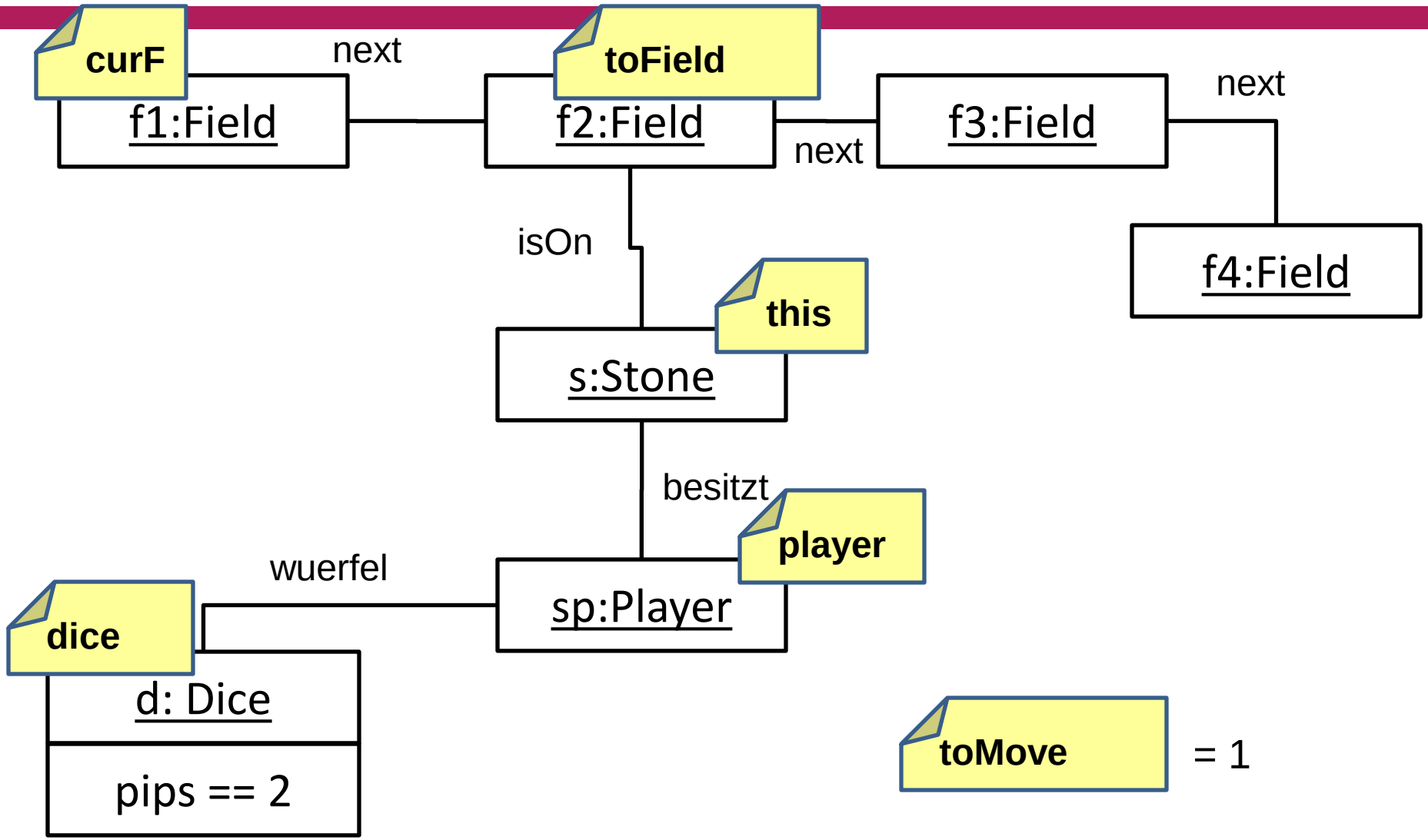
- **Mögliche Implementierung**

```
public void move()
    {
        Player player = getPlayer();
        Dice dice = player.getDice();
        if (dice != null)
        {
            int toMove = dice.getPips();

            while (toMove > 0)
            {
                Field curF = getIsOn();
                Field toField = curF.getTo();
                setIsOn(toField);
                toMove--;
            }
        }
    }
```

- **Verifikation durch Zetteltest**

= 1

Methodenentwurf XIV - Zetteltest



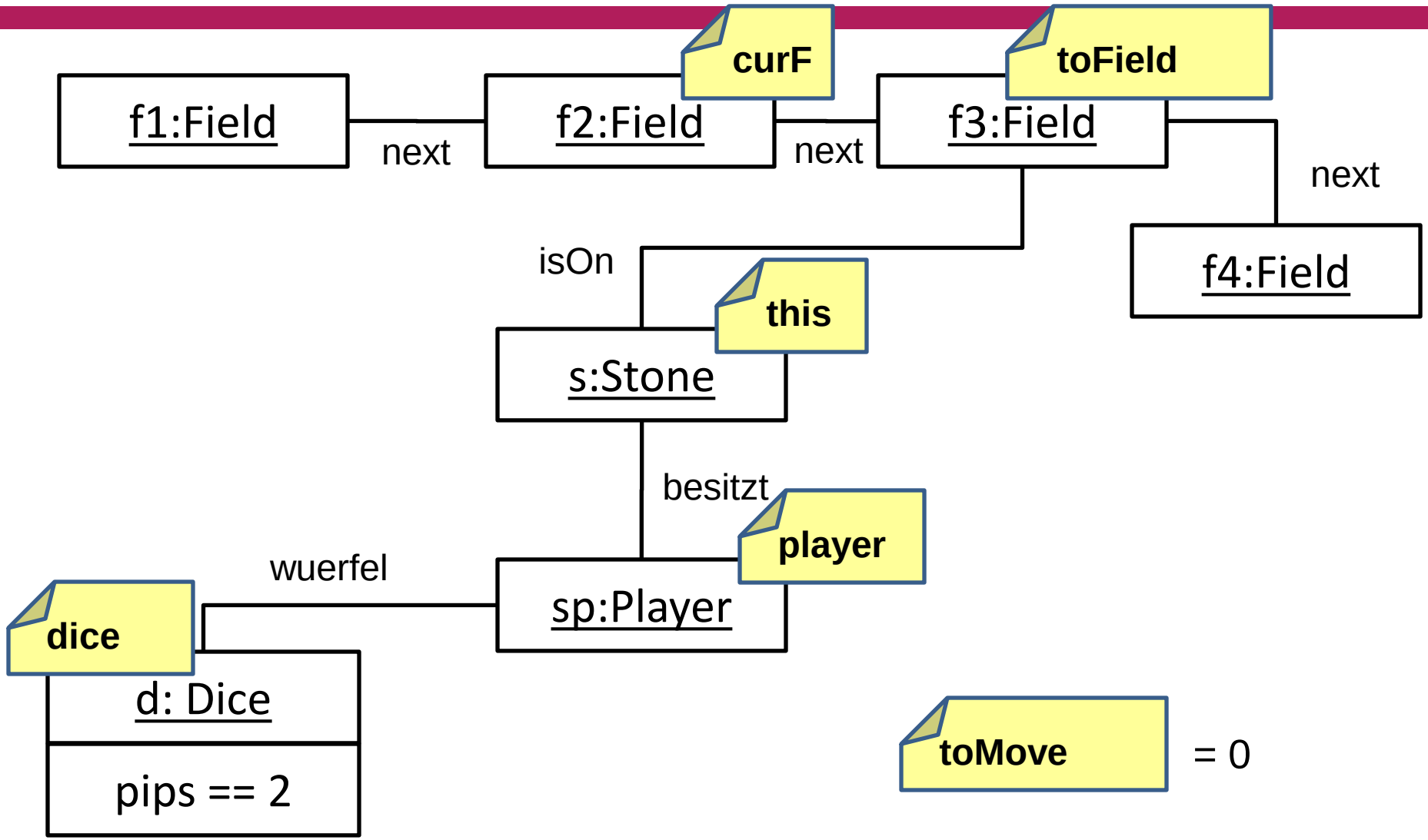
Methodenentwurf XV

- **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**

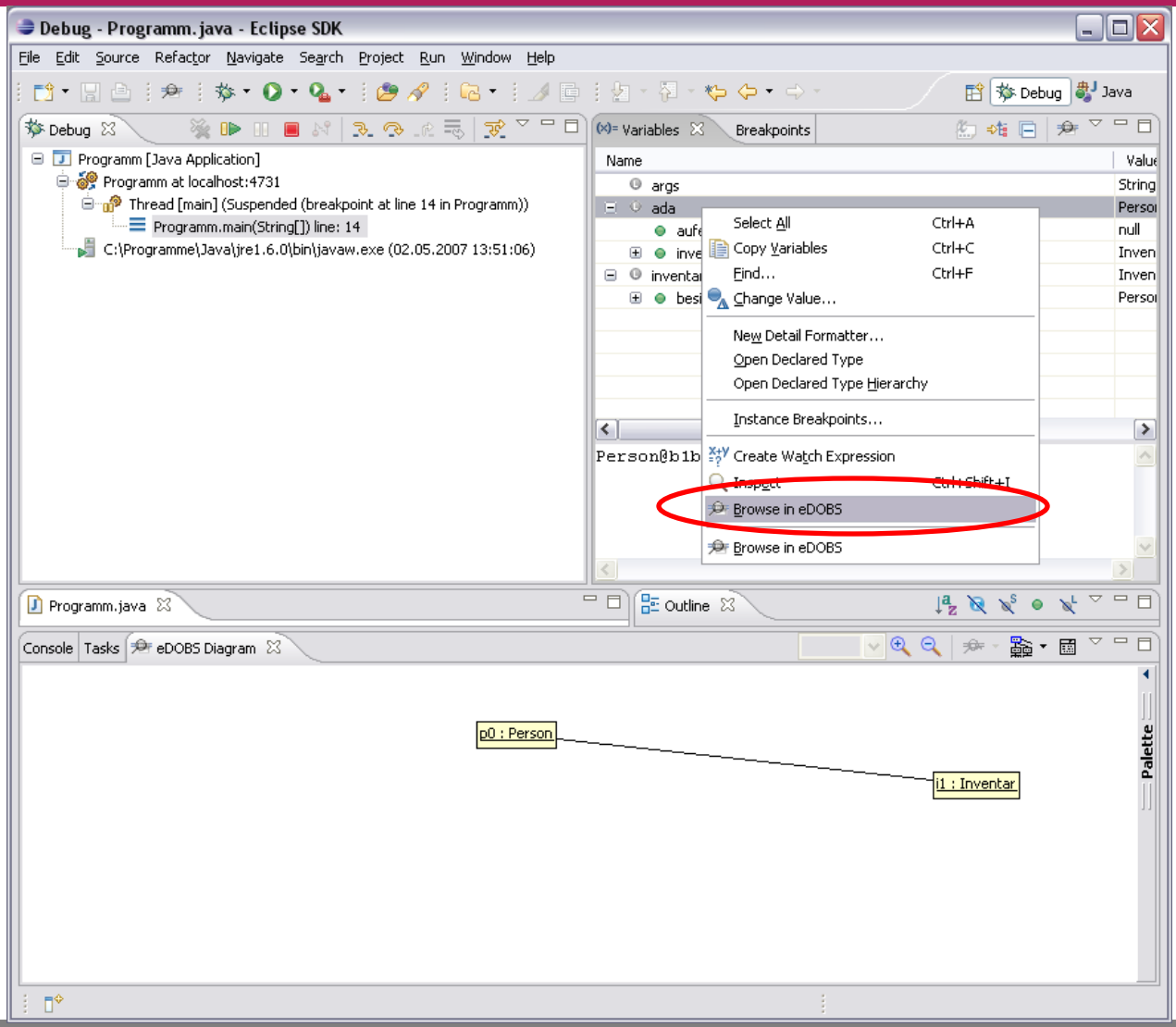
Methodenentwurf XVI - Zetteltest



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

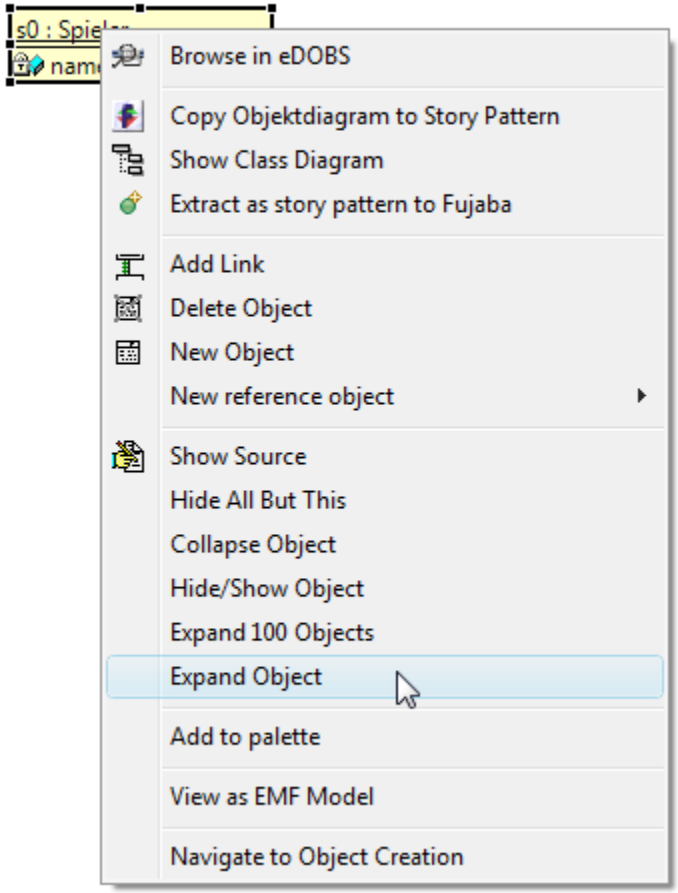

Debug - Programm.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Java
Debug
Programm [Java Application]
Programm at localhost:4731
Thread [main] (Suspended (breakpoint at line 14 in Programm))
Programm.main(String[]) line: 14
C:\Programme\Java\jre1.6.0\bin\javaw.exe (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

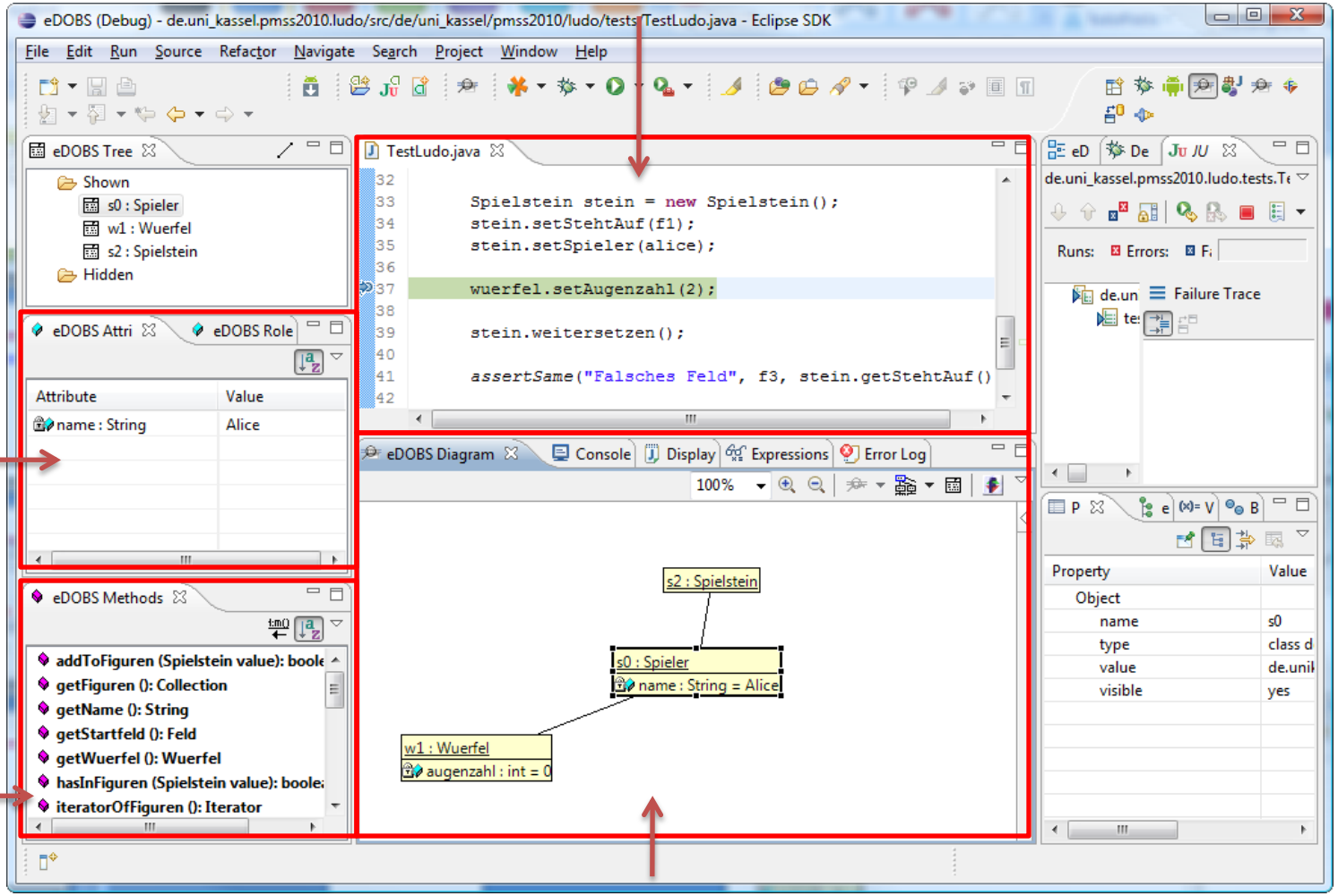
- **Breakpoint im Programm setzen**
- **Starte das Programm im Debug-Modus**
  - Debug as -> Java Application, NICHT Debug as eDOBS VM
- **Warte bis der Debugger auf dem Breakpoint hält**
- **Wähle eine Objekt-Variable im Variables-View (z.b. this), Rechtsklick, „Browse in eDOBS“**
- **Wechsle die Perspektive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

eDOBS IV

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



eDobs V

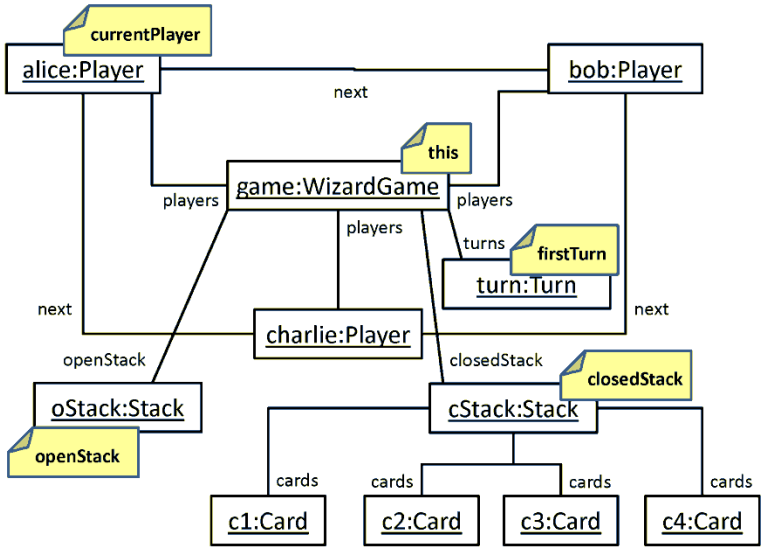

Quelltext
eDOBS (Debug) - de.uni_kassel.pmss2010.ludo/src/de/uni_kassel/pmss2010/ludo/tests/TestLudo.java - Eclipse SDK
File Edit Run Source Refactor Navigate Search Project Window Help
eDOBS Tree
Shown
s0 : Spieler
w1 : Wuerfel
s2 : Spielstein
Hidden
TestLudo.java
Spielstein stein = new Spielstein();
stein.setStehtAuf(f1);
stein.setSpieler(alice);
wuerfel.setAugenzahl(2);
stein.weitersetzen();
assertSame("Falsches Feld", f3, stein.getStehtAuf()
eDOBS Attri
eDOBS Role
Attribute
Value
name : String
Alice
eDOBS Methods
addToFiguren (Spielstein value): boole
getFiguren (): Collection
getName (): String
getStartfeld (): Feld
getWuerfel (): Wuerfel
hasInFiguren (Spielstein value): boole
iteratorOfFiguren (): Iterator
eDOBS Diagram
Console
Display
Expressions
Error Log
100%
s2 : Spielstein
s0 : Spieler
name : String = Alice
w1 : Wuerfel
augenzahl : int = 0
Runs:
Errors:
Failure Trace
Property
Value
Object
name
s0
type
class d
value
de.unil
visible
yes
Attribute anzeigen/ ändern
Methoden aufrufen
Objektdiagramm

Vorschau HA3 I

- **Abgabe in 2 Wochen (02.06.2010, 23.59 Uhr)**
- **Vorbereitung:**
  - eDOBS installieren
  - Wizard Projekt herunterladen (optional)
    - Enthält bereits Implementierung des Klassendiagramms und der Methode `WizardGame::startGame():void`
- **Aufgabe 1**
  - `WizardGame::init(…):void` implementieren
    - Für jeden per Parameter übergebenen Namen einen Spieler erzeugen
    - „Open-“ und „Closed-“ Stack erzeugen
    - 60 Karten erzeugen

Vorschau HA3 II

- **Aufgabe 2**
  - eDOBS Screenshot von Objektsituation am ende von `WizardGame::init (…):void`
- **Aufgabe 3**
  - Startsituation ist vorgegeben
  - Zetteltest zur Methode `WizardGame::startGame():void`
  - eDOBS Screenshot



Vorschau HA3 III

- **Zusatzaufgabe**
  - JUnit Test schreiben der die Korrektheit der ersten Vorhersagerunde testet
    - Jeder Spieler muss ein „Forecast“-Objekt besitzen, welches der aktuellen Runde zugeordnet ist
    - Das „trickcount“ Attribut der Klasse „Forecast“ darf nicht größer sein als das „number“ Attribut des zugeordneten „Turn“-Objekts
  - Methode `WizardGame::startGame():void` erweitern, `Player::doForecast():void` implementieren
  - Zwei eDOBS Screenshots:
    - Startsituation
    - Endsituation

Ende

Schönes WE!