

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**

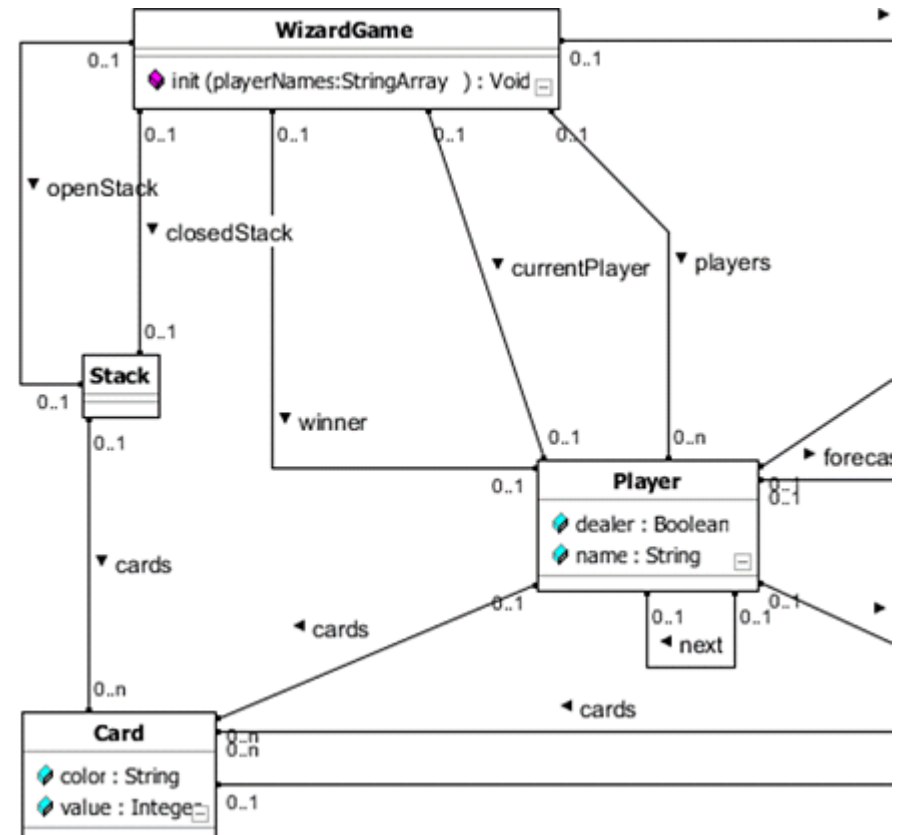
- Implementierung erfolgt strukturiert durch vorgegebenes Schema

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();

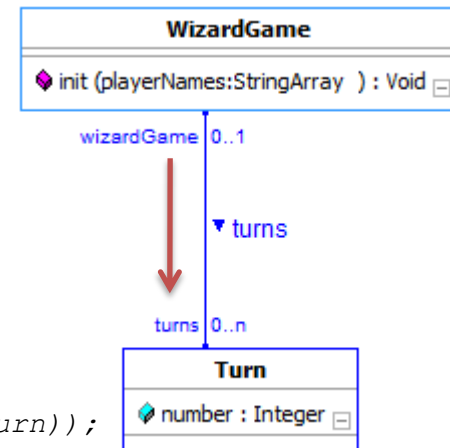
    // Add the turn to the game
    wizardGame.addToTurns(turn);

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

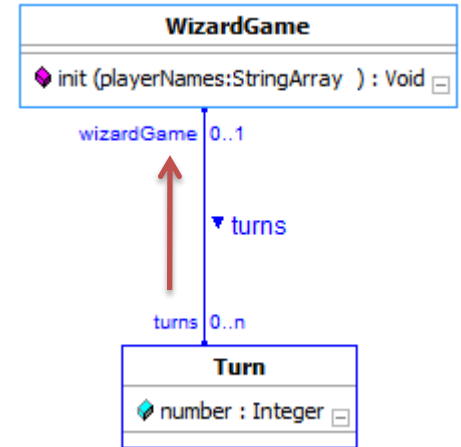
    // Set wizard game
    turn.setWizardGame (wizardGame);

    // Check referential integrity
    assertTrue ("Turn was not added to wizard game", wizardGame.hasInTurns (turn));
    assertEquals ("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 ());
}

```

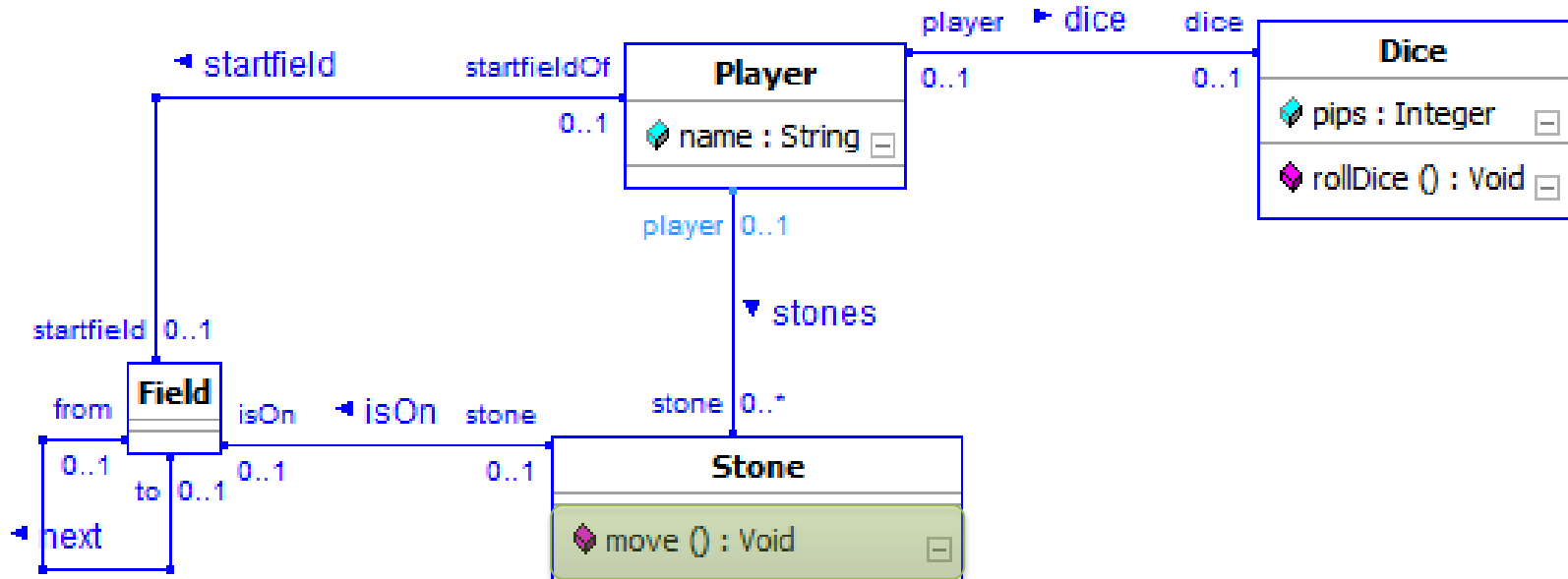


Methodenentwurf I

- 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 weitersetzt werden, wie der Würfel anzeigt

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 letzten Feld steht
- Methode `move():void` der Klasse `Stone` nach textuellem Entwurf implementieren

Methodenentwurf V

- **Mögliche Implementierung**

```
public void move ()  
{
```

```
}
```

- **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

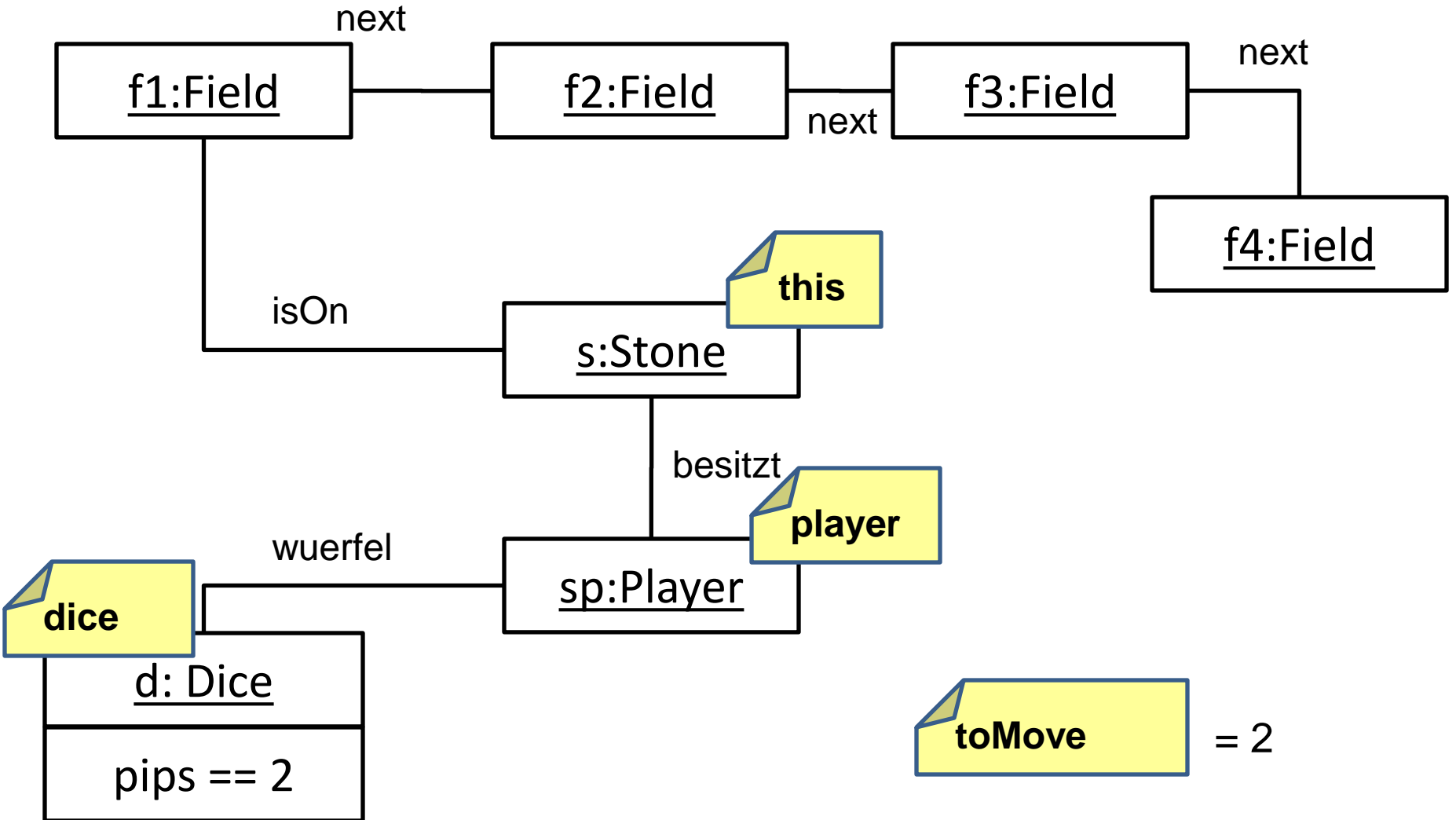
- **Mögliche Implementierung**

```
public void move()  
{
```

```
}
```

- **Verifikation durch Zetteltest**

Methodenentwurf IX - Zetteltest



Methodenentwurf X

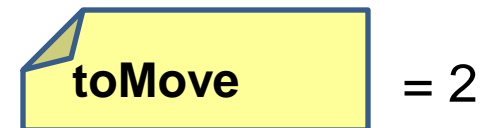
- Mögliche Implementierung

```
public void move ()
{
```

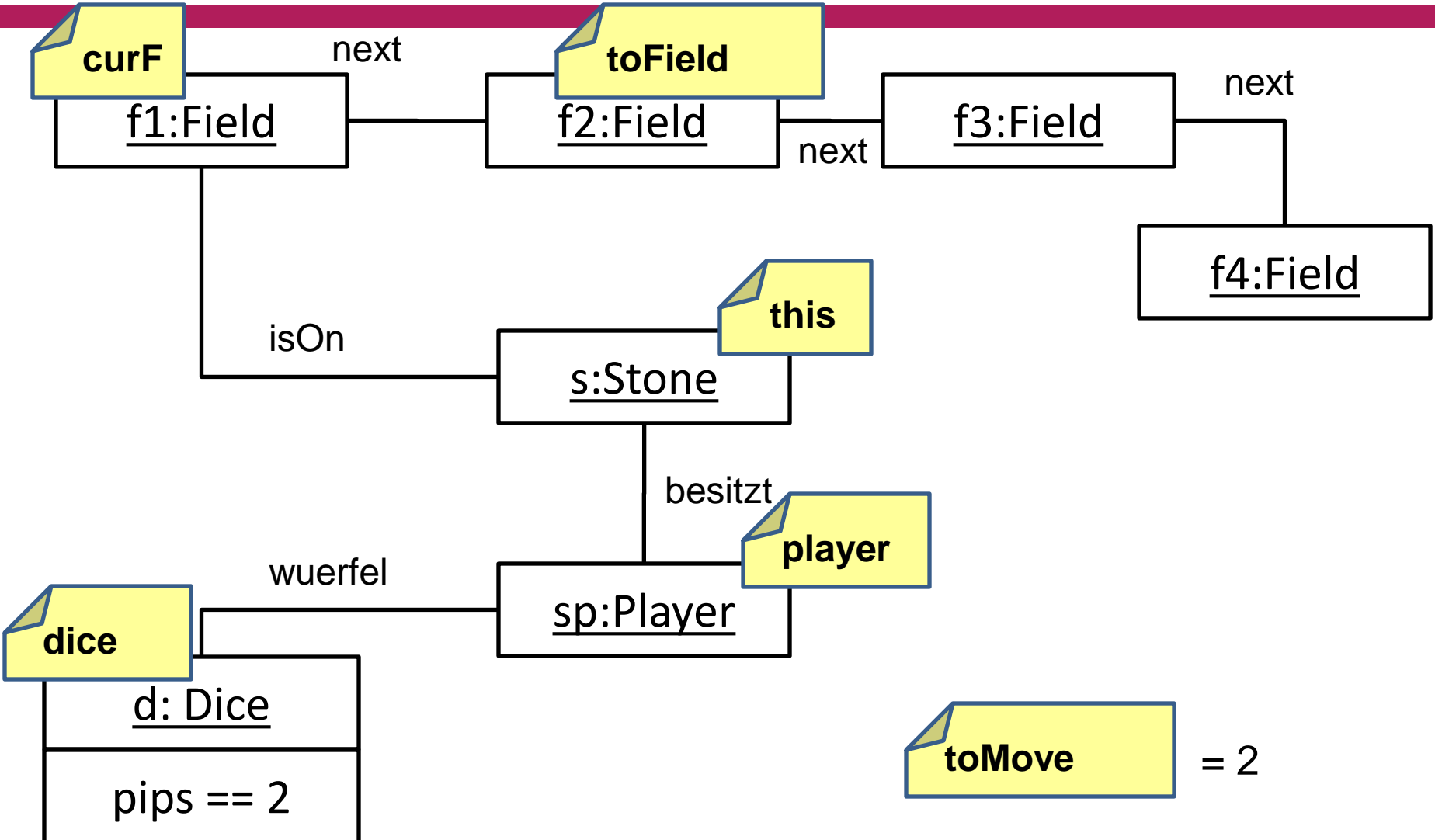


```
}
```

- Verifikation durch Zetteltest



Methodenentwurf XI - Zetteltest



Methodenentwurf XIII

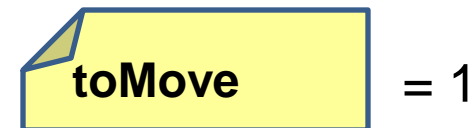
- Mögliche Implementierung

```
public void move ()
{
```

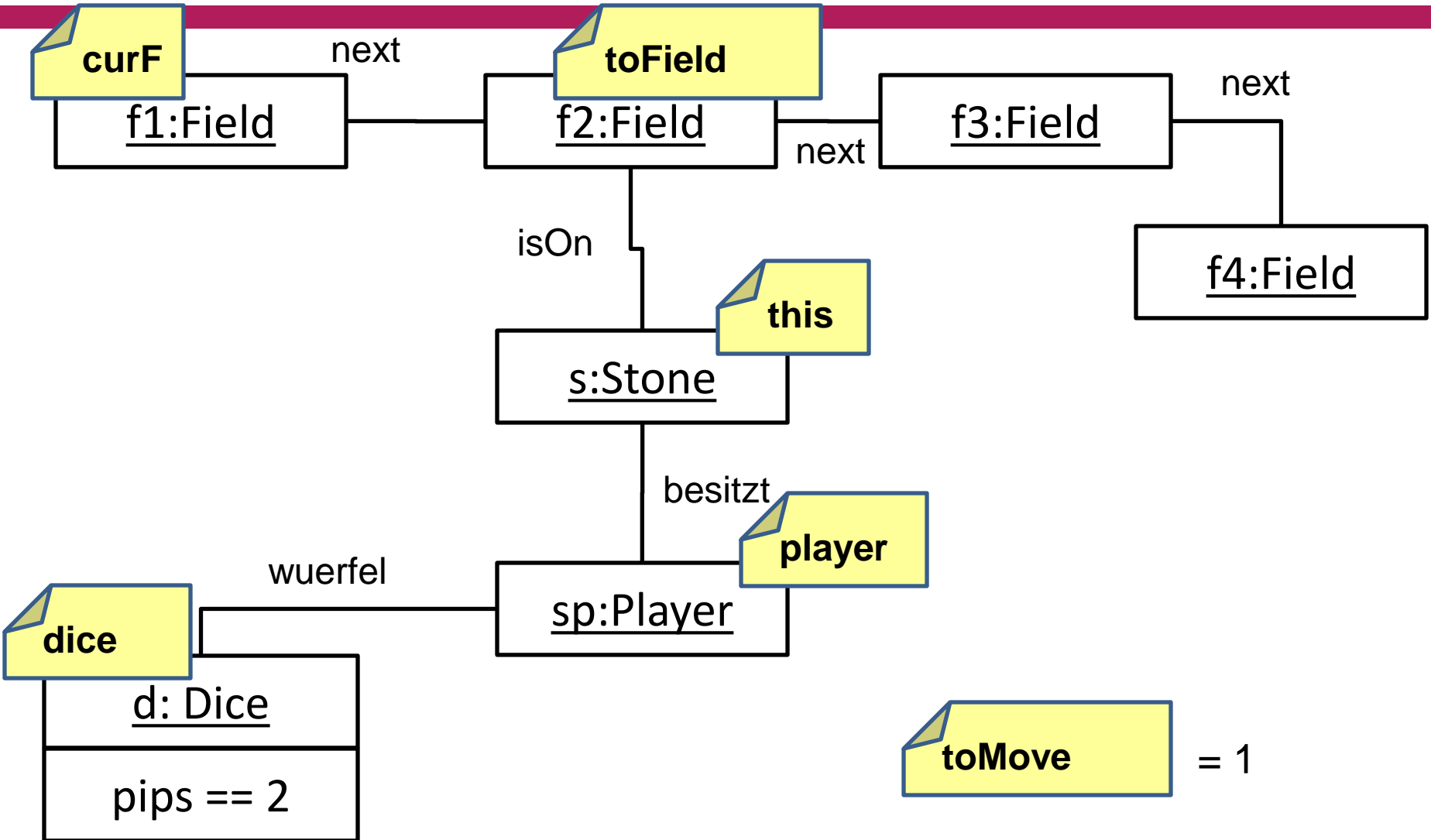


```
}
```

- Verifikation durch Zetteltest



Methodenentwurf XIV - Zetteltest



Methodenentwurf XV

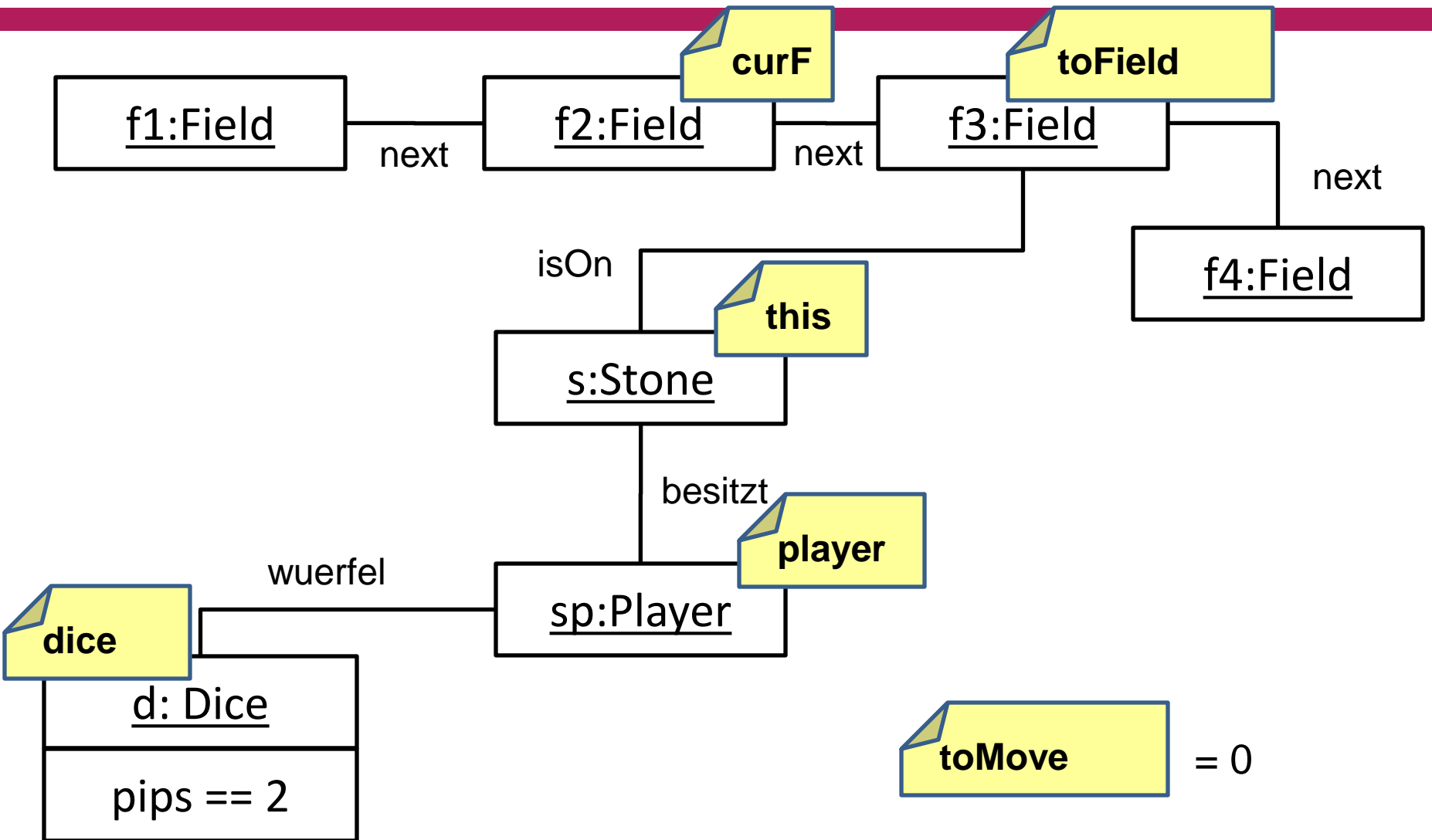
- Mögliche Implementierung



- Verifikation durch Zetteltest

nochGehen = 0

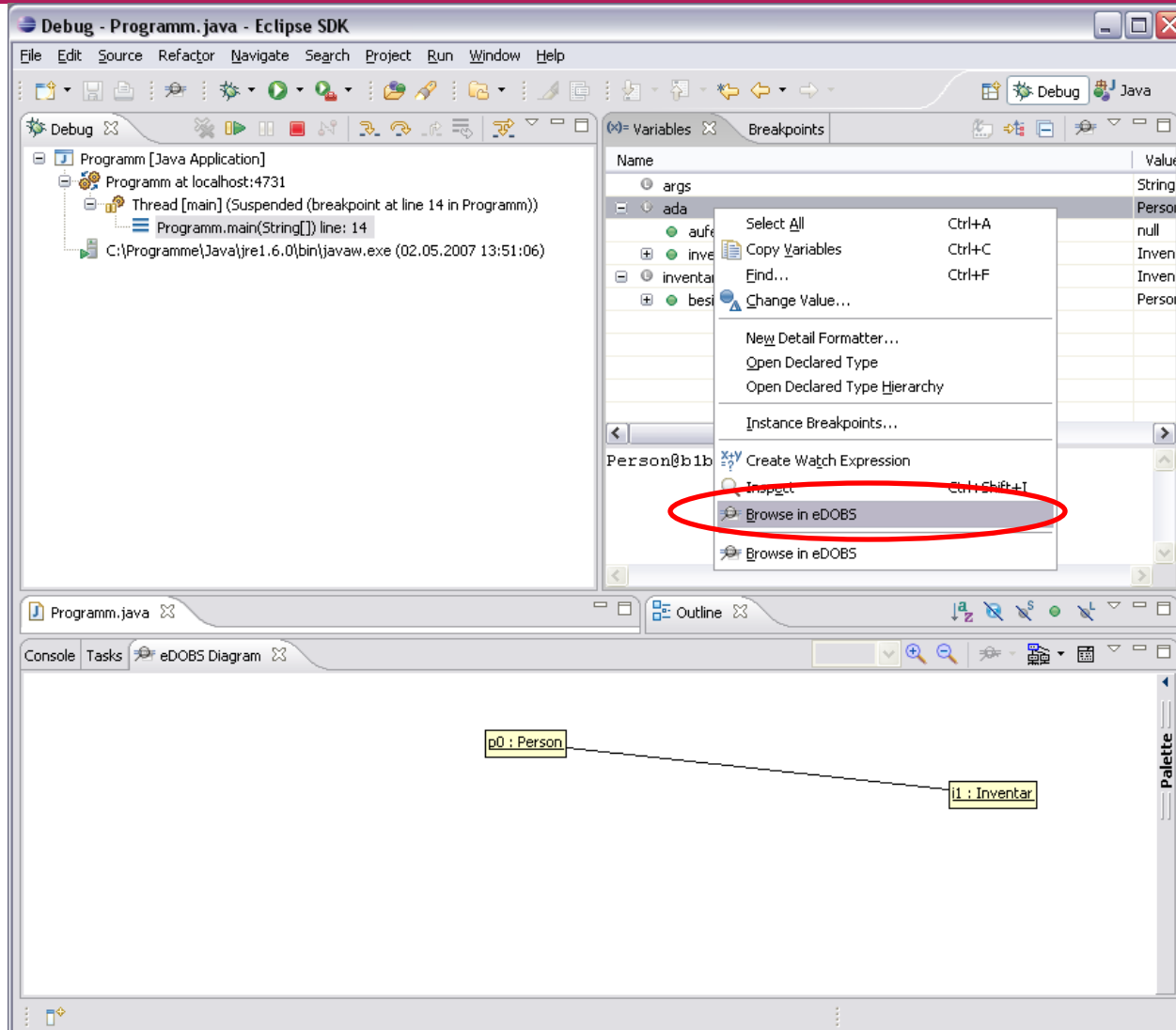
Methodenentwurf XVI - Zetteltest



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

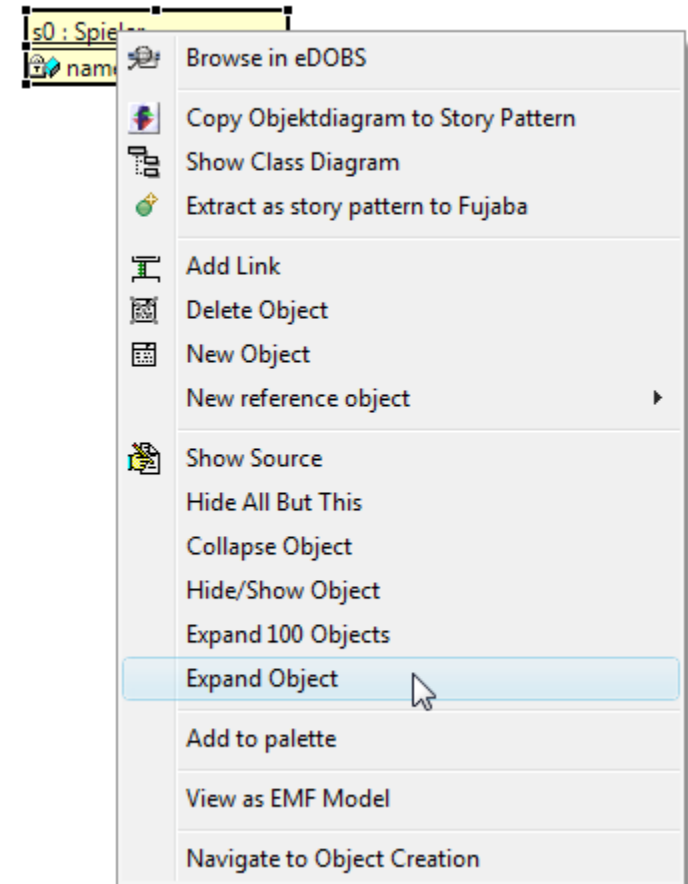


eDOBS III

- **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

The screenshot shows the Eclipse IDE with the following components:

- Source Editor:** Displays Java code for `TestLudo.java`. Line 37 is highlighted: `wuerfel.setAugenzahl(2);`. Other lines include `Spielstein stein = new Spielstein();`, `stein.setStehtAuf(f1);`, `stein.setSpieler(alice);`, `stein.weitersetzen();`, and `assertSame("Falsches Feld", f3, stein.getStehtAuf());`.
- eDOBS Tree:** Shows a hierarchy with 'Shown' objects: `s0: Spieler`, `w1: Wuerfel`, and `s2: Spielstein`, and a 'Hidden' folder.
- eDOBS Attri:** A table with columns 'Attribute' and 'Value'. It contains one entry: `name: String` with value `Alice`.
- eDOBS Methods:** A list of methods for the selected object, including `addToFiguren (Spielstein value): boole`, `getFiguren (): Collection`, `getName (): String`, `getStartfeld (): Feld`, `getWuerfel (): Wuerfel`, `hasInFiguren (Spielstein value): boole`, and `iteratorOfFiguren (): Iterator`.
- eDOBS Diagram:** A visual representation of objects and their relationships. It shows nodes for `w1: Wuerfel` (with `augenzahl: int = 0`), `s0: Spieler` (with `name: String = Alice`), and `s2: Spielstein`.
- Property Table:** Located on the right, it shows properties for the selected object: `name` (value `s0`), `type` (value `class d`), `value` (value `de.unil`), and `visible` (value `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

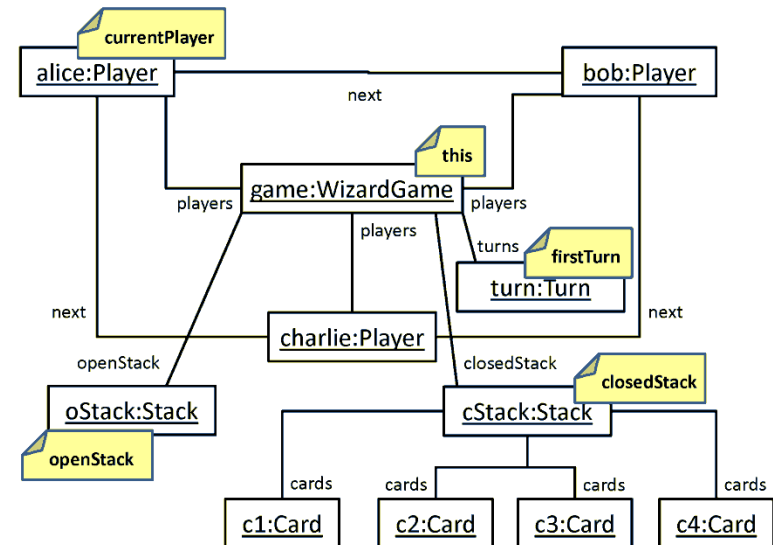
Vorschau HA3 II

- **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!