UNIKASSEL
VERSITÄT

Research Group
Software Engineering

# Programmiermethodik

# Übung 6

Wintersemester 2011 / 12

Fachgebiet Software Engineering

Tobias George

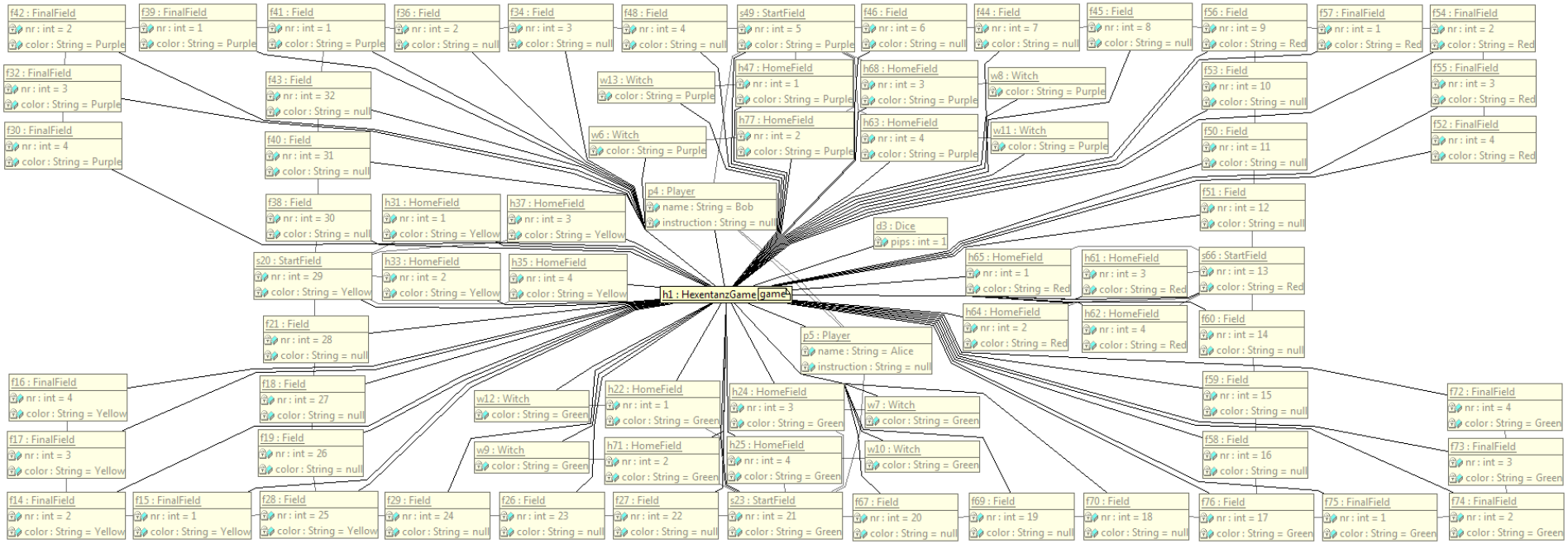
george@uni-kassel.de

# Agenda

- **Besprechung HA4**
- **Zetteltest**
- **Fujaba4Eclipse**
  - Installieren,
  - (Projekt anlegen, Klassendiagramm anlegen, Alternate Editing Mode, Code generieren)
- **Fujaba 4 Eclipse: Storyboards**
- **Praktische Übung: Storyboards**
- **Vorschau HA5**

# Besprechung HA4

- **Aufgabe 1: HexentanzGame::init(playerNames:StringArray) implementieren**
- **Aufgabe 2: eDobs Screenshot am Ende von init(…)**



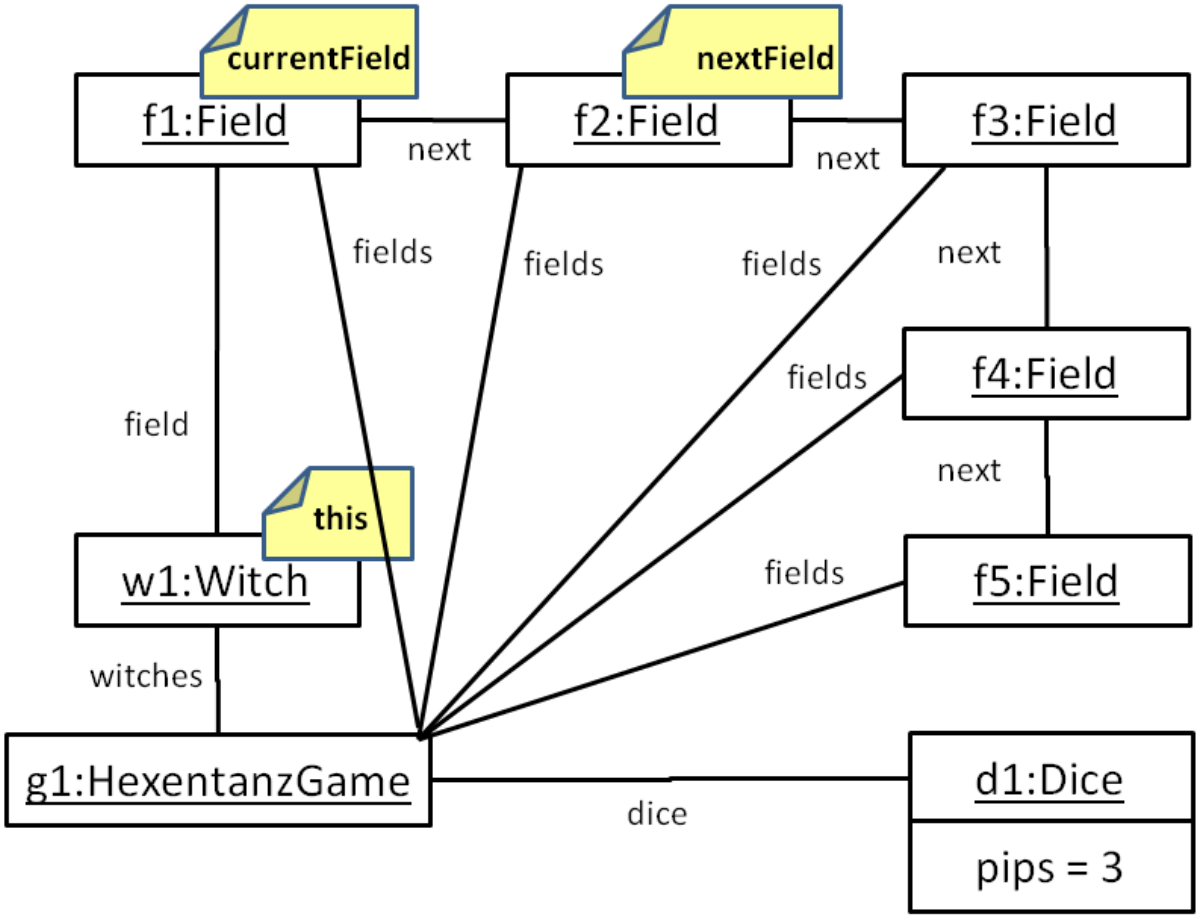
# Zetteltest I

- **Methode : Witch.move()**

```
1  public void move()
2  {
3     Field currentField = this.getField();
4     Game game = this.getGame();
5     Dice dice = game.getDice();
6     Field nextField = currentField.getNext();
7
8     for(int pips = dice.getPips(); pips > 0; pips--)
9     {
10       currentField.removeFromWitches(this);
11       nextField.addToWitches(this);
12
13       currentField = nextField;
14       nextField = nextField.getNext();
15    }
16 }
```

# Zetteltest II

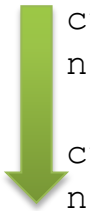
- **Zetteltest Schritt 1**



# Zetteltest I

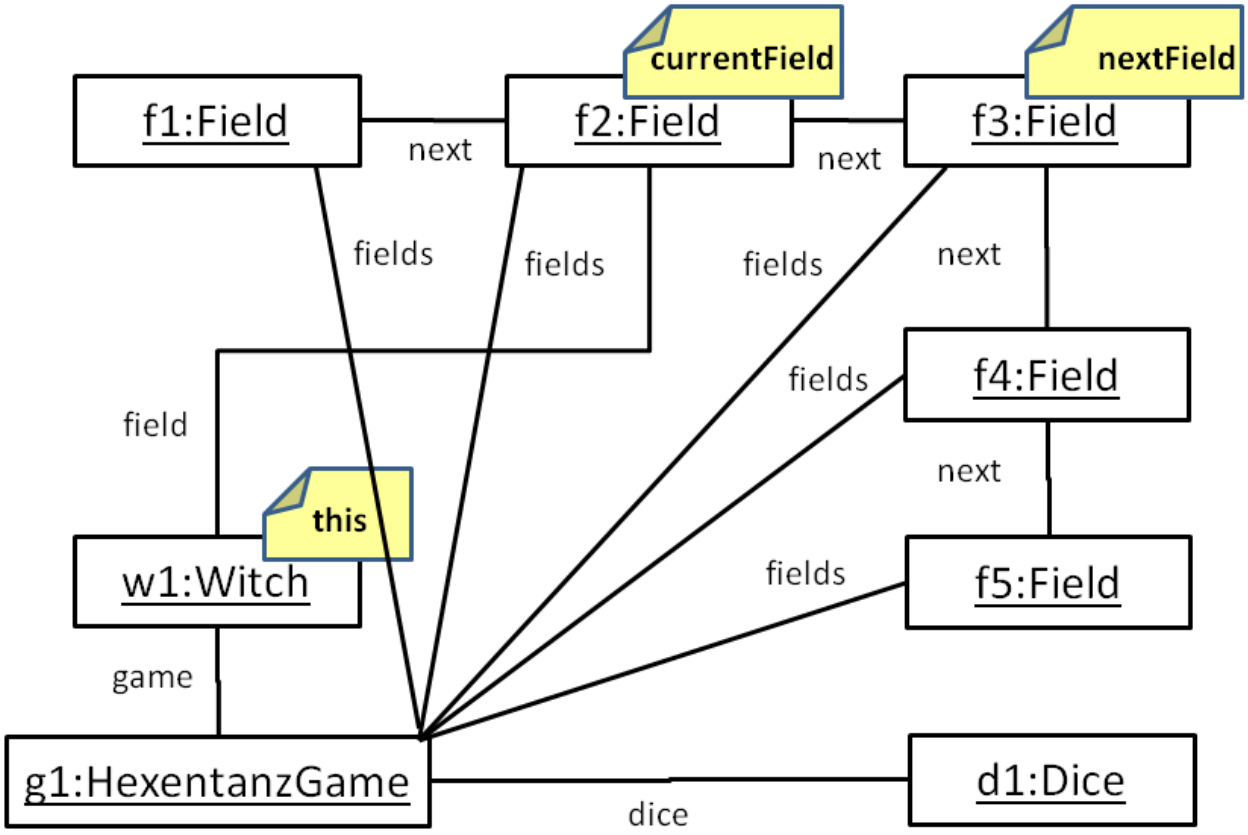
- **Methode : Witch.move()**

```
1  public void move()
2  {
3      Field currentField = this.getField();
4      Game game = this.getGame();
5      Dice dice = game.getDice();
6      Field nextField = currentField.getNext();
7
8      for(int pips = dice.getPips(); pips > 0; pips--)
9      {
10         currentField.removeFromWitches(this);
11         nextField.addToWitches(this);
12
13         currentField = nextField;
14         nextField = nextField.getNext();
15     }
16 }
```



# Zetteltest III

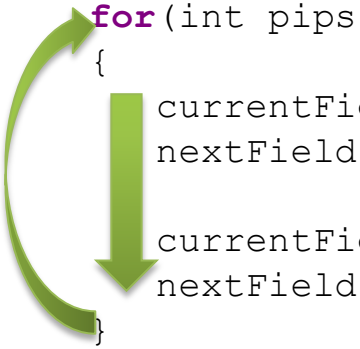
- **Zetteltest Schritt 2**



# Zetteltest I

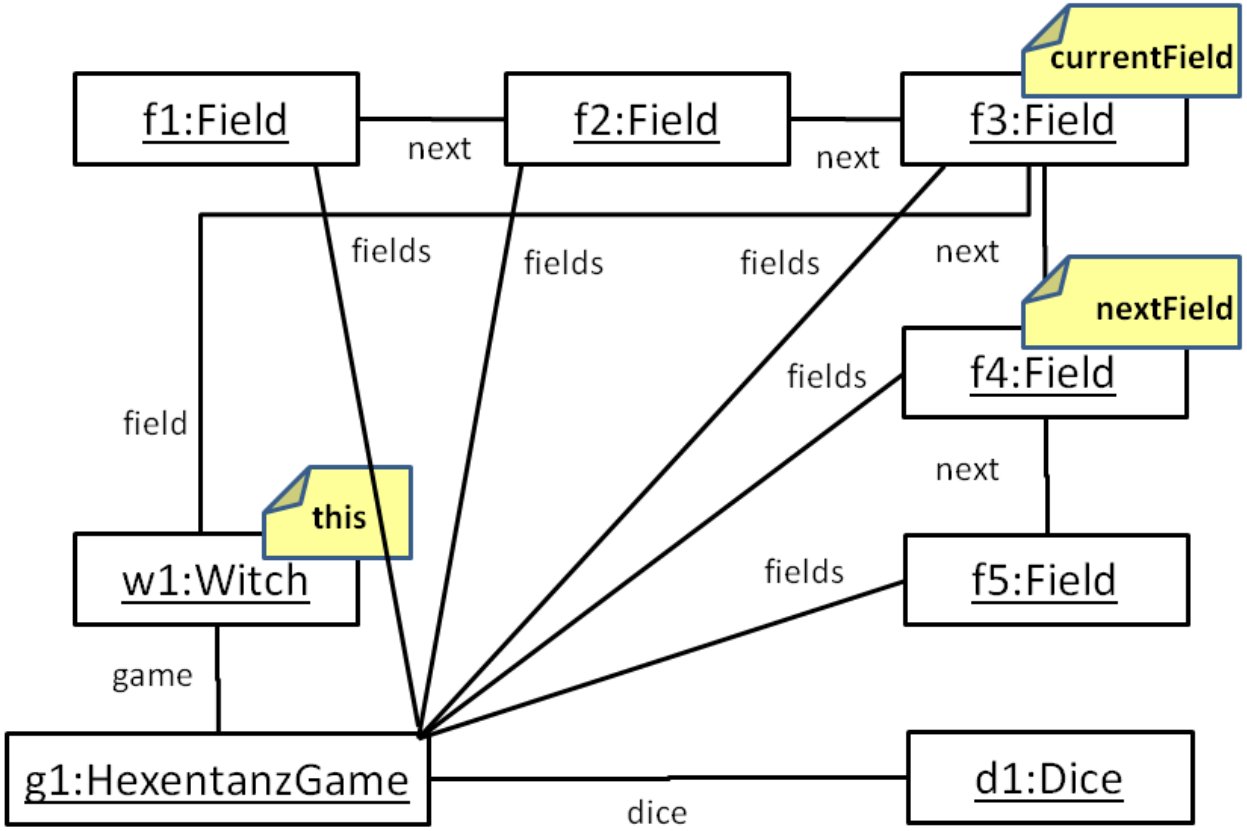
- **Methode : Witch.move()**

```
1  public void move()
2  {
3      Field currentField = this.getField();
4      Game game = this.getGame();
5      Dice dice = game.getDice();
6      Field nextField = currentField.getNext();
7
8      for(int pips = dice.getPips(); pips > 0; pips--)
9      {
10         currentField.removeFromWitches(this);
11         nextField.addToWitches(this);
12
13         currentField = nextField;
14         nextField = nextField.getNext();
15     }
16 }
```



# Zetteltest IV

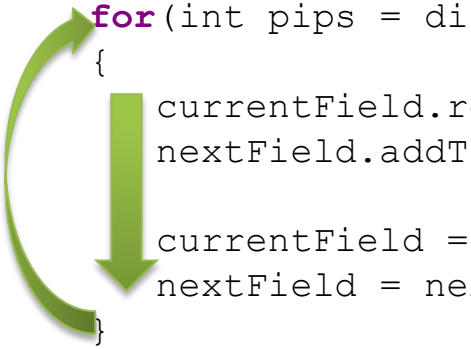
- **Zetteltest Schritt 3**



# Zetteltest I

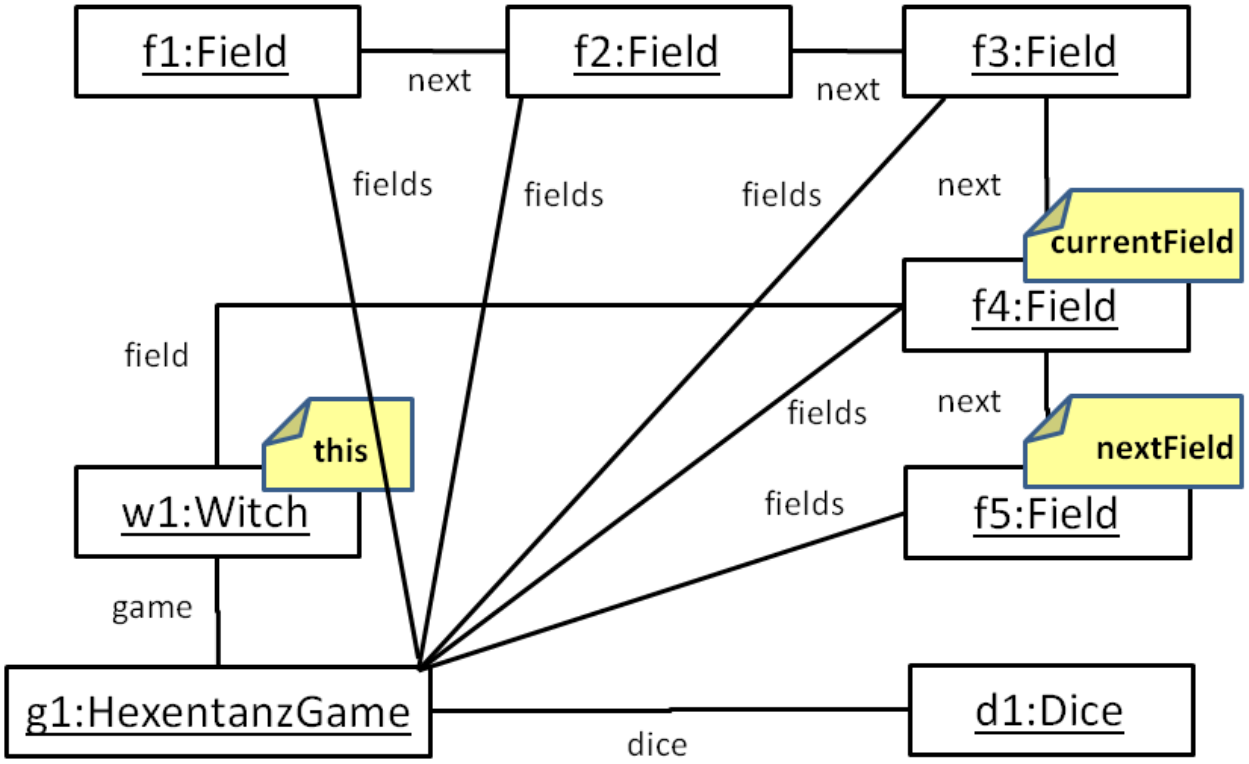
- **Methode : Witch.move()**

```
1  public void move()
2  {
3      Field currentField = this.getField();
4      Game game = this.getGame();
5      Dice dice = game.getDice();
6      Field nextField = currentField.getNext();
7
8      for(int pips = dice.getPips(); pips > 0; pips--)
9      {
10         currentField.removeFromWitches(this);
11         nextField.addToWitches(this);
12
13         currentField = nextField;
14         nextField = nextField.getNext();
15     }
16 }
```



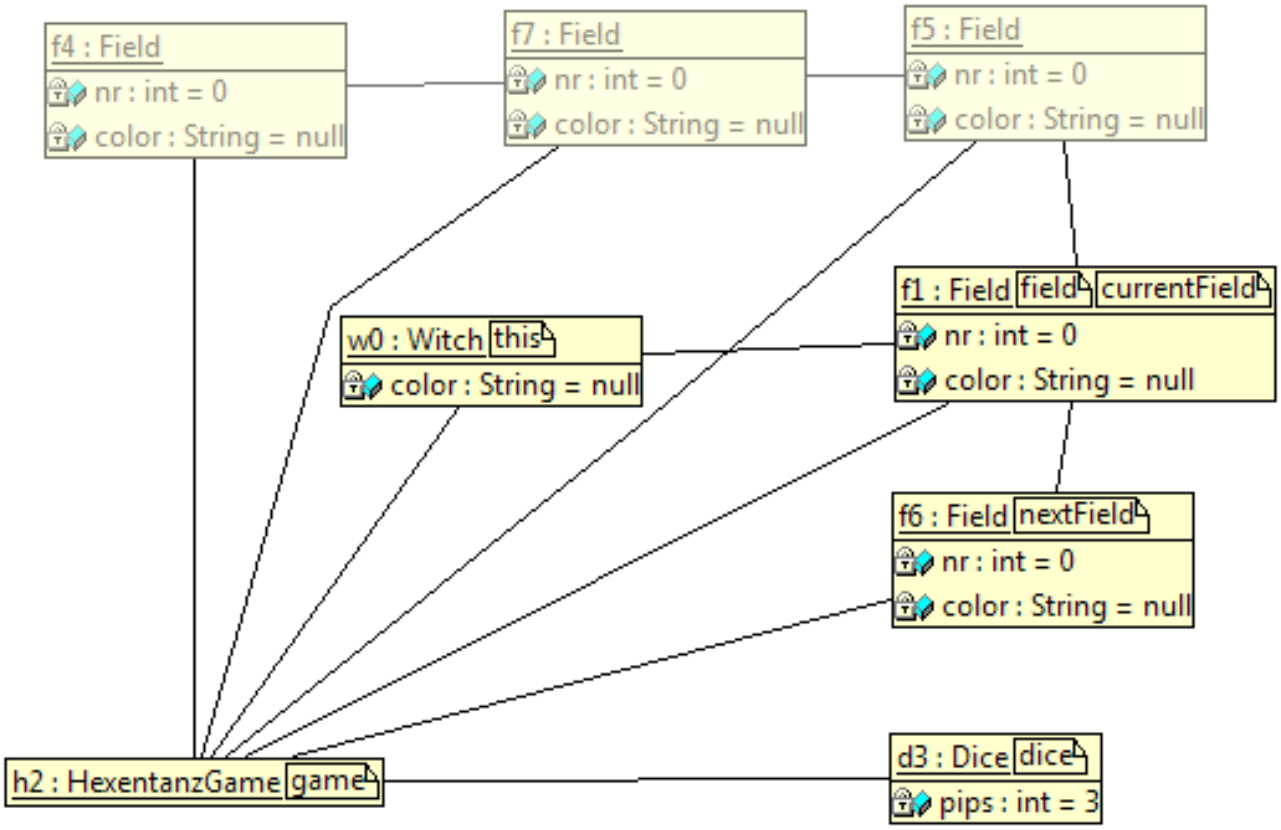
# Zetteltest V

- **Zetteltest Schritt 4**



# Zetteltest VI

- **eDOBS Screenshot nach `Witch:move()`**

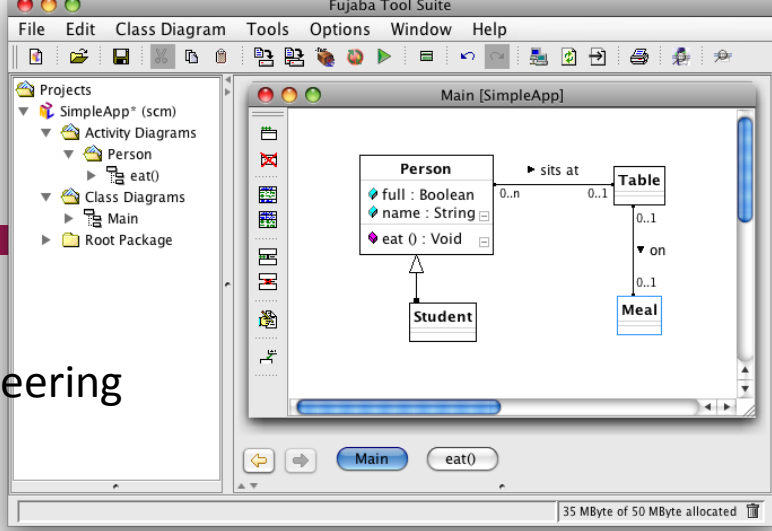


# Rückblick: Implementierung eines KD von Hand

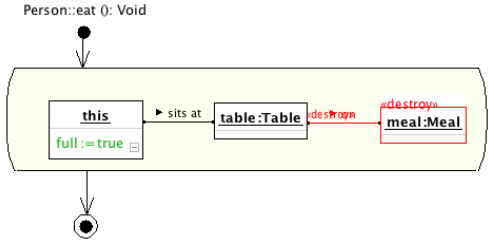
- Dauert lang
- Fehleranfällig (z.B. bezüglich der Referenziellen Integrität)
- „Man schreibt ständig das gleiche!“

**Das lässt sich automatisieren: Codegenerierung!**

# Fujaba




- **From UML to Java And Back Again**
  - Intention: Code generation and Reverse Engineering
- **Erlaubt u.a.:**
  - Das Erstellen von UML Klassendiagrammen (statische Struktur von Programmen)
  - Das Erstellen von Story- bzw. Aktivitätsdiagrammen (zur Modellierung von Verhalten)
  - Aus Diagrammen (Java, EMF, C++...) Code zu generieren
- **Vorerst: Klassendiagram-Anteil!**



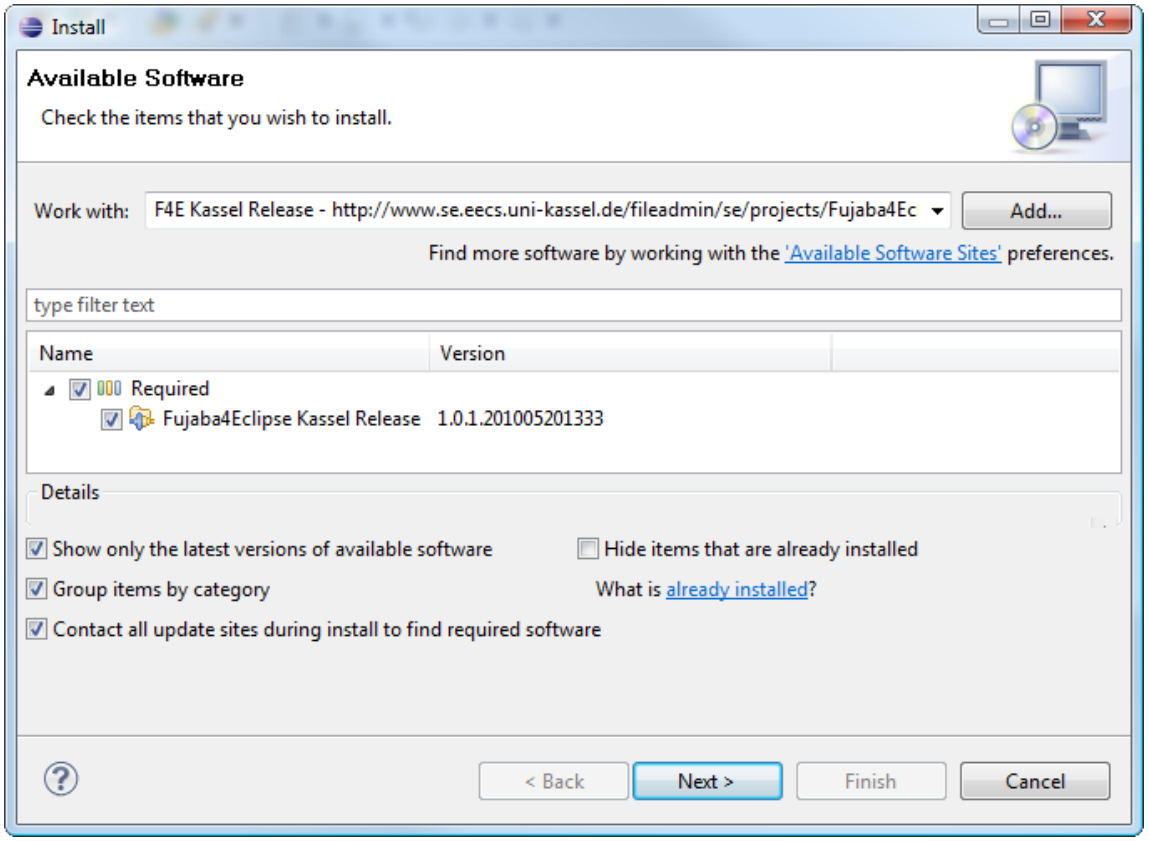
# Fujaba4Eclipse

- **Ist ein Eclipse Plugin → Fujaba mit Eclipse Integration**
  - Ein Fujaba Projekt/Modell ist Teil eines Eclipse Projekts. Ein Eclipse-Projekt hat (u.U.) viele Fujaba-Modelle
- **Durch Integration in Eclipse gibt quasi umsonst:**
  - JDT, Debugger, Compiler, Run environment
- **Vorteile:**
  - Zeitersparnis
  - Keine (Flüchtigkeits-)Fehler mehr in der Implementierung
  - Man kann sich auf das Wesentliche (das Modell) konzentrieren
  - Besserer Überblick durch Diagramme statt Code
  - …

# F4E: Installieren

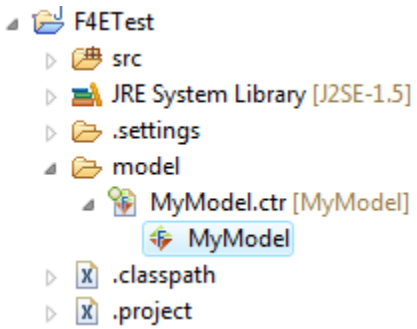
- **Über Eclipse Update Mechanismus. Update-Site URL:**

  **http://www.se.eecs.uni-kassel.de/fileadmin/se/update/**



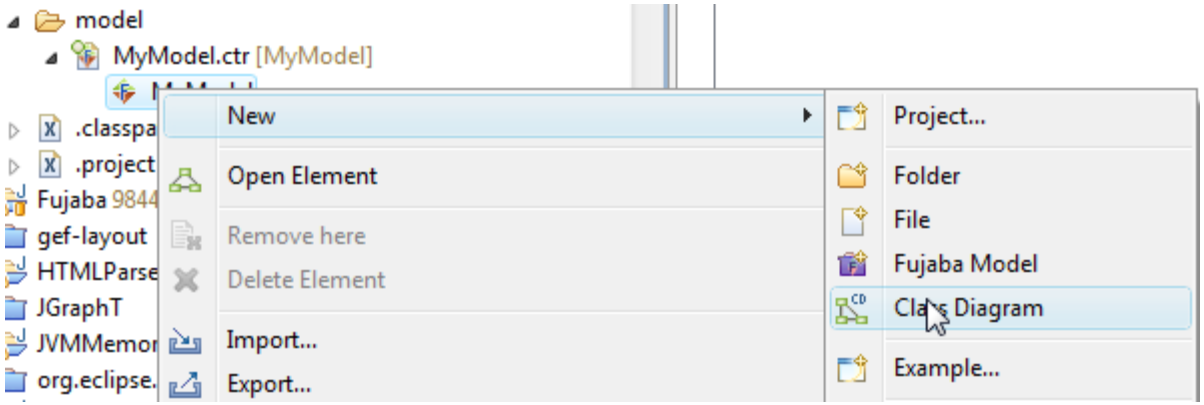
# F4E: Projekt anlegen

1. **Normales Java Projekt anlegen**
2. **(Neuer Unterordner „model“)**
3. **Fujaba Model erstellen**
   - Rechtsklick auf Java Projekt -> New -> Other -> Fujaba Model
   - .ctr-Datei im Projekt- oder „model“ Ordner speichern
   - (Erstellt auch gleich ein Klassendiagramm, man kann mehrere anlegen...)
4. **Die Fujaba4Eclipse Perspektive wählen**
   - Window -> Open Perspective -> Other… -> Fujaba4Eclipse
5. **Nun kann die .ctr Datei im *Project Explorer* (!) „aufgeklappt“ werden**



# F4E: Klassendiagramm erstellen

- **Klassendiagramm erstellen**
  - Rechtsklick auf Fujaba Projekt -> New -> Class Diagram




  - Name eingeben
- **„Alternate Editing Mode“ einschalten**
  - Vereinfacht das Anlegen von Klassen, Attributen, Methoden und Assoziationen

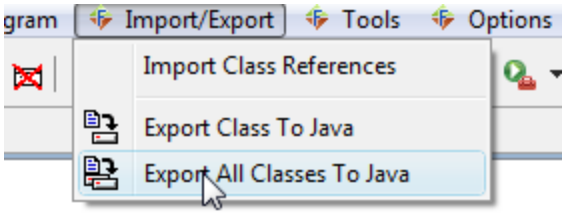
# F4E: Alternate Editing Mode

- **Maus:**
  - Klasse durch Aufziehen eines Kastens erstellen
  - Eine Assoziation erstellen, indem die eine Klasse auf die andere gezogen wird

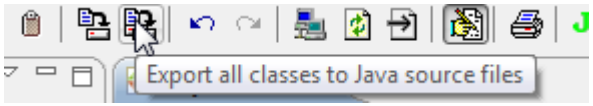

- **Tastatur**
  - Umbenennen von Elementen (Klassen, Attribute, Assoziationen) durch Anklicken und lostippen
  - Erstellen von Attributen und Methoden durch Selektieren einer Klasse und Eingabe der Signatur:
    - myAttribute : Integer
    - aMethod():Void

# F4E: Code generieren I

- **Zwei Möglichkeiten:**
  - Fujaba Menü -> Import/Export -> Export all Classes to Java

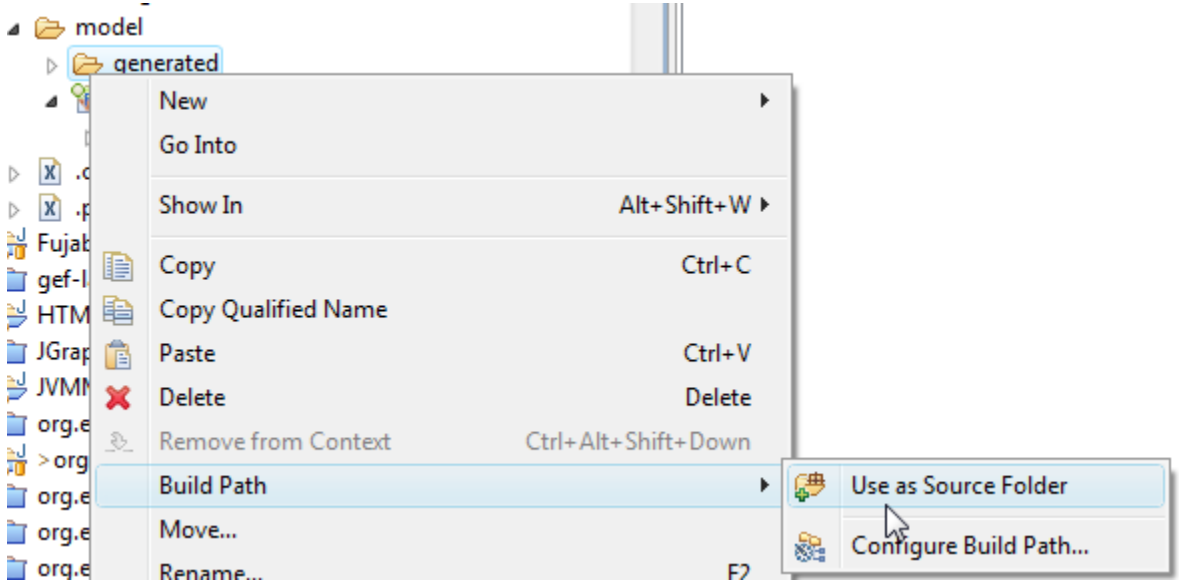



  - „Export all Classes to Java source files“ Button



# F4E: Code generieren II

- **Generierte Klassen zum Build Path hinzufügen**
  - Rechtsklick auf „generated“ Verzeichnis -> Build Path -> Use as Source Folder

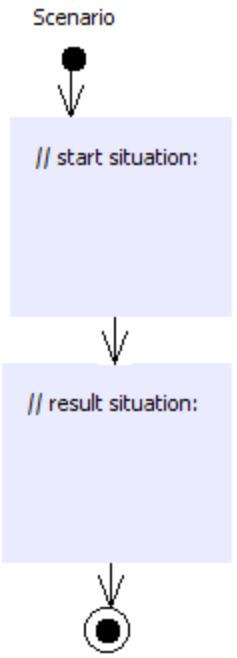


# Fujaba Storyboards I

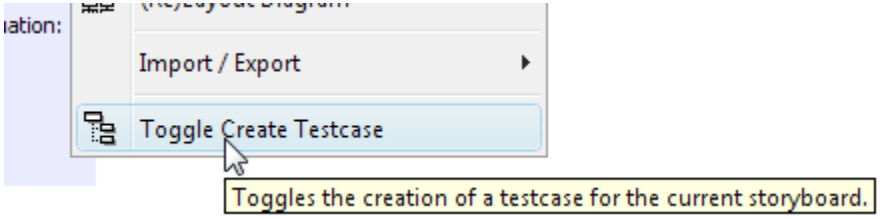
- **Tests schreiben ist aufwendig -> Storyboards**
- **„Closed World Assumption“: Objekte die nicht gezeichnet wurden gibt es nicht**
- **Ein Storyboard ist – wie ein „normaler“ JUnit Test auch – nicht allgemein gehalten sondern testet IMMER einen konkreten Fall.**

# Fujaba Storyboards II

- **Fujaba Model öffnen -> Diagrams -> New Story Board**
- **Start situation:**
  - Startsituation eines Szenarios herstellen
  - Methode aufrufen
- **Result situation:**
  - Endsituation prüfen

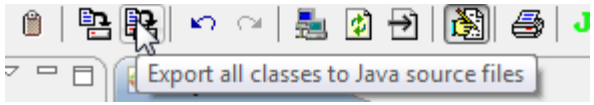



- **Wichtig: Rechtsklick auf das Diagramm -> Toggle Create Testcase**

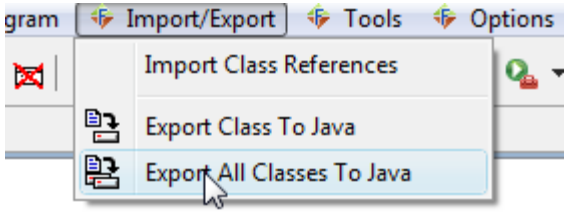


# Fujaba Storyboards III

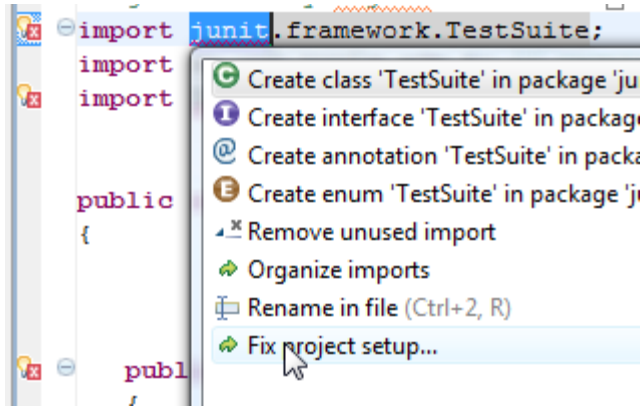
- **Wie gewohnt Code generieren**

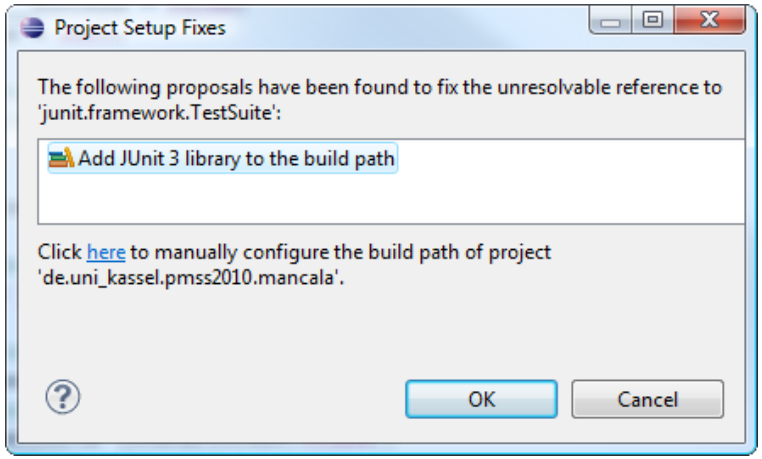



oder



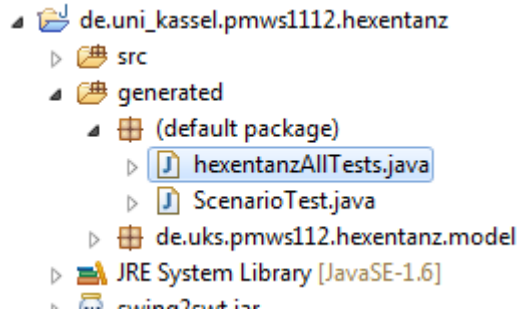

- **Neben dem Modell werden JUnit3 Testklassen generiert**
- **Falls nötig, JUnit3 Dependency hinzufügen**



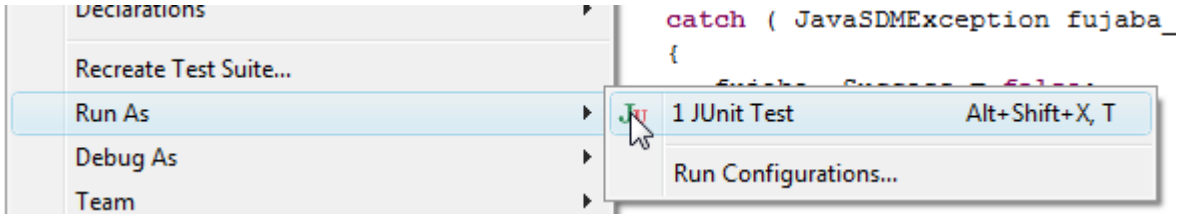



# Fujaba Storyboards IV

- **Generierte Tests ausführen:**
  - Rechtsklick




  - Run As -> JUnit Test



# Praktische Übung: Erstellen von Storyboards

- **Hexentanz Projekt vom PM Blog runterladen**
- **Storyboard zu** HexentanzGame::init(..) **erstellen**
- **Startsituation:**
  - HexentanzGame – Objekt
  - Methodenaufruf: `init(new String[]{„Alice“, „Bob“})`
- **Endsituation:**
  - Erwartete Objektstruktur nach init(…)

# Vorschau HA5

- **Deadline: 08.12.2011, 23:59 Uhr**
- **Aufgabe 1: Storyboards mit Fujaba4Eclipse:**
  - Hexe bewegen, mit Rausschmeissen
  - Rausschmeissen mit besetzem 7. Previous-Feld
  - Eigene Hexe in Endzone bewegen
  - Letzte eigene Hexe in Endzone bewegen
- **Randbedingungen:**
  - 1 HexentanzGame
  - 2 Spieler (Alice, Bob)

# Ende

## Jetzt: Betreutes Arbeiten

## Ansonsten: Schönes WE!