

## Vorlesung



University of Applied Sciences

# Programmieren I und II

## Unit 10

Objektorientierter Entwurf und (objektorientierte) Designprinzipien

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

1



University of Applied Sciences



**Prof. Dr. rer. nat.  
Nane Kratzke**

*Praktische Informatik und  
betriebliche Informationssysteme*

- Raum: 17-0.10
- Tel.: 0451 300 5549
- Email: [kratzke@fh-luebeck.de](mailto:kratzke@fh-luebeck.de)



@NaneKratzke

Updates der Handouts auch über Twitter #prog\_inf  
und #prog\_itd

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

2

## Units



University of Applied Sciences

Unit 1 Einleitung und Grundbegriffe	Unit 2 Grundelemente imperativer Programme	Unit 3 Selbstdefinierbare Datentypen und Collections	Unit 4 Einfache I/O Programmierung
Unit 5 Rekursive Programmierung und rekursive Datenstrukturen	Unit 6 Einführung in die objektorientierte Programmierung + UML	Unit 7 Konzepte objektorientierter Programmiersprachen	Unit 8 Testen (objektorientierter) Programme
Unit 9 Generische Datentypen	Unit 10 Objektorientierter Entwurf und objektorientierte Designprinzipien	Unit 11 Graphical User Interfaces	Unit 12 Multithread Programmierung



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

3

## Abgedeckte Ziele dieser UNIT



University of Applied Sciences

Kennen existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)	Verstehen des Unterschieds zwischen Werte- und Referenzsemantik
Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen	Kennen des Algorithmusbegriffs, Implementieren einfacher Algorithmen	Kennen objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung	Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)
Kennen von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML)	Kennen der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software	Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien	Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Multithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

4

## Themen dieser Unit



University of Applied Sciences

### OO Entwurf

- Beispiel Tic Tac Toe für erweiterbare Software
- Einfaches Vorgehensmodell

### OO Entwurfsprinzipien

- Lenkende Prinzipien bei OO Entwicklung
- Ein paar Regeln pro Prinzipien

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

5

## Zum Nachlesen ...



University of Applied Sciences



### Kapitel 4

#### Die Struktur objektorientierter Software

- 4.1 Die Basis von allem: das Objekt
- 4.2 Klassen: Objekte haben Gemeinsamkeiten
- 4.3 Beziehungen zwischen Objekten

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

6

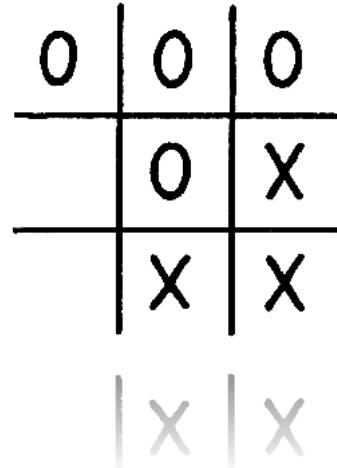
## Struktur objektorientierter Software am Beispiel des Spiels Tic Tac Toe



University of Applied Sciences

Klassisches,  
Zwei Personen  
Strategiespiel

Bereits im 12.  
Jh. v. Chr.  
bekannt



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

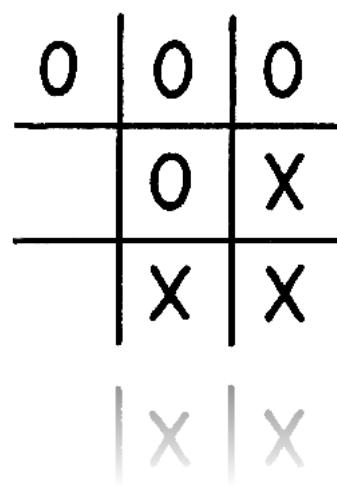
7

## Tic Tac Toe Spielverlauf und Regeln



University of Applied Sciences

- Auf einem 3×3 Felder großen Spielfeld machen die beiden Spieler abwechselnd ihre Zeichen (ein Spieler Kreuze, der andere Kreise).
- Der Spieler, der als erstes drei seiner Zeichen in eine Reihe, Spalte oder eine der beiden Hauptdiagonalen setzen kann, gewinnt.
- **Wenn allerdings beide Spieler optimal spielen, kann keiner gewinnen, und es kommt zu einem Unentschieden.**



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

8

## Struktur objektorientierter Software



University of Applied Sciences

Klasse	Objekt	Objektorientierte Abläufe
<ul style="list-style-type: none"><li>• Objekte haben Gemeinsamkeiten</li><li>• Modellierungsmittel</li><li>• Klassen sind Datentypen</li><li>• Sichtbarkeiten</li></ul>	<ul style="list-style-type: none"><li>• Konstruktoren/ Destruktoren</li><li>• Zustand</li><li>• Verhalten</li><li>• Ausprägungen von Klassen</li></ul>	<ul style="list-style-type: none"><li>• Interaktion zwischen Objekten</li><li>• Kontrakte und Exceptions</li></ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

9

## Tic Tac Toe Requirements



University of Applied Sciences

- Es soll eine T3Engine (Spiel) entwickelt werden, die es ermöglicht, zwei beliebige Strategien (Spieler) gegeneinander spielen zu lassen.
- Es sollen Regelverstöße erfasst und dem verursachenden Spieler zugeordnet werden.
- Laufzeitfehler eines Spielers sind als Regelverstöße zu werten.
- Begeht ein Spieler einen Regelverstoß, gewinnt automatisch der andere Spieler.
- Ein Regelverstoß soll durch das Spiel dokumentiert (ausgegeben) werden.
- Jeder Spieler hat einen Namen.
- Das Spiel erteilt den Spielern X und O wechselseitig das Zugrecht und ist für die Feststellung von Regelverstößen sowie Sieg, Niederlagen und Unentschieden zuständig.
- Der Spieler X beginnt das Spiel.
- Einmal gemachte Zeichen dürfen nicht überschrieben oder gelöscht werden.
- Der Spieler am Zug muss ein leeres Element des Felds mit seinem Zeichen belegen.
- Ein Spieler gewinnt, wenn er eine Spalte, Zeile oder Diagonale mit seinem Zeichen (X oder O) belegen konnte.
- Das Spiel endet unentschieden, wenn kein Spieler gewonnen hat und alle Felder belegt sind.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

10

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

Ableitung der Methoden pro Kernklasse

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

Sichtbarkeiten und Änderbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

11

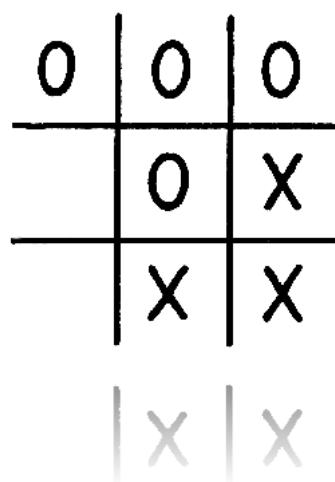
## Objekte in Tic Tac Toe

Was für „Akteure“ leiten Sie aus Tic Tac Toe ab?



University of Applied Sciences

- Ein Tic Tac Toe **Spiel** wird auf einem 3×3 Felder großen Spielfeld gespielt. Auf diesem Feld machen die beiden Spieler abwechselnd ihre Zeichen (ein Spieler Kreuze, der andere Kreise).
- Der **Spieler**, der als erstes drei seiner Zeichen in eine Reihe, Spalte oder eine der beiden Hauptdiagonalen setzen kann, gewinnt.



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

12

## Objekte in Tic Tac Toe

### Was für „Akteure“ leiten Sie aus Tic Tac Toe ab?



University of Applied Sciences

T3Spieler
Zustand des Spielers (Variablen)
Routinen des Spielers (Verhalten)

T3Spiel
Zustand des Spiels (Variablen)
Routinen des Spiels (Verhalten)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

13

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

Ableitung der Methoden pro Kernklasse

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

Sichtbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

14

## Tic Tac Toe Requirements

### Ableitung der Zustandsbeschreibung für T3Spiel



University of Applied Sciences

- Es soll eine T3Engine (Spiel) entwickelt werden, die es ermöglicht, zwei beliebige Strategien (Spieler) gegeneinander spielen zu lassen.
- Es sollen Regelverstöße erfasst und dem verursachenden Spieler zugeordnet werden.
- Laufzeitfehler eines Spielers sind als Regelverstöße zu werten.
- Begeht ein Spieler einen Regelverstoß, gewinnt automatisch der andere Spieler.
- Ein Regelverstoß soll durch das Spiel dokumentiert (ausgegeben) werden.
- Jeder Spieler hat einen Namen.
- Das Spiel erteilt den Spielern X und O wechselseitig das Zugrecht und ist für die Feststellung von Regelverstößen sowie Sieg, Niederlagen und Unentschieden zuständig.
- Der Spieler X beginnt das Spiel.
- Einmal gemachte Zeichen dürfen nicht überschrieben oder gelöscht werden.
- Der Spieler am Zug muss ein leeres Element des Felds mit seinem Zeichen belegen.
- Ein Spieler gewinnt, wenn er eine Spalte, Zeile oder Diagonale mit seinem Zeichen (X oder O) belegen konnte.
- Das Spiel endet unentschieden, wenn kein Spieler gewonnen hat und alle Felder belegt sind.

T3Spiel	
feld	anz_leere_felder
X_am_zug	O_am_zug
spielerX	spielerO
Routinen des Spiels (Verhalten)	

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

15

## Tic Tac Toe Requirements

### Ableitung der Zustandsbeschreibung für T3Spieler



University of Applied Sciences

- Es soll eine T3Engine (Spiel) entwickelt werden, die es ermöglicht, zwei beliebige Strategien (Spieler) gegeneinander spielen zu lassen.
- Es sollen Regelverstöße erfasst und dem verursachenden Spieler zugeordnet werden.
- Laufzeitfehler eines Spielers sind als Regelverstöße zu werten.
- Begeht ein Spieler einen Regelverstoß, gewinnt automatisch der andere Spieler.
- Ein Regelverstoß soll durch das Spiel dokumentiert (ausgegeben) werden.
- Jeder Spieler hat einen Namen.
- Das Spiel erteilt den Spielern X und O wechselseitig das Zugrecht und ist für die Feststellung von Regelverstößen sowie Sieg, Niederlagen und Unentschieden zuständig.
- Der Spieler X beginnt das Spiel.
- Einmal gemachte Zeichen dürfen nicht überschrieben oder gelöscht werden.
- Der Spieler am Zug muss ein leeres Element des Felds mit seinem Zeichen belegen.
- Ein Spieler gewinnt, wenn er eine Spalte, Zeile oder Diagonale mit seinem Zeichen (X oder O) belegen konnte.
- Das Spiel endet unentschieden, wenn kein Spieler gewonnen hat und alle Felder belegt sind.

T3Spieler	
name	regelverstoesse
Routinen des Spielers (Verhalten)	

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

16

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

**Ableitung der Methoden pro Kernklasse**

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

Sichtbarkeiten und Änderbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

17

## Tic Tac Toe Requirements

### Ableitung der Methoden für T3Spiel



University of Applied Sciences

- Es soll eine T3Engine (Spiel) entwickelt werden, die es ermöglicht, zwei beliebige Strategien (Spieler) gegeneinander spielen zu lassen.
- Es sollen Regelverstöße erfasst und dem verursachenden Spieler zugeordnet werden.
- Laufzeitfehler eines Spielers sind als Regelverstöße zu werten.
- Begeht ein Spieler einen Regelverstoß, gewinnt automatisch der andere Spieler.
- Ein Regelverstoß soll durch das Spiel dokumentiert (ausgegeben) werden.
- Jeder Spieler hat einen Namen.
- Das Spiel erteilt den Spielern X und O wechselseitig das Zugrecht und ist für die Feststellung von Regelverstößen sowie Sieg, Niederlagen und Unentschieden zuständig.
- Der Spieler X beginnt das Spiel.
- Einmal gemachte Zeichen dürfen nicht überschrieben oder gelöscht werden.
- Der Spieler am Zug muss ein leeres Element des Felds mit seinem Zeichen belegen.
- Ein Spieler gewinnt, wenn er eine Spalte, Zeile oder Diagonale mit seinem Zeichen (X oder O) belegen konnte.
- Das Spiel endet unentschieden, wenn kein Spieler gewonnen hat und alle Felder belegt sind.

T3Spiel	
feld	
anz_leere_felder	
X_am_zug	
O_am_zug	
spielerX	
spielerO	
leite_spiel	
gewonnen	
unentschieden	
setze_auf_feld (inkl. Prüfungen)	
schiedsrichter_information	

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

18

## Tic Tac Toe Requirements

### Ableitung der Methoden für T3Spieler



University of Applied Sciences

- Es soll eine T3Engine (Spiel) entwickelt werden, die es ermöglicht, zwei beliebige Strategien (Spieler) gegeneinander spielen zu lassen.
- Es sollen Regelverstöße erfasst und dem verursachenden Spieler zugeordnet werden.
- Laufzeitfehler eines Spielers sind als Regelverstöße zu werten.
- Begeht ein Spieler einen Regelverstoß, gewinnt automatisch der andere Spieler.
- Ein Regelverstoß soll durch das Spiel dokumentiert (ausgegeben) werden.
- Jeder Spieler hat einen Namen.
- Das Spiel erteilt den Spielern X und O wechselseitig **das Zugrecht** und ist für die Feststellung von Regelverstößen sowie Sieg, Niederlagen und Unentschieden zuständig.
- Der Spieler X beginnt das Spiel.
- Einmal gemachte Zeichen dürfen nicht überschrieben oder gelöscht werden.
- Der Spieler am Zug muss ein leeres Element des Felds mit seinem Zeichen belegen.
- Ein Spieler gewinnt, wenn er eine Spalte, Zeile oder Diagonale mit seinem Zeichen (X oder O) belegen konnte.
- Das Spiel endet unentschieden, wenn kein Spieler gewonnen hat und alle Felder belegt sind.

T3Spieler	
name	regelverstoesse
am_zug	melde_regelverstoess

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

19

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

Ableitung der Methoden pro Kernklasse

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

Sichtbarkeiten und Änderbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

20

## Typisierung der Klassen



University of Applied Sciences

### T3Spiel

```
feld  
anz_leere_felder  
X_am_zug  
O_am_zug  
spielerX  
spielerO
```

```
leite_spiel  
gewonnen  
unentschieden  
setze_auf_feld (inkl. Prüfungen)  
schiedsrichter_information
```

### T3Spiel

```
char[][] feld = new char[3][3];  
int anz_leere_felder = 9;  
boolean X_am_zug = true;  
boolean O_am_zug = false;  
T3Spieler spielerX  
T3Spieler spielerO
```

```
T3Spiel(T3Spieler, T3Spieler) // Konstruktor
```

```
char leite_spiel()  
boolean gewonnen(char)  
boolean unentschieden()  
void setze_auf_feld(T3Spieler, char, int, int)  
throws Exception  
String schiedsrichter_information(char, Exception)
```

Prof. Dr. rer. nat. Nane Kratzke

21

## Typisierung der Klassen



University of Applied Sciences

### T3Spieler

```
name  
regelverstoesse
```

```
am_zug  
melde_regelverstoss
```

### T3Spieler

```
String name = "";  
int regelverstoesse = 0;
```

```
T3Spieler(String) // Konstruktor
```

```
void am_zug(char, T3Spiel) throws Exception  
void melde_regelverstoss()
```

Prof. Dr. rer. nat. Nane Kratzke

22

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

Ableitung der Methoden pro Kernklasse

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

Sichtbarkeiten und Änderbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

23

## Kontrakte



University of Applied Sciences

Im vorhergehenden Schritt wurde die Syntax festgelegt:

- Rückgabetyp
- Name
- Zahl und Typ der Aufrufparameter
- Strukturelle Festlegungen

Es wurde aber noch nicht die Semantik festgelegt:

- Was soll eine Methode leisten?
- Wie soll sie sich verhalten?
- Inhaltliche Festlegungen

Hierzu dienen Kontrakte

- Diese können informell sein, bspw. Durch textuelle Beschreibung der Funktionsweise einer Methode im Quelltext mittels Kommentaren
- Formell – z.B. mit einer Spezifikationssprache wie Object-Z die mathematische Voraus- und Nachbedingungen zu einer Operation festlegt.

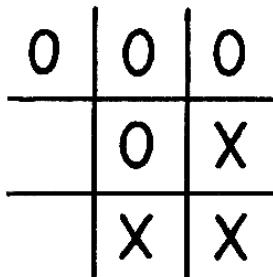
Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

24

## Kontrakte der T3Spiel Methoden



University of Applied Sciences



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

25

## Kontrakte der T3Spiel-Methoden Konstruktor



University of Applied Sciences

T3Spiel
char[][] feld = new char[3][3]; int anz_leere_felder = 9; boolean X_am_zug = true; boolean O_am_zug = false; T3Spiel spielerX T3Spiel spielerO
<b>T3Spiel(T3Spieler, T3Spieler) // Konstruktor</b>
char leite_spiel() boolean gewonnen(char) boolean unentschieden() void setze_auf_feld(T3Spieler, char, int, int) throws Exception String schiedsrichter_information(char, Exception)

### Informeller Kontrakt für Konstruktor T3Spiel

Der Konstruktor weist die Rollen in einem Spiel zu.

Der Aufruf

T3Spiel s = new  
T3Spiel(s1, s2);

bedeutet, dass s1 die Rolle X und s2 die Rolle O im Spiel s einnimmt  
(spielerX == s1 und spielerO == s2)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

26

## Kontrakte der T3Spiel-Methoden

### Methode **leite\_spiel**



University of Applied Sciences

T3Spiel
<pre>char[] feld = new char[3][3]; int anz_leere_felder = 9; boolean X_am_zug = true; boolean O_am_zug = false; T3Spiel spielerX T3Spiel spielerO</pre>
<pre>T3Spiel(T3Spieler, T3Spieler) // Konstruktor  <b>char leite_spiel()</b> boolean gewonnen(char) boolean unentschieden() void setze_auf_feld(T3Spieler, char, int, int) throws Exception String schiedsrichter_information(char, Exception)</pre>

#### Informeller Kontrakt für Methode **leite\_spiel**

Die Methode startet ein Spiel zwischen spielerX und spielerO.

Die Methode liefert folgende Rückgaben:  
 •X (wenn spielerX gewinnt)  
 •O (wenn spielerO gewinnt)  
 •Leerzeichen (wenn unentschieden)

Begehen spielerX oder spielerO Regelverstöße wird deren Methode **melde\_regelverstoss** aufgerufen.

## Kontrakte der T3Spiel-Methoden

### Methode **gewonnen**



University of Applied Sciences

T3Spiel
<pre>char[] feld = new char[3][3]; int anz_leere_felder = 9; boolean X_am_zug = true; boolean O_am_zug = false; T3Spiel spielerX T3Spiel spielerO</pre>
<pre>T3Spiel(T3Spieler, T3Spieler) // Konstruktor  <b>char leite_spiel()</b> boolean gewonnen(char) boolean unentschieden() void setze_auf_feld(T3Spieler, char, int, int) throws Exception String schiedsrichter_information(char, Exception)</pre>

#### Informeller Kontrakt für Methode **gewonnen**

Eingabeparameter v (char).  
 Die Methode prüft ob v (X oder O) gem. der Feldbelegung gewonnen hat.

Die Methode liefert folgende Rückgaben:  
 •true (wenn in feld eine Spalte, Reihe oder Diagonale mit v durchgängig belegt sind)  
 •false sonst

## Kontrakte der T3Spiel-Methoden

### Methode **unentschieden**



University of Applied Sciences

#### T3Spiel

```
char[] feld = new char[3][3];
int anz_leere_felder = 9;
boolean X_am_zug = true;
boolean O_am_zug = false;
T3Spiel spielerX
T3Spiel spielerO

T3Spiel(T3Spieler, T3Spieler) // Konstruktor

char leite_spiel()
boolean gewonnen(char)
boolean unentschieden()
void setze_auf_feld(T3Spieler, char, int, int)
    throws Exception
String schiedsrichter_information(char, Exception)
```

#### Informeller Kontrakt für Methode **unentschieden**

Die Methode prüft ob ein Unentschieden vorliegt.

Die Methode liefert folgende Rückgaben:

- true (wenn
  - gewonnen(X) == false und
  - gewonnen(O) == false und
  - anz\_leere\_felder == 0)
- false sonst

## Kontrakte der T3Spiel-Methoden

### Methode **setze\_auf\_feld**



University of Applied Sciences

#### T3Spiel

```
char[] feld = new char[3][3];
int anz_leere_felder = 9;
boolean X_am_zug = true;
boolean O_am_zug = false;
T3Spiel spielerX
T3Spiel spielerO

T3Spiel(T3Spieler, T3Spieler) // Konstruktor

char leite_spiel()
boolean gewonnen(char)
boolean unentschieden()
void setze_auf_feld(T3Spieler, char, int, int)
    throws Exception
String schiedsrichter_information(char, Exception)
```

#### Informeller Kontrakt für Methode **setze\_auf\_feld**

Parameter:  
T3Spieler s,  
char v (X oder O),  
int x, int y

Die Methode setzt fuer Spieler s, den Wert v auf das Spielfeld feld an Position x und y. Es wird eine Exception ausgelöst, wenn eine der folgenden Bedingungen gilt:

- s in Rolle v nicht am Zug
- x,y keine zulässige Pos.
- x,y bereits belegt
- v nicht O oder X ist

## Kontrakte der T3Spiel-Methoden

### Methode **schiedsrichter\_information**



University of Applied Sciences

T3Spiel
char[][] feld = new char[3][3]; int anz_leere_felder = 9; boolean X_am_zug = true; boolean O_am_zug = false; T3Spiel spielerX T3Spiel spielerO
T3Spiel(T3Spieler, T3Spieler) // Konstruktor  char leite_spiel() boolean gewonnen(char) boolean unentschieden() void setze_auf_feld(T3Spieler, char, int, int) throws Exception <b>String schiedsrichter_information(char, Exception)</b>

#### Informeller Kontrakt für Methode schiedsrichter\_info

Parameter:  
char v (X oder O),  
Exception ex

Die Methode erzeugt eine Fehlermeldung, wenn eine Exception durch einen Spieler ausgelöst wurde. Es werden die Spieler spielerX und spielerO, die Rolle v in der die Exception ausgelöst wurde und die Feldbelegung von feld sowie ein erläuternder Text der Exception ex ausgegeben.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

31

## Kontrakte der T3Spieler Methoden



University of Applied Sciences



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

32

## Kontrakte der T3Spieler-Methoden

### Konstruktor



University of Applied Sciences

```
T3Spieler  
-----  
String name = "";  
int regelverstoesse = 0;  
  
-----  
T3Spieler(String) // Konstruktor  
void am_zug(char, T3Spiel) throws Exception  
void melde_regelverstoss()
```

#### Informeller Kontrakt für Konstruktor T3Spieler

Parameter:  
String n

Dieser Konstruktor legt einen Spieler mit dem Namen n an.

Der Aufruf

```
T3Spieler s = new  
T3Spieler("Max Mustermann  
");
```

bedeutet, dass der Spieler mit dem Namen Max Mustermann angelegt wird. (name == "Max Mustermann")

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

33

## Kontrakte der T3Spieler-Methoden

### Methode am\_zug



University of Applied Sciences

```
T3Spieler  
-----  
String name = "";  
int regelverstoesse = 0;  
  
-----  
T3Spieler(String) // Konstruktor  
void am_zug(char, T3Spiel) throws Exception  
void melde_regelverstoss()
```

#### Informeller Kontrakt für Methode am\_zug

Parameter:  
char v (X oder O),  
T3Spiel s

Die Methode ist ein „Hook“, die aufgerufen wird, wenn der Spieler im Spiel s in der Rolle v am Zug ist.

In dieser Methode wird die Spielstrategie implementiert. Die Spielstrategie kann hierzu den Zustand des Spiels s auswerten und in einen Zug mittels s.setze\_auf\_feld umsetzen.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

34

## Kontrakte der T3Spieler-Methoden

### Methode **melde\_regelverstoss**



University of Applied Sciences

```
T3Spieler  
  
String name = "";  
int regelverstoesse = 0;  
  
T3Spieler(String) // Konstruktor  
void am_zug(char, T3Spiel) throws Exception  
void melde_regelverstoss()
```

#### Informeller Kontrakt für Methode **melde\_regelverstoss**

Die Methode wird aufgerufen, wenn ein Regelverstoß erkannt worden ist.

Die Methode inkrementiert regelverstoesse um eins.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

35

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

Ableitung der Methoden pro Kernklasse

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

Sichtbarkeiten und Änderbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

36

## Sichtbarkeiten, Änderungsmöglichkeiten und Kapselung



University of Applied Sciences

Im Rahmen des objektorientierten Entwurfs geht es nicht nur um die zu implementierende Funktionalität,

sondern auch darum, festzulegen, welche Erweiterungspunkte vorzusehen sind und

welche **Zugriffsmöglichkeiten** von diesen Erweiterungspunkten aus eingeräumt werden sollen.

Und welche **Änderungsmöglichkeiten** an diesen Erweiterungspunkten eingeräumt werden.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

37

## Einschränkungen von Sichtbarkeiten und Änderungen (Kapselung)



University of Applied Sciences

Folgende **Änderungseinschränkungen** sind in OO-Sprachen bekannt:

Final

- Methode darf nicht mehr überladen werden.

Abstract

- Methode muss noch implementiert werden.

Kein Qualifier

- Methode kann, muss aber nicht überladen werden.

Folgende **Zugriffseinschränkungen** sind in OO-Sprachen bekannt:

Private

- Zugriff nur aus der definierenden Klasse heraus möglich

Protected

- Zugriff nur aus demselben Paket oder allen abgeleiteten Klassen möglich

Public

- Zugriff aus allen Paketen
- Und allen Klassen möglich

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

38

## Erweiterungspunkte in Tic Tac Toe

Zentraler Erweiterungspunkt für Spielstrategien ist die Klasse T3Spieler

Diese Implementierungen (Ableitungen von T3Spieler) sind daher nicht kontrollierbar und aus Sicht der Engine „mit Vorsicht zu genießen“.

Von T3Spieler abgeleitete Strategien dürfen nicht

- Den Zustand des Spiels unkontrolliert ändern
- Das eigene Regelverstoßkonto manipulieren können

Von T3Spieler abgeleitete Strategien müssen

- den Belegungszustand des Feldes auslesen können
- den Belegungszustand des Feldes kontrolliert durch Ihren Spielzug ändern können
- minimalen Zugriff auf den Spielzustand haben.

## Zugriffe beschränken (Daten kapseln) am Beispiel T3Spiel

0	0	0
0	X	
X		X
X	X	

T3Spiel	
protected	char[][] feld = new char[3][3];
protected	int anz_leere_felder = 9;
protected	boolean X_am_zug = true;
protected	boolean O_am_zug = false;
protected	T3Spiel spielerX
protected	T3Spiel spielerO
public	T3Spiel(T3Spieler, T3Spieler)
public	char leite_spiel()
protected	boolean gewonnen(char)
protected	boolean unentschieden()
public	void setze_auf_feld(T3Spieler, char, int, int)
	throws Exception
protected	String schiedsrichter_information(char, Exception)

Keine unkontrollierten Spielstandsänderungen.

Kontrollierte Spielstandsänderungen zu lassen.

Minimaler Zugriff auf Spielstand.

Lesenden Zugriff auf Feld einräumen

### Lesenden Zugriff auf Attribute mittels „getter“-Methoden realisieren

<pre> T3Spiel protected char[][] feld = new char[3][3]; protected int anz_leere_felder = 9; protected boolean X_am_zug = true; protected boolean O_am_zug = false; protected T3Spieler spielerX; protected T3Spieler spielerO;  public T3Spiel(T3Spieler, T3Spieler) public char leite_spiel() protected boolean gewonnen(char) protected boolean unentschieden() public void setze_auf_feld(T3Spieler, char, int, int) throws Exception protected String schiedsrichter_information(char, Exception) public char[] lese_feld() </pre>	<table border="1" style="border-collapse: collapse; width: 100%; height: 100px;"> <tr><td style="width: 33%;">0</td><td style="width: 33%;">0</td><td style="width: 33%;">0</td></tr> <tr><td>0</td><td>X</td><td></td></tr> <tr><td>X</td><td>X</td><td></td></tr> <tr><td>X</td><td>X</td><td></td></tr> </table>	0	0	0	0	X		X	X		X	X	
0	0	0											
0	X												
X	X												
X	X												

**Informeller Kontrakt für Methode lese\_feld**

Die Methode erzeugt eine Kopie des Attributs feld und liefert diese an den Aufrufer zurück.

So kann sichergestellt werden, dass die Feldbelegung gelesen aber nicht verändert werden kann.

**Lesenden Zugriff auf Feld einräumen**



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 41

### Änderungen einschränken am Beispiel T3Spieler

<pre> T3Spieler private String name = ""; private int regelverstoesse = 0;  public T3Spieler(String) public abstract void am_zug(char, T3Spiel) throws Exception public final void melde_regelverstoss() </pre>	 <p><b>Keine Manipulation des Regelverstoßkontos durch Erweiterungen</b></p> <p><b>Implementierung der eigenen Spielstrategie erforderlich</b></p> <p><b>Lesenden Zugriff auf Regelverstoßkonto</b></p> 
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 42

## Änderungen einschränken am Beispiel T3Spieler



```

T3Spieler

private String name = "";
private int regelverstoesse = 0;

public T3Spieler(String)
public abstract void am_zug(char, T3Spiel)
throws Exception
public final void melde_regelverstoss()
public final int anz_regelverstoesse()
  
```

**Informeller Kontrakt für  
Methode  
anz\_regelverstoesse**

Die Methode liefert den  
Wert, der im Attribut  
regelverstoesse  
gespeichert ist.

Lesenden Zugriff auf  
Regelverstoßkonto

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

43

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

Ableitung der Methoden pro Kernklasse

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

Sichtbarkeiten und Änderbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

44

## Implementierung der T3Spieler Methoden



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

45

## Implementierung der T3Spieler-Methoden Konstruktor

### Informeller Kontrakt für Konstruktor T3Spieler

Parameter:  
String n

Dieser Konstruktor legt  
einen Spieler mit dem Namen  
n an.

Der Aufruf

```
T3Spieler s = new  
T3Spieler("Max Mustermann");
```

bedeutet, dass der Spieler  
mit dem Namen Max Mustermann  
angelegt wird. (name == "Max  
Mustermann")

```
public abstract class T3Spieler {  
  
    private String name = "";  
  
    ...  
  
    public T3Spieler(String n) {  
        this.name = n;  
    }  
  
    ...  
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

46

## Implementierung der T3Spieler-Methoden

### Methode am\_zug



University of Applied Sciences

#### Informeller Kontrakt für Methode am\_zug

Parameter:  
char v (X oder O),  
T3Spiel s

Die Methode ist ein „Hook“, die aufgerufen wird, wenn der Spieler im Spiel s in der Rolle v am Zug ist.

In dieser Methode wird die Spielstrategie implementiert. Die Spielstrategie kann hierzu den Zustand des Spiels s auswerten und in einen Zug mittels s.setze\_auf\_feld umsetzen.

```
public abstract class T3Spieler {  
    ...  
    public abstract void am_zug(char v,  
                                T3Spiel s);  
    ...  
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

47

## Abstrakte Klassen



University of Applied Sciences

- Eine abstrakte Klasse bezeichnet in der OO-Programmierung eine spezielle Klasse mit mindestens einer, abstrakten Methode (Nur Methodensignatur ohne Implementierung).
- Aus abstrakten Klassen können keine Objekte erzeugt (instantiiert) werden.
- Schnittstellen (Interfaces) sind rein abstrakte Klassen, die nur Methodensignaturen deklarieren.
- Als Basisklassen in einer Klassenhierarchie können abstrakte Klassen grundlegende Eigenschaften ihrer Unterklassen festlegen, ohne diese bereits konkret zu implementieren.
- Leitet eine Klasse von einer abstrakten Klasse ab, müssen alle vererbten abstrakten Methoden überschrieben und implementiert werden, damit die erbende Klasse selbst nicht abstrakt ist.
- Abstrakte Klassen werden dazu genutzt, Teile des Quelltextes allgemein zu halten.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

48

## Implementierung der T3Spieler-Methoden

### Methode **melde/anz\_regelverstoesse**



University of Applied Sciences

Informeller Kontrakt für Methode **melde\_regelverstoss**

Die Methode wird aufgerufen, wenn ein Regelverstoß erkannt worden ist.

Die Methode inkrementiert regelverstoesse um eins.

Informeller Kontrakt für Methode **anz\_regelverstoesse**

Die Methode liefert den Wert, der im Attribut regelverstoesse gespeichert ist.

```
public abstract class T3Spieler {  
  
    private int regelverstoesse = 0;  
  
    ...  
  
    public final void melde_regelverstoss() {  
        this.regelverstoesse++;  
    }  
  
    public final int anz_regelverstoesse() {  
        return this.regelverstoesse;  
    }  
  
    ...  
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

49

## Implementierung der T3Spiel Methoden



University of Applied Sciences

0	0	0
0	X	
X	X	

| X | X |

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

50

## Implementierung der T3Spiel-Methoden

### Konstruktor



University of Applied Sciences

**Informeller Kontrakt für Konstruktor T3Spiel**

Der Konstruktor weist die Rollen in einem Spiel zu.

Der Aufruf

```
T3Spiel s = new T3Spiel(s1, s2);
```

bedeutet, dass s1 die Rolle X und s2 die Rolle O im Spiel s einnimmt  
(spielerX == s1 und spielerO == s2)

```
public class T3Spiel {  
    ...  
    protected T3Spieler spielerX;  
    protected T3Spieler spielerO;  
    ...  
    public T3Spiel(T3Spieler s1,  
                  T3Spieler s2) {  
        this.spielerX = s1;  
        this.spielerO = s2;  
    }  
    ...  
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

51

## Implementierung der T3Spiel-Methoden

### Methode leite\_spiel



University of Applied Sciences

**Informeller Kontrakt für Methode leite\_spiel**

Die Methode startet ein Spiel zwischen spielerX und spielerO.

Die Methode liefert folgende Rückgaben:

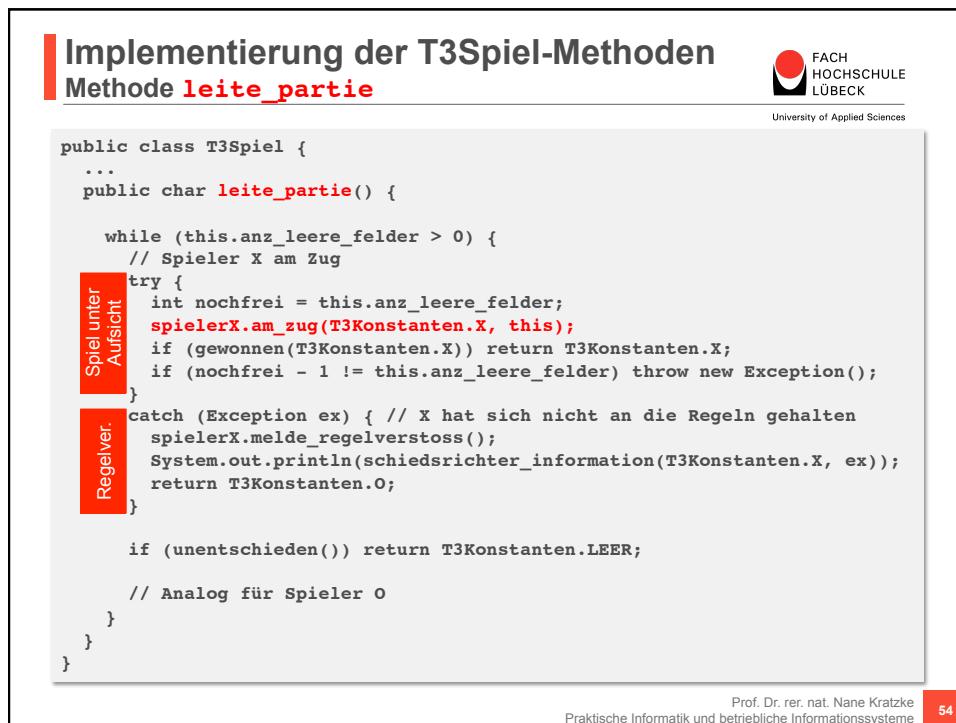
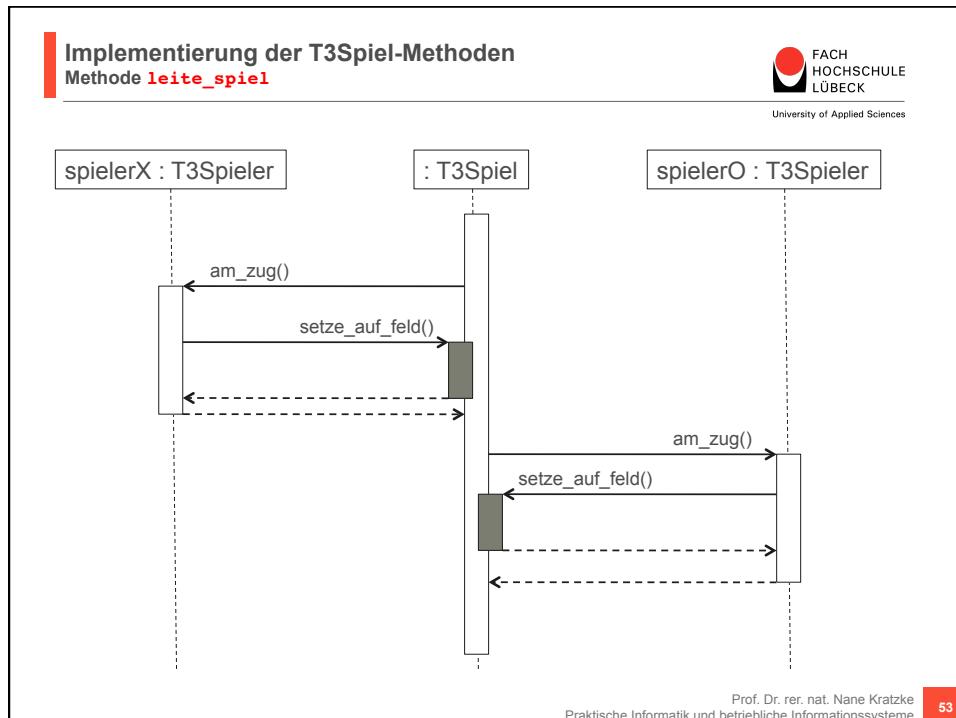
- X (wenn spielerX gewinnt)
- O (wenn spielerO gewinnt)
- Leerzeichen (wenn unentschieden)

Begehen spielerX oder spielerO Regelverstöße wird deren Methode melde\_regelverstoss aufgerufen.



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

52



## Implementierung der T3Spiel-Methoden

### Methode **gewonnen**

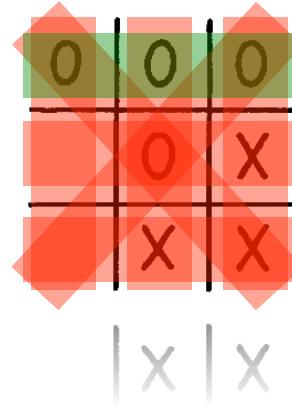
Informeller Kontrakt für Methode gewonnen

Eingabeparameter v (char).

Die Methode prüft ob v (X oder O) gem. der Feldbelegung gewonnen hat.

Die Methode liefert folgende Rückgaben:

- true (wenn in feld eine Spalte, Reihe oder Diagonale mit v durchgängig belegt sind)
- false sonst



## Implementierung der T3Spiel-Methoden

### Methode **gewonnen**

```
public class T3Spiel {  
    ...  
    protected boolean gewonnen(char v) {  
        boolean diag1 = true;  
        boolean diag2 = true;  
  
        for (int i = 0; i < T3Konstanten.BREITE; i++) {  
            boolean spalte = true;  
            boolean zeile = true;  
            for (int j = 0; j < T3Konstanten.BREITE; j++) {  
                spalte = spalte && this.feld[i][j] == v;  
                zeile = zeile && this.feld[j][i] == v;  
            }  
  
            if (spalte || zeile) return true;  
  
            diag1 = diag1 && this.feld[i][i] == v;  
            diag2 = diag2 && this.feld[i][T3Konstanten.BREITE - 1 - i] == v;  
        }  
  
        return diag1 || diag2;  
    }  
    ...  
}
```

## Implementierung der T3Spiel-Methoden

### Methode **unentschieden**



University of Applied Sciences

#### Informeller Kontrakt für Methode unentschieden

Die Methode prüft ob ein Unentschieden vorliegt.

Die Methode liefert folgende Rückgaben:

- true (wenn
  - gewonnen(X) == false und
  - gewonnen(O) == false und
  - anz\_leere\_felder == 0)
- false sonst

```
public class T3Spiel {  
    ...  
    protected boolean unentschieden() {  
        return this.anz_leere_felder == 0 &&  
            !gewonnen(T3Konstanten.X) &&  
            !gewonnen(T3Konstanten.O);  
    }  
    ...  
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

57

## Implementierung der T3Spiel Methoden

### Methode **lese\_feld**



University of Applied Sciences

#### Informeller Kontrakt für Methode lese\_feld

Die Methode erzeugt eine Kopie des Attributs feld und liefert diese an den Aufrufer zurück.

So kann sichergestellt werden, dass die Feldbelegung gelesen aber nicht verändert werden kann.

```
public class T3Spiel {  
    ...  
    public char[][] lese_feld() {  
        char[][] clone = this.feld.clone();  
        for (int i = 0; i < clone.length; i++)  
            clone[i] = this.feld[i].clone();  
        return clone;  
    }  
    ...  
}
```

Wieso geht diese Variante nicht?

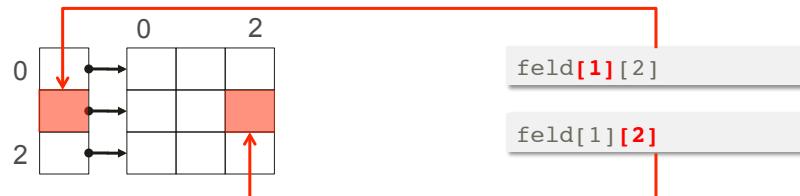
```
public class T3Spiel {  
    ...  
    public char[][] lese_feld() {  
        return this.feld.clone();  
    }  
    ...  
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

58

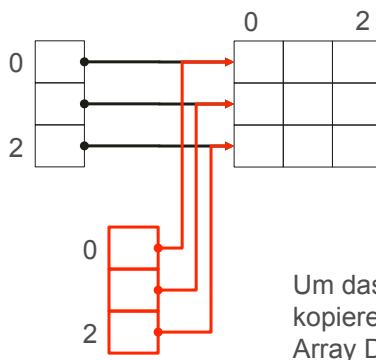
## Mehrdimensionale Arrays kopieren (I)

Mehrdimensionale Arrays werden als Arrays von Arrays angelegt.



## Mehrdimensionale Arrays kopieren (II)

`feld.clone` hat folgenden Effekt



Um das gesamte 2D-Array zu kopieren, müssen Sie also jedes Array Dimension für Dimension klonen (deepclone).

## Implementierung der T3Spiel-Methoden

### Methode **setze\_auf\_feld**



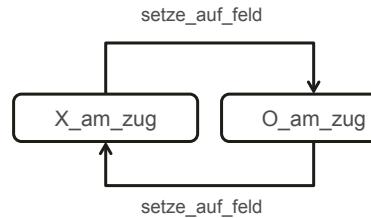
University of Applied Sciences

#### Informeller Kontrakt für Methode **setze\_auf\_feld**

Parameter:  
T3Spieler s,  
char v (X oder O),  
int x, int y

Die Methode setzt fuer Spieler s, den Wert v auf das Spielfeld feld an Position x und y. Es wird eine Exception ausgelöst, wenn eine der folgenden Bedingungen gilt:

- s in Rolle v nicht am Zug
- x,y keine zulässige Pos.
- x,y bereits belegt
- v nicht O oder X ist



$\begin{array}{ c c c } \hline 0 & 0 & 0 \\ \hline 0 & X & \\ \hline X & X & X \\ \hline \end{array}$	anz_leere_felder--
$\begin{array}{ c c c } \hline X & X & \\ \hline \end{array}$	

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

61

## Implementierung der T3Spiel-Methoden

### Methode **setze\_auf\_feld**



University of Applied Sciences

```

public class T3Spiel {
    ...
    public void setze_auf_feld(T3Spieler s, char v, int x, int y)
        throws Exception {
        Regelkonformer Zug?
        if ((v == T3Konstanten.X) && !X_am_zug) throw new Exception();
        if ((v == T3Konstanten.O) && !O_am_zug) throw new Exception();
        if ((v != T3Konstanten.O) && (v != T3Konstanten.X))
            throw new Exception();
        if (x < 0 || x >= T3Konstanten.BREITE) throw new Exception();
        if (y < 0 || y >= T3Konstanten.BREITE) throw new Exception();
        if (feld[x][y] != T3Konstanten.LEER) throw new Exception();

        Status
        this.feld[x][y] = v; Belege das Feld

        this.anz_leere_felder--;
        this.X_am_zug = !this.X_am_zug;
        this.O_am_zug = !this.O_am_zug;
    }
    ...
}
  
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

62

## Implementierung der T3Spiel-Methoden

### Methode **schiedsrichter\_information**



University of Applied Sciences

```
Informeller Kontrakt für
Methode
schiedsrichter_info

Parameter:
char v (X oder O),
Exception ex

Die Methode erzeugt eine
Fehlermeldung, wenn eine
Exception durch einen
Spieler ausgelöst wurde.
Es werden die Spieler
spielerX und spielerO, die
Rolle v in der die
Exception ausgelöst wurde
und die Feldbelegung von
feld sowie ein
erläuternder Text der
Exception ex ausgegeben.
```

### Bsp. T3-Fehlermeldung

Folgende Regelverletzung
ist durch 0 begonnen
worden: Division by zero

X: Max Mustermann

O: Sabine Sauertopf

```
X| |
-+--|
| |
-+--|
| |
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

63

## Implementierung der T3Spiel-Methoden

### Methode **schiedsrichter\_information**



University of Applied Sciences

```
protected String schiedsrichter_information(char durch, Exception ex) {
    String message = "Folgende Regelverletzung ist durch " + durch +
        " begonnen worden: " + ex.getMessage() + "\n";
    message += "X: " + this.spielerX + "\n";
    message += "O: " + this.spielerO + "\n";
    message += this.toString();
    return message;
}

public String toString() {
    String ret = "";
    for (char[] zeilen : feld) {
        String zeile = "";
        for (char spalte : zeilen) zeile += spalte + T3Konstanten.HSEP;
        ret += zeile.substring(0, zeile.length() - 1) + "\n";
        ret += T3Konstanten.VSEP + "\n";
    }
    return ret.substring(0, ret.length() - T3Konstanten.VSEP.length() - 1);
}
```

Ausgabe  
Fehlermeldung

Tic Tac Toe Feld in  
String wandeln

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

64

## Zusammenfassung Architektur des T3-Frameworks

```

classDiagram
    class IhreSpielstrategien
    class T3Spieler
    class T3Starter
    class T3SpielEngine

    IhreSpielstrategien --> T3Spieler : <<extends>>
    T3Spieler --> T3SpielEngine : spielerX
    T3Spieler --> T3SpielEngine : spielerO
    T3Starter --> T3SpielEngine : <<uses>>
    T3SpielEngine --> T3Starter : <<starts>>
  
```

A thought bubble contains the text: "Sie sollen in der Übung nun eine eigene T3-Strategie implementieren". To the right, there is a drawing of a person sitting at a desk with a computer monitor, and another drawing of a bear holding a sign that says "A+".

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

65

## Themen dieser Unit

**OO Entwurf**

- Beispiel Tic Tac Toe für erweiterbare Software
- Einfaches Vorgehensmodell

**OO Entwurfsprinzipien**

- Lenkende Prinzipien bei OO Entwicklung
- Ein paar Regeln pro Prinzipien

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

66

## Zum Nachlesen ...



University of Applied Sciences



### Kapitel 3

#### Prinzipien des objektorientierten Entwurfs

- 3.1 Prinzip der einzigen Verantwortung
- 3.2 Trennen der Anliegen
- 3.3 Wiederholungen vermeiden
- 3.4 Offen für Erweiterungen, geschlossen für Änderungen
- 3.5 Trennung von Schnittstelle und Implementierung
- 3.6 Umkehr der Abhängigkeiten
- 3.7 Mach es testbar

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

67

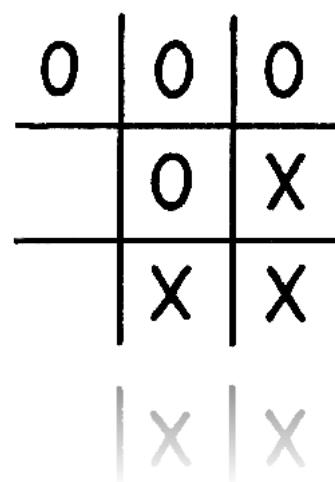
## Struktur objektorientierter Software am Beispiel des Spiels Tic Tac Toe



University of Applied Sciences

Klassisches,  
Zwei Personen  
Strategiespiel

Bereits im 12.  
Jh. v. Chr.  
bekannt



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

68

## Tic Tac Toe Requirements



University of Applied Sciences

- Es soll eine T3Engine (Spiel) entwickelt werden, die es ermöglicht, zwei beliebige Strategien (Spieler) gegeneinander spielen zu lassen.
- Es sollen Regelverstöße erfasst und dem verursachenden Spieler zugeordnet werden.
- Laufzeitfehler eines Spielers sind als Regelverstöße zu werten.
- Begeht ein Spieler einen Regelverstoß, gewinnt automatisch der andere Spieler.
- Ein Regelverstoß soll durch das Spiel dokumentiert (ausgegeben) werden.
- Jeder Spieler hat einen Namen.
- Das Spiel erteilt den Spielern X und O wechselseitig das Zugrecht und ist für die Feststellung von Regelverstößen sowie Sieg, Niederlagen und Unentschieden zuständig.
- Der Spieler X beginnt das Spiel.
- Einmal gemachte Zeichen dürfen nicht überschrieben oder gelöscht werden.
- Der Spieler am Zug muss ein leeres Element des Felds mit seinem Zeichen belegen.
- Ein Spieler gewinnt, wenn er eine Spalte, Zeile oder Diagonale mit seinem Zeichen (X oder O) belegen konnte.
- Das Spiel endet unentschieden, wenn kein Spieler gewonnen hat und alle Felder belegt sind.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

69

## Vorgehen



University of Applied Sciences

Ableitung der zentralen Akteure (Kernklassen)

Ableitung der Zustandsbeschreibungen der Kernklassen

Ableitung der Methoden pro Kernklasse

Typisierung der Kernklassen (Zustände und Methoden)

Kontrakte der Methoden

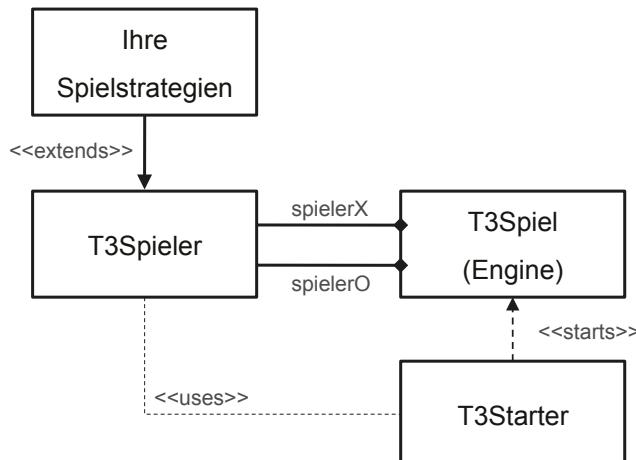
Sichtbarkeiten und Änderbarkeiten der Zustände und Methoden (Kapselung)

Implementierung der Methoden

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

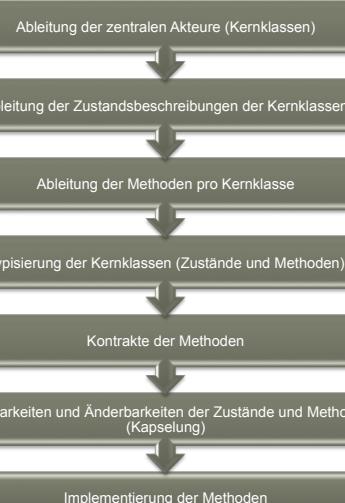
70

## Resultat: Architektur und Realisierung des T3-Frameworks



## Prinzipiengeleitetes Entwurfsvorgehen

**Prinzipien des objektorientierten Entwurfs ...**  
... werden bei jeder einzelnen Entwurfs- und Implementierungsentscheidung berücksichtigt.



## Prinzipien des objektorientierten Entwurfs



University of Applied Sciences

**1**

- Prinzip einer einzigen Verantwortung
- Single Responsibility

**2**

- Trennung der Anliegen
- Separation of Concerns

**3**

- Wiederholungen vermeiden
- Don't repeat yourself

**4**

- Offen für Erweiterungen, geschlossen für Änderungen
- Open-Closed-Principle

**5**

- Trennung von Schnittstelle und Implementierung
- Program to interfaces

**6**

- Umkehr der Abhängigkeiten (des Kontrollflusses)
- Dependency Inversion Principle (Inversion of Control)

**7**

- Mach es testbar
- Unit-Tests

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

73

## Prinzip einer einzigen Verantwortung Single Responsibility



University of Applied Sciences

Jedes Modul soll genau eine Verantwortung übernehmen

Jede Verantwortung soll genau einem Modul zugeordnet werden

Erhöhung der Wartbarkeit

Erhöhung der Wiederverwendbarkeit

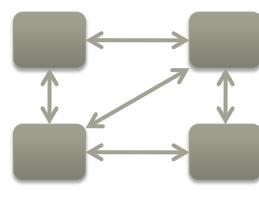
Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

74

## Prinzip einer einzigen Verantwortung Zu beachtende Regeln

### Regel 1:

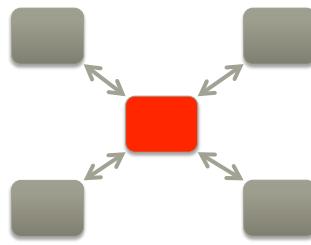
- Kohäsion maximieren
- Unabhängige Teile in Teilmodule zerlegen



Hoher Grad der Kopplung

### Regel 2:

- Kopplung minimieren
- Kopplung zwischen Modulen gering halten
- Einführen von Koordinatoren (neues Modul)

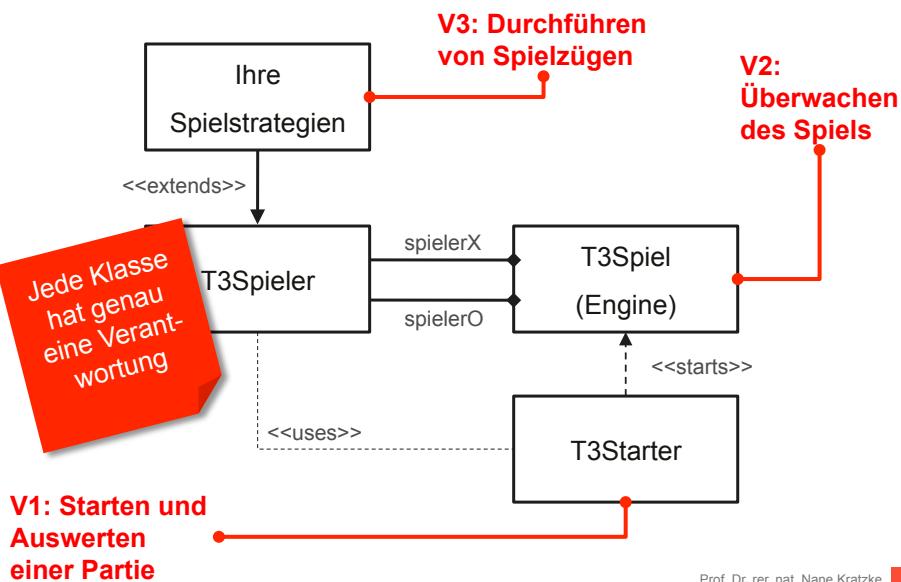


Reduzierter Grad der Kopplung

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

75

## Prinzip der einzigen Verantwortung am Bsp. Tic Tac Toe



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

76

## Prinzipien des objektorientierten Entwurfs



University of Applied Sciences

1

- Prinzip einer einzigen Verantwortung
- Single Responsibility

2

- Trennung der Anliegen
- Separation of Concerns

3

- Wiederholungen vermeiden
- Don't repeat yourself

4

- Offen für Erweiterungen, geschlossen für Änderungen
- Open-Closed-Principle

5

- Trennung von Schnittstelle und Implementierung
- Program to interfaces

6

- Umkehr der Abhängigkeiten (des Kontrollflusses)
- Dependency Inversion Principle (Inversion of Control)

7

- Mach es testbar
- Unit-Tests

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

77

## Trennung der Anliegen Separation of Concerns



University of Applied Sciences

### Ein Anliegen ist

- formulierbare Aufgabe
- zusammenhängend
- abgeschlossen
- und in verschiedenen Kontexten und Anwendungen nutzbar

### Beispiele

- Protokollierung von Aktionen, Fehlern, etc.
- Autorisierung von Benutzern
- Prüfung von Zugriffsrechten
- Transaktionsverarbeitung

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

78

## Trennung der Anliegen Probleme



University of Applied Sciences

In verschiedenen Kontexten und Anwendungen nutzbare Funktionalitäten

Lassen sich schwer in Modulen lokalisieren

Beispiel: Zugriffskontrolle

- Die muss in dem Modul angestoßen werden, aus dem der Zugriff heraus geschieht
- So etwas ist schwer zu lokalisieren

Mit OO alleine nur anteilig lösbar, hier hilft die Aspekt-orientierte Programmierung weiter (die in dieser VL nicht behandelt wird)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

79

## Prinzipien des objektorientierten Entwurfs



University of Applied Sciences

1

- Prinzip einer einzigen Verantwortung
- Single Responsibility

2

- Trennung der Anliegen
- Separation of Concerns

3

- Wiederholungen vermeiden
- Don't repeat yourself

4

- Offen für Erweiterungen, geschlossen für Änderungen
- Open-Closed-Principle

5

- Trennung von Schnittstelle und Implementierung
- Program to interfaces

6

- Umkehr der Abhängigkeiten (des Kontrollflusses)
- Dependency Inversion Principle (Inversion of Control)

7

- Mach es testbar
- Unit-Tests

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

80

## **Wiederholungen vermeiden** **Don't repeat yourself**



University of Applied Sciences

Eine identifizierbare Funktionalität eines Softwaresystems sollte innerhalb dieses Systems nur einmal implementiert sein.

 Erhöht die Wartbarkeit

 Reduziert die Fehleranfälligkeit

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

81

## **Wiederholungen vermeiden** **Zu beachtende Regeln**



University of Applied Sciences

### **Nutze Konstanten**

- Die lassen sich an einer Stelle im Quelltext ändern
- Es muss bei Änderungen nicht nach allen Vorkommen einer Konstante im Quelltext gesucht werden

### **Kopiere keinen Quelltext**

- Wenn Quelltext kopiert werden kann, um ein Problem zu lösen,
- frag dich, wie aus dem Quelltext eine parametrisierbare Methode gemacht werden kann.
- Ansonsten wird eine zukünftig geänderte Funktionalität nur an einer Stelle, anstatt an allen Kopievorkommen geändert.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

82

## Wiederholungen vermeiden am Beispiel Tic Tac Toe (I)



University of Applied Sciences

### Regel: Nutze Konstanten

```
public class T3Konstanten {  
  
    public final static char X = 'X';  
    public final static char O = 'O';  
    public final static char LEER = ' ';  
    public final static int BREITE = 3;  
  
    [...]  
  
}
```

In der Klasse T3Konstanten wurden Konstanten definiert, die genutzt werden sollten. Nicht X sondern T3Konstanten.X, usw. Auch die Breite wurde als Konstante genutzt. Möchte man Tic Tac Toe auf einem 4x4 Spielfeld spielen, dann lässt sich das durch Änderung an einer Stelle realisieren, sofern alle Routinen konsequent diese Konstanten nutzen.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

83

## Wiederholungen vermeiden am Beispiel Tic Tac Toe (II)



University of Applied Sciences

### Regel: Kopiere keine Quelltexte

```
public class T3VersierterSpieler  
extends T3Spieler {  
  
    protected List<T3Pos> leere_felder(char[][] feld);  
    protected List<T3Pos> gewinnfelder(char v, char[][] feld);  
  
}
```

Sie haben in der Übung aus der abstrakten Klasse T3Spieler die Klasse T3VersierterSpieler abgeleitet und in ihr die oben stehenden Methoden implementiert, die man für jede vernünftige, d.h. nicht triviale, Tic Tac Toe Strategie benötigt.

So konnte jeder von Ihnen eine oder mehrere Strategien auf Basis T3VersierterSpieler implementieren, ohne diese Grundfunktionalitäten jedesmal neu implementieren oder kopieren zu müssen.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

84

## Wiederholungen vermeiden am Beispiel Tic Tac Toe (III)



University of Applied Sciences

### Regel: Kopiere keine Quelltexte

```
public class T3Routinen
{
    public static char[][] deepclone(char[][] feld);
    public static boolean gewonnen(char v, char[][] feld);
}
```

In der T3 Engine wurden Routinen zentral in der Klasse T3Routinen definiert, die an mehreren Stellen eines Tic Tac Toe Spiels genutzt werden.

**deepclone** um ein Spielfeld zu kopieren.

**gewonnen** in ihren Strategieimplementierungen und in der Klasse T3Spiel im Rahmen der Spielüberwachung.

## Prinzipien des objektorientierten Entwurfs



University of Applied Sciences

1

- Prinzip einer einzigen Verantwortung
- Single Responsibility

2

- Trennung der Anliegen
- Separation of Concerns

3

- Wiederholungen vermeiden
- Don't repeat yourself

4

- Offen für Erweiterungen, geschlossen für Änderungen
- Open-Closed-Principle

5

- Trennung von Schnittstelle und Implementierung
- Program to interfaces

6

- Umkehr der Abhängigkeiten (des Kontrollflusses)
- Dependency Inversion Principle (Inversion of Control)

7

- Mach es testbar
- Unit-Tests

**Offen für Erweiterungen, geschlossen für Änderungen**  
Open-Closed Principle



University of Applied Sciences

**Ein Modul soll für Erweiterungen offen sein**

- Definierte Funktionalität soll angepasst/erweitert werden können.
- Die Erweiterung soll nur die Ergänzung beinhalten, keinesfalls Teile des Originalcodes.

Steigerung der Wiederverwendbarkeit

**Für Erweiterungen sind keine Änderungen am Modul erforderlich**

- Es sind keine Änderungen am Originalcode eines Modul für Erweiterungen erforderlich.
- Ungewünschte Erweiterungen des Moduls werden strukturell unterbunden.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

87

**Offen für Erweiterungen, geschlossen für Änderungen**  
Zu beachtende Regeln



University of Applied Sciences

**Definiere „Hooks“  
(Erweiterungspunkte)**

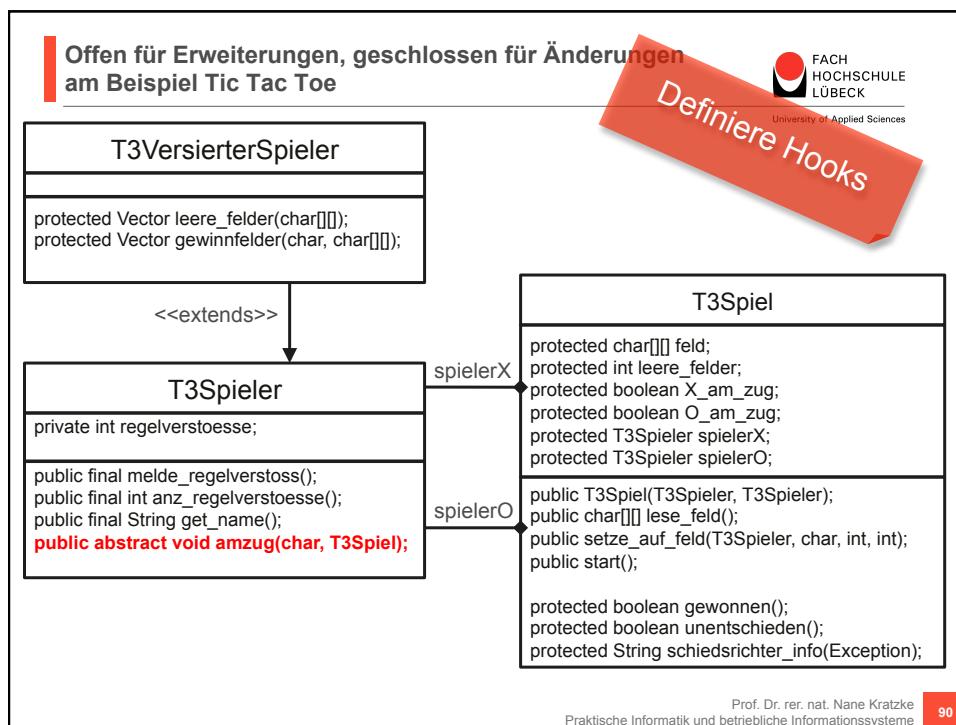
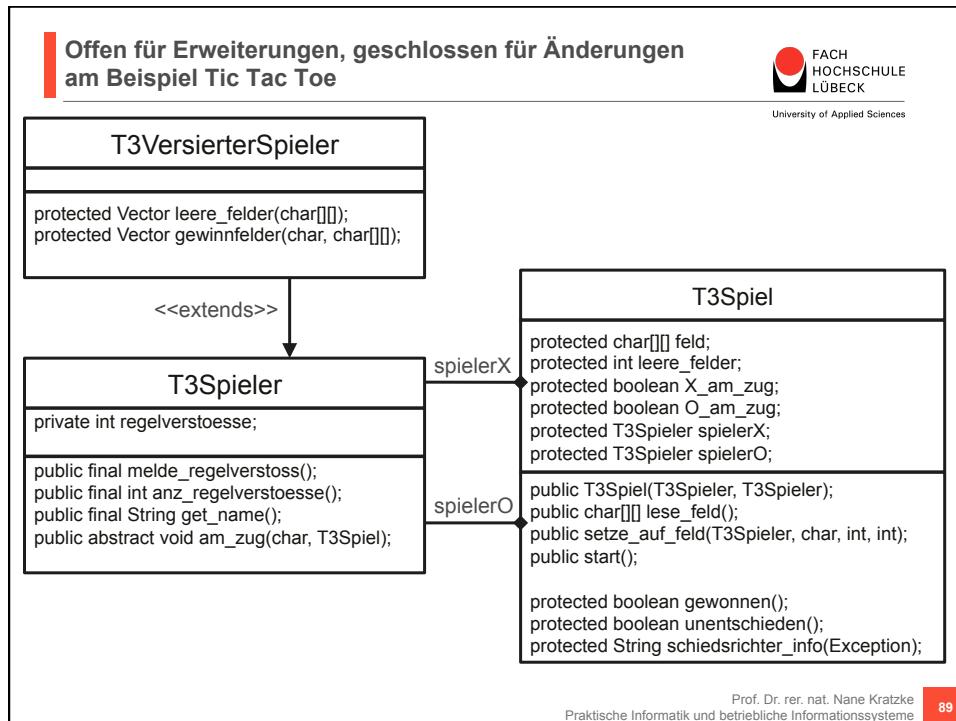
- Zu ändernde Funktionalität sollte durch Hooks definiert werden.
- An diese „Haken“ kann man dann die Erweiterungsfunktionalität hängen.
- Hooks sind zu dokumentieren
- da Hooks nicht am Quelltext zu erkennen sind. (Es gibt kein JAVA Schlüsselwort dafür)

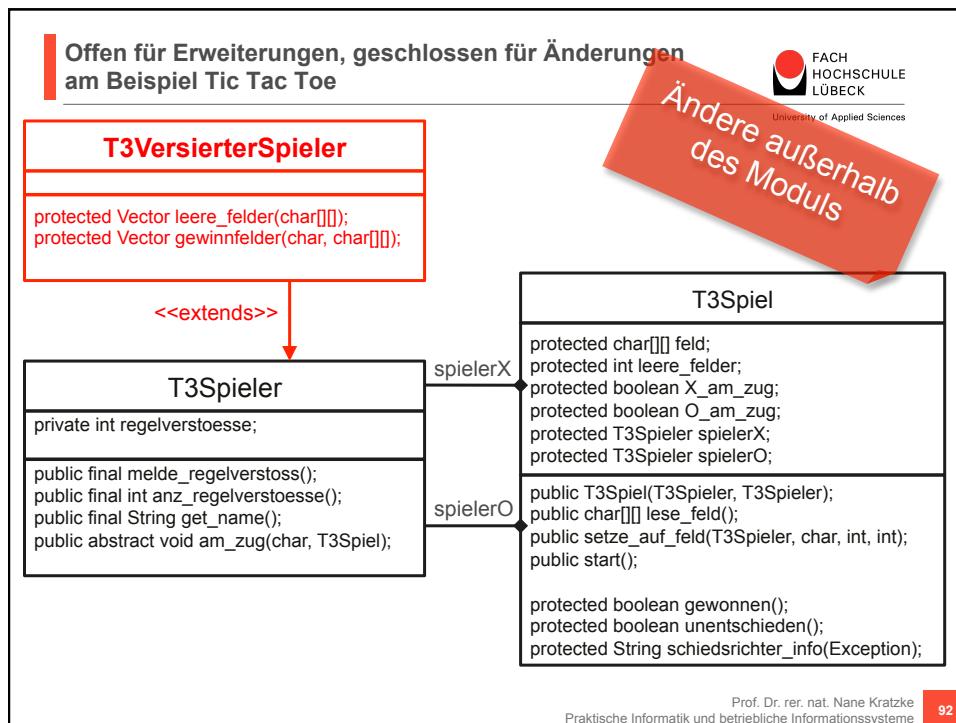
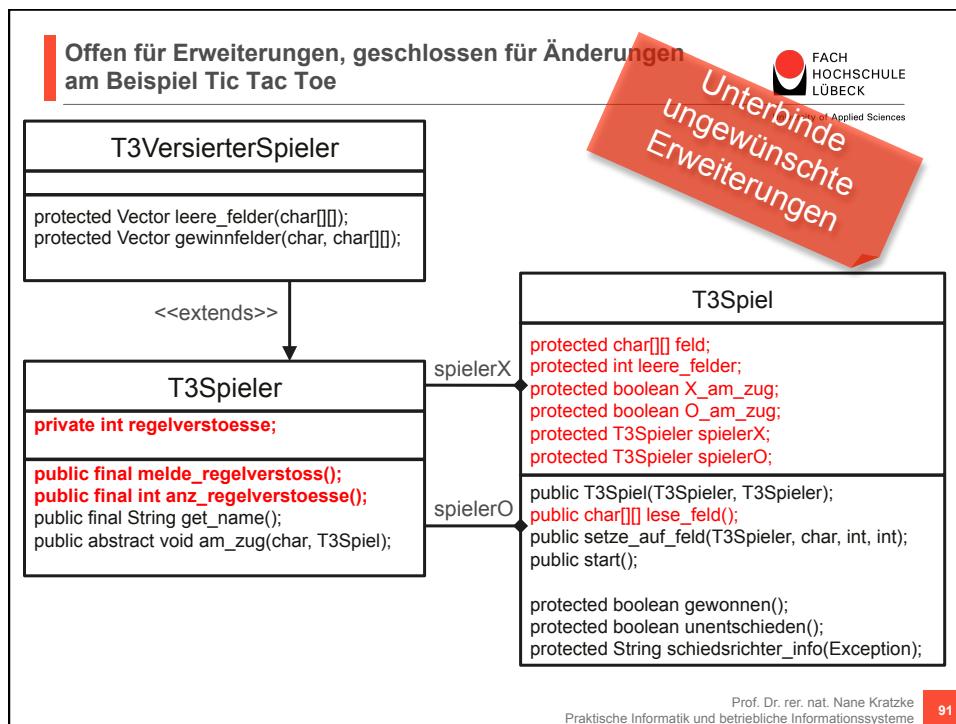
**Nutze im Modul Indirektionen**

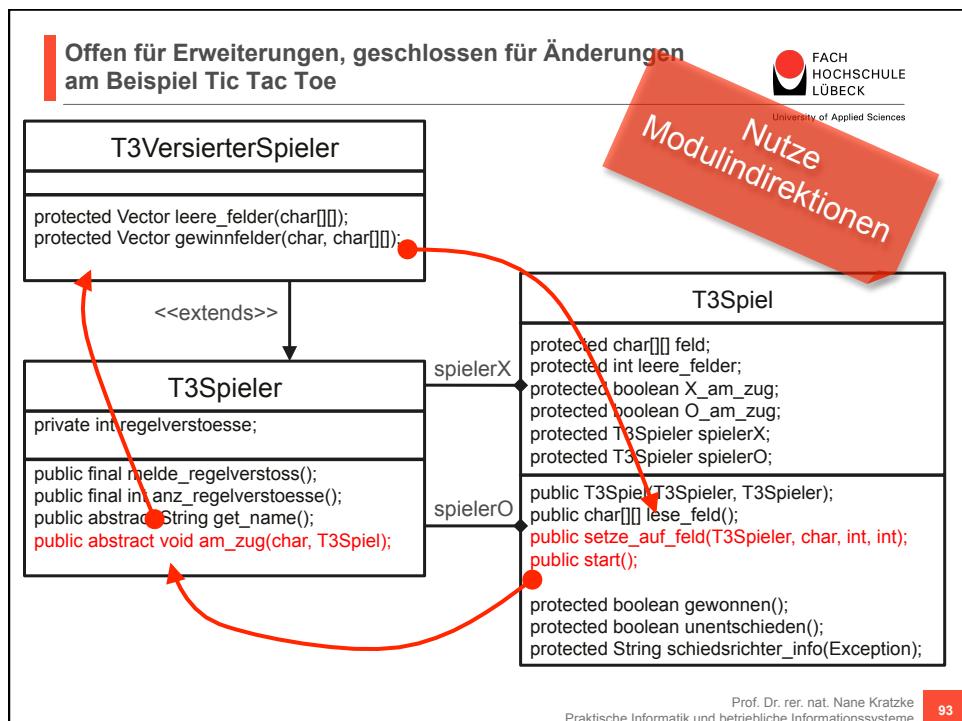
- Das erweiterbare Modul darf keine Varianten-spezifische Funktionalität nutzen
- Das Modul darf nur ihm bekannte „Hooks“ und Schnittstellen/(abstrakte) Klassen aufrufen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

88







## Trennung von Schnittstelle und Implementierung Program to Interfaces

Jede Abhängigkeit zwischen zwei Modulen explizit dokumentieren

Ein Modul sollte nur von Schnittstellen und deren Spezifikation abhängen

Ein Modul sollte niemals von nicht spezifizierten oder beeinflussbaren Seiteneffekten abhängen oder Implementierungen

Vermeidung „stiller“ Kopplungen

Erhöhung der Wiederverwendbarkeit

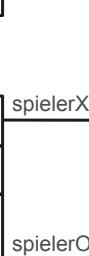
## Trennung von Schnittstelle und Implementierung am Beispiel Tic Tac Toe

T3Spiel muss bspw. nicht wissen, wie „hinter“ T3Spieler die Strategie implementiert wurde.

Sie hätten ja z.B. die T3 World Cup Series Datenbank abfragen können, um die erfolgreichsten Züge zu bestimmen.

```

T3VersierterSpieler
protected Vector leere_felder(char[][]);
protected Vector gewinnfelder(char, char[][]);
protected boolean gewonnen(char, char[][]);
<<extends>>
T3Spieler
private int regelverstoesse;
public final melde_regelverstoss();
public final int anz_regelverstoesse();
public final String get_name();
public abstract void am_zug(char, T3Spiel);
    
```



## Program to Interfaces



University of Applied Sciences

**Goldene Regel:** Programmiere nie nach dem **WIE** etwas implementiert wurde, sondern **WAS** spezifiziert wurde.

Programmiere nach dem Vertrag einer Methode

Wie nennt man die Lösungen, die entstehen, wenn man nach dem WIE implementiert?

Workaround

Workarounds sind fehlerumgehende Programmierlösungen, die gewählt werden, weil genutzte Module nicht ihrer Spezifikation entsprechen.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

97

## Prinzipien des objektorientierten Entwurfs



University of Applied Sciences

1

- Prinzip einer einzigen Verantwortung
- Single Responsibility

2

- Trennung der Anliegen
- Separation of Concerns

3

- Wiederholungen vermeiden
- Don't repeat yourself

4

- Offen für Erweiterungen, geschlossen für Änderungen
- Open-Closed-Principle

5

- Trennung von Schnittstelle und Implementierung
- Program to interfaces

6

- Umkehr der Abhängigkeiten (des Kontrollflusses)
- Dependency Inversion Principle (Inversion of Control)

7

- Mach es testbar
- Unit-Tests

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

98

## Umkehr der Abhangigkeiten Dependency Inversion Principle



University of Applied Sciences

### Umkehr der Abhangigkeiten

- Ein Entwurf soll sich auf Abstraktionen stutzen.
- Er soll sich nicht auf Spezialisierungen stutzen.

### Umkehr des Kontrollflusses

- Ein spezifisches Modul
- wird von einem mehrfach verwendbaren Modul aufgerufen.

Erhohung der  
Austauschbarkeit

Reduzierung der Kopplung

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

99

## Regel:

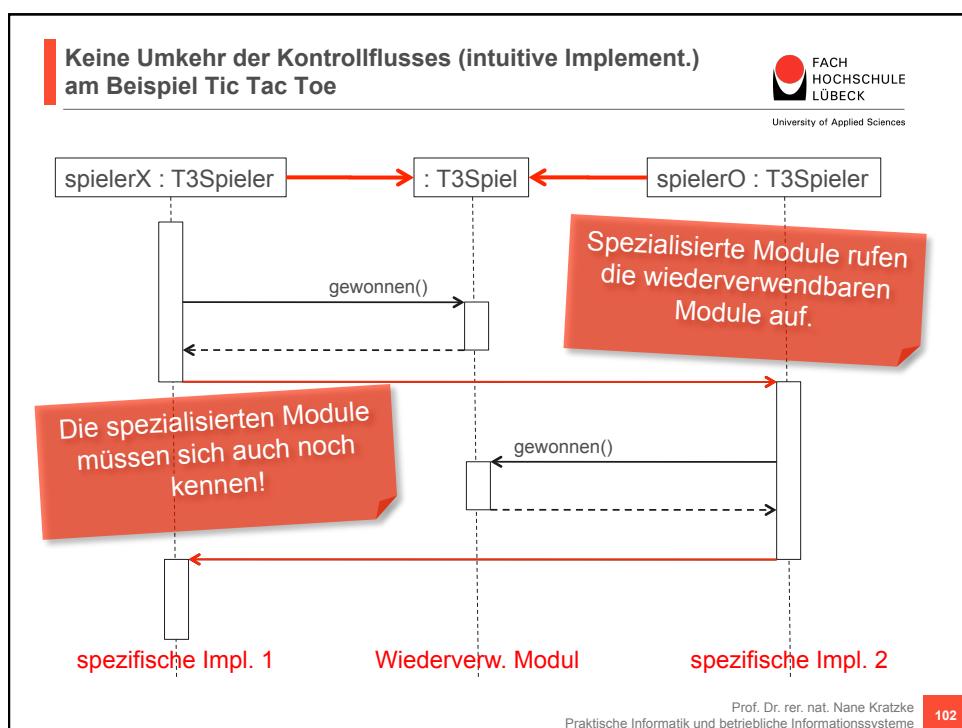
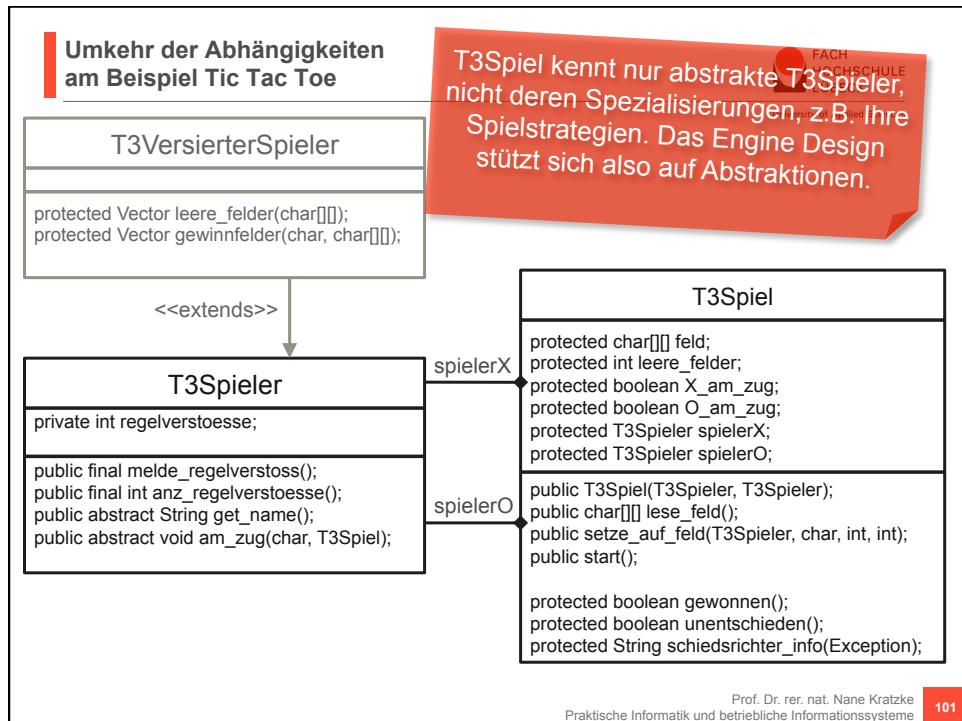


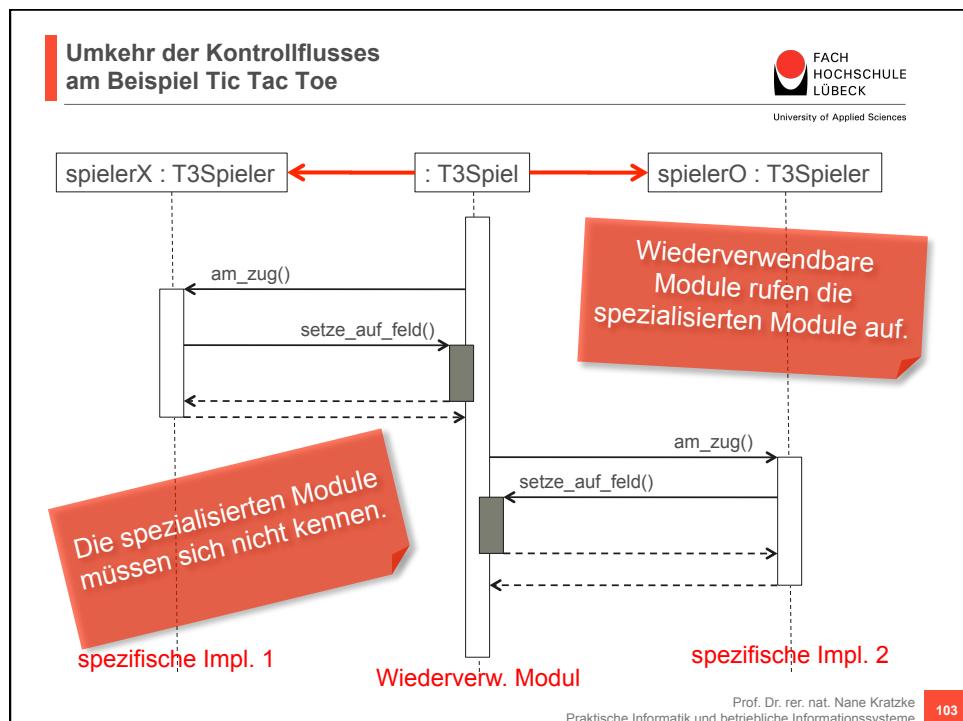
University of Applied Sciences

- Erganzungen werden von Kernmodulen initial aufgerufen!
- Niemals umgekehrt!

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

100





## Mach es testbar Unit Tests



University of Applied Sciences

### Unit-Tests

- sind Testprogramme
- die die Korrektheit von SW-prüfen.

### Unit-Tests

- sind automatisierbar
- und helfen nach Änderungen Fehler schneller zu erkennen

 Erhöhung der Korrektheit

 Erhöhung der Wartbarkeit

 Erhöhung der Testbarkeit

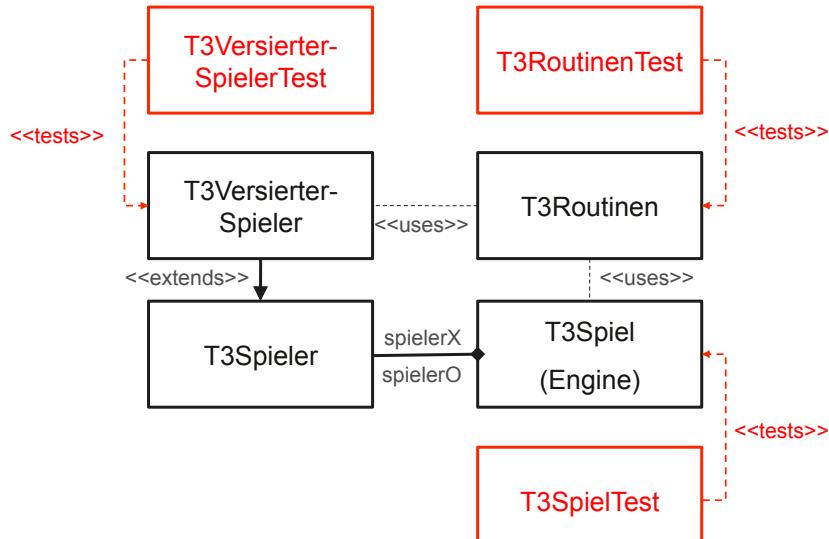
## Regel:



University of Applied Sciences

- Nutze UNIT Tests !
- UNIT Tests sind Freunde, kein Aufwand

## Unit Tests testen Komponenten Bsp. Tic Tac Toe



## Beispiel eines Unit-Tests in Tic Tac Toe

```

public class T3SpieltTest {

    public Katastrophenspieler k = new Katastrophenspieler("K");
    public ZufallsSpieler z = new ZufallsSpieler("Z");

    @Test
    public void testT3Spiel_Ablauf() {

        // Teste ob katastrophale Programmierungen den Absturz bringen
        Assert.assertEquals(-20,
                            T3Starter.starte_partie(10,k,z,false));
        Assert.assertEquals(20,
                            k.anz_regelverstoesse());
        // [...]
    }
}
  
```

### Bsp. Units Tests in Tic Tac Toe

#### Automatisierte Unit Tests in einer IDE

The screenshot shows the Eclipse IDE interface with several windows open:

- Java - GOOP-TicTacToe/src/de/fhi/goop/tictactoe/engine/T3spielTest.java - Eclipse - /Users/Nane/Documents/workspace**: The main Java file containing unit test code for a Tic Tac Toe engine.
- Outline View**: Shows the class structure and methods.
- Run View**: Displays the test results: 4/4 runs, 0 errors, 0 failures.
- Failure Trace**: Shows the failure details for a specific test case, indicating a catastrophe due to a strategy error.
- Problems View**: Lists any errors or warnings in the project.
- Javadoc View**, **Declaration View**, **Console View**: Standard Eclipse developer tools.

**Annotations:**

- A blue line points from the text "Automatisierter Start, Auswertung und Überblick von Tests in Eclipse" to the Run View window.
- The text "Run as JUnit Testcase" is located below the Run View window.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme | 109

### Zusammenfassung: Prinzipien des (OO) Entwurfs

The diagram illustrates the Seven Principles of Design (SOLID + DRY + KISS) as a series of seven green boxes numbered 1 to 7, each pointing to the right:

- Prinzip einer einzigen Verantwortung  
• Single Responsibility
- Trennung der Anliegen  
• Separation of Concerns
- Wiederholungen vermeiden  
• Don't repeat yourself
- Offen für Erweiterungen, geschlossen für Änderungen  
• Open-Closed-Principle
- Trennung von Schnittstelle und Implementierung  
• Program to interfaces
- Umkehr der Abhängigkeiten (des Kontrollflusses)  
• Dependency Inversion Principle (Inversion of Control)
- Mach es testbar  
• Unit-Tests

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme | 110