

Vorlesung

Programmieren I und II

Unit 10

Objektorientierter Entwurf und (objektorientierte) Designprinzipien



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

*Praktische Informatik und
betriebliche Informationssysteme*

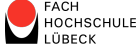
- Raum: 17-0.10
- Tel.: 0451 300 5549
- Email: kratzke@fh-luebeck.de



@NaneKratzke

Updates der Handouts auch über Twitter **#prog_inf** und **#prog_itd**

Units



FACH
HOCHSCHULE
LÜBECK

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 und UML

Unit 7
 Konzepte objektorientierter Programmiersprachen

Unit 8
 Testen (objektorientierter) Programme

Unit 9
 Generische Datentypen

Unit 10
 Objektorientierter Entwurf und objektorientierte Designprinzipien

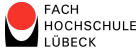
Unit 11
 Graphical User Interfaces

Unit 12
 Multithread Programmierung

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

3

Abgedeckte Ziele dieser UNIT



FACH
HOCHSCHULE
LÜBECK

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

OO Entwurf

- Beispiel Tic Tac Toe für erweiterbare Software
- Einfaches Vorgehensmodell

OO Entwurfsprinzipien

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

Zum Nachlesen ...



Kapitel 4

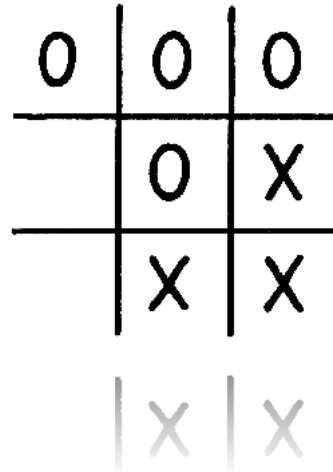
Die Struktur objektorientierter Software

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

Struktur objektorientierter Software am Beispiel des Spiels Tic Tac Toe

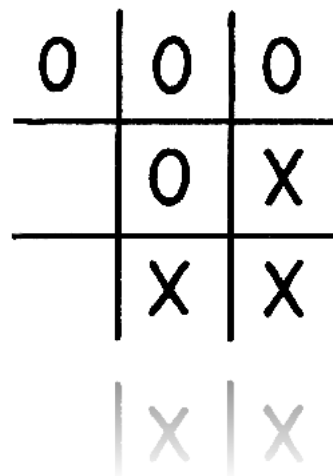
Klassisches,
Zwei Personen
Strategiespiel

Bereits im 12.
Jh. v. Chr.
bekannt



Tic Tac Toe Spielverlauf und Regeln

- 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.**

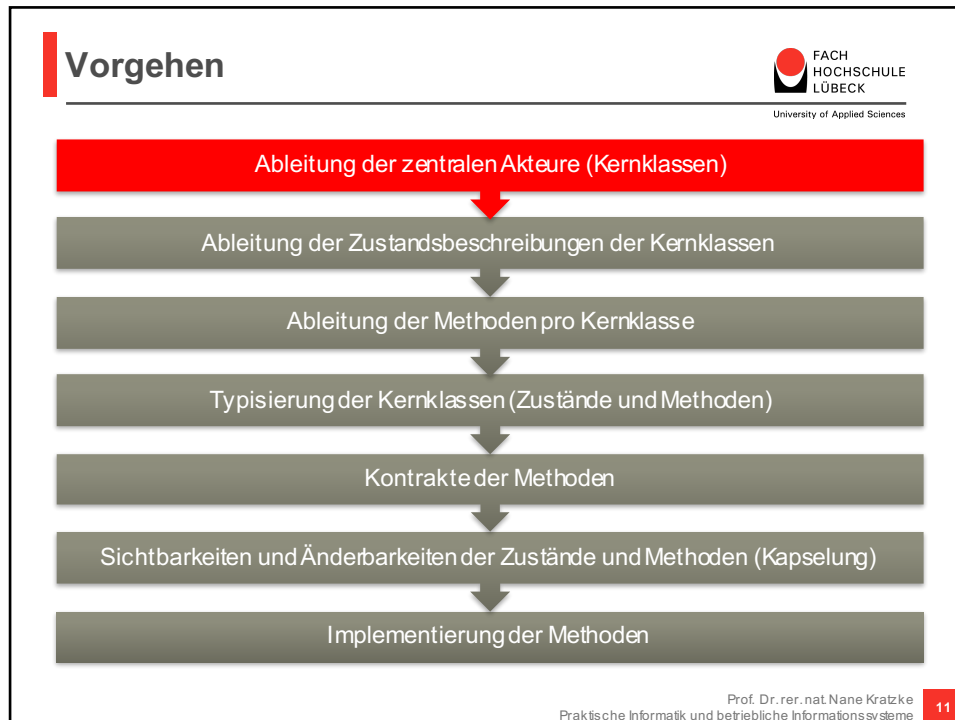


Struktur objektorientierter Software

Klasse	Objekt	Objektorientierte Abläufe
• Objekte haben Gemeinsamkeiten • Modellierungsmittel • Klassen sind Datentypen • Sichtbarkeiten	• Konstruktoren/Destruktoren • Zustand • Verhalten • Ausprägungen von Klassen	• Interaktion zwischen Objekten • Kontrakte und Exceptions

Tic Tac Toe Requirements

- 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.



Objekte in Tic Tac Toe

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

FACH HOCHSCHULE LÜBECK
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?

T3Spieler
Zustand des Spielers (Datenfelder)
Methoden des Spielers (Verhalten)

T3Spiel
Zustand des Spiels (Datenfelder)
Methoden des Spiels (Verhalten)

Vorgehen



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
Methoden des Spiels (Verhalten)

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

15

Tic Tac Toe Requirements

Ableitung der Zustandsbeschreibung für T3Spieler



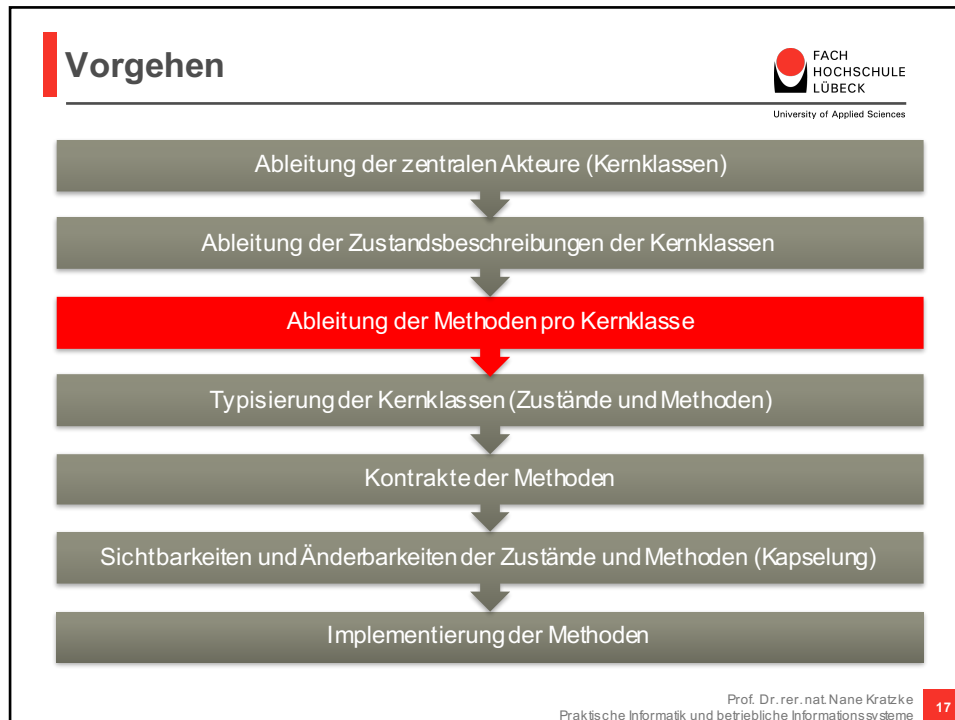
University of Applied Sciences

- 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
Methoden des Spielers (Verhalten)

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

16



Tic Tac Toe Requirements

Ableitung der Methoden für T3Spiel

FACH HOCHSCHULE LÜBECK
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

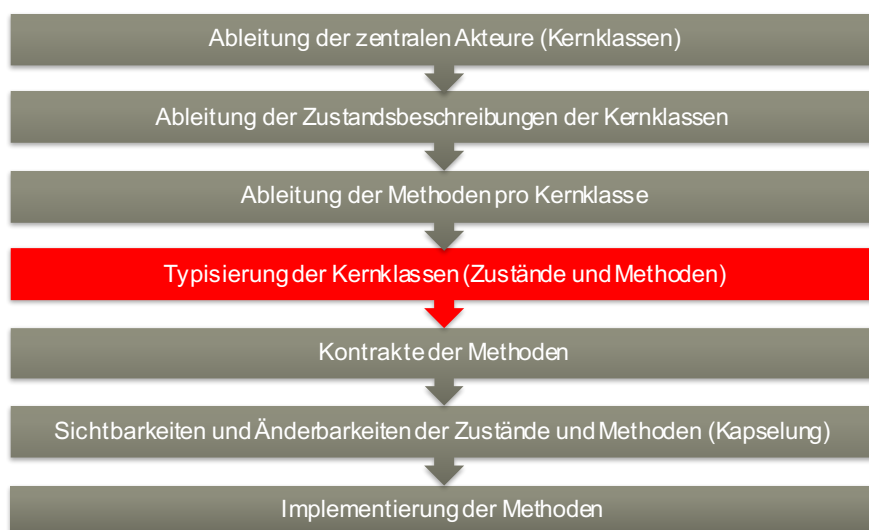
18

Tic Tac Toe Requirements Ableitung der Methoden für T3Spieler

- 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_regelverstoss

Vorgehen

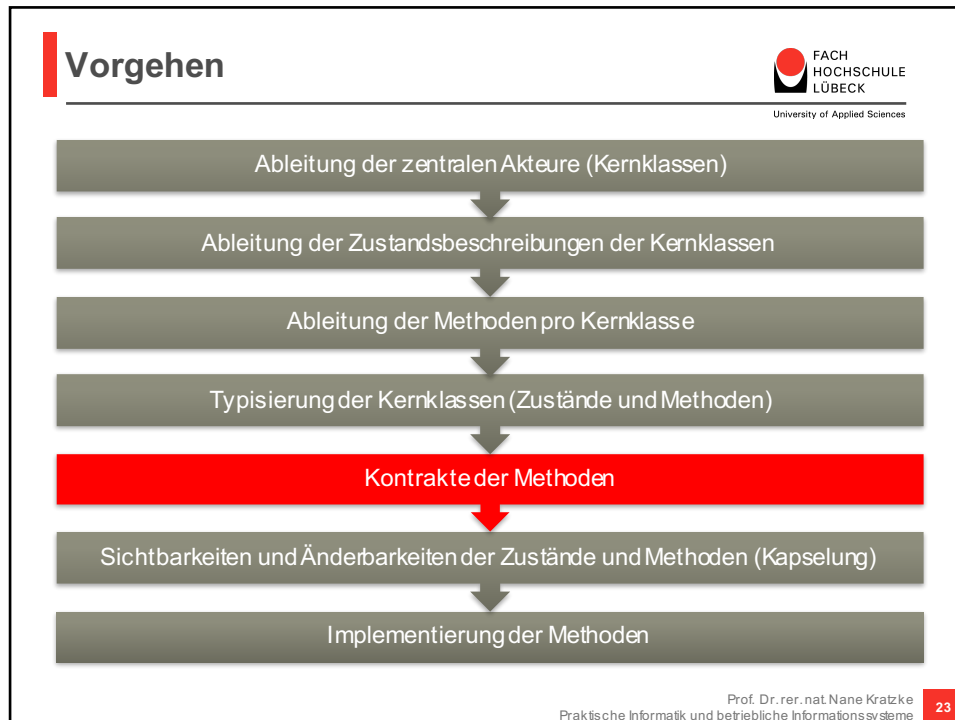


Typisierung der Klassen

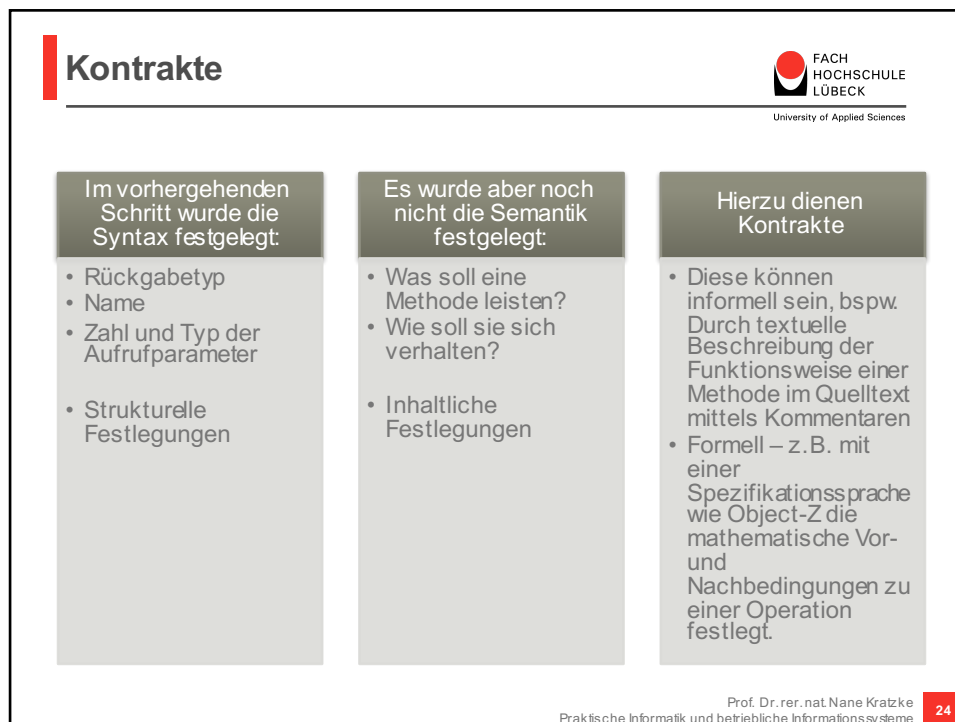
T3Spiel	T3Spiel
feld anz_leere_felder X_am_zug O_am_zug spielerX spielerO	<pre>char[][] feld = new char[3][3]; int anz_leere_felder = 9; boolean X_am_zug = true; boolean O_am_zug = false; T3Spieler spielerX T3Spieler spielerO</pre>
leite_spiel gewonnen unentschieden setze_auf_feld (inkl. Prüfungen) schiedsrichter_information	<pre>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)</pre>

Typisierung der Klassen

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



23



24

Kontrakte der T3Spiel Methoden

0	0	0
	0	X
	X	X

	x	x
--	---	---

Kontrakte der T3Spiel-Methoden Konstruktor

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 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)

Kontrakte der T3Spiel-Methoden

Methode **leite_spiel**



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 **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

27

Kontrakte der T3Spiel-Methoden

Methode **gewonnen**



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 **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

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

28

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

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

29

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

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

30

Kontrakte der T3Spiel-Methoden

Methode **schiedsrichter_information**



FACH
HOCHSCHULE
LÜBECK

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 **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

Implementierung der T3Spiel-Methoden

Methode **schiedsrichter_information**



FACH
HOCHSCHULE
LÜBECK

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

32

Kontrakte der T3Spieler Methoden



Kontrakte der T3Spieler-Methoden Konstruktor

T3Spieler

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

T3Spieler(String) // Konstruktor

```
void am_zug(char, T3Spieler) throws Exception  
void melde_regelverstoess()
```

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")

Kontrakte der T3Spieler-Methoden

Methode **am_zug**



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

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

35

Kontrakte der T3Spieler-Methoden

Methode **melde_regelverstoess**



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

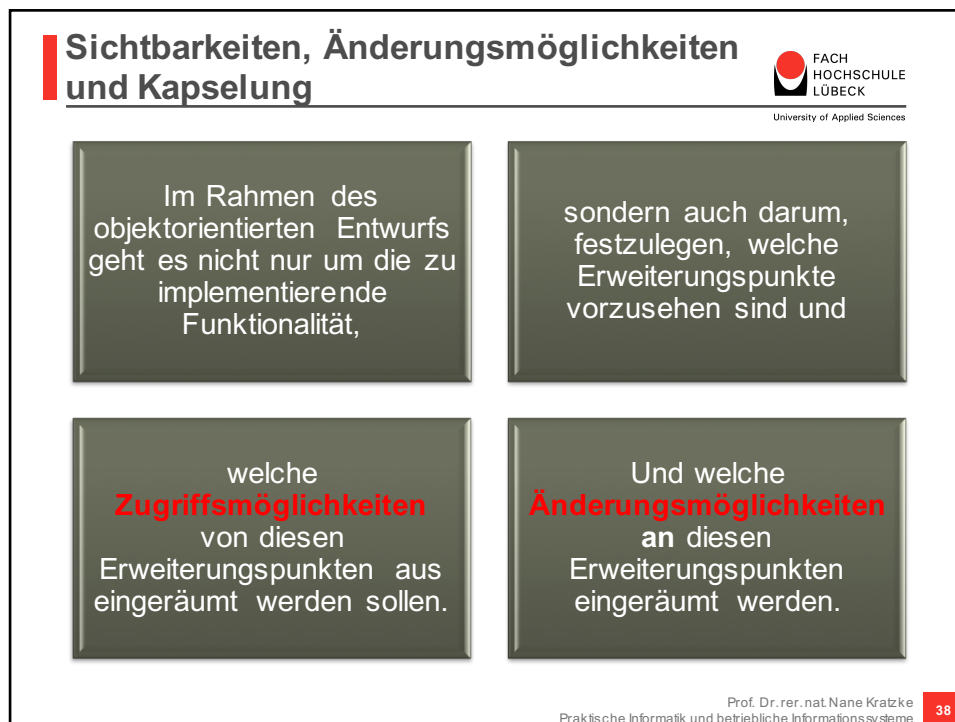
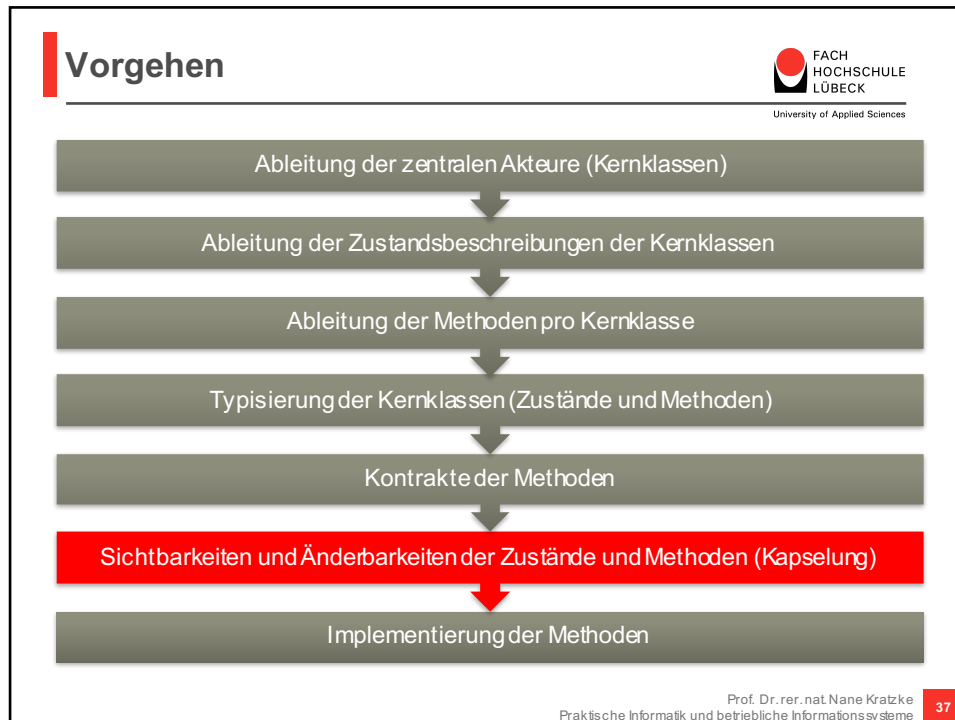
Informeller Kontrakt für Methode **melde_regelverstoess**

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

36



Einschränkungen von Sichtbarkeiten und Änderungen (Kapselung)



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

Final	Abstract	Kein Qualifier
• Methode darf nicht mehr überladen werden.	• Methode muss noch implementiert werden.	• Methode kann, muss aber nicht überladen werden.

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

Private	Protected	Public
• Zugriff nur aus der definierenden Klasse heraus möglich	• Zugriff nur aus demselben Paket • oder allen abgeleiteten Klassen möglich	• Zugriff aus allen Paketen • Und allen Klassen möglich

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

39

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	Von T3Spieler abgeleitete Strategien müssen
• Den Zustand des Spiels unkontrolliert ändern • Das eigene Regelverstoßkonto manipulieren können	• den Belegungszustand des Feldes auslesen können • den Belegungszustand des Feldes kontrolliert durch Ihren Spielzug ändern können • minimalen Zugriff auf den Spielzustand haben.

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

40

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



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

41

Lesenden Zugriff auf Attribute mittels „getter“-Methoden realisieren

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)
public	char[][] lese_feld()

**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

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_regelverstoess()

Keine Manipulation des
Regelverstoßkontos durch
Erweiterungen

Implementierung der
eigenen Spielstrategie
erforderlich

Lesenden Zugriff auf
Regelverstoßkonto



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

43

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	void melde_regelverstoess()
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

44

Vorgehen



Implementierung der T3Spieler Methoden



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

47

Implementierung der T3Spieler-Methoden Methode **am_zug**



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

48

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 erbbende 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

49

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

50

Implementierung der T3Spiel Methoden

0	0	0
	0	X
	X	X

	X		X	
--	---	--	---	--

Implementierung der T3Spiel-Methoden Konstruktor

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;  
    }  
  
    ...  
}
```

Implementierung der T3Spiel-Methoden Methode **leite_spiel**

Informeller Kontrakt für Methode **leite_spiel**

Die Methode startet ein Spiel zwischen `spielerX` und `spielerO`.

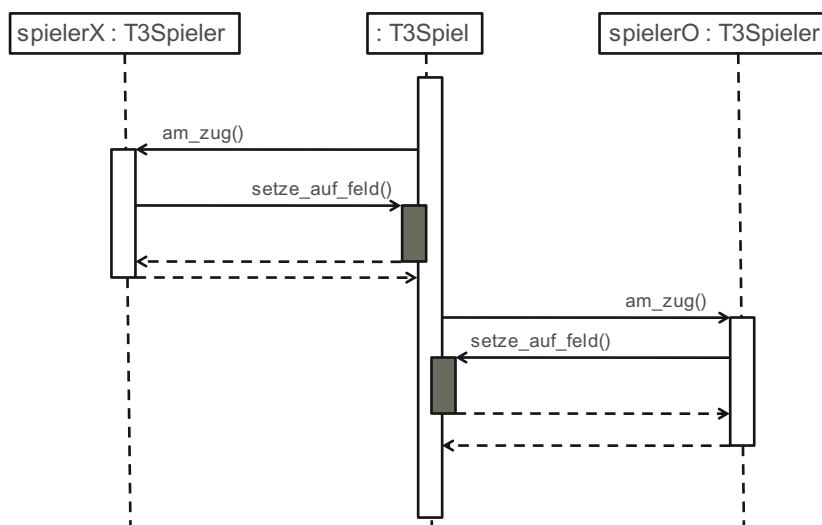
Die Methode liefert folgende Rückgaben:

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

Begehen `spielerX` oder `spielerO` Regelverstöße wird deren Methode `melde_regelverstoss` aufgerufen.



Implementierung der T3Spiel-Methoden Methode **leite_spiel**



Implementierung der T3Spiel-Methoden

Methode **leite_partie**



```
public class T3Spiel {
    ...
    public char leite_partie() {

        while (this.anz_leere_felder > 0) {
            // Spieler X am Zug
            try {
                int nochfrei = this.anz_leere_felder;
                spielerX.am_zug(T3Konstanten.X, this);
                if (gewonnen(T3Konstanten.X)) return T3Konstanten.X;
                if (nochfrei - 1 != this.anz_leere_felder) throw new Exception();
            }
            catch (Exception ex) { // X hat sich nicht an die Regeln gehalten
                spielerX.melde_regelverstoss();
                System.out.println(schiedsrichter_information(T3Konstanten.X, ex));
                return T3Konstanten.O;
            }

            if (unentschieden()) return T3Konstanten.LEER;

            // Analog für Spieler O
        }
    }
}
```

Spiel unter
Aufsicht
Regelver.

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

55

Implementierung der T3Spiel-Methoden

Methode **gewonnen**

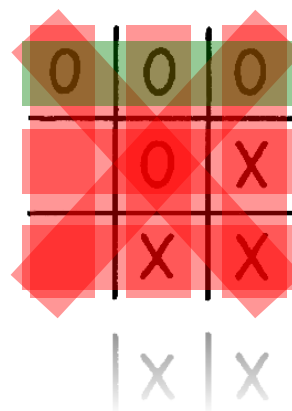


Informeller Kontrakt für Methode gewonnen

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



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

56

Implementierung der T3Spiel-Methoden

Methode **gewonnen**



University of Applied Sciences

```
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;
    }
    ...
}
```

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

57

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

58

Implementierung der T3Spiel Methoden

Methode `lese_feld`

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();  
    }  
    ...  
}
```

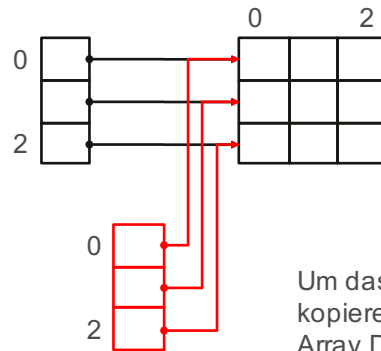
Mehrdimensionale Arrays kopieren (I)

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).

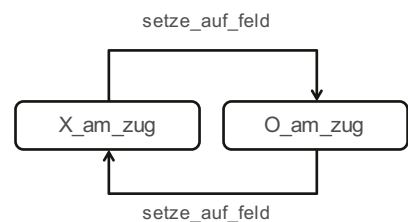
Implementierung der T3Spiel-Methoden Methode `setze_auf_feld`

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



0	0	0
	0	X
X	X	X

anz_leere_felder--

Implementierung der T3Spiel-Methoden

Methode **setze_auf_feld**



University of Applied Sciences

```
public class TSpiel {
    ...
    public void setze_auf_feld(T3Spieler s, char v, int x, int y)
        throws Exception {

        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();

        this.feld[x][y] = v;

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

Regelkonformer Zug?

Status

Belege das Feld

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

63

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

64

Implementierung der T3Spiel-Methoden

Methode **schiedsrichter_information**



```
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);  
}
```

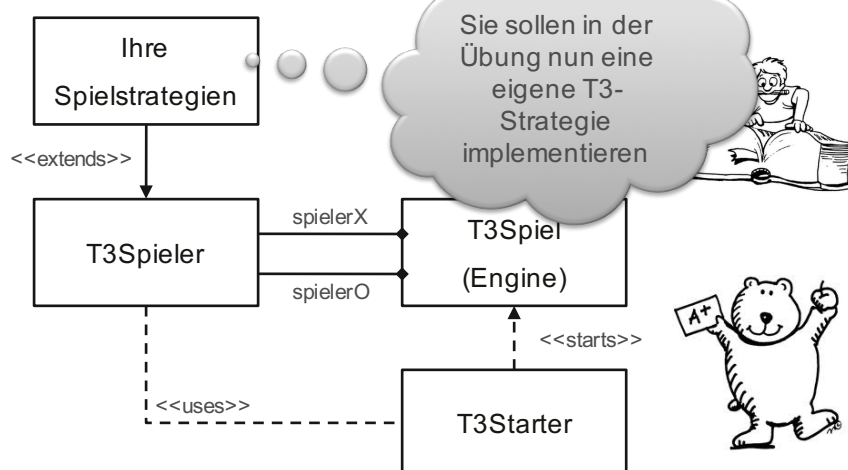
Ausgabe
Fehlermeldung

Tic Tac Toe Feld in
String wandeln

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

65

Zusammenfassung Architektur des T3-Frameworks



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

66

Themen dieser Unit

FACH HOCHSCHULE LÜBECK
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

67

Zum Nachlesen ...

FACH HOCHSCHULE LÜBECK
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

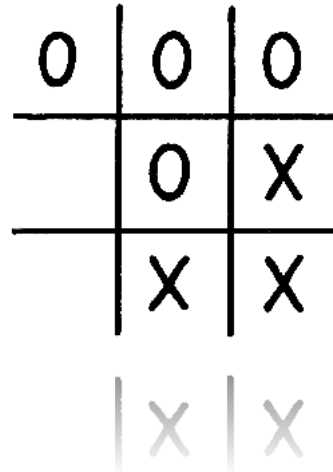
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

68

Struktur objektorientierter Software am Beispiel des Spiels Tic Tac Toe

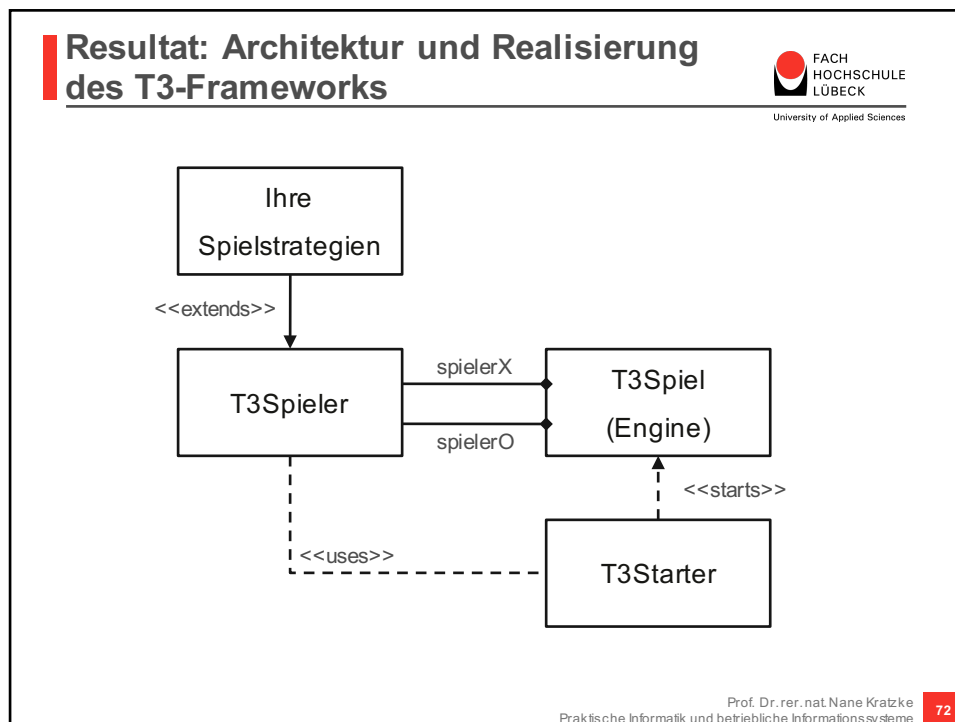
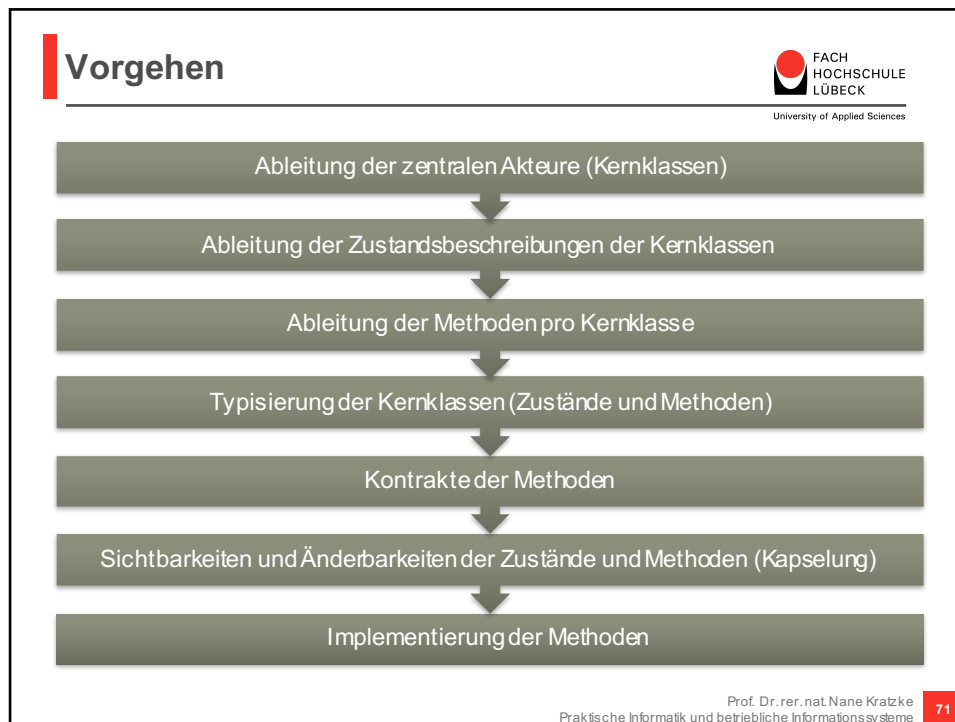
Klassisches,
Zwei Personen
Strategiespiel

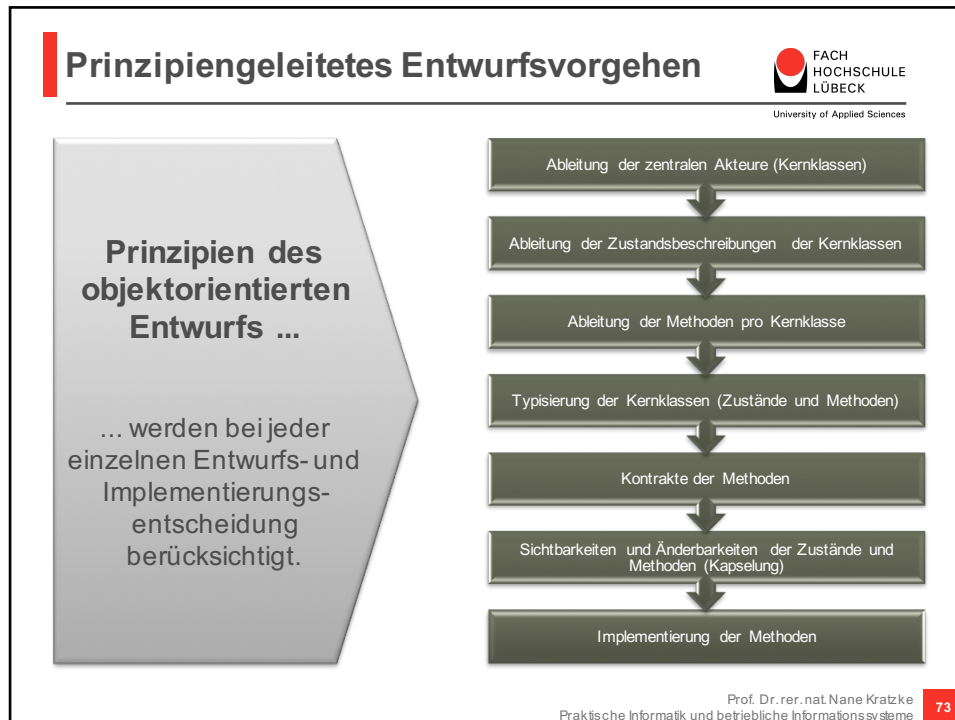
Bereits im 12.
Jh. v. Chr.
bekannt



Tic Tac Toe Requirements

- 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.





Prinzip einer einzigen Verantwortung Single Responsibility

Jedes Modul soll
genau eine
Verantwortung
übernehmen

Jede Verantwortung
soll genau einem
Modul zugeordnet
werden

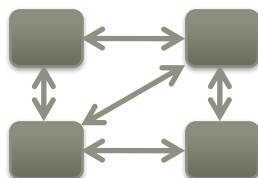
Erhöhung der Wartbarkeit

Erhöhung der Wiederverwendbarkeit

Prinzip einer einzigen Verantwortung Zu beachtende Regeln

Regel 1:

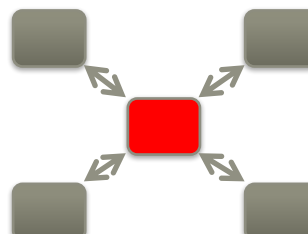
- Kohäsion maximieren
- Unabhängige Teile in Teilmodule zerlegen



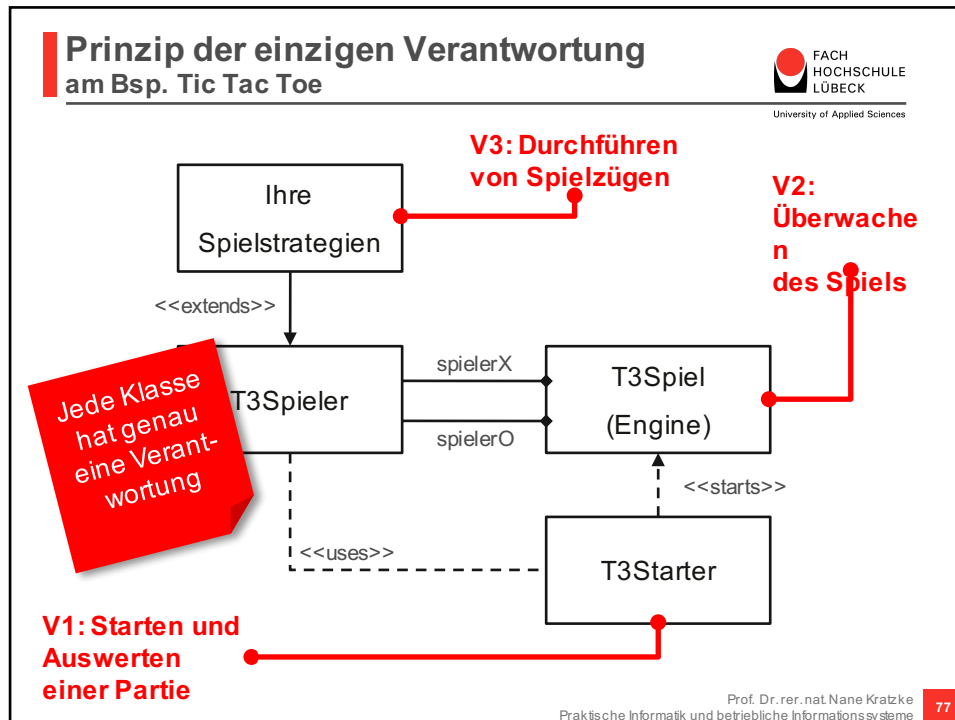
Hoher Grad der Kopplung

Regel 2:

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



Reduzierter Grad der Kopplung



Trennung der Anliegen Separation of Concerns

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

Trennung der Anliegen Probleme

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)

Prinzipien des objektorientierten Entwurfs



- 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

81

Wiederholungen vermeiden Don't repeat yourself



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

82

Wiederholungen vermeiden Zu beachtende Regeln

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 Kopiervorkommen geändert.

Wiederholungen vermeiden am Beispiel Tic Tac Toe (I)

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.

Wiederholungen vermeiden am Beispiel Tic Tac Toe (II)

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.

Wiederholungen vermeiden am Beispiel Tic Tac Toe (III)

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



- 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

87

Offen für Erweiterungen, geschlossen für Änderungen Open-Closed Principle



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

88

Offen für Erweiterungen, geschlossen für Änderungen
Zu beachtende Regeln


FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

**Definiere „Hooks“
(Erweiterungspunkte)**

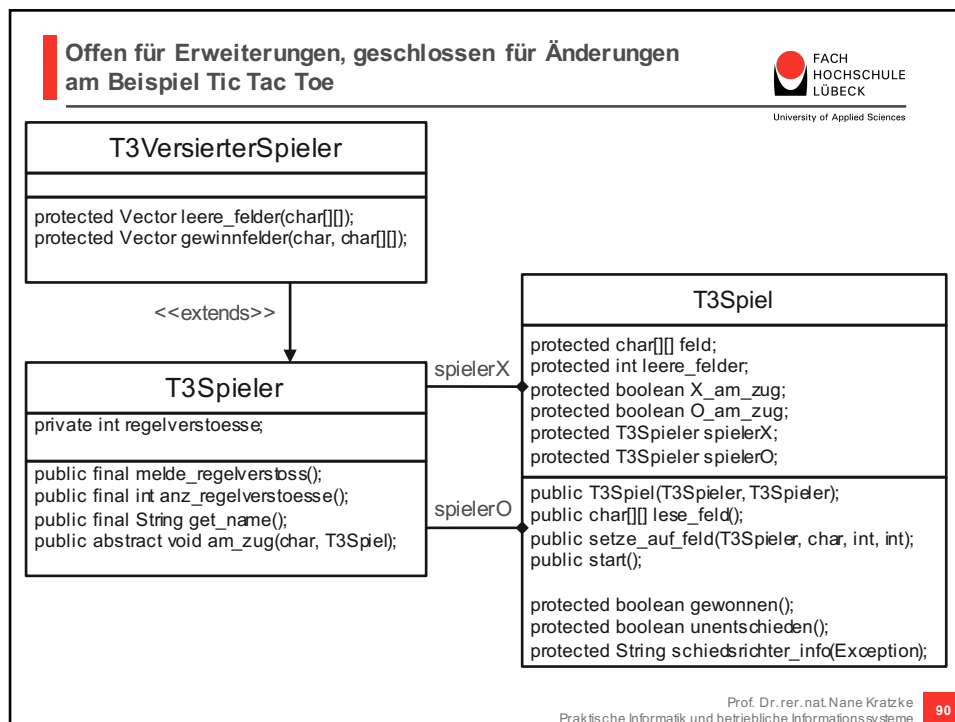
- Zu ändernde Funktionalität sollte durch Hooks definiert werden.
- An diese „Haken“ kann man dann die Erweiterungs-funktionalität hängen.
- Hooks sind zu dokumentieren
- da Hooks nicht am Quelltext zu erkennen sind. (Es gibt kein JAVA Schlüsselwort dafür)

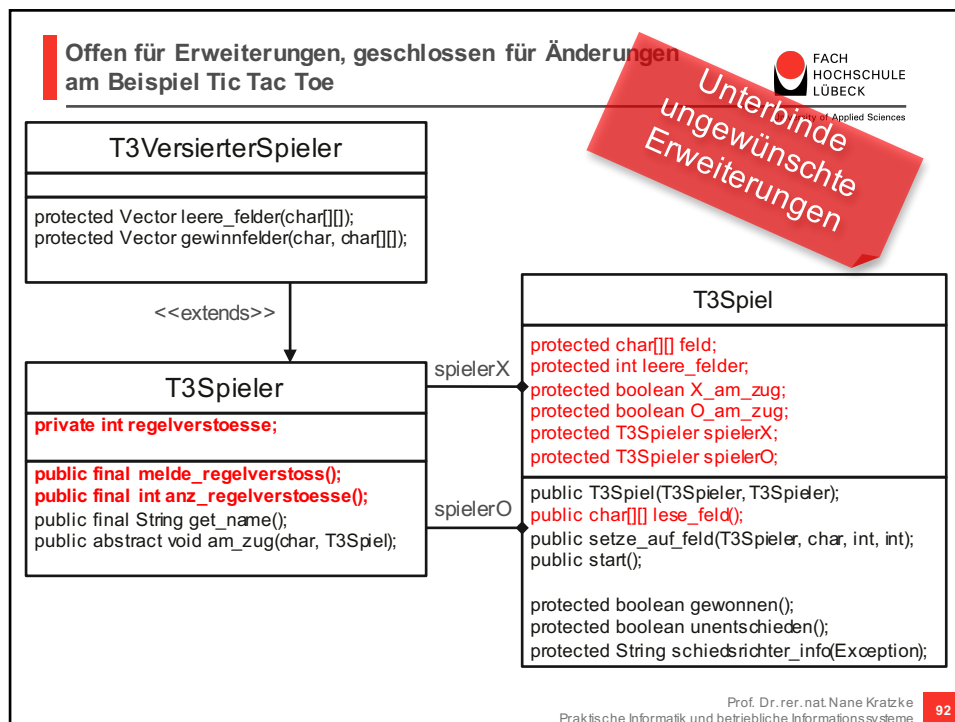
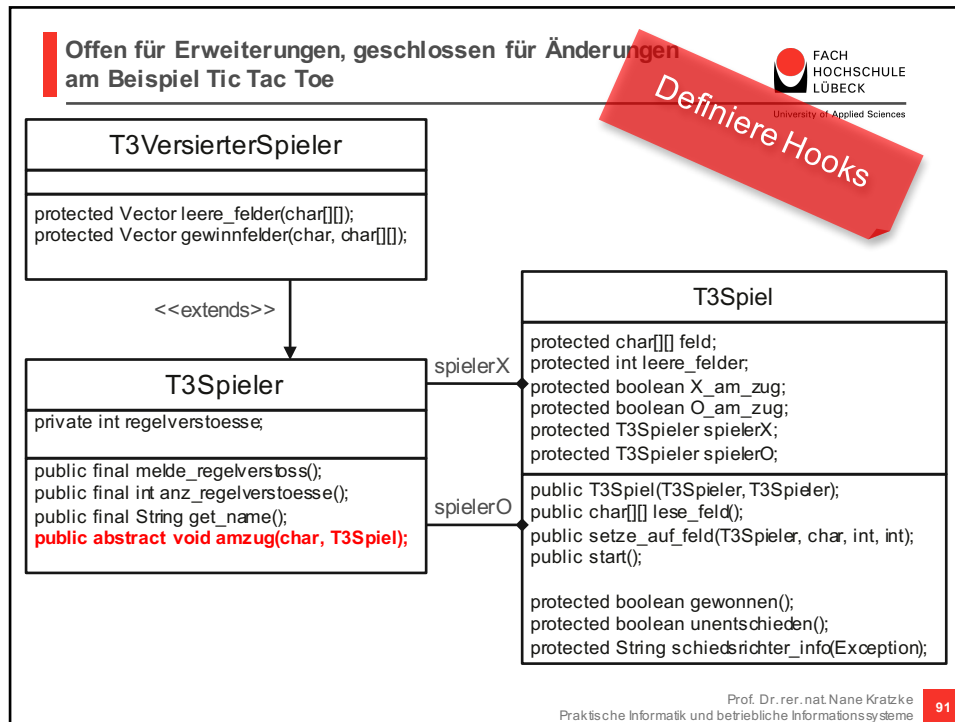
**Nutze im Modul
Indirektionen**

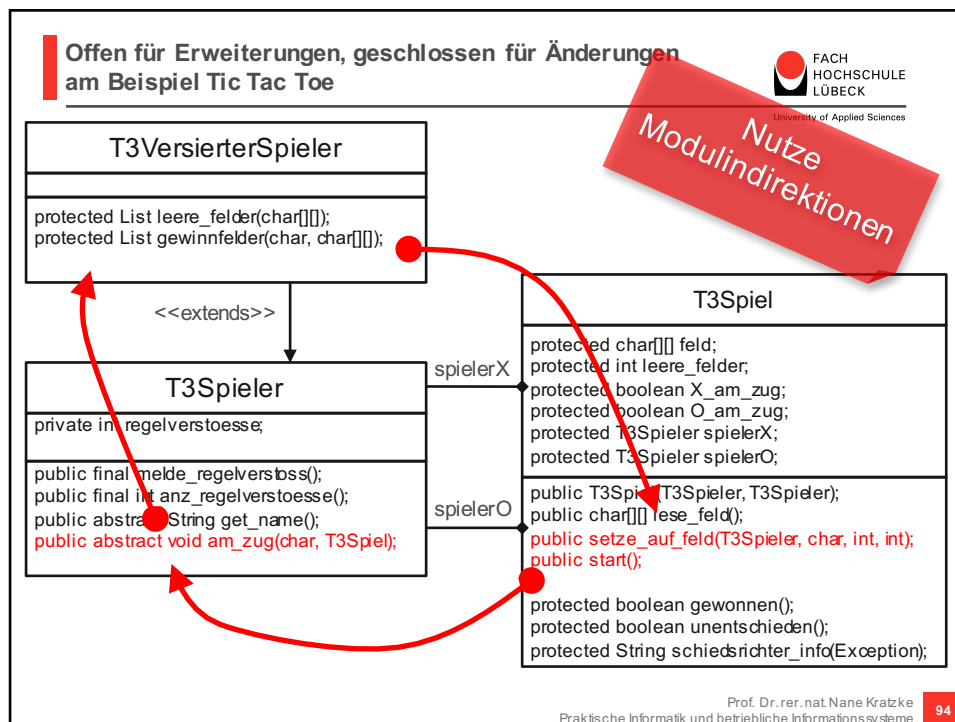
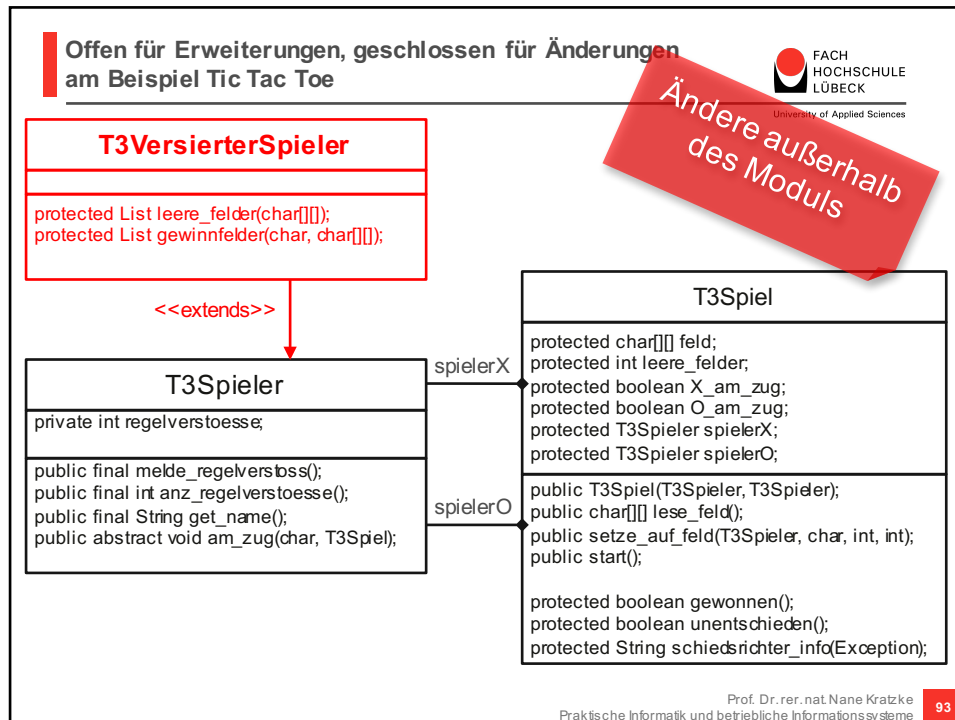
- Das erweiterbare Modul darf keine Variantenspezifische Funktionalität nutzen
- Das Modul darf nur ihm bekannte „Hooks“ und Schnittstellen/(abstrakte) Klassen aufrufen

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

89







Prinzipien des objektorientierten Entwurfs



- 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

95

Trennung von Schnittstelle und Implementierung Program to Interfaces

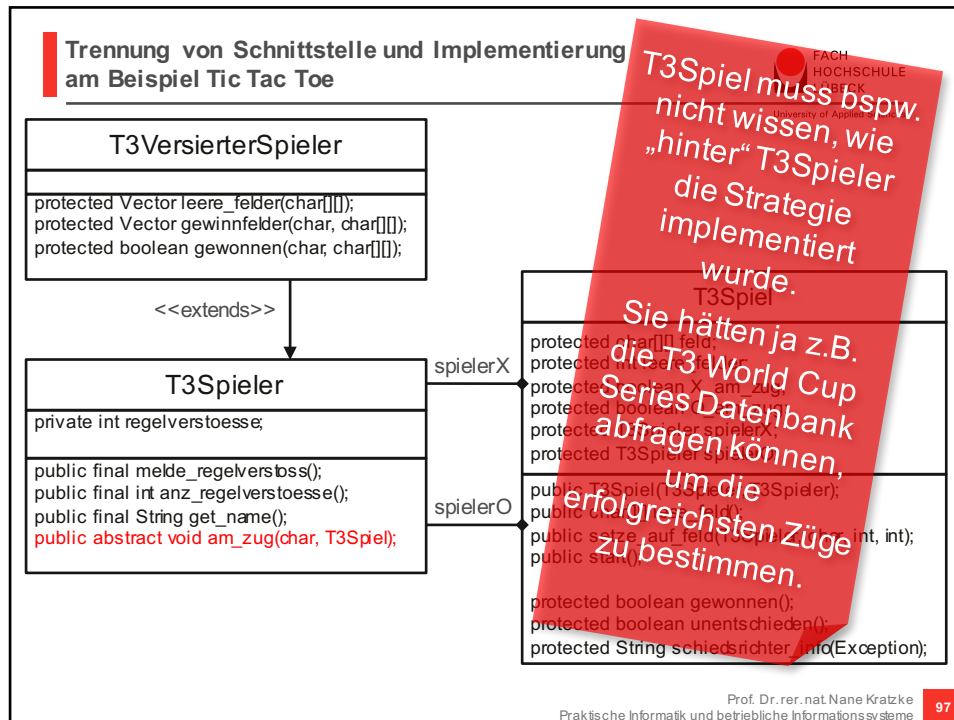


- Jede Abhängigkeit zwischen zwei Modulen explizit dokumentieren
- Ein Modul sollte nur von Schnittstellen und deren Spezifikation abhängen
- Ein Modul sollte niemals von nicht spezifizierten oder beeinflussbaren Seiteneffekten abhängen oder Implementierungen

- Vermeidung „stiller“ Kopplungen
- Erhöhung der Wiederverwendbarkeit

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

96



Program to Interfaces

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

Programmierenie nach dem Vertrag einer Methode

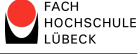
Wie nennt man die Lösungen, die entstehen, wenn man nach dem **WIE** implementiert?

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

Prinzipien des objektorientierten Entwurfs

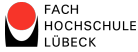

University of Applied Sciences

- 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

99

Umkehr der Abhängigkeiten Dependency Inversion Principle


University of Applied Sciences

Umkehr der Abhängigkeiten	Umkehr des Kontrollflusses
• Ein Entwurf soll sich auf Abstraktionen stützen. • Er soll sich nicht auf Spezialisierungen stützen.	• Ein spezifisches Modul • wird von einem mehrfach verwendbaren Modul aufgerufen.

 Erhöhung der Austauschbarkeit

 Reduzierung der Kopplung

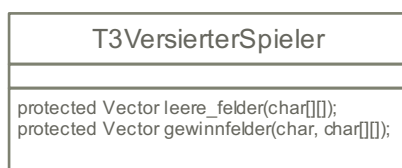
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

100

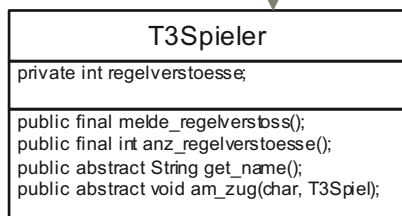
Regel:

- Ergänzungen werden von Kernmodulen initial aufgerufen!
- Niemals umgekehrt!

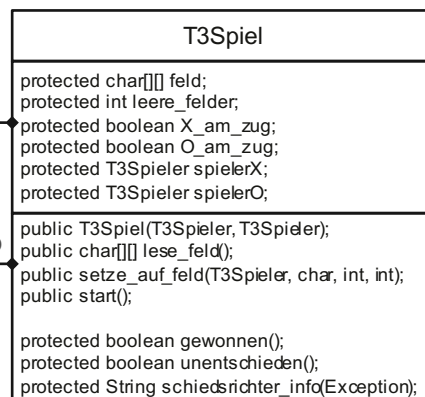
Umkehr der Abhängigkeiten am Beispiel Tic Tac Toe

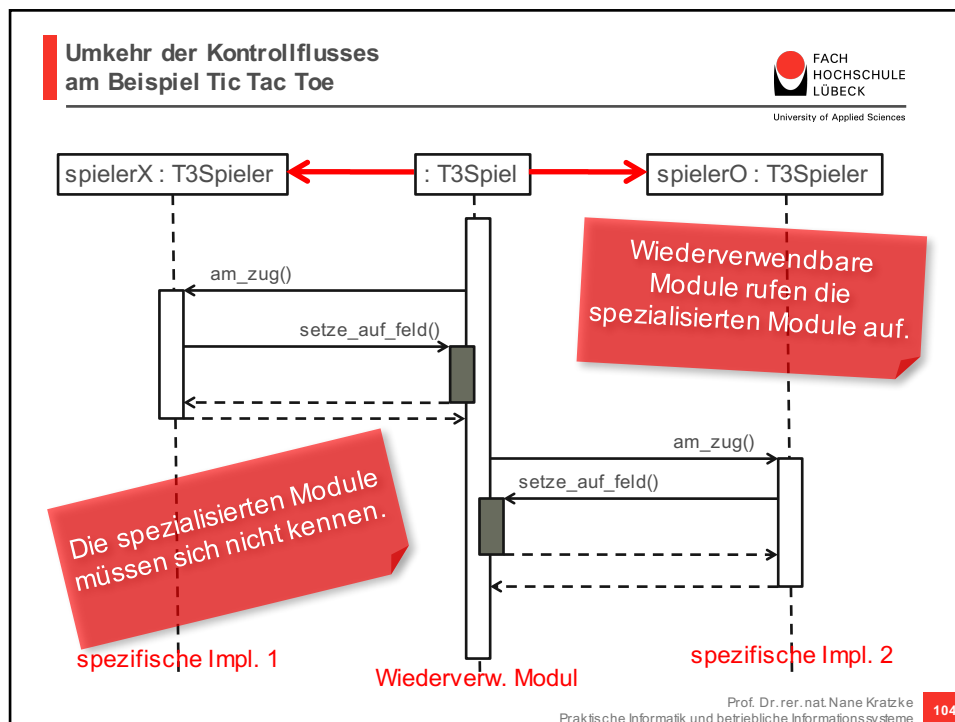
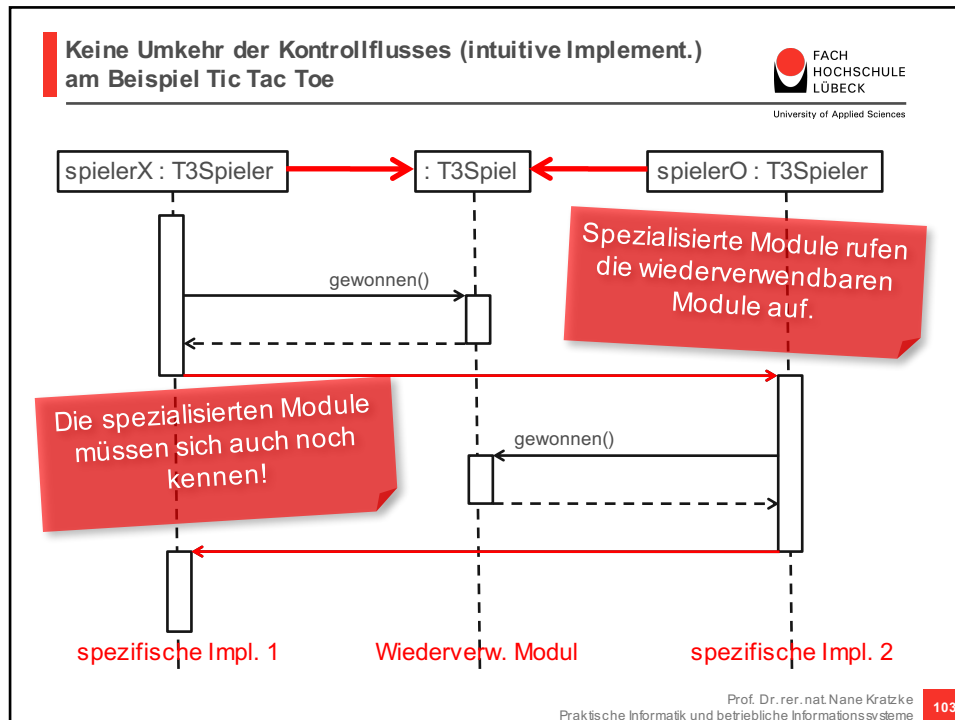


<<extends>>



T3Spiel kennt nur abstrakte T3Spieler, nicht deren Spezialisierungen, z.B. Ihre Spielstrategien. Das Engine Design stützt sich also auf Abstraktionen.





Prinzipien des objektorientierten Entwurfs

FACH HOCHSCHULE LÜBECK
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 105

Mach es testbar Unit Tests

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Unit-Tests	Unit-Tests
• sind Testprogramme • die die Korrektheit von SW-prüfen.	• sind automatisierbar • und helfen nach Änderungen Fehler schneller zu erkennen

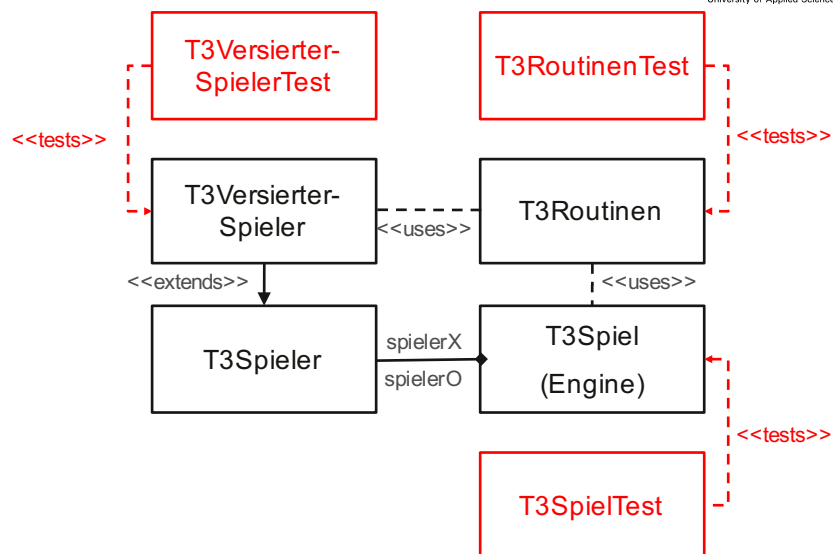
- Erhöhung der Korrektheit
- Erhöhung der Wartbarkeit
- Erhöhung der Testbarkeit

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 106

Regel:

- 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 T3SpielTest {

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

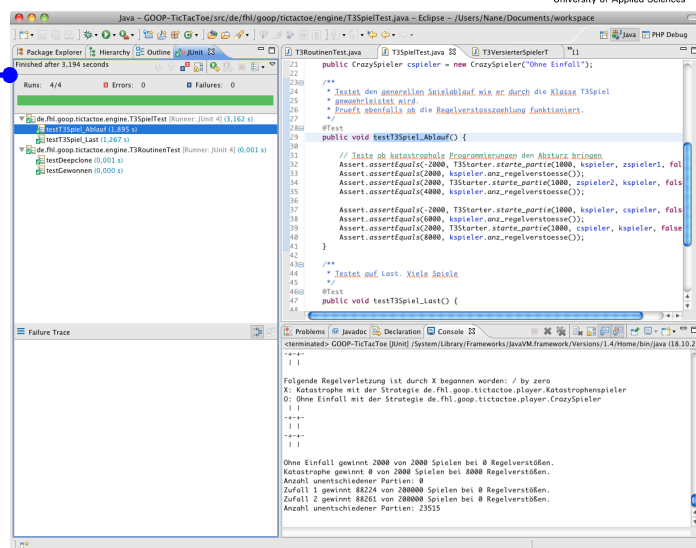
    @Test
    public void testT3Spiel_Ablauf() {

        // 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

Automatisierter Start,
Auswertung
und
Überblick
von Tests in
Eclipse

Run as
JUnit
Testcase



Zusammenfassung: Prinzipien des (OO) Entwurfs



- 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

111