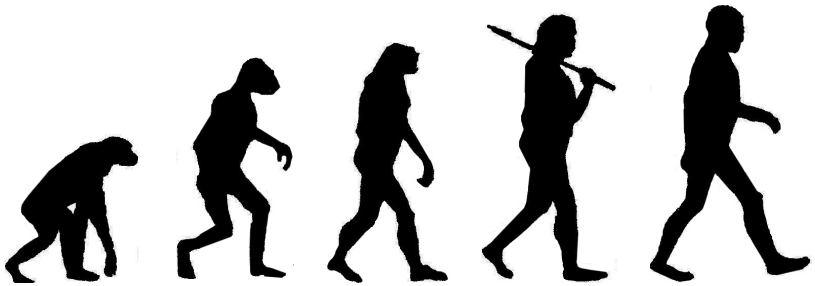
Karlsruhe Institute of Technology

VKSI Verein der Karlsruher Software-Ingenieure

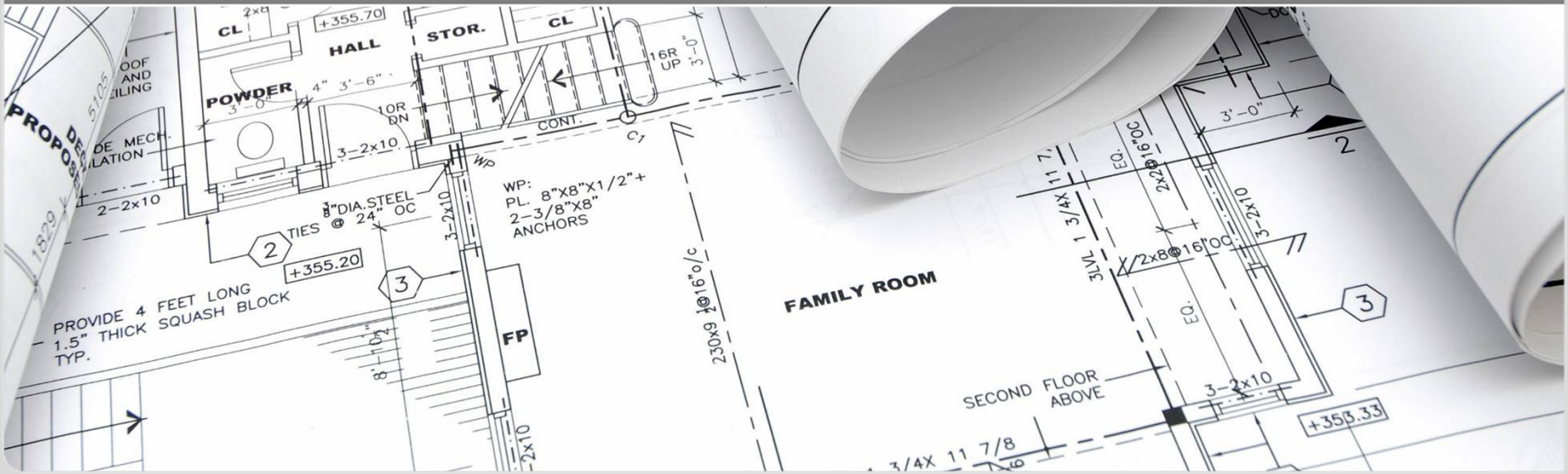
Design Driven Development (D3)

Agilität und Qualität im SE



Karlsruhe, 18. November 2011
Hagen Buchwald

Institut AIFB – KOMPLEXITÄTSMANAGEMENT



VKSI Verein der Karlsruher Software-Ingenieure

Home Login Kontakt Impressum

Software means Engineering

Home | Der Verein | Sneak Preview | Veranstaltungen | Newsletter | Magazin | BOGY-Praktikum

Home

suchen GO!

VKSI-Magazin Nr. 4 - Lebenslanges Lernen in der IT-Region Karlsruhe

Das vierte VKSI-Magazin ist als Sonderausgabe zum Thema "Lebenslangem Lernen in der IT-Region Karlsruhe" erschienen und berichtet über Aus- und Weiterbildungsmöglichkeiten in der Region sowie der Bedeutung der stetigen Weiterentwicklung im IT Berufsleben. Weitere Informationen und das Heft als kostenlosen Download und Online Ausgabe finden Sie **hier auf der Magazin Seite**.

Willkommen beim Verein der Karlsruher Software-Ingenieure (VKSI)

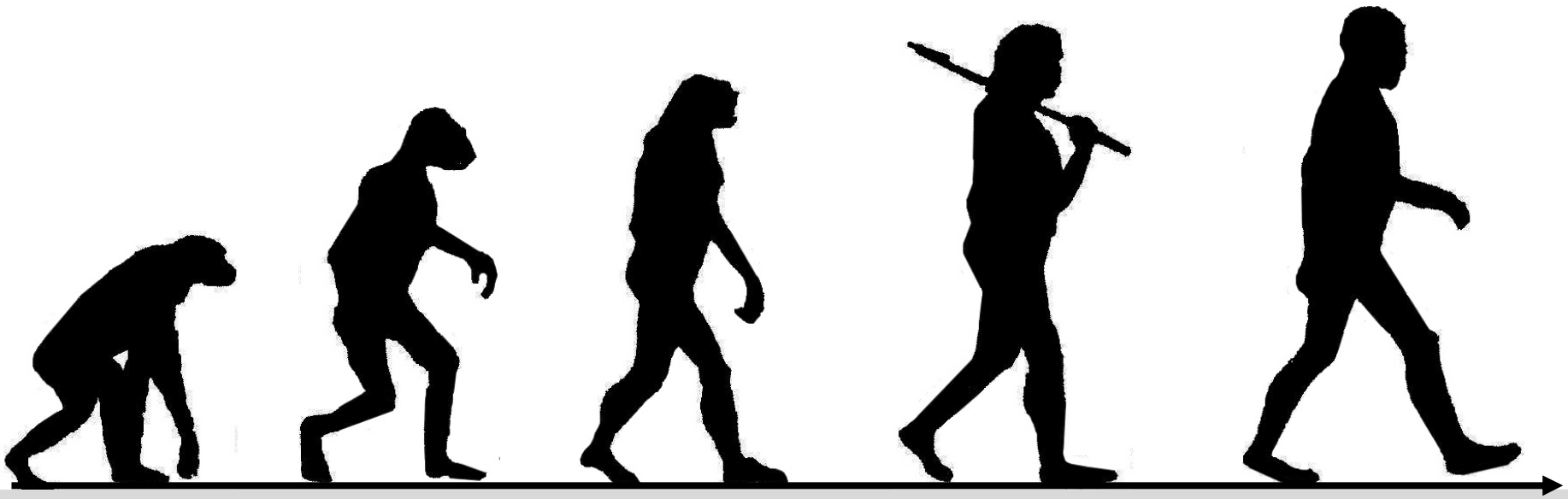
Softwaretechnik (oder Software-Engineering) ist eine der Kerndisziplinen der Informatik und Grundlage fast

Aktuelles

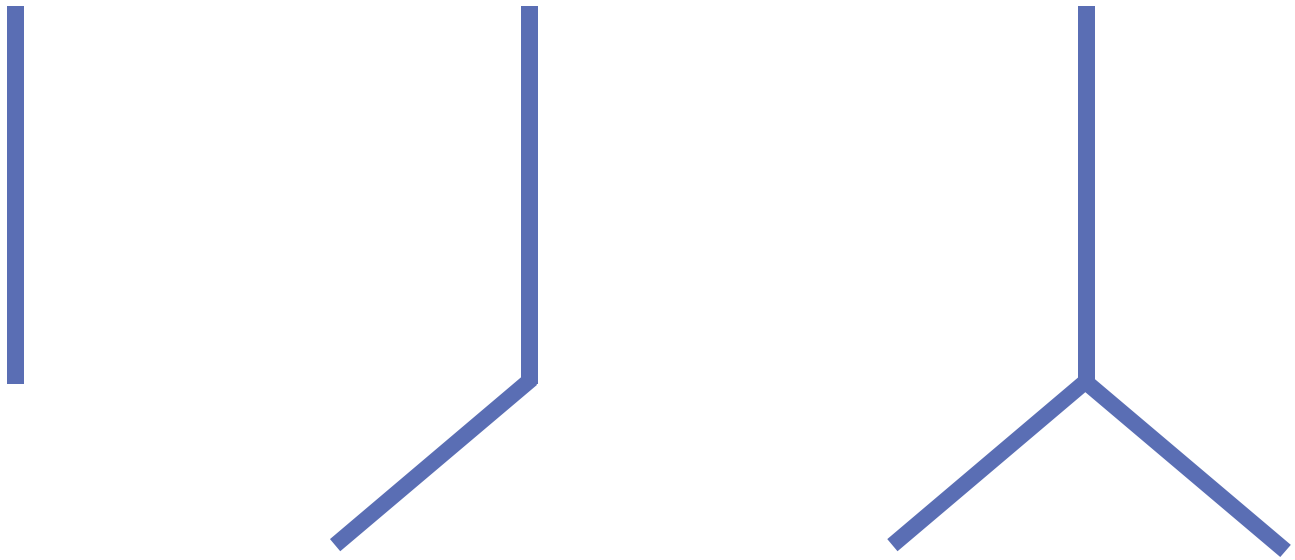
Requirements Engineering & Management
ein effektiver 3-tägiger Workshop. Reduzierten Satz für Mitglieder des VKSI bzw. Mitarbeiter von VKSI Mitgliedsfirmen.
Weitere Informationen

Sie entwickeln Software in der Region?
Wenn auch Sie oder Ihr Unternehmen dem Verein beitreten wollen, finden Sie den
Aufnahmeantrag hier.

Wo stehen wir im Software Engineering?

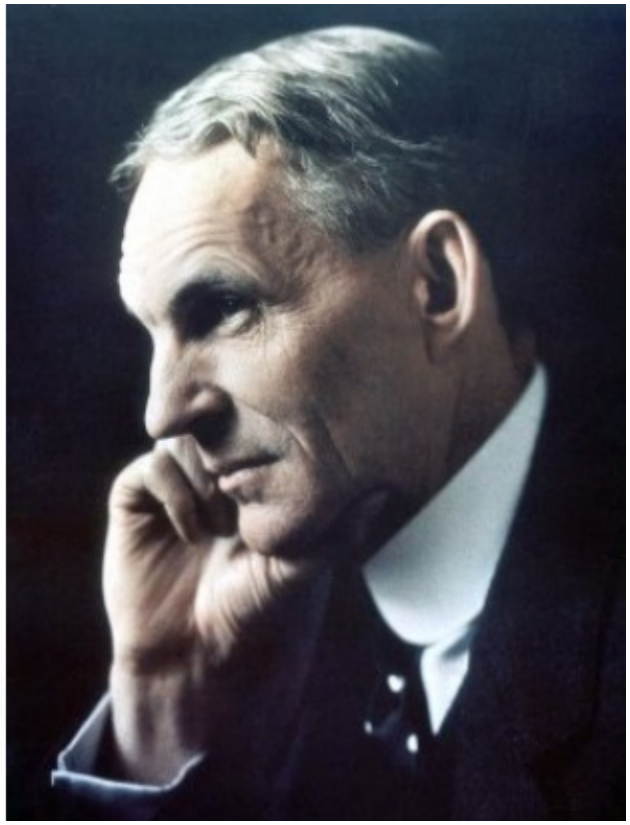


Warum sind diese Symbole so wichtig für die Software-Qualität?

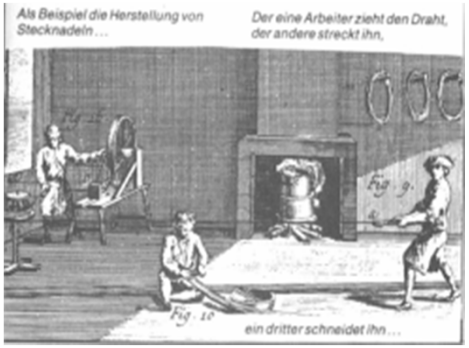


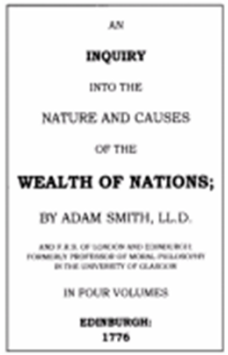
Was hat Poker mit Design by Contract zu tun?

Und was haben Adam Smith und Henry Ford mit Software-Entwicklung zu tun?

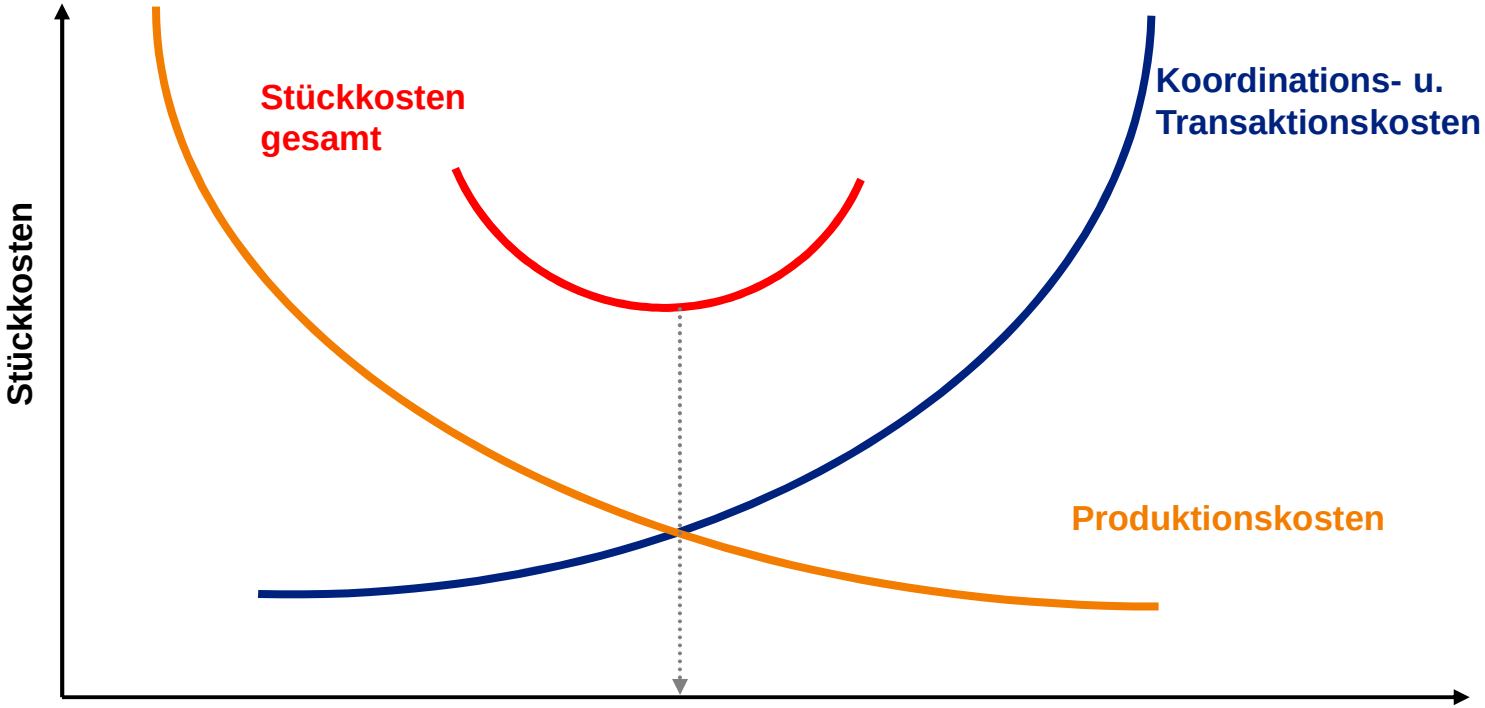


„The division of work shall improve the productivity of work more than any other means.“Adam Smith (1776)



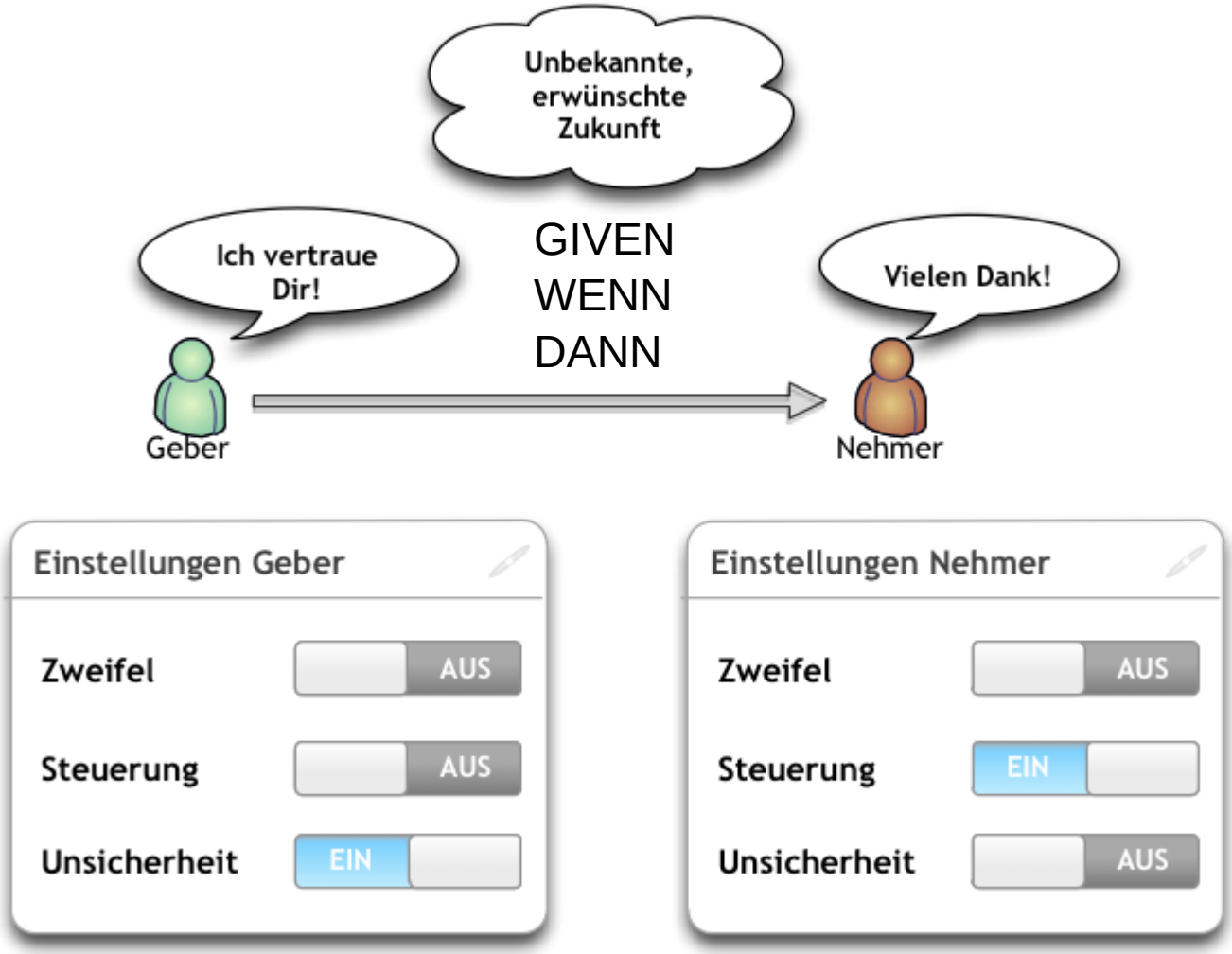


Adam Smith
1723 - 1790



Quelle: Gary Becker und Kevin M. Murphy, 1992

„Vertrauen reduziert soziale Komplexität.“
M. Bohlen (XP Days 2011, „Ein Team und sein Vertrag“)



Quelle: Matthias Bohlen (www.mbohlen.de), XP Days 2011, Vortrag „Ein Team und sein Vertrag“

Exkurs: Was kann passieren, wenn die GIVEN-WENN-DANN-Kette bricht? Das Phantom von Heilbronn.

GIVEN

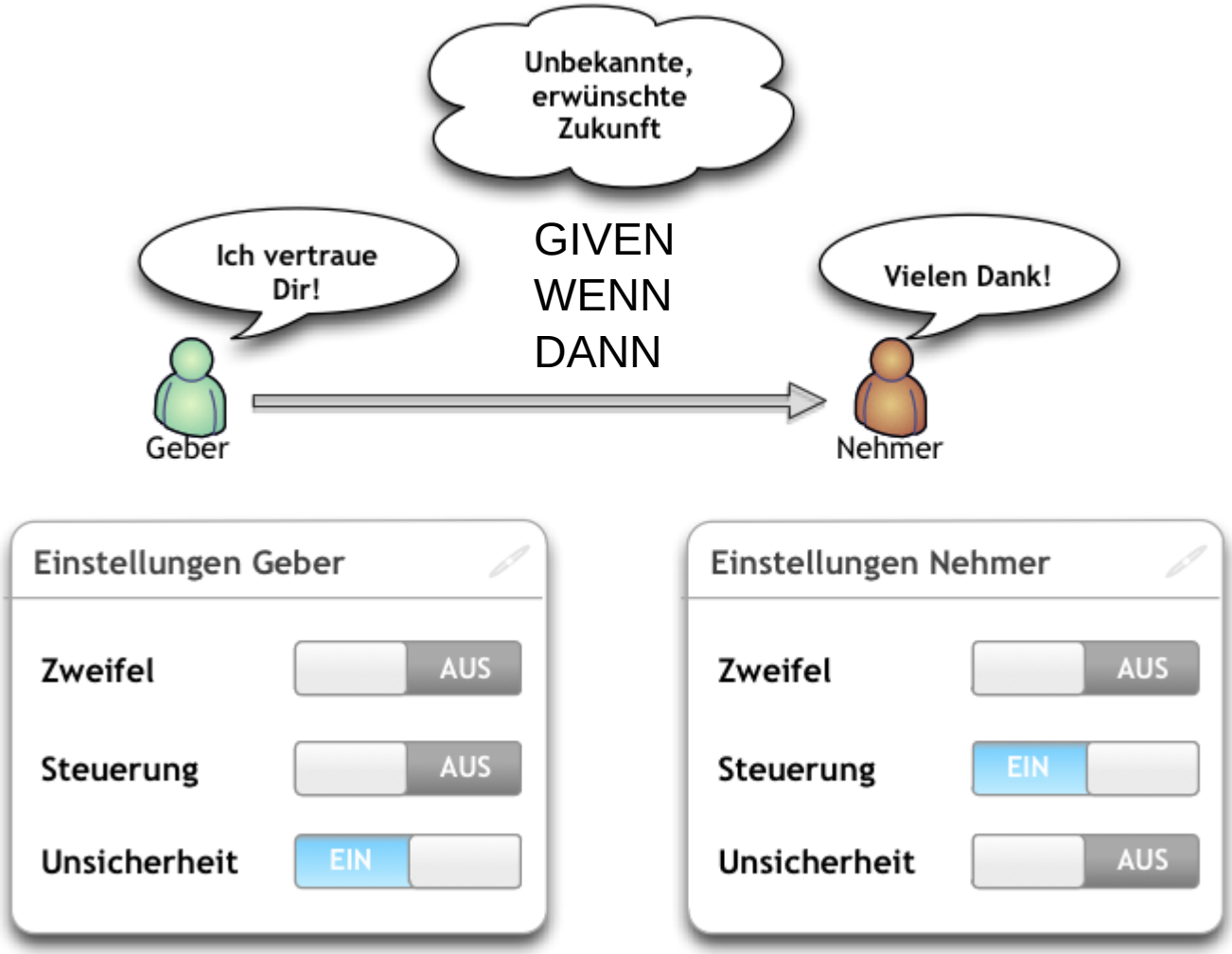
WENN

WENN

DANN

- Falsche Annahmen können hohe Kosten verursachen – und unter Umständen Menschenleben kosten!

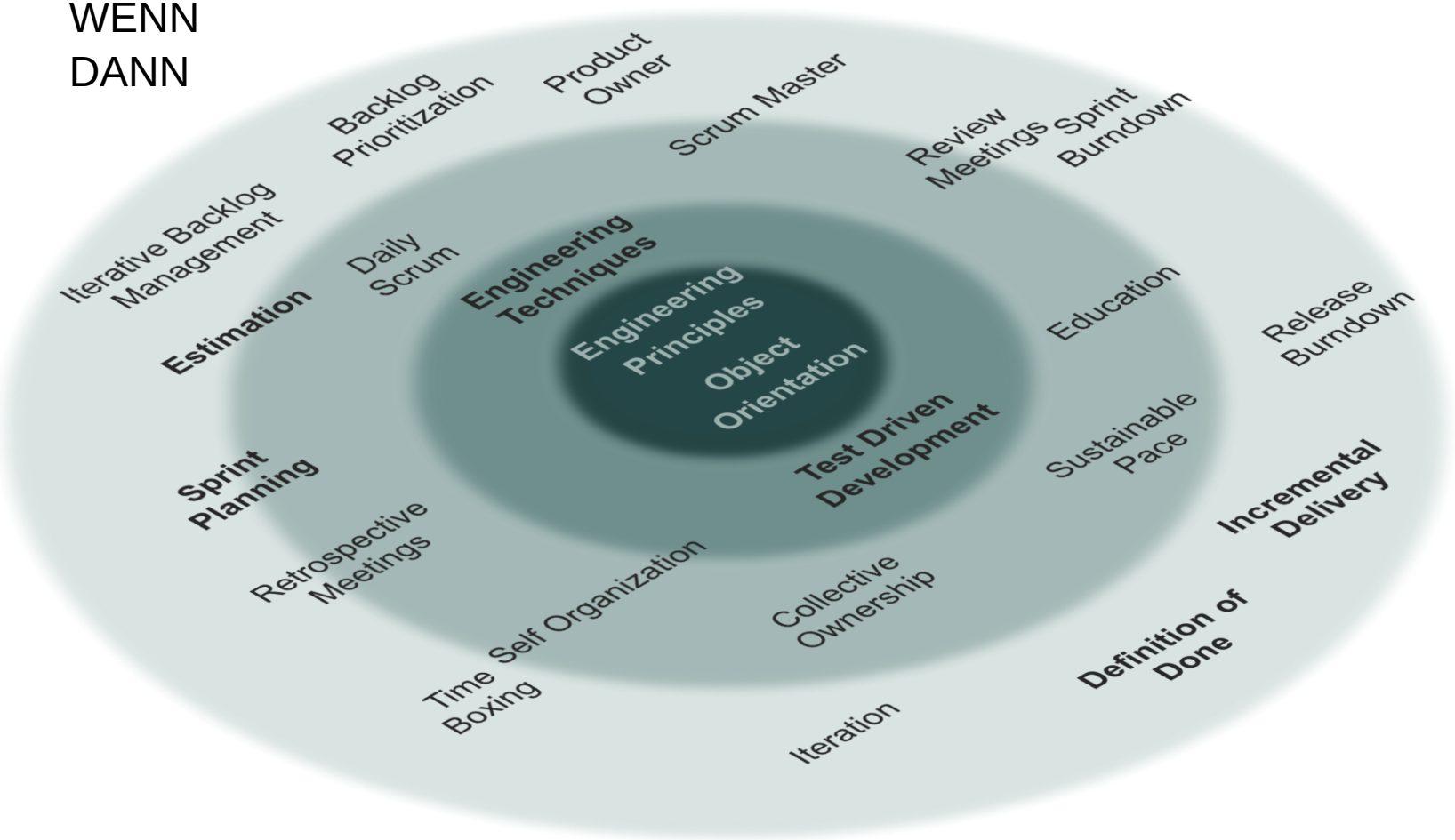
„Vertrauen reduziert soziale Komplexität.“
M. Bohlen (XP Days 2011, „Ein Team und sein Vertrag“)



Quelle: Matthias Bohlen (www.mbohlen.de), XP Days 2011, Vortrag „Ein Team und sein Vertrag“

Agiles Software-Engineering basiert in seinem Kern auf der Beherrschung der Objektorientierten Programmierung!

GIVEN
WENN
DANN

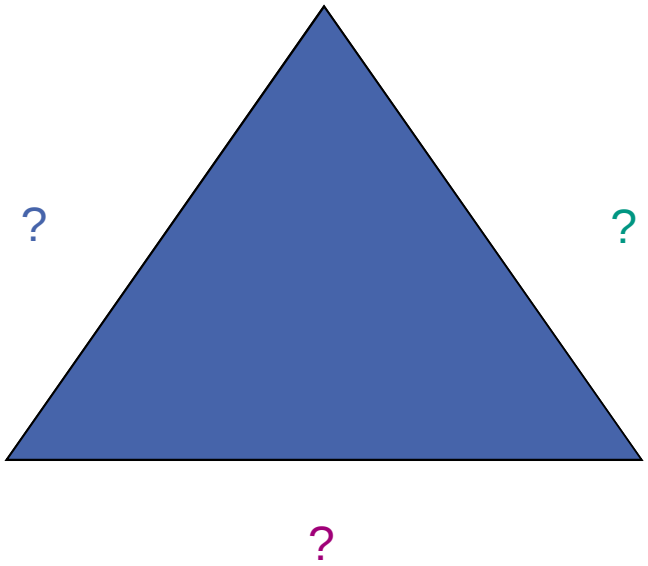


D3 – Agilität und Qualität im Software-Engineering
Agenda

Karlsruher Institut für Technologie

1. Was versteht man konkret unter Objektorientierter Programmierung?
2. Ist Java eine vollständig objektorientierte Programmiersprache?
3. Wie können in Java Abstrakte Datentypen modelliert werden?
4. Was ist Design by Contract – und wie funktioniert es in Java?
5. Was verändert sich, wenn wir Design by Contract in Java nutzen?
6. Wie stehen DbC und TDD zueinander?
7. Fragen & Antworten

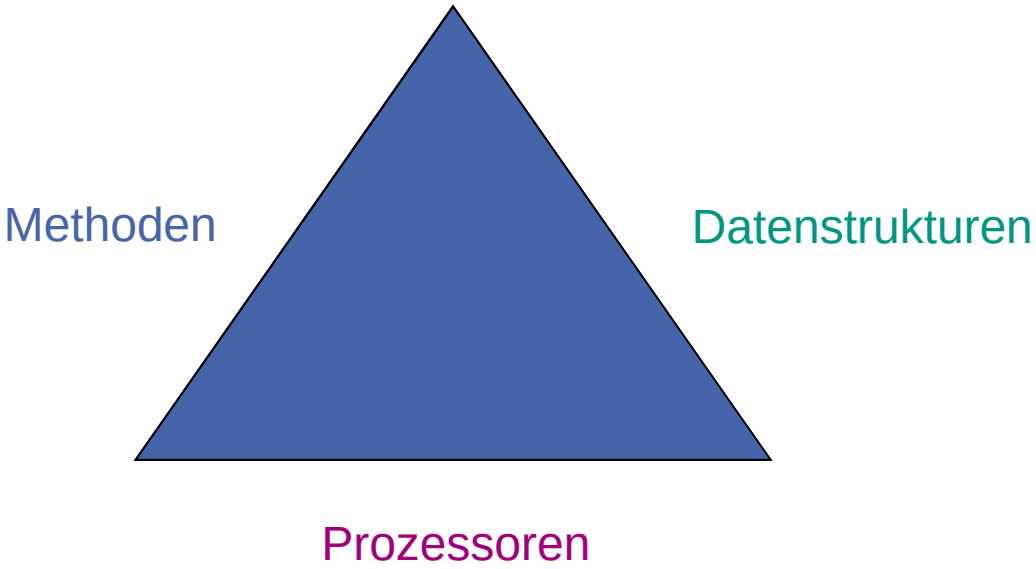
Welches sind die drei Kernelemente eines Softwaresystems?



In Anlehnung an: Bertrand Meyer, Object-Oriented Software Construction, 1997

Die drei Kernelemente eines Softwaresystems sind: Methoden, Datenstrukturen und Prozessoren.

Ein Softwaresystem besteht aus Prozessoren,
die Methoden auf Datenstrukturen ausführen.



In Anlehnung an: Bertrand Meyer, Object-Oriented Software Construction, 1997

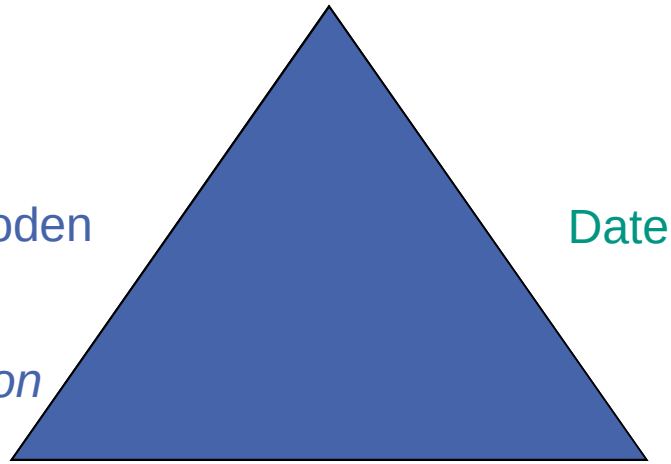
Objekt-Orientierte Programmierung (OOP). Definition nach Bertrand Meyer.

Ein Softwaresystem besteht aus Prozessoren,
die Methoden auf Datenstrukturen ausführen.

„Prädikat“ ï Methoden

AM

„Was ist die Hauptfunktion des Systems?“

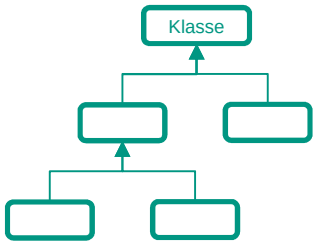


Datenstrukturen ð Objekt

ADT

„Auf was arbeitet das System?“

Prozessoren



OOP ist die Erstellung von Softwaresystemen
als strukturierte Sammlung von Implementierungen
Abstrakter Datentypen (ADTs).

Quelle: Bertrand Meyer, Object-Oriented Software Construction, 1988

D3 – Agilität und Qualität im Software-Engineering
Agenda

Karlsruher Institut für Technologie

1. Was versteht man konkret unter Objektorientierter Programmierung?
2. Ist Java eine vollständig objektorientierte Programmiersprache?
3. Wie können in Java Abstrakte Datentypen modelliert werden?
4. Was ist Design by Contract – und wie funktioniert es in Java?
5. Was verändert sich, wenn wir Design by Contract in Java nutzen?
6. Wie stehen DbC und TDD zueinander?
7. Fragen & Antworten

Die 7 Stufen zur Objektorientierung am Beispiel `Eiffel`

Karlsruher Institut für Technologie

Ü **Stufe 7**

Mehrfaches und wiederholtes Erben

Man kann Klassen deklarieren, die Erben von mehr als einer Klasse sind und mehr als einmal erben.

Ü **Stufe 6**

Polymorphismus und dynamisches Binden

Programm-Elemente dürfen sich auf Objekte aus mehr als einer Klasse beziehen, und Operationen dürfen unterschiedliche Realisierungen in unterschiedlichen Klassen haben.

Ü **Stufe 5**

Vererbung

Eine Klasse kann als Einschränkung oder Erweiterung einer anderen definiert werden.

Ü **Stufe 4**

Klassen

Jeder nicht-einfache Typ ist ein Modul, und jeder Modul höherer Stufe ist ein Typ.

Ü **Stufe 3**

Automatische Speicherplatzverwaltung

Unbenutzte Objekte sollen ohne Programmiereingriff vom unterliegenden Sprachsystem freigegeben werden.

Ü **Stufe 2**

Datenabstraktion

Objekte müssen als Implementierung abstrakter Datentypen (ADTs) beschrieben werden.

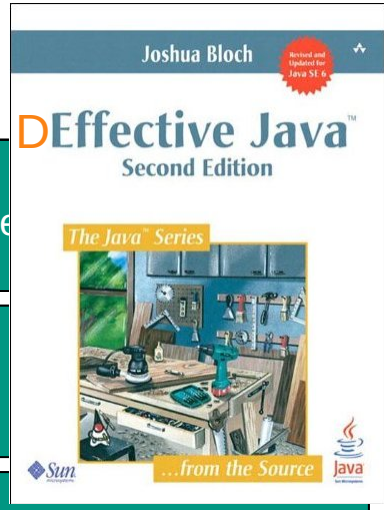
Ü **Stufe 1**

Objektbasierte, modulare Struktur

Systeme werden auf Grundlage ihrer Datenstruktur modularisiert.

Quelle: Bertrand Meyer, Object-Oriented Software Construction, 1988

Die 7 Stufen zur Objektorientierung am Beispiel Java



Stufe 7
Mehrfaches und wiederholtes Erben
Man kann Klassen deklarieren, die Erben von mehr als einer Klasse und mehr als einmal erben.

Stufe 6
Polymorphismus und dynamisches Binden
Programm-Elemente dürfen sich auf Objekte aus mehr als einer Klasse und Operationen dürfen unterschiedliche Realisierungen in unterschiedlichen

Stufe 5
Vererbung
Eine Klasse kann als Einschränkung oder Erweiterung einer anderen definiert werden.

Stufe 4
Klassen
Jeder nicht-einfache Typ ist ein Modul, und jeder Modul höherer Stufe ist ein Typ.

Stufe 3
Automatische Speicherplatzverwaltung
Unbenutzte Objekte sollen ohne Programmiereingriff vom unterliegenden Sprachsystem freigegeben werden.

Stufe 2
Datenabstraktion
Objekte müssen als Implementierung abstrakter Datentypen (ADTs) beschrieben werden.

Stufe 1
Objektbasierte, modulare Struktur
Systeme werden auf Grundlage ihrer Datenstruktur modularisiert.

Quelle: Bertrand Meyer, Object-Oriented Software Construction, 1986

D3 – Agilität und Qualität im Software-Engineering
Agenda

Karlsruher Institut für Technologie

1. Was versteht man konkret unter Objektorientierter Programmierung?
2. Ist Java eine vollständig objektorientierte Programmiersprache?
3. Wie können in Java Abstrakte Datentypen modelliert werden?
4. Was ist Design by Contract – und wie funktioniert es in Java?
5. Was verändert sich, wenn wir Design by Contract in Java nutzen?
6. Wie stehen DbC und TDD zueinander?
7. Fragen & Antworten

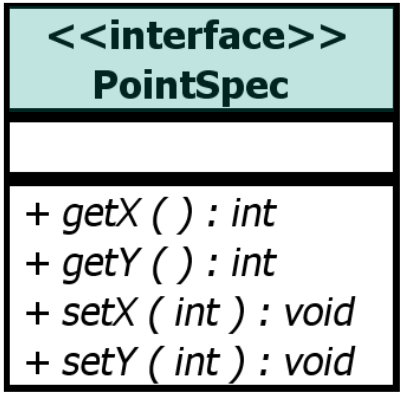
Die Klasse `Point` beschrieben in Form eines Abstrakten Datentyps (ADT).

```
TYPE
    PointSpec

FUNCTIONS
    getX() : int
    getY() : int
    setX(int x) : void
    setY(int y) : void


PRE-CONDITIONS
    setX(int x) : x >= 0
    setY(int y) : y >= 0

AXIOMS (POST-CONDITIONS)
    setX(int x) : getX() == x
    setY(int y) : getY() == y
```



Stufe 2

Datenabstraktion

Objekte müssen als Implementierung abstrakter Datentypen (ADTs) beschrieben werden.

Derived from: Bertrand Meyer, Object-Oriented Software Construction, 1988

Assertions in Java can *only* be applied to *implemented* methods, not to *abstract* methods or *abstract* classes.

- Syntax von Assertions in Java:

Syntaxregel

```
assert «AUSDRUCK» : «AUSDRUCK» ;
```

`boolean` Ausdruck

`String` Ausdruck

- Beispiel:

```
public double inverse(double x)
{
    assert x != 0 : "x_not_0";
    double result = 0;
    result = 1.0 / x;
    return result;
}
```

Assertions in Java can not be applied on an abstract level and can not be inherited to a subclass!

Syntax rule taken from: Ratz, Seese: Grundkurs Programmieren in Java, 2010

Java ist gemäß der Definition von B. Meyer *keine* vollständig objektorientierte Programmiersprache!

Karlsruher Institut für Technologie

Stufe 7

Mehrfaches und wiederholtes Erben

Man kann Klassen deklarieren, die Erben von mehr als einer Klasse sind und mehr als einmal erben.

Stufe 6

Polymorphismus und dynamisches Binden

Programm-Elemente dürfen sich auf Objekte aus mehr als einer Klasse beziehen, und Operationen dürfen unterschiedliche Realisierungen in unterschiedlichen Klassen haben.

Stufe 5

Vererbung

Eine Klasse kann als Einschränkung oder Erweiterung einer anderen definiert werden.

Stufe 4

Klassen

Jeder nicht-einfache Typ ist ein Modul, und jeder Modul höherer Stufe ist ein Typ.

Stufe 3

Automatische Speicherplatzverwaltung

Unbenutzte Objekte sollen ohne Programmiereingriff vom unterliegenden Sprachsystem freigegeben werden.

Stufe 2

Datenabstraktion

Objekte müssen als Implementierung abstrakter Datentypen (ADTs) beschrieben werden.

Stufe 1

Objektbasierte, modulare Struktur

Systeme werden auf Grundlage ihrer Datenstruktur modularisiert.

Quelle: Bertrand Meyer, Object-Oriented Software Construction, 1986

D3 – Agilität und Qualität im Software-Engineering
Agenda

Karlsruher Institut für Technologie

1. Was versteht man konkret unter Objektorientierter Programmierung?
2. Ist Java eine vollständig objektorientierte Programmiersprache?
3. Wie können in Java Abstrakte Datentypen modelliert werden?
4. Was ist Design by Contract – und wie funktioniert es in Java?
5. Was verändert sich, wenn wir Design by Contract in Java nutzen?
6. Wie stehen DbC und TDD zueinander?
7. Fragen & Antworten

Was tun Kinder zuerst, wenn sie miteinander spielen wollen?

Sie legen die Spielregeln fest!

Was machen Unternehmer, wenn sie kooperieren möchten?

Kunde

Rechte

Pflichten

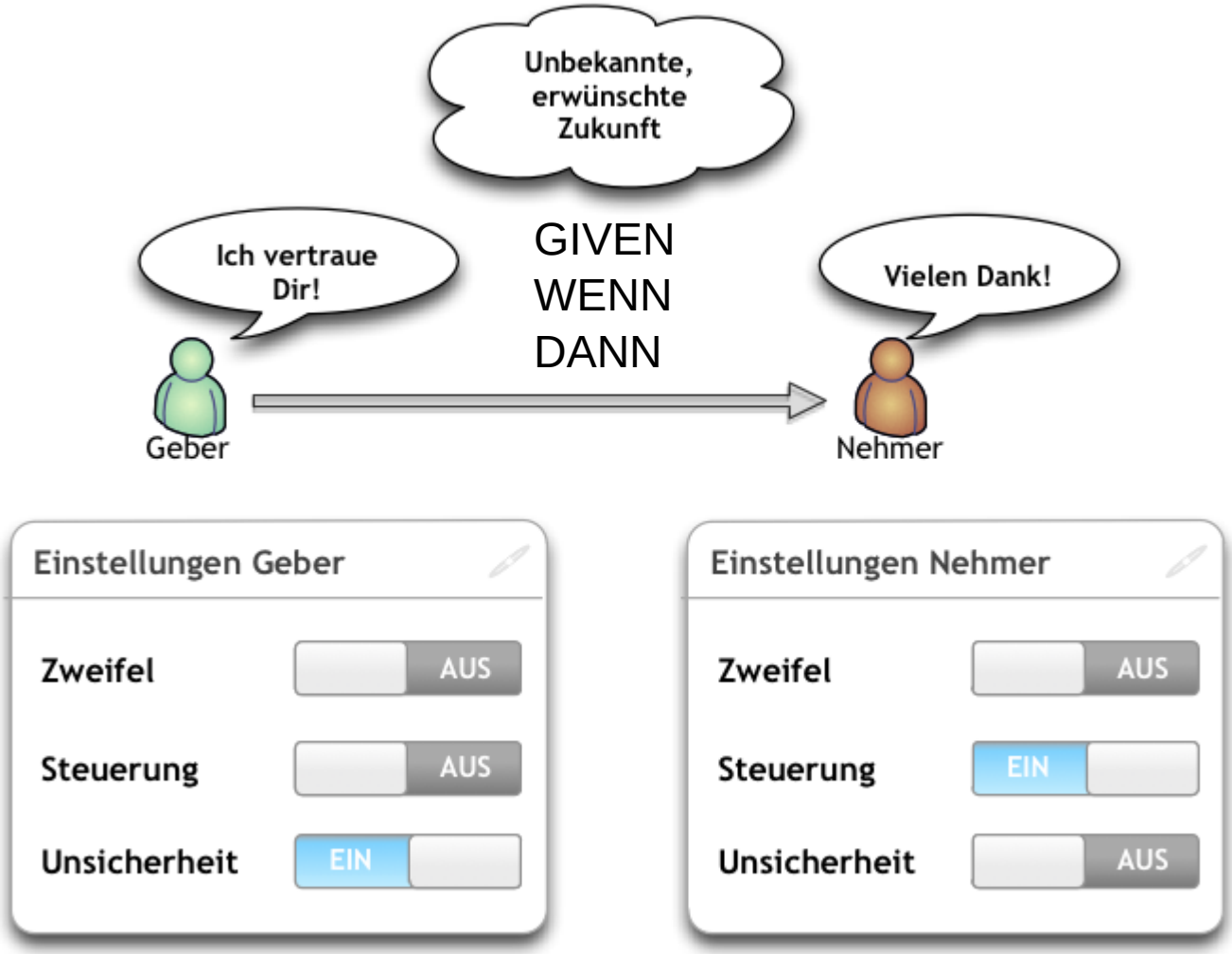
Anbieter

Rechte

Pflichten

Sie schließen einen Vertrag ab!

„Vertrauen reduziert soziale Komplexität.“
M. Bohlen (XP Days 2011, „Ein Team und sein Vertrag“)



Quelle: Matthias Bohlen (www.mbohlen.de), XP Days 2011, Vortrag „Ein Team und sein Vertrag“

Die Klasse `Point` beschrieben in Form eines Abstrakten Datentyps (ADT).

```
TYPE
    PointSpec

FUNCTIONS
    getX() : int
    getY() : int
    setX(int x) : void
    setY(int y) : void
```

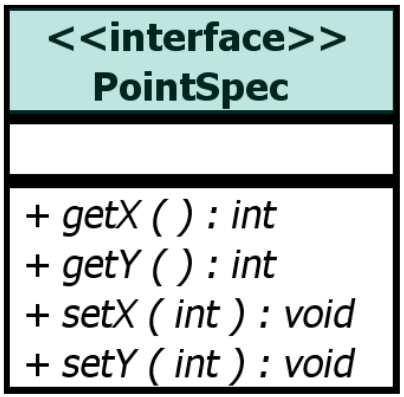
GIVEN

```
PRE-CONDITIONS
    setX(int x) : x >= 0
    setY(int y) : y >= 0
```

WENN

```
AXIOMS (POST-CONDITIONS)
    setX(int x) : getX() == x
    setY(int y) : getY() == y
```

DANN



Stufe 2

Datenabstraktion

Objekte müssen als Implementierung abstrakter Datentypen (ADTs) beschrieben werden.

Derived from: Bertrand Meyer, Object-Oriented Software Construction, 1988

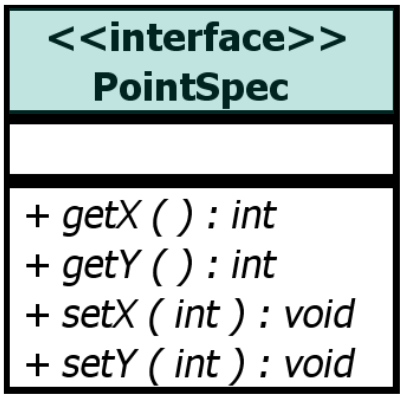
Die Klasse `Point` beschrieben in Form eines Abstrakten Datentyps (ADT).

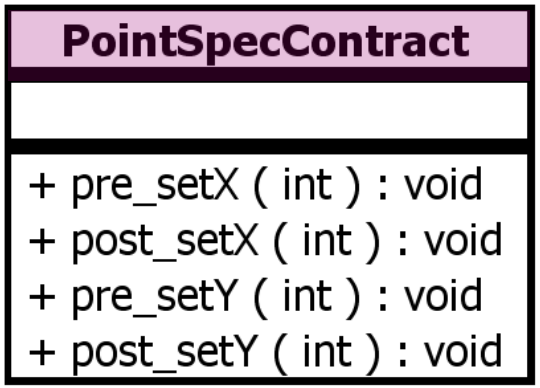
```
TYPE
    PointSpec

FUNCTIONS
    getX() : int
    getY() : int
    setX(int x) : void
    setY(int y) : void


PRE-CONDITIONS
    setX(int x) : x >= 0
    setY(int y) : y >= 0

AXIOMS (POST-CONDITIONS)
    setX(int x) : getX() == x
    setY(int y) : getY() == y
```





Stufe 2

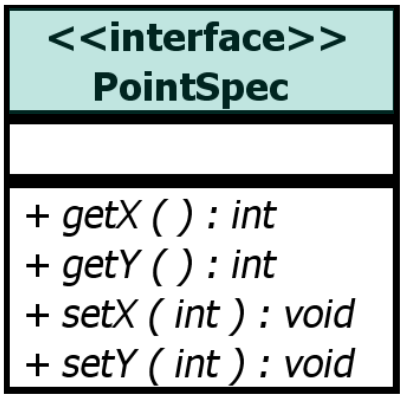
Datenabstraktion

Objekte müssen als Implementierung abstrakter Datentypen (ADTs) beschrieben werden.

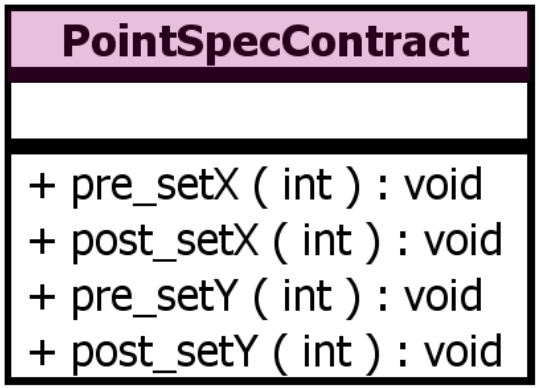
Derived from: Bertrand Meyer, Object-Oriented Software Construction, 1988

Der Abstrakte Datentyp (ADT) Point formuliert in Java.

```
import net.sourceforge.c4j.*;
@ContractReference(contractClassName = "PointSpecContract")
public interface PointSpec
{
    @Pure int getX();
    @Pure int getY();
    void setX(int x);
    void setY(int y);
}
```



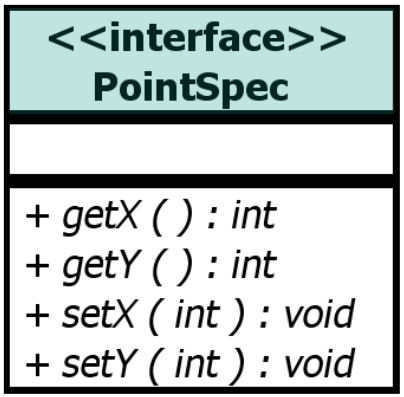

```
import net.sourceforge.c4j.*;
public class PointSpecContract
             extends ContractBase<PointSpec> {
    public void pre_setX(int x) {
        assert x >= 0 : "x_>=_0";
    }
    public void post_setX(int x) {
        assert m_target.getX() == x : "x_set";
    }
    public void pre_setY(int y) {
        assert y >= 0 : "y_>=_0";
    }
    public void post_setY(int y) {
        assert m_target.getY() == y : "y_set";
    }
}
```



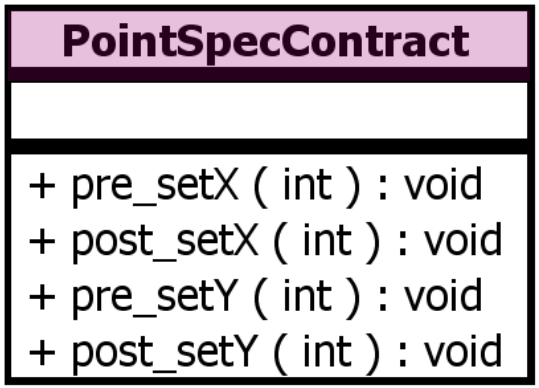
Quelle: Bertrand Meyer, Object-Oriented Software Construction, 1986

Der Abstrakte Datentyp (ADT) Point formuliert in Java.

```
import net.sourceforge.c4j.*;
@ContractReference(contractClassName = "PointSpecContract")
public interface PointSpec
{
    @Pure int getX();
    @Pure int getY();
    void setX(int x);
    void setY(int y);
}
```




```
import net.sourceforge.c4j.*;
public class PointSpecContract
            extends ContractBase<PointSpec> {
    public void pre_setX(int x) {
        assert x >= 0 : "x_>=_0";
    }
    public void post_setX(int x) {
        assert m_target.getX() == x : "x_set";
    }
    …
```



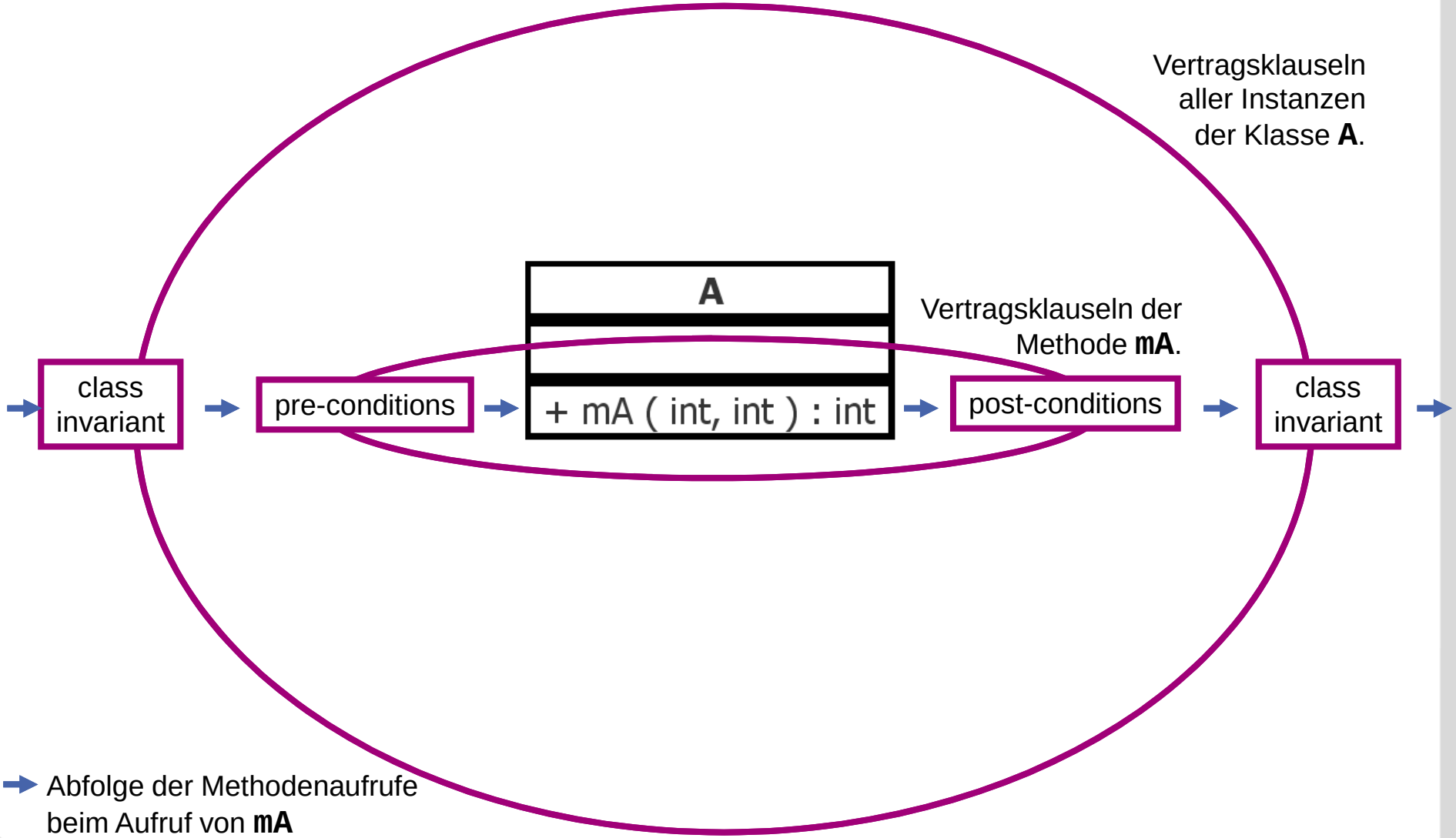
Stufe 2

Datenabstraktion

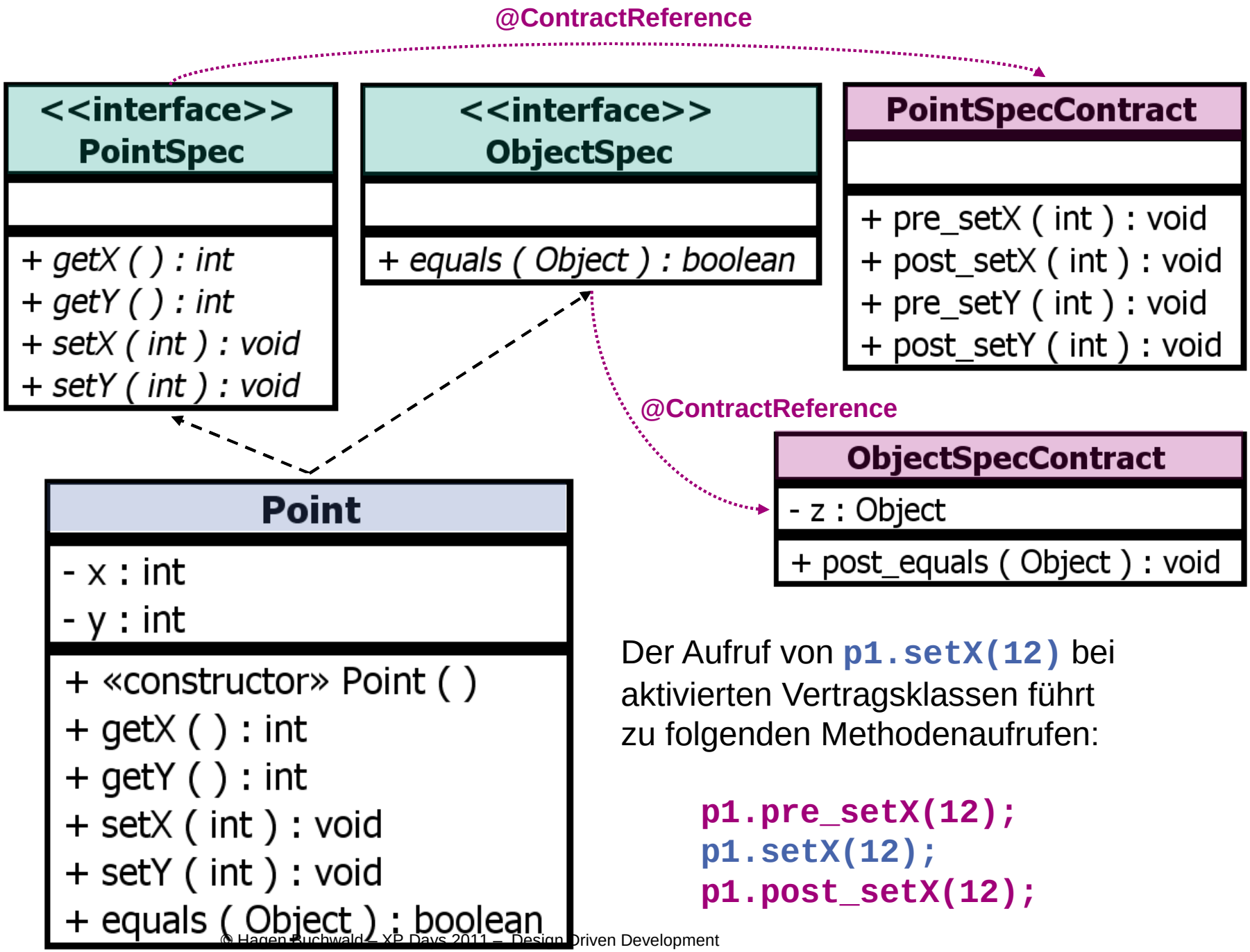
Objekte müssen als Implementierung abstrakter Datentypen (ADTs) beschrieben werden.

Quelle: Bertrand Meyer, Object-Oriented Software Construction, 1986

Vertragsklassen bewirken eine klar definierte Abfolge von Methodenaufrufen beim Aufruf der Methode `mA`.

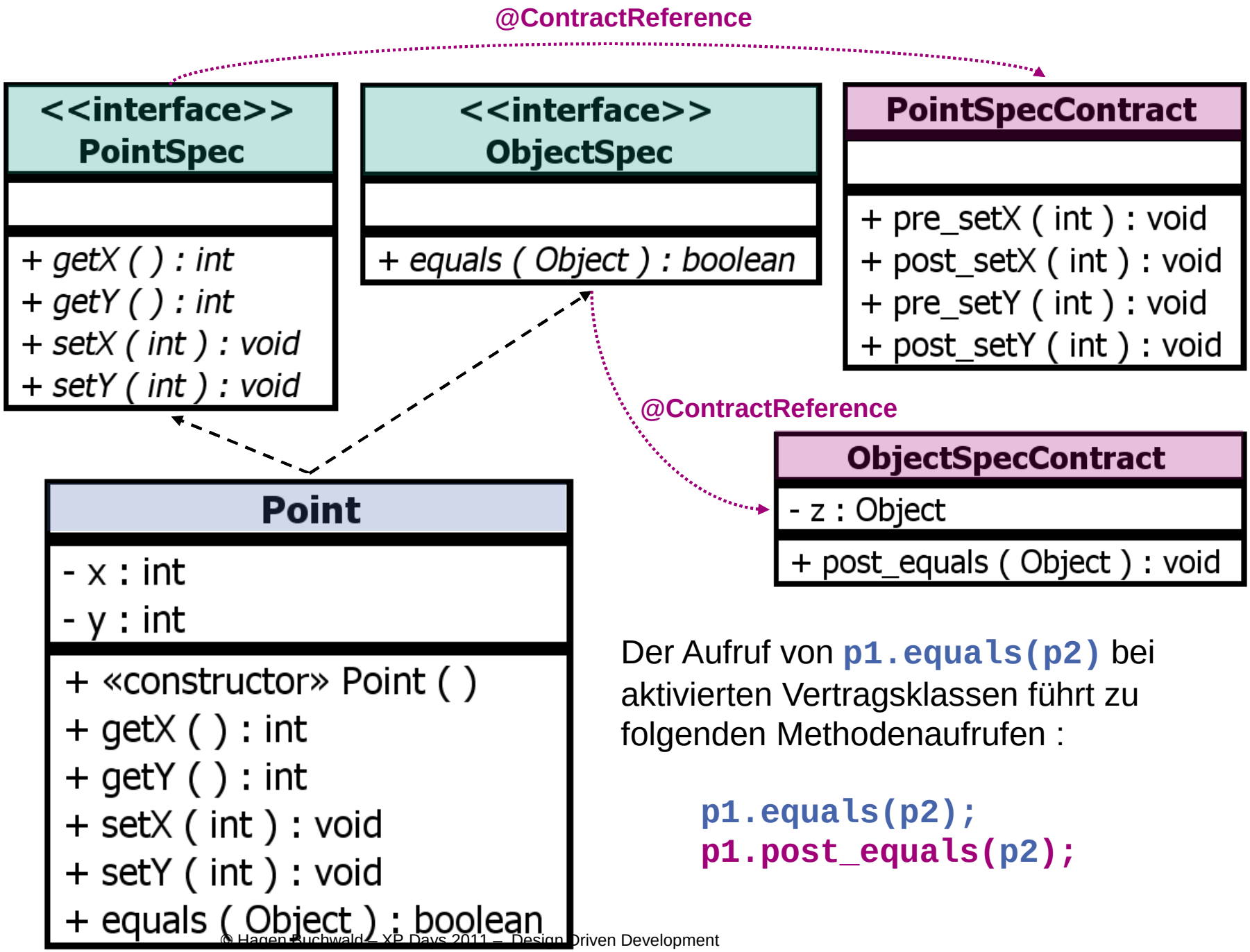


→ Abfolge der Methodenaufrufe beim Aufruf von **mA**



Der Aufruf von `p1.setX(12)` bei aktivierten Vertragsklassen führt zu folgenden Methodenaufrufen:

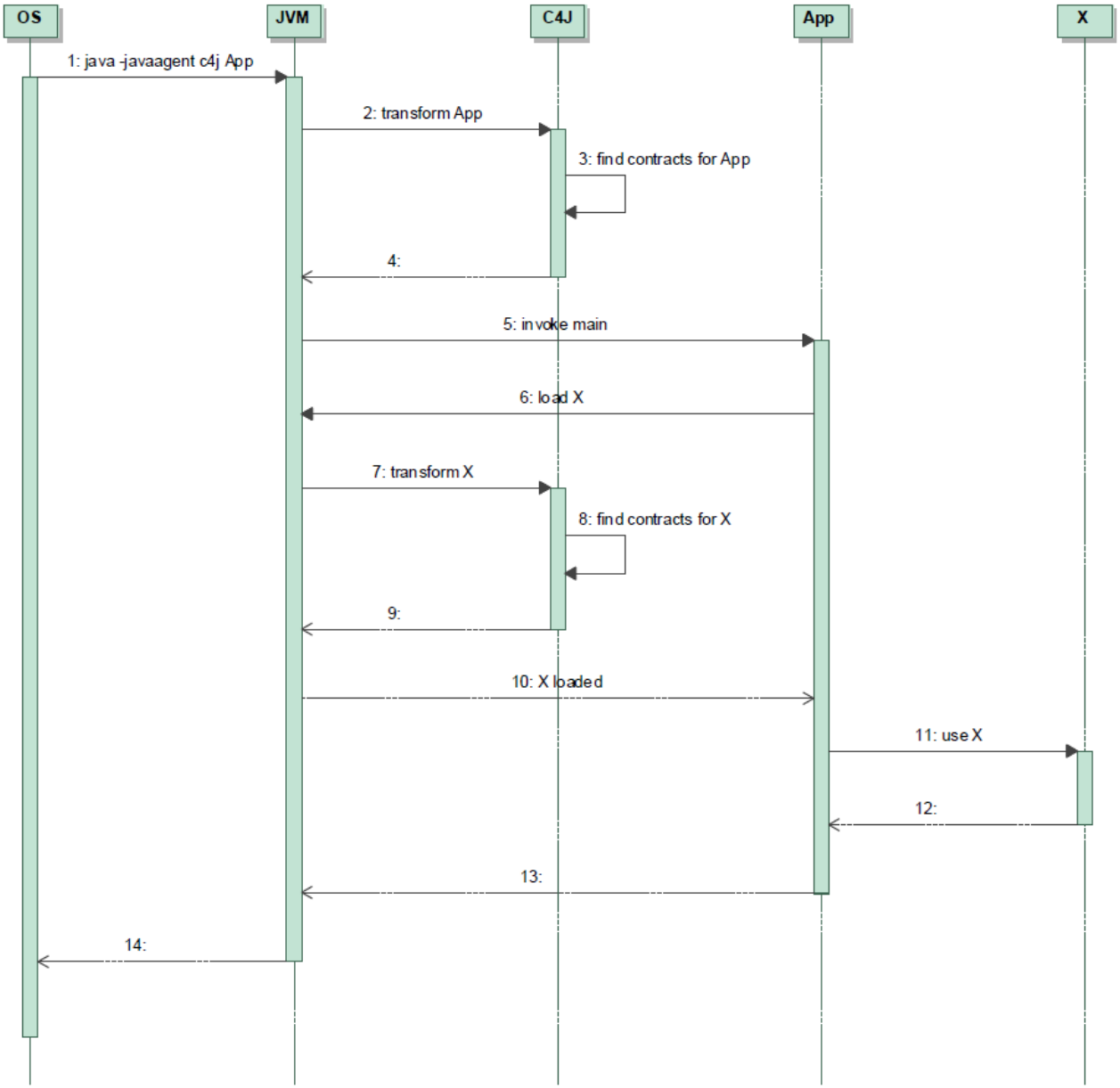
```
p1.pre_setX(12);
p1.setX(12);
p1.post_setX(12);
```



Der Aufruf von `p1.equals(p2)` bei aktivierten Vertragsklassen führt zu folgenden Methodenaufrufen :

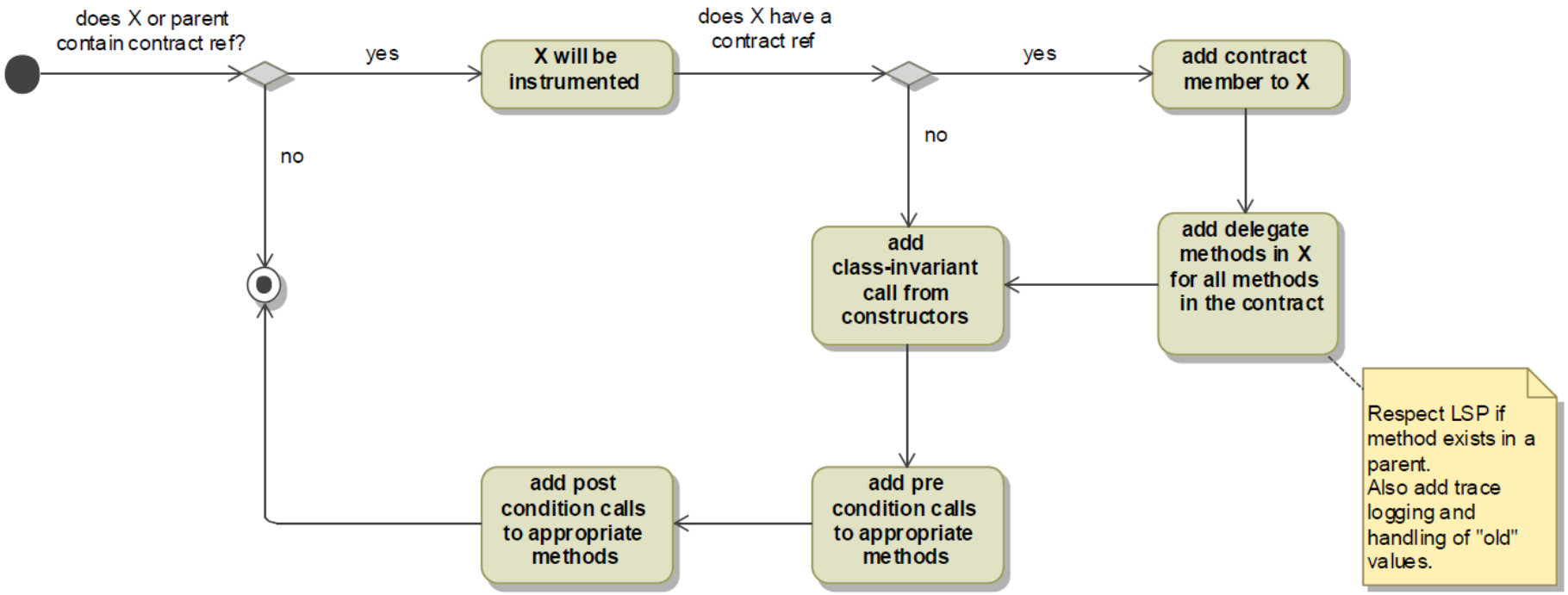
```
p1.equals(p2);
p1.post_equals(p2);
```

C4J
DbC for Java.



Source: Jonas Bergström: Test by Contract presentation on Karlsruher Entwicklertag 2011

C4J (Contracts for Java) uses class load interception and instruments guarded classes.



Source: Jonas Bergström: Test by Contract presentation on Karlsruher Entwicklertag 2011

What is C4J – Contracts for Java?

C4J

Design By Contract for Java

Navigation

- Overview
- Ease of use
- Powerful
- Class invariant example
- Pre condition example
- Post condition example
- Interface example
- Target members
- Inheritance of contracts
- Under the hood
- Running
- Todo
- Downloads
- About

News

2011-04-02 - C4J 2.7.5

Overview

Contracts for Java (C4J) is a *Design By Contract* (DBC, see Wikipedia DBC definition and Eiffel DBC docs) framework for Java 1.5 and later. The primary goal for C4J is ease of use. DBC is about design and quality, aspects of programming that a lot programmers don't spend enough time and energy on.
Therefore a DBC framework must be simple and painless to use. At the same time the framework must be powerful.
C4J is simple *and* powerful.

I am not going to try to convince any readers that DBC indeed is a very powerful design and quality assurance technique, so if you are not already convinced of that, please follow the links above and you may be convinced to try this tool out! These are my two favorites though:

- To be able to define meaningful contracts you are forced to split lengthy methods in to small, well defined, methods with a single responsibility. This single "side effect" of DBC has very positive effects on the code quality, maintainability, and readability.
- Your tests are verified against the *real* usage of your application, not against some test cases that may not even be real use cases.
- If you are dealing with legacy code that you are afraid of refactoring, contracts are perfect to add to existing code with no risk involved.

Source: Jonas Bergström: http://c4j.sourceforge.net/

C4J is easy to use, as you express the contracts completely in Java.

Karlsruher Institut für Technologie

Ease of use

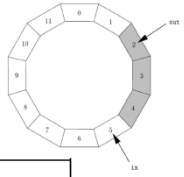
Adding class invariants, pre conditions, and post conditions to your regular Java objects is easy with C4J. Contracts for a class (the target) are implemented in a separate class (the contract). The contract is linked to the target at compile time using an annotation, and the actual contract verification code is tied to the target using instrumentation during class loading at run time.

There are several reasons for defining contracts in plain Java and in a separate class:

- You don't have to learn any new language constructs.
- You will get help from your IDE when implementing the contract (as opposed to implementing contracts in comment blocks).
- It is possible to put break points in your contract if you want to debug them.
- Separating the contract code from the application makes it possible to remove the risk of accidentally deploying the contracts together with your application.
- You get to keep your beautiful code nice and clean without all the ugly assertions.

Source: Jonas Bergström: http://c4j.sourceforge.net/

Keep your Code nice and clean! Gegenbeispiel: ModernJass.



```
@ModelDefinitions({
  @Model( name = "mTheBuffer", type = Object[].class),
  @Model( name = "mStoreIndex", type = Integer.class),
  @Model( name = "mExtractIndex", type = Integer.class)})
@InvariantDefinitions({
  @Invariant("size() >= 0"),
  @Invariant("(0 <= mStoreIndex - mExtractIndex) &&" +
      " (mStoreIndex - mExtractIndex <= mTheBuffer.length)") })
public interface BufferSpec<T> {

  @Pure
  @Post("@Result == mTheBuffer.length")
  public int size();

  @Pure
  @Post("@Result == (mStoreIndex == mExtractIndex)")
   public boolean empty();

  @Post("@Result == (mStoreIndex - mExtractIndex == mTheBuffer.length)")
  @Pure public boolean full();

  @Also({
    @SpecCase(
      pre = "o != null && !full()",
      post = "@Old(mStoreIndex) == mStoreIndex - 1"),
    @SpecCase(
      pre="o == null", signals = NullPointerException.class ) })
  public void add(@Name("o") T o);

  @Pre("!empty()")
  public T getNext();

  @SpecCase(
    pre="!empty()", post = "@Old(mExtractIndex) == mExtractIndex - 1")
  public T remove();

  @Pure
  @Post("@Result == @Exists(Object tmp : mTheBuffer;
    tmp != null && o.equals(tmp))")
  public boolean contains(@NonNull @Name("o") T o);

  @Post("@Result != null")
  public BufferSpec<T> copy();
}
```

Listing 7.1: Declaration and specification of the ring buffer interface.

```
public interface BufferSpec<T> {

  public int size();

  public boolean empty();

  public boolean full();

  public void add(T o);

  public T getNext();

  public T remove();

  public boolean constains(T o);

  public BufferSpec<T> copy();
}
```


Source: Johannes Rieken, Modern Jass – Design by Contract for Java, Master Thesis 2007

C4J respects LSP in an elaborate way.

C4J support full contract inheritance, both for target classes and for contracts themselves. If a class is protected by a contract, all extending classes will automatically enforce those contracts as well. Also, if a parent class is protected by a contract and an extending class is protected by another contract, the contracts will be combined for the extending class. Furthermore, contracts defined for super interfaces to interfaces that a class implements will also be enforced.

Contract inheritance in C4J works as follows:

- Class invariants are merged (AND'ed), except for in constructors where only the target class' own class invariants are verified.
- Pre conditions may not be strengthened, i.e. the contract in the extending class may not disallow a method call that the contract in the parent allows.
- Post conditions may not be weakened, i.e. the contract in the extending class may not allow more outcomes than the contract in the parent.
- Contracts defined on the same inheritance level, e.g. by implementing two interfaces which both has contracts for the same methods, are merged.

Source: Jonas Bergström: http://c4j.sourceforge.net/

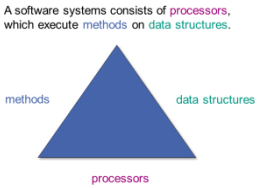
D3 – Agilität und Qualität im Software-Engineering
Agenda

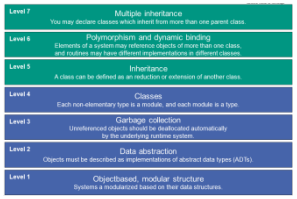
Karlsruher Institut für Technologie

1. Was versteht man konkret unter Objektorientierter Programmierung?
2. Ist Java eine vollständig objektorientierte Programmiersprache?
3. Wie können in Java Abstrakte Datentypen modelliert werden?
4. Was ist Design by Contract – und wie funktioniert es in Java?
5. Was verändert sich, wenn wir Design by Contract in Java nutzen?
6. Wie stehen DbC und TDD zueinander?
7. Fragen & Antworten

Design by Contract am KIT – ein Experiment über 3 Jahre hinweg mit 600 Studenten pro Jahr.

Programmieren I: Java im WS 2010-11
Aufbau der Vorlesung





Karlsruher Institut für Technologie

V01 V02 V03 V04 V05 V06 V07 V08 V09 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21

V00 : Organisatorisches

Prozedurales Programmieren

V01 P01 T01 1. – 4.1 Grundbegriffe
V02 4.2 Erste Schritte in Java
V03 P02 T02 4.3 Einfache Datentypen
4.4 Umgang mit einfachen Datentypen
V04 P03 T03 4.5 Ablaufsteuerung I
V05 4.5 Ablaufsteuerung II
V06 P04 T04 5.1 Referenzdatentypen und Felder
V07 6.1 Methoden
6.3 Die Methode **`main`**
V08 P05 T05 6.2 Rekursiv definierte Methoden

Objektbasiertes Programmieren

V09 A1 Die 7 Stufen zur Objektorientierung
A2 Modularität und Schnittstellen
V10 5.2 Klassen
6.4 Klassenmethoden
P07 T07 6.5 Instanzmethoden
12.2 Wrapper-Klassen (Hüllklassen)
V11 A3 Modul-Design mit C4J 2.6 – Design by Contract in Java
V12 P07 T07 A4 Modul-Test mit JUnit 4.8
A5 Einfach-verkettete Listen

Objektorientiertes Programmieren

V13 7. OOP-Philosophie
A6 Garbage Collection
V14 P08 T08 8. Klassen und Objekte
8.4.4 Mechanismus d. Objekterzeugung
V15 9.1 + 9.2 Vererbung
8.4.4 Vollständiger Mechanismus der Objekterzeugung mit Vererbung
V16 P09 T09 9.3 + 9.4 Polymorphismus
V17 9.6 Mehrfacherben u. Abstrakte Klassen
V18 9.8.1 Pakete
9.8.2 Zugriffsrechte
9.8.3 Innere Klassen
P10 T10 10. Exceptions und Errors
V19 11.1 Aufzählungstypen
11.2 Generische Datentypen
V20 A7 Listen –
P11 T11 ein Entwurf, drei Implementationen
A8 Binäre Bäume
V21 Ausblick: Parallele Programmierung

GFS Große Fragestunde zur Klausur

Evolution des Software-Engineerings am Beispiel der Klasse `Point`.

```
// Point
int x;
int y;
```

```
class Point
{
  int x;
  int y;
}
```

Point
- x : int
- y : int

+ «constructor» Point (int, int)
+ getX ( ) : int
+ getY ( ) : int
+ setX (int) : void
+ setY (int) : void

<<interface>> PointSpec

+ getX ( ) : int
+ getY ( ) : int
+ setX (int) : void
+ setY (int) : void

<<interface>> ObjectSpec

+ equals (Object) : boolean

PointSpecContract

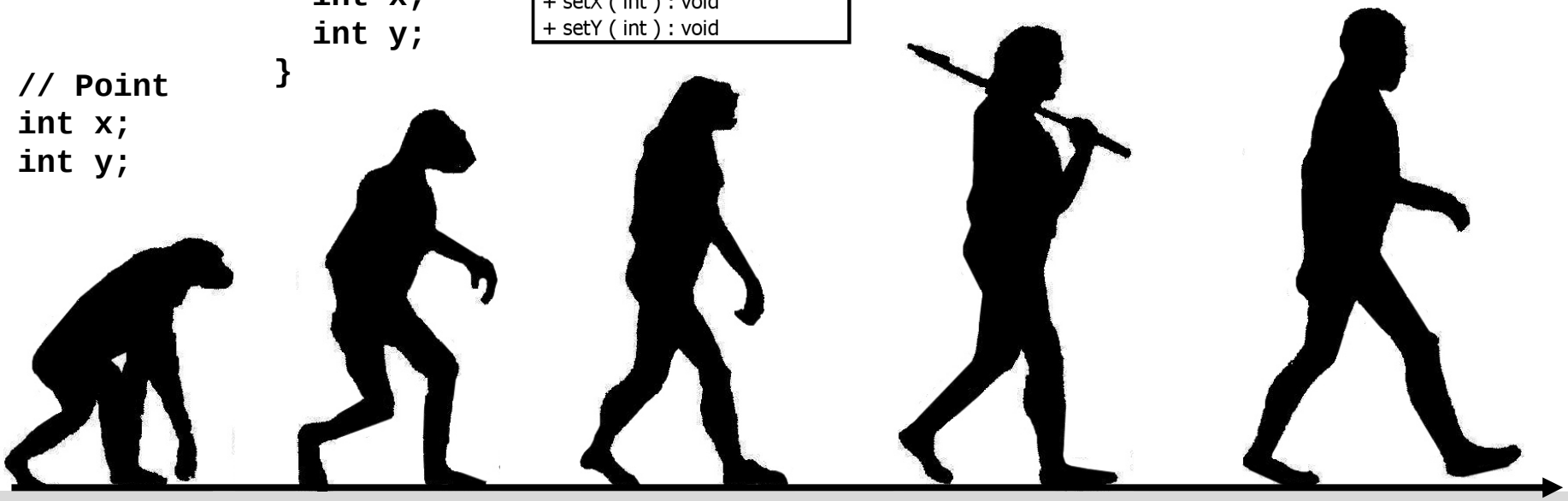
+ pre_setX (int) : void
+ post_setX (int) : void
+ pre_setY (int) : void
+ post_setY (int) : void

Point
- x : int
- y : int

+ «constructor» Point ( )
+ getX ( ) : int
+ getY ( ) : int
+ setX (int) : void
+ setY (int) : void
+ equals (Object) : boolean

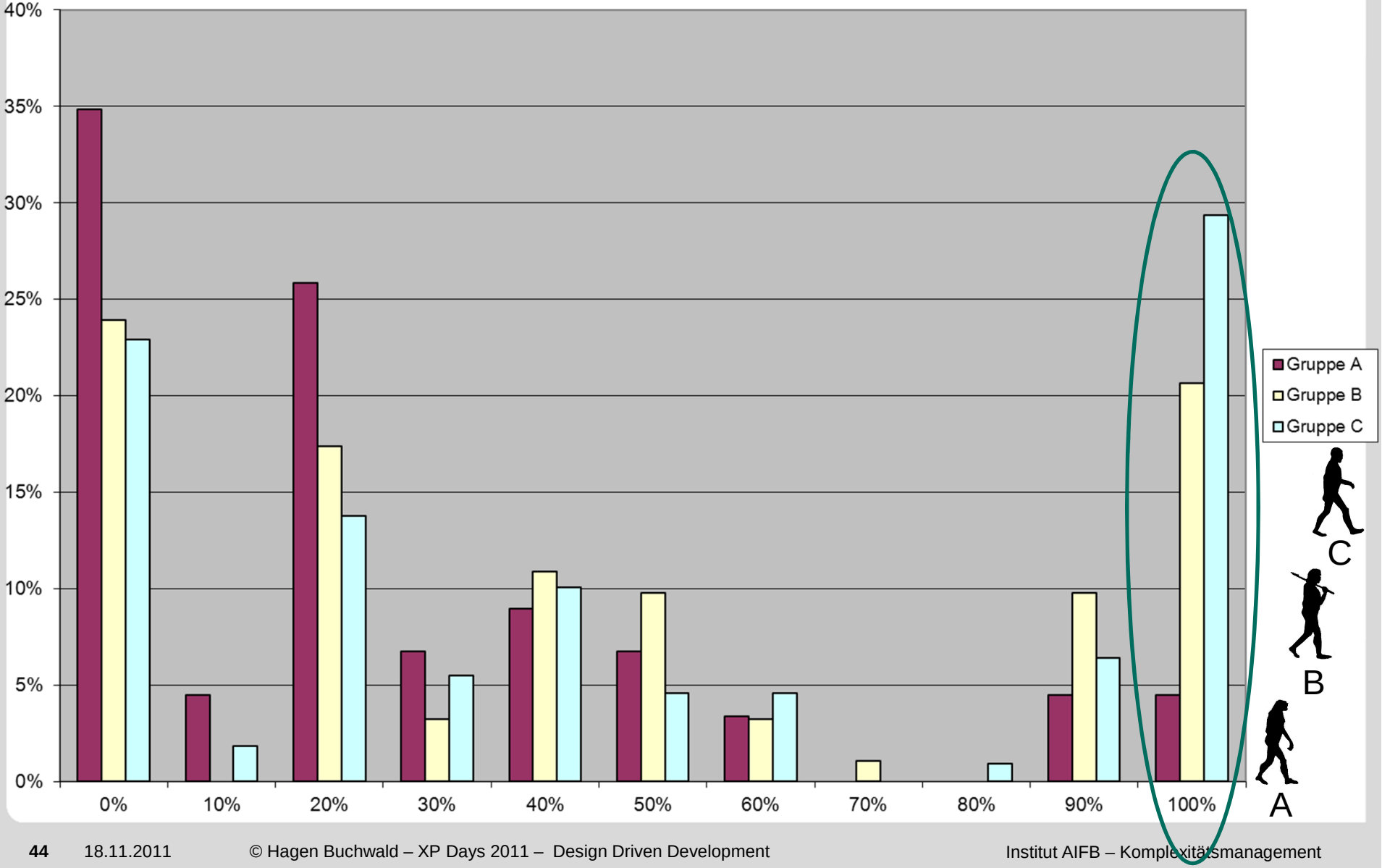
ObjectSpecContract
- z : Object

+ post_equals (Object) : void



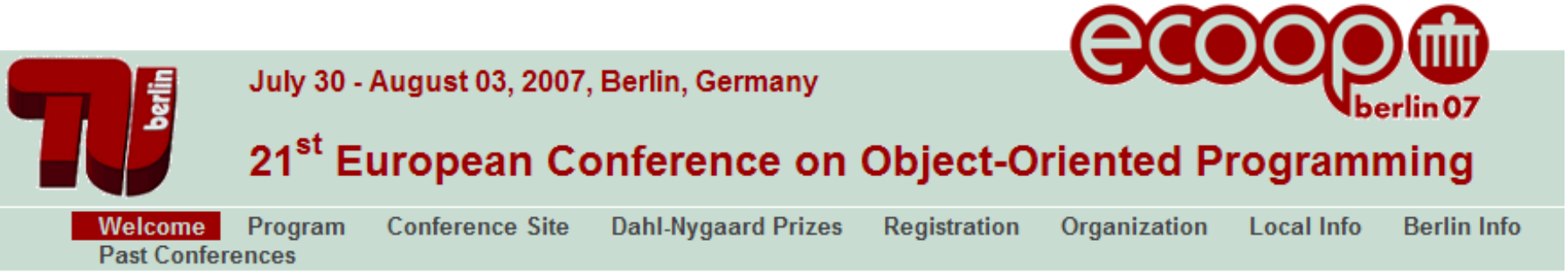
homo structurlos | homo methodicus | homo modulus | homo abstractus | homo objectus

Die Vervollständigung von Java um Ebene 2 (ADTs) steigert die Produktivität der Softwareentwicklung signifikant!



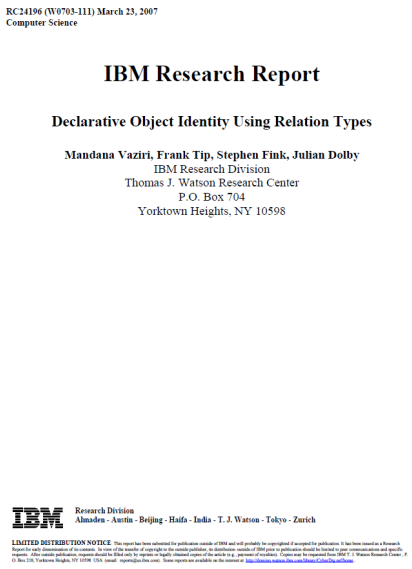
Which effects does DbC show in software development? Example: Implementing the `equals()` method in Java.

Karlsruher Institut für Technologie



„We found buggy or fragile **equals()** methods in nearly every Java application that we examined.“

Mandana Vaziri
IBM Research



RC24196 (W0703-111) March 23, 2007
Computer Science

IBM Research Report

Declarative Object Identity Using Relation Types

Mandana Vaziri, Frank Tip, Stephen Fink, Julian Dolby
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

IBM Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Copies may be requested from IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home.

Quelle: Declarative Object Identity Using Relation Types, IBM Research Report, RC24196, 2007

Which effects does DbC show in software development? Example: Implementing the `equals()` method in Java.

- Article of Angelika Langer, 2002
- Implementing the equals()-Method

- Article of Martin Odersky, 2009
- How to write an Equality-Method in Java

Angelika Langer - Training & Consulting

HOME | SEMINARS | TALKS | ARTICLES | BOOKS | LINKS | IOSTREAMS | GENERICS | ABOUT | NEWSLETTER | CONTACT | SITEMAP

HOME
SEMINARS
TALKS

ARTICLES
OVERVIEW

BY TOPIC
JAVA
C++

BY COLUMN
EFFECTIVE JAVA
EFFECTIVE STDLIB

BY MAGAZINE
JAVA MAGAZIN
JAVA SPEKTRUM
JAVA WORLD
JAVA SOLUTIONS
JAVA PRO
C++ REPORT
CUJ
OBJEKT SPEKTRUM

OTHER

BOOKS
LINKS
IOSTREAMS
GENERICS
ABOUT
NEWSLETTER
CONTACT
SITEMAP

Implementing the equals() Method - Part 1

Objektvergleich
Wie, wann und warum implementiert man die equals()-Methode?

Teil 1: Die Prinzipien der Implementierung von equals()

JavaSPEKTRUM, Januar 2002
Klaus Kreft & Angelika Langer

Dies ist das Manuskript eines Artikels, der im Rahmen einer Kolumne mit dem Titel "Effective Java" im JavaSPEKTRUM erschienen ist. Die übrigen Artikel dieser Serie sind ebenfalls verfügbar (click here).

Vorbemerkung

Mit diesem Artikel beginnen wir unter dem Titel "Effective Java" eine Kolumne, die sich mit der Programmiersprache Java auseinandersetzen wird. Wir haben dabei bewusst den Titel "Effective Java" gewählt, um an die Tradition von Scott Meyers anzuknüpfen, der den Begriff "Effective..." durch seine "Effective C++"-Bücher populär gemacht hat. Betrachtungen unter dem Motto "Effective" wenden sich typischerweise der Tücke des Objekts zu und so werden wir uns in dieser Kolumne den mehr oder weniger offensichtlichen Fallstricken der Programmiersprache Java widmen.

Nun werden unter dem Begriff "Java" unzählige Aspekte subsummiert, von Realtime-Programmierung unter speziellen virtuellen Maschinen über GUI-Programmierung bis hin zur Applikationsentwicklung auf Basis von EJB und JSP. Es wäre vermessen, über "Java" in dieser Gesamtheit schreiben zu wollen. Wir beschränken uns daher bewusst auf den Kern von Java: die Programmiersprache selbst, wesentliche Aspekte der virtuellen Maschine und einige grundlegende APIs aus den Bibliotheken der Java 2 Standard Edition (J2SE). Unser Ziel ist es, über genau den Teil von Java zu schreiben, der jeden Java-Programmierer angeht, ganz egal in welcher Domain er oder sie programmiert. Dabei wollen wir uns die weniger offensichtlichen und bisweilen überraschenden Effekte in Java ansehen.

Objekt-Infrastruktur in Java

Beginnen wir mit scheinbaren Trivialitäten wie "Kopieren von Objekten" und "Vergleichen von Objekten". Gemeint sind die Methoden clone(), equals() und einige andere, die zusammen so etwas wie die "Infrastruktur" eines Objekts ausmachen. Was meinen wir mit "Infrastruktur"?

Alle Klassen in Java sind implizit von der Klasse Object abgeleitet und erben daher alle Methoden aus Object. Zu diesen geerbten Methoden gehören die public Methoden equals() und hashCode(). equals() vergleicht zwei Objekte miteinander, während hashCode() einen integralen Wert (den sogenannten Hashcode) berechnet. Mit den Details dieser Methoden werden wir uns in diesem und den folgenden Artikeln noch eingehend beschäftigen. An dieser Stelle nur soviel: beide Methoden werden u.a. gebraucht, um Java-Objekte in hash-basierten Containern wie beispielsweise HashSet ablegen zu können.

Wegen der automatischen Ableitung von der Superklasse Object sind equals() und hashCode() Teil der Schnittstelle einer jeder Java-Klasse, d.h. man kann auf allen Objekten in Java equals() und hashCode() aufrufen. Es gibt auch immer eine Implementierung dieser Methoden, nämlich entweder die aus Object geerbte Default-Implementierung oder eine klassenspezifische Implementierung, wenn die betreffende Klasse die geerbte Methode überschrieben hat.

Methoden wie equals() und hashCode() stellen Basisfunktionalität zur Verfügung, die man von allen Objekten in Java erwartet. Die Menge der Basisfunktionalitäten bezeichnet man bisweilen als "Infrastruktur" eines Objekts. Zur Infrastruktur gehören nicht nur equals() und hashCode(), sondern auch Funktionalität für Initialisierung und Aufräumen von Objekten sowie für Kopieren und Vergleichen von Objekten. Initialisierung geschieht üblicherweise mittels Konstruktoren, Aufräumen mittels finalize()-Methode, Kopieren mittels clone()-Methode, Vergleichen mittels equals() und compareTo()-Methode.

How to Write an Equality Method in Java
by Martin Odersky, Lex Spoon, and Bill Venners
June 1, 2009

Summary
This article describes a technique for overriding the `equals` method that preserves the contract of `equals` even when subclassses of concrete classes add new fields.

ADVERTISEMENT
Java Performance Tuning Workshop
An in-depth training course taught by Kirk Pepperdine
Four days: May 2 - 5, 2011
Oakland, California
Register Now!

In Item 8 of *Effective Java*[1], Josh Bloch describes the difficulty of preserving the `equals` contract when subclassing as a "fundamental problem of equivalence relations in object-oriented languages." Bloch writes:

> *There is no way to extend an instantiable class and add a value component while preserving the equals contract, unless you are willing to forgo the benefits of object-oriented abstraction.*

Chapter 28 of *Programming in Scala* shows an approach that allows subclasses to extend an instantiable class, add a value component, and nevertheless preserve the `equals` contract. Although in the technique is described in the book in the context of defining Scala classes, it is also applicable to classes defined in Java. In this article, we present the technique using text adapted from the relevant section of *Programming in Scala*, but with the code examples translated from Scala into Java.

Commmon equality pitfalls

Class `java.lang.Object` defines an `equals` method, which subclasses may override. Unfortunately, it turns out that writing a correct equality method is surprisingly difficult in object-oriented languages. In fact, after studying a large body of Java code, the authors of a 2007 paper concluded that almost all implementations of `equals` methods are faulty.[2]

This is problematic, because equality is at the basis of many other things. For one, a faulty equality method for a type `C` might mean that you cannot reliably put an object of type `C` in a collection. You might have two elements `elem1`, `elem2` of type `C` that are equal, *i.e.*, `*elem1.equals(elem2)*` yields `true`. Nevertheless, with commonly occurring faulty implementations of the `equals` method, you could still see behavior like the following:

```
set<C> hashSet = new java.util.Hashset<C>();
hashset.add(elem1);
hashset.contains(elem2);    // returns false!
```

Here are four common pitfalls[3] that can cause inconsistent behavior when overriding `equals`:

1. Defining `equals` with the wrong signature.
2. Changing `equals` without also changing `hashCode`.
3. Defining `equals` in terms of mutable fields.
4. Failing to define `equals` as an equivalence relation.

These four pitfalls are discussed in the remainder of this section.

Pitfall #1: Defining `equals` with the wrong signature.

Consider adding an equality method to the following class of simple points:

```
public class Point {

    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
```

Which effects does DbC show in software development? Example: Implementing the `equals()` method in Java.

Karlsruher Institut für Technologie

```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

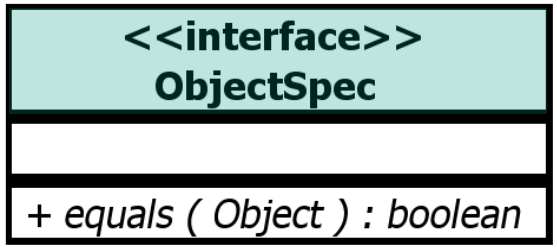
Class `Object` as abstract data type (ADT)

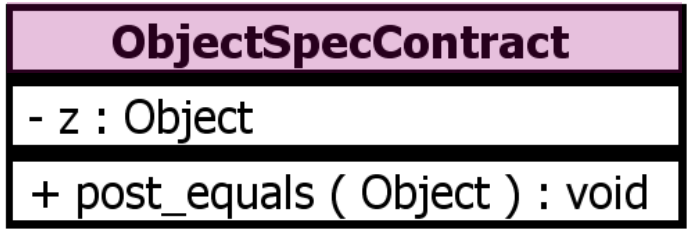
```
TYPE
    ObjectSpec

FUNCTIONS
    equals(Object x) : boolean


PRE-CONDITIONS
    ---

AXIOME (POST-CONDITIONS)
    equals(Object x) : if x != null
                       then x.equals(x)
    equals(Object x) : if x != null
                       then x.equals(y) == y.equals(x)
    equals(Object x) : if x != null
                       then if x.equals(y) and y.equals(z)
                            then x.equals(z)
    equals(Object x) : if x != null
                       then x.equals(y) == x.equals(y)
    equals(Object x) : if x == null
                       then result == false
```





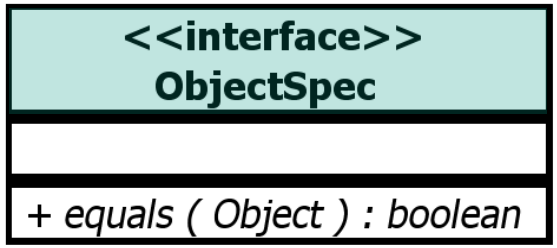
The abstract data type (ADT) `Object` expressed in `Java`.

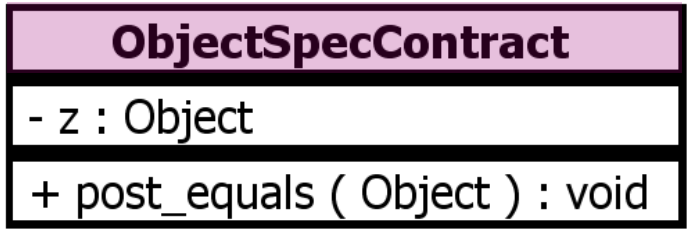
```
import net.sourceforge.c4j.*;
@ContractReference(contractClassName = "ObjectSpecCor
public interface ObjectSpec
{
    @Pure boolean equals(Object obj);
}

import net.sourceforge.c4j.ContractBase;
public class ObjectSpecContrac
             extends ContractBase<ObjectSpec>
{
    public void post_equals (Object obj)
    {
        …
        if (obj != null)
        {
            assert x.equals(x) : "is_reflexive";

            assert x.equals(y) == y.equals(x) : "is_symmetric";

            if (x.equals(y) && y.equals(z)) {
                assert x.equals(z) : "is_transitive";
            }
            assert x.equals(y) == x.equals(y) : "is_consistent_with_equals";
        }
        else {
            assert result == false : "if_null_then_false";
        }
    }
}
```





The abstract data type (ADT) `Object` expressed in `Java`.

```
public void post_equals (Object obj)
{
    boolean result = (Boolean) getReturnValue();
    Object x = m_target;
    Object y = obj;
    if (z == null) {
        z = obj;
    }

    if (obj != null)
    {
        assert x.equals(x) : "is_reflexive";

        assert x.equals(y) == y.equals(x) : "is_symmetric";

        if (x.equals(y) && y.equals(z)) {
            assert x.equals(z) : "is_transitive";
        }
        assert x.equals(y) == x.equals(y) : "is_consistent_with_equals";
    }
    else {
        assert result == false : "if_null_then_false";
    }
}
```

ObjectSpecContract
- z : Object
+ post_equals (Object) : void

The C4J based EJE Plugin D3-Explorer for KIT students integrates DbC for Java in an effective and efficient way.

Karlsruher Institut für Technologie

EJE - Editing Java Easily 6.0 (Corporate Academic License)

Datei Bearbeiten Java Tools Plugins Hilfe

Stack.java

```
1  import net.sourceforg
2  import java.util.Arra
3
4  @ContractReference(co
5  public class Stack<T>
6  {
7      private int capac
8      private ArrayList
9
10     public Stack (int
11     {
12         this.capacity
13         values = new
14     }
15
16     @Override
17     public boolean is
18     {
19         boolean resul
20         result = coun
21         return result
22     }
23
24     @Override
```

Konsolenfenster

```
Compiliere ...

Programm compiliert.
```

Projekt-Explorer

- Stack6
  - (default source folder)
    - (default package)
      - Stack
      - StackContract
      - StackSpec
      - StackSpecContract
      - StackTest

Interface	Interface...	Class	ClassCon...	ClassTest
Stack6				
package (default package)				
StackSpec	StackSpec...	Stack	StackContr...	StackTest
StackSpec	StackSpec...	Stack	StackContr...	StackTest
StackSpec	StackSpec...	Stack	StackContr...	StackTest
StackSpec	StackSpec...	Stack	StackContr...	StackTest
StackSpec	StackSpec...	Stack	StackContr...	StackTest

Fertig

Fertig ... | 1:30 | 06.07.2011 14:17:59

Source: Fabian Rigterink, KIT student, Bachelor thesis

The C4J based Eclipse Plugin D3-Explorer for professionals integrates DbC for Java in an effective and efficient way.

Karlsruher Institut für Technologie

Java - Test02/src/test/java/projekt/util/StackTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer | JUnit

Finished after 0,078 seconds

Runs: 9/9 Errors: 0 Failures: 0

projekt.util.StackTest [Runner: JUnit 4] (0

- StackTestRight (0,000 s)
- isFullTestRight (0,000 s)
- capacityTestRight (0,000 s)
- addTestErrorCondition1 (0,000 s)
- addTestErrorCondition2 (0,000 s)
- addTestRight (0,000 s)
- removeTestRight (0,000 s)
- getTestRight (0,000 s)
- countTestRight (0,000 s)

Failure Trace

StackSpecContract.ja | Stack.java | StackContract.java | StackTest.java

```
    @Before
    public void setUpTest ()
    {
        // TODO Auto-generated method stub
        m_target = new Stack<Integer>(C);
    }

    @After
    public void tearDownTest ()
    {
        // TODO Auto-generated method stub
        m_target = null;
    }

    @Test
```

Task List

Find | All | Activate...

Uncategorized

Connect Mylyn

Connect to your task and ALM tools.

Outline

- projekt.util
- import declarations
  - org.junit.*
  - org.junit.Assert.*
- StackTest
  - m_target : Stack<Integer>

Problems | Javadoc | Declaration | Console | Tasks | D3-Explorer

	Interface		InterfaceContract		Class		ClassContract		ClassTest	
Test02	Test02									
src/main/design										
projekt.util	package projekt.util									
StackSpec	StackSpec		StackSpecContract		Stack		StackContract		StackTest	
src/main/java										
projekt.util	package projekt.util									
src/test/java										
projekt.util	package projekt.util									

Done

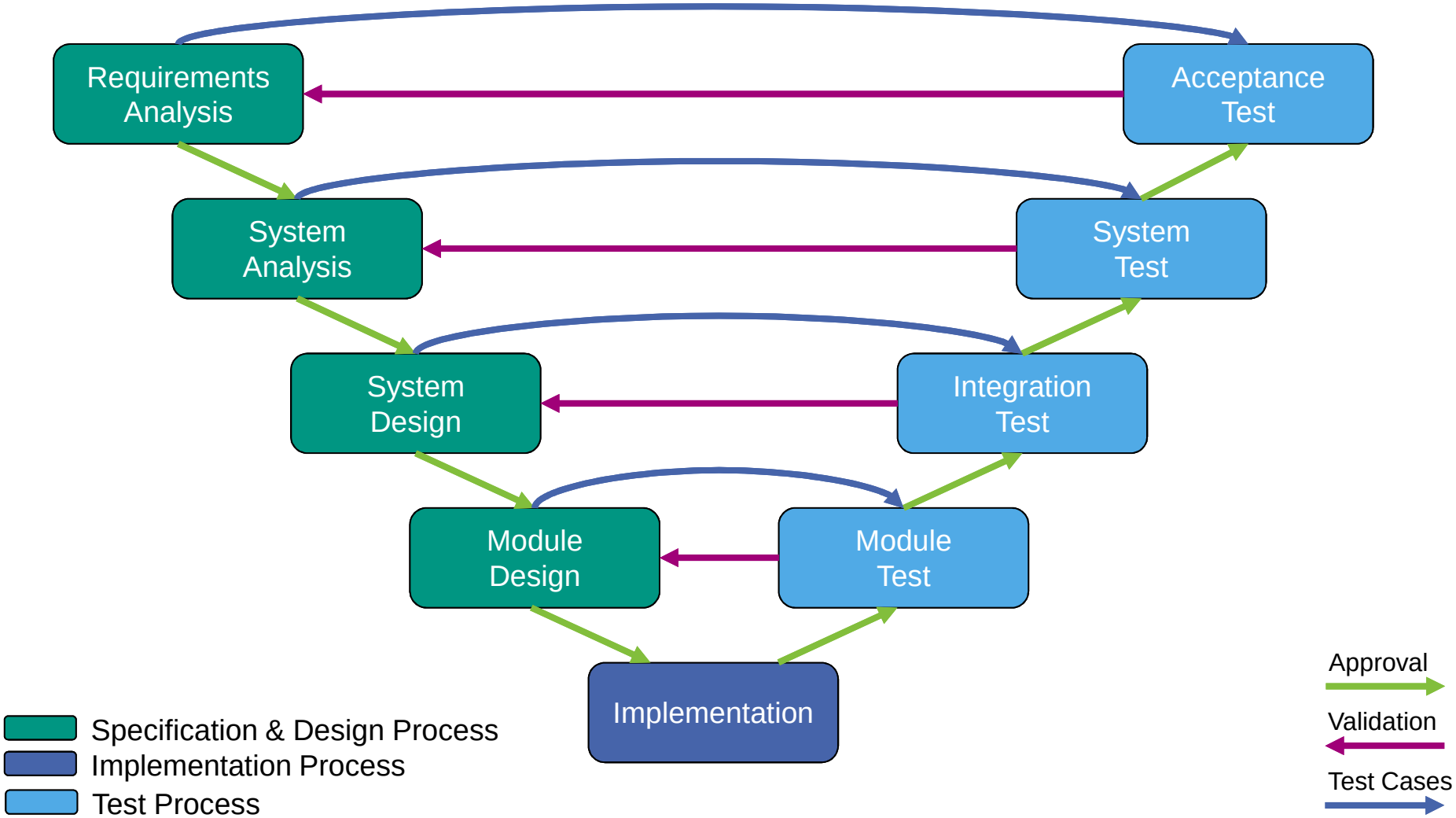
Source: Yana Stoeva, KIT student, Diploma thesis

D3 – Agilität und Qualität im Software-Engineering
Agenda

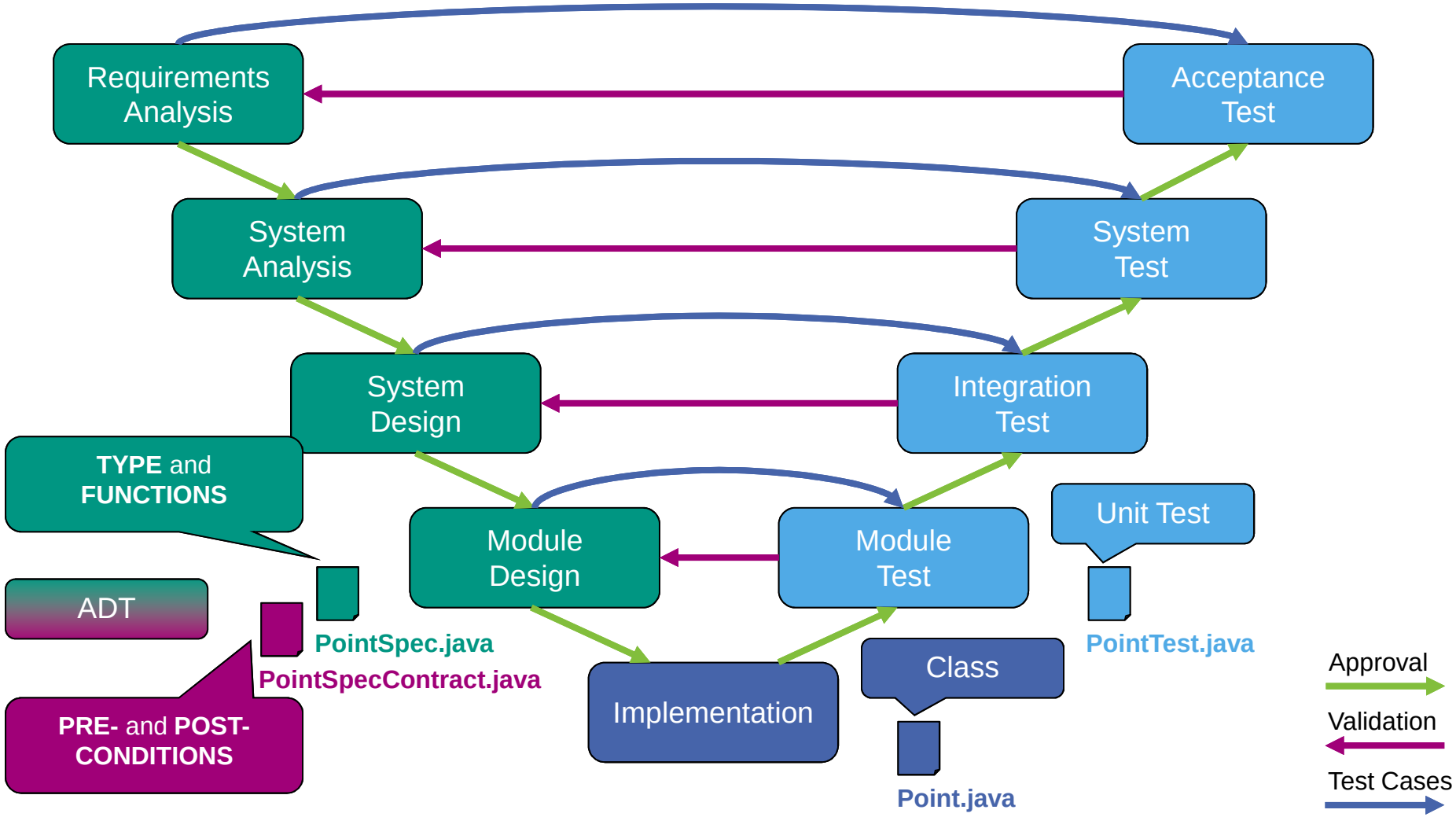
Karlsruher Institut für Technologie

1. Was versteht man konkret unter Objektorientierter Programmierung?
2. Ist Java eine vollständig objektorientierte Programmiersprache?
3. Wie können in Java Abstrakte Datentypen modelliert werden?
4. Was ist Design by Contract – und wie funktioniert es in Java?
5. Was verändert sich, wenn wir Design by Contract in Java nutzen?
6. Wie stehen DbC und TDD zueinander?
7. Fragen & Antworten

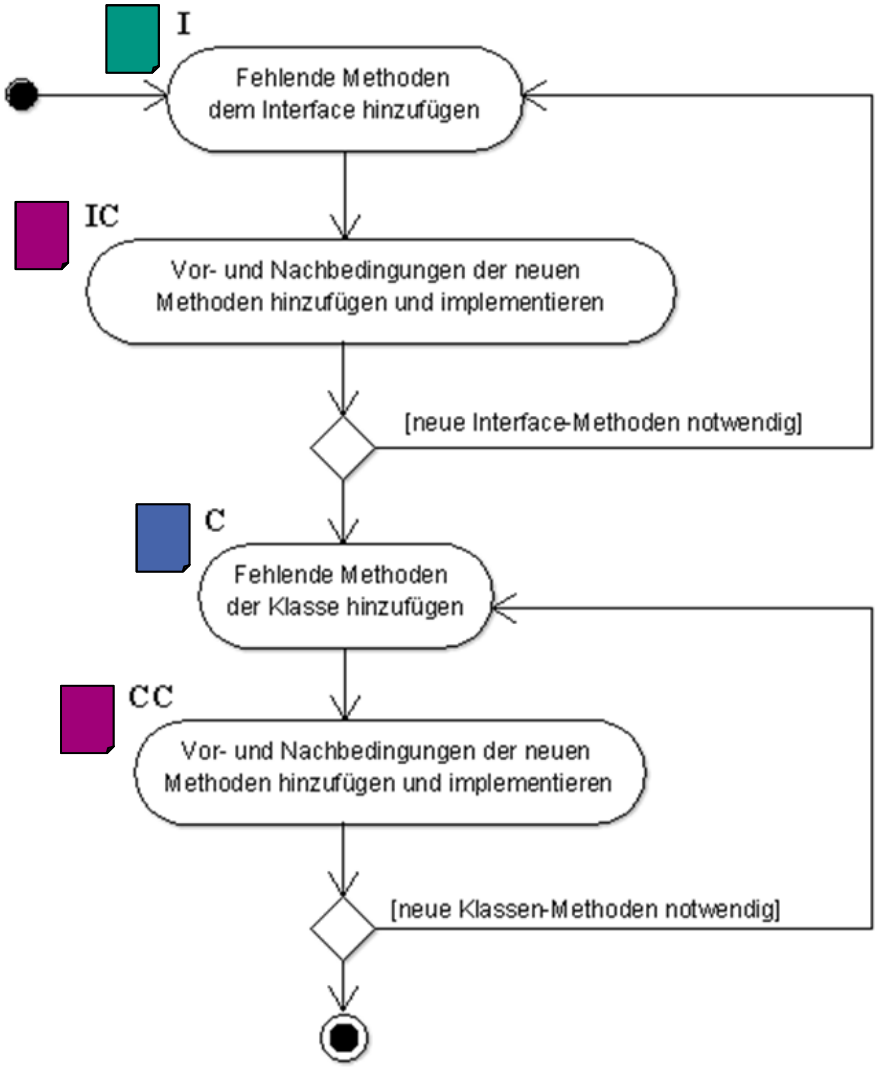
Which requirements does *DbC for Java* raise on the methods level?



Die Verbindung des Design by Contract Zyklus mit dem TDD Zyklus führt zu stabilen Interfaces und Implementierungen.

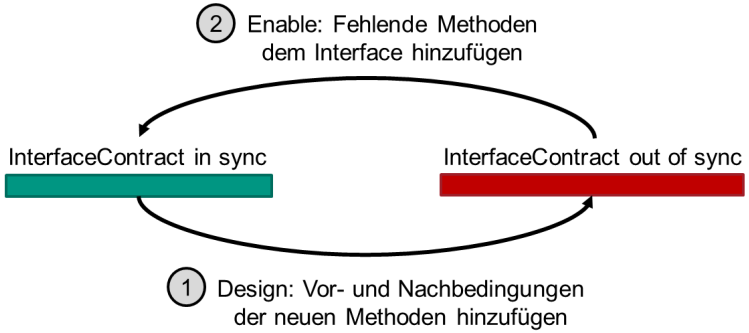


Der DbC-*design/enable-Zyklus* spezifiziert eine Klasse anhand ihres Interfaces und InterfaceContracts.



Design-Phase

I = Interface
IC = InterfaceContract
C = Class
CC = ClassContract
CT = ClassTest

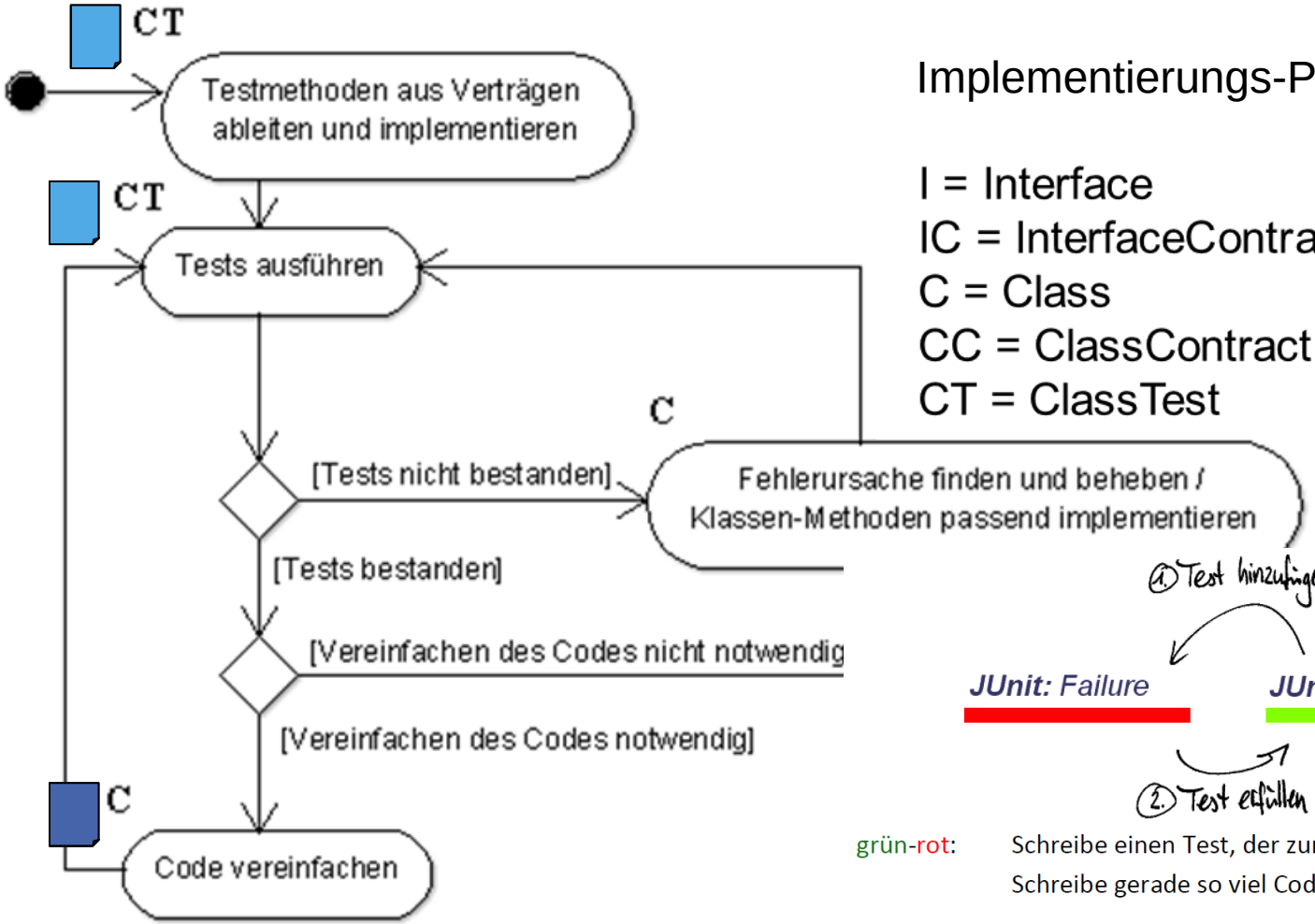


grün–rot : Deklariere eine neue Methode im Interface und füge die zugehörigen Vor- und Nachbedingungen dem InterfaceContract hinzu.

rot–grün : Deklariere die durch die neuen Vertragsklauseln notwendig gewordenen Methoden als neue Methoden im Interface.

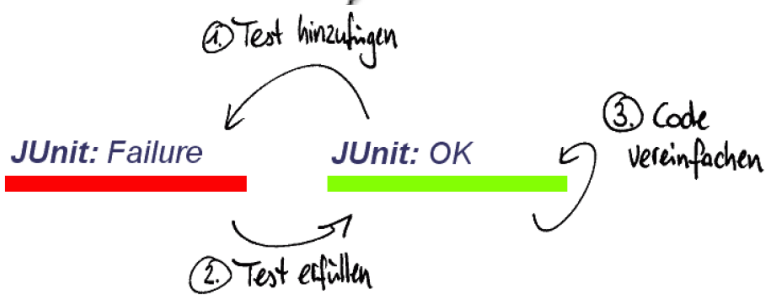
Derived from: Yana Stoeva, KIT student, Diploma thesis

Der TDD-*test/code/refactor-Zyklus* treibt die Implementierung der Klasse anhand ihres Unit-Tests.



Implementierungs-Phase

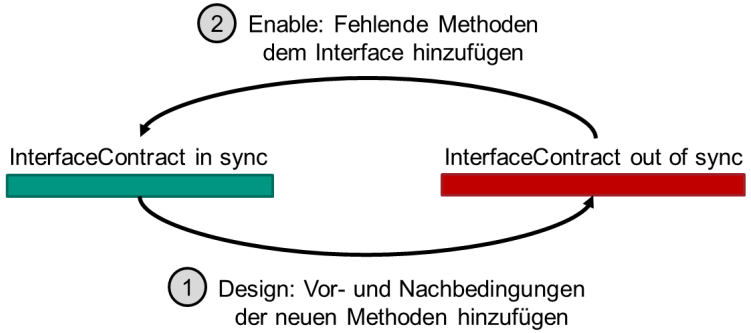
I = Interface
IC = InterfaceContract
C = Class
CC = ClassContract
CT = ClassTest

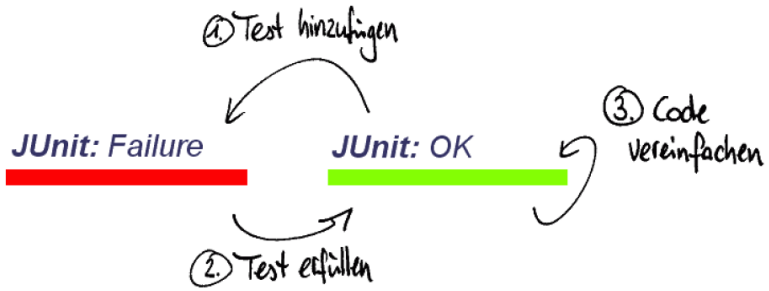


grün-rot: Schreibe einen Test, der zunächst fehlschlagen sollte.
Schreibe gerade so viel Code, dass der Test kompiliert.

rot-grün: Schreibe gerade so viel Code, dass alle Tests laufen.

D3 ist die Kombination des DbC-*design/enable-Zyklus* mit dem TDD- *test/code/refactor-Zyklus*.



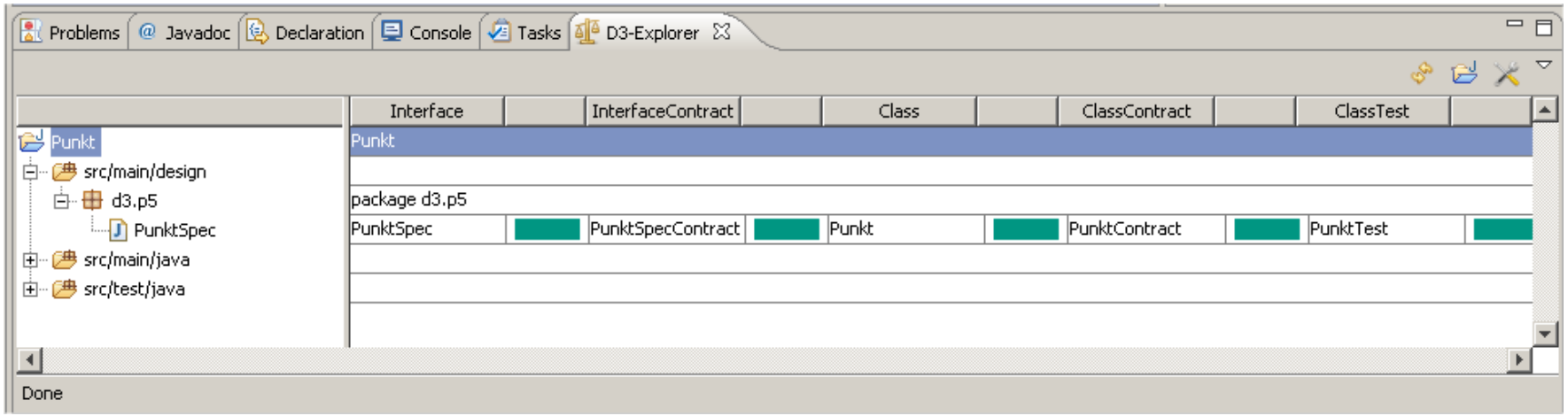


grün–rot : Deklariere eine neue Methode im Interface und füge die zugehörigen Vor- und Nachbedingungen dem InterfaceContract hinzu.

rot–grün : Deklariere die durch die neuen Vertragsklauseln notwendig gewordenen Methoden als neue Methoden im Interface.

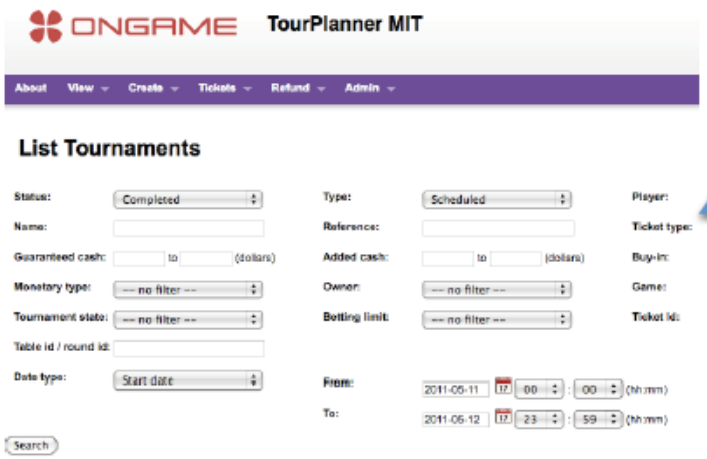
grün-rot: Schreibe einen Test, der zunächst fehlschlagen sollte. Schreibe gerade so viel Code, dass der Test kompiliert.

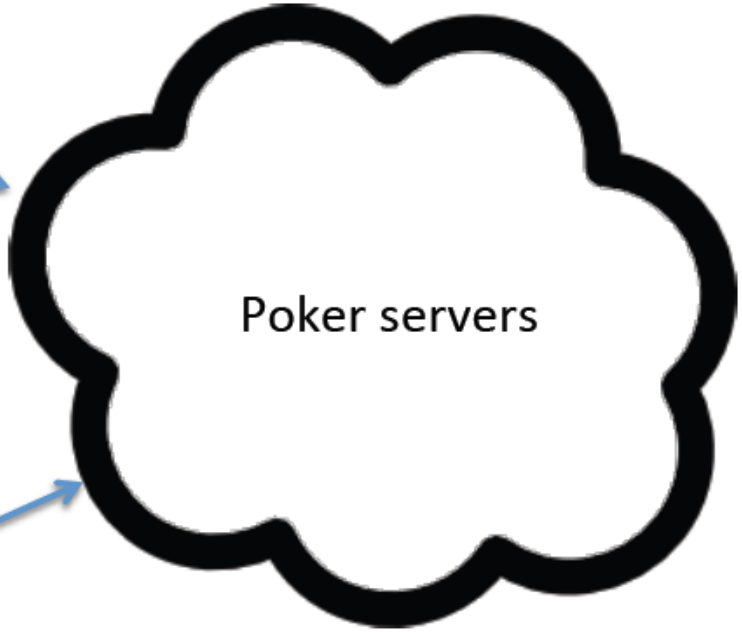
rot-grün: Schreibe gerade so viel Code, dass alle Tests laufen.



Was hat Poker mit Design by Contract zu tun?

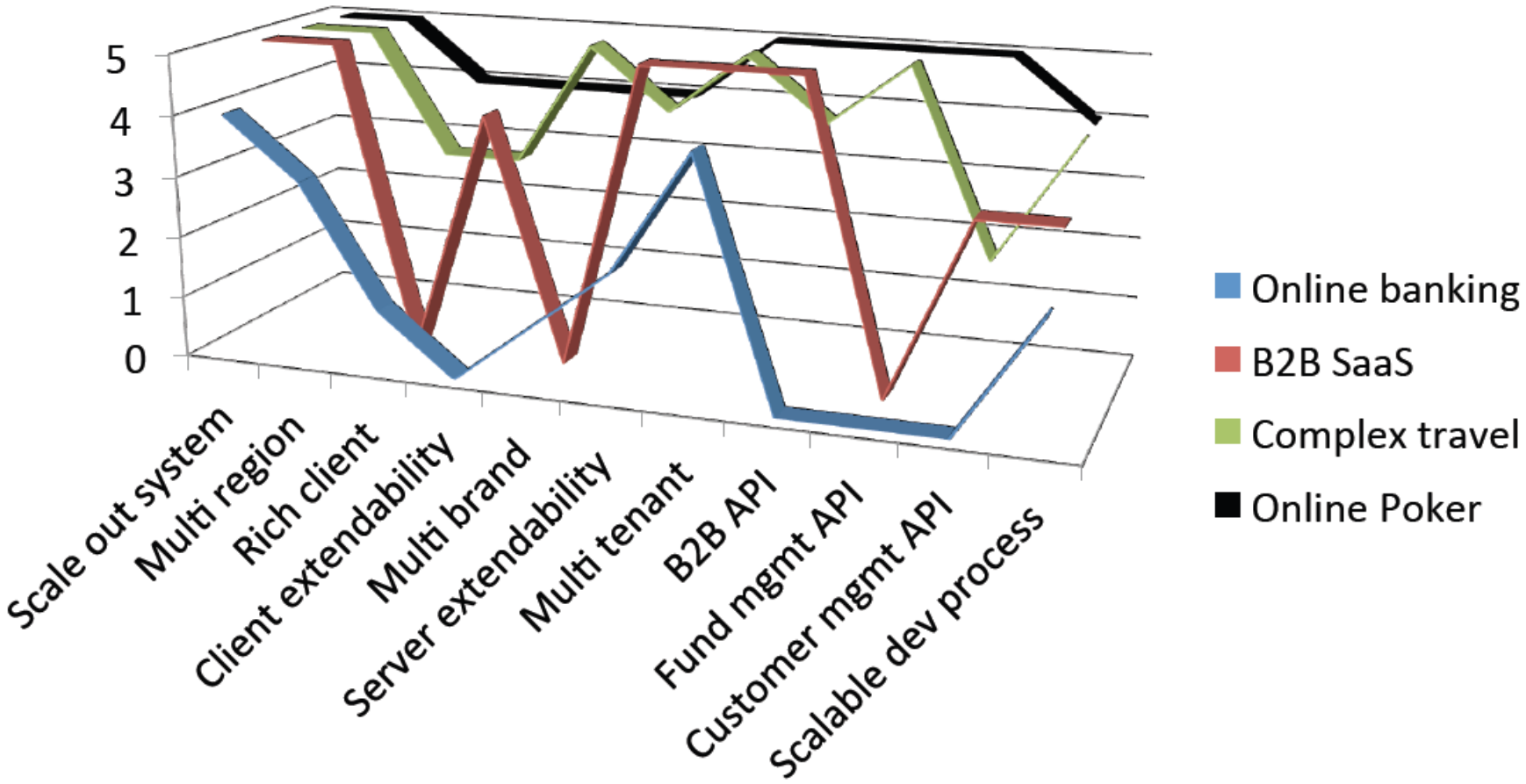
Was hat Poker mit Design by Contract zu tun?





Source: Jonas Bergström: Vortrag auf dem Karlsruher Entwicklertag 2011

Die Anforderungen an ein white label online Poker-System sind sehr hoch!



Source: Jonas Bergström: Vortrag auf dem Karlsruher Entwicklertag 2011

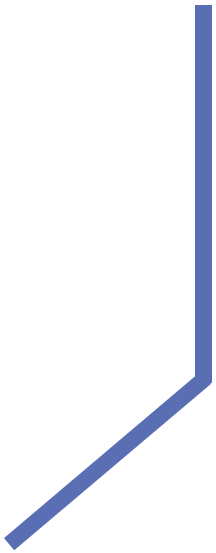
PokerBots simulieren Spieler und testen die Poker-Engine bei aktivierten Verträgen!

Source: Jonas Bergström: Vortrag auf dem Karlsruher Entwicklertag 2011

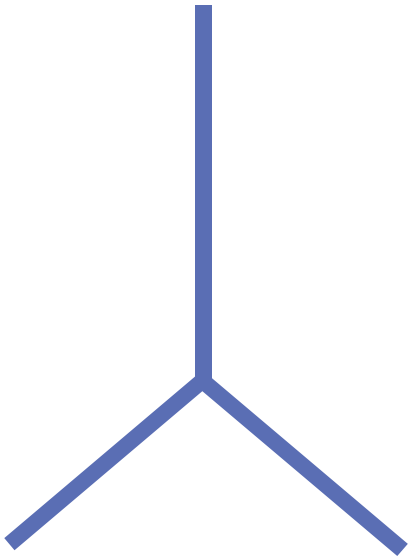
Warum sind diese Symbole so wichtig für die Software-Qualität?

Boundary Condition

Edge Condition



Corner Condition



D3 – Agilität und Qualität im Software-Engineering
Agenda

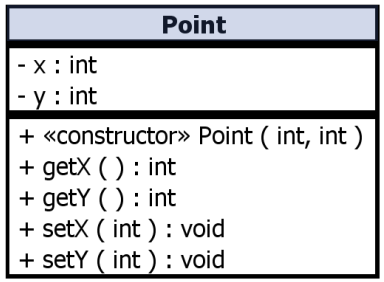
Karlsruher Institut für Technologie

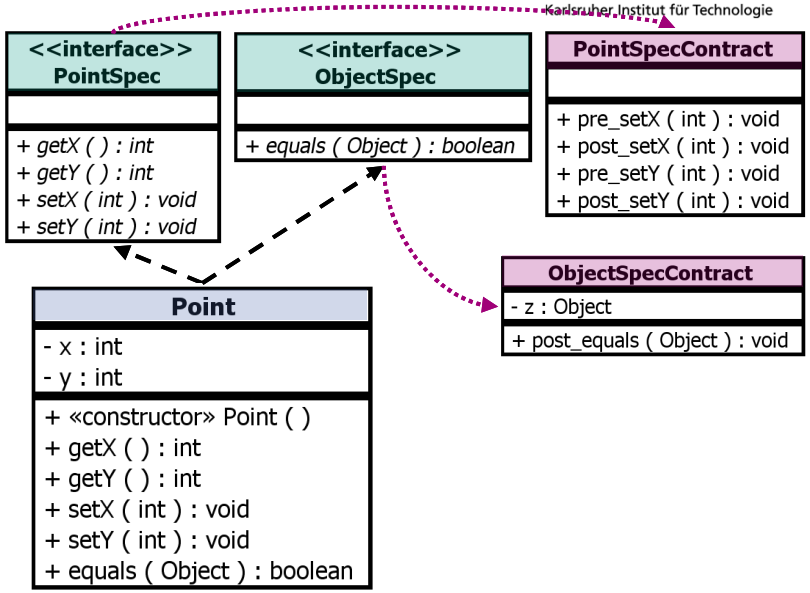
1. Was versteht man konkret unter Objektorientierter Programmierung?
2. Ist Java eine vollständig objektorientierte Programmiersprache?
3. Wie können in Java Abstrakte Datentypen modelliert werden?
4. Was ist Design by Contract – und wie funktioniert es in Java?
5. Was verändert sich, wenn wir Design by Contract in Java nutzen?
6. Wie stehen DbC und TDD zueinander?
7. Fragen & Antworten

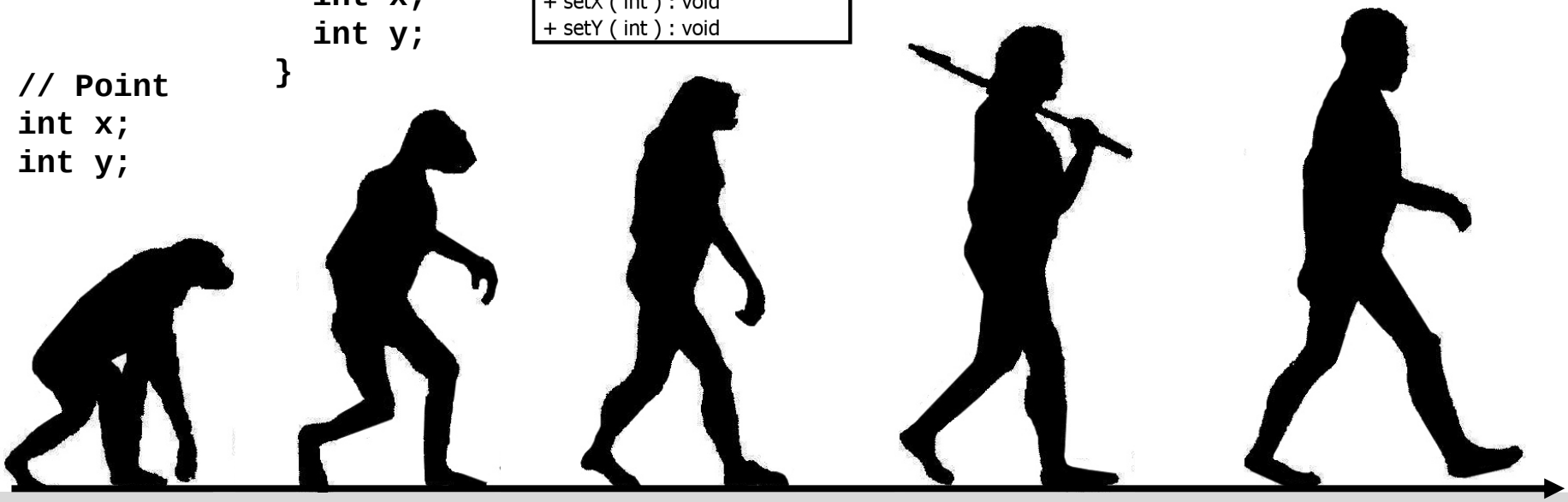
Fragen & Antworten

```
// Point
int x;
int y;
```

```
class Point
{
   int x;
   int y;
}
```







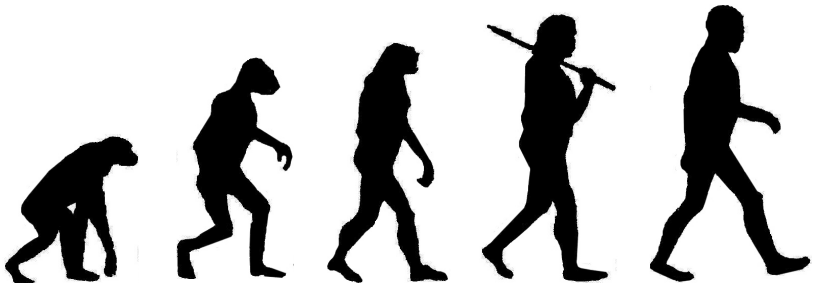
homo structurlos | homo methodicus | homo modulus | homo abstractus | homo objectus

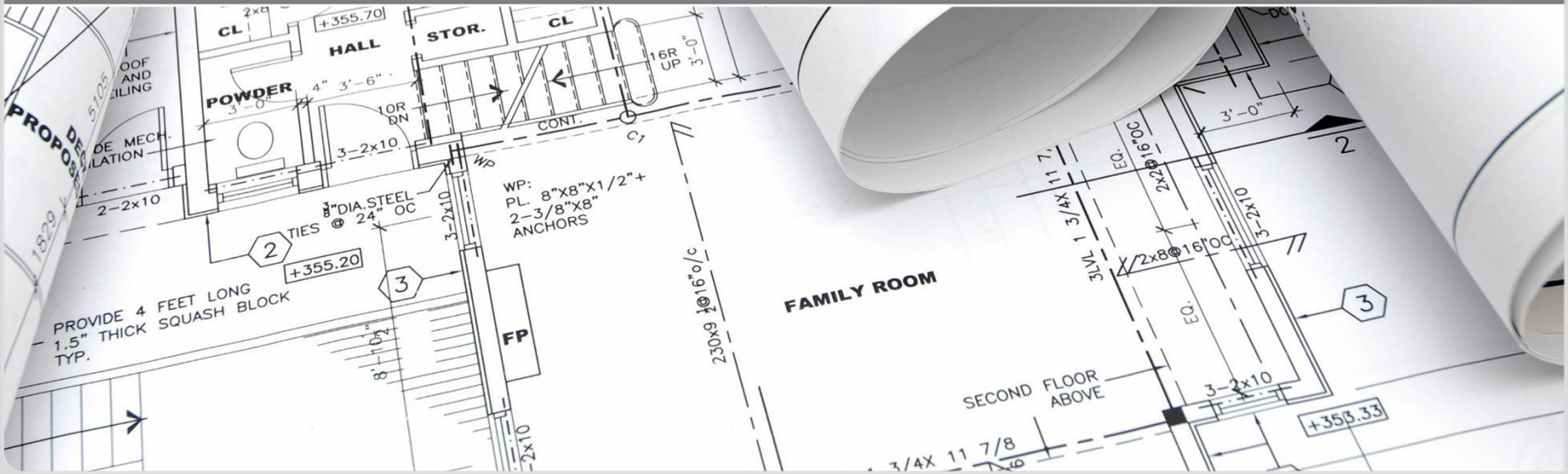
Karlsruhe Institute of Technology

VKSI Verein der Karlsruher Software-Ingenieure

Design Driven Development (D3)

Agilität und Qualität im SE



Karlsruhe, 18. November 2011
Hagen Buchwald

Institut AIFB – KOMPLEXITÄTSMANAGEMENT