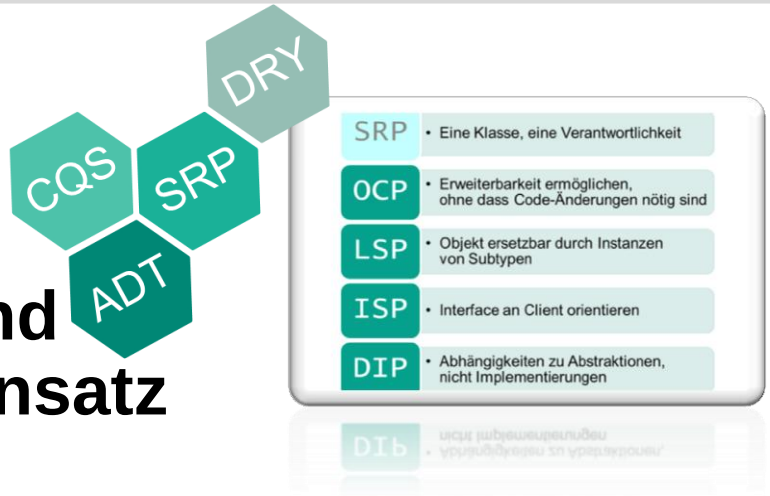
VKSI Verein der Karlsruher Software-Ingenieure



Typen in Java mit Interfaces und Unit Contracts - ein SOLIDer Ansatz

Clean Code Days 2017
Hagen Buchwald, Lars Alvincz
andrena objects

VKSI SIG C4J - Special Interest Group „Contracts for Java“ des Vereins Karlsruher Software Ingenieure

Inhaltsübersicht
Unit Contracts – ein SOLIDer Ansatz

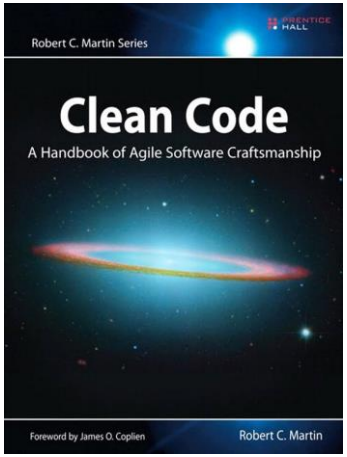
Was ist Clean Code?

Bob Martin: Smells & Heuristics

Meaningful Names

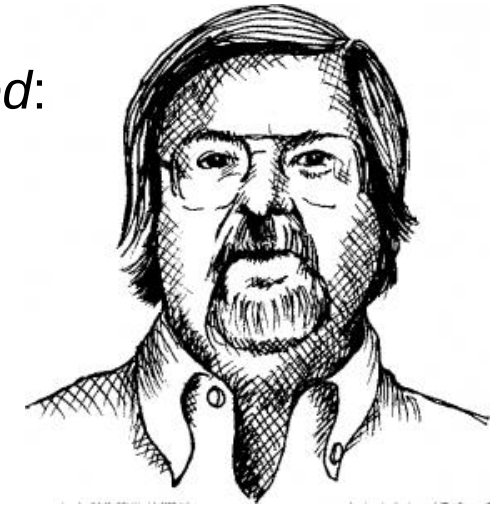
DRY – Don‘t Repeat Yourself!

CQS – Command Query Separation



Ron Jeffries, author of *Extreme Programming Installed*:

- *Abstraction often calls my attention to what‘s „really“ going on.*
- *Reduced duplication, high expressiveness, and early building of simple abstractions: That‘s what makes clean code for me.*



Die 5 SOLID-Prinzipien

SRP	• Eine Klasse, eine Verantwortlichkeit
OCP	• Erweiterbarkeit ermöglichen, ohne dass Code-Änderungen nötig sind
LSP	• Objekt ersetzbar durch Instanzen von Subtypen
ISP	• Interface an Client orientieren
DIP	• Abhängigkeiten zu Abstraktionen, nicht Implementierungen

Quelle: Bob Martin, z.B. Design Principles and Design Patterns (2000)

Inhaltsübersicht
Unit Contracts – ein SOLIDer Ansatz

Das Prinzip eines Vertrags: Die Rechte des Kunden sind die Pflichten des Anbieters – und umgekehrt!

VKSI

Kunde

Rechte

Pflichten

Anbieter

Rechte

Pflichten

Beispiel eines Vertrags:
Ein Kunde kauft ein Brötchen beim Anbieter.

VKSI

Kunde

Rechte

„Ich bekomme ein Brötchen geliefert.“

Pflichten

„Ich muss 50 Cent zahlen.“

Anbieter

Rechte

„Ich bekomme 50 Cent bezahlt.“

Pflichten

„Ich muss ein Brötchen liefern.“

Es reicht aus, den Vertrag bei dem Anbieter zu hinterlegen und dort auch zu überprüfen.

VKSI

Kunde

Rechte

Pflichten

Anbieter

Rechte

Pflichten

Im Brötchen-Beispiel reicht es aus, den Vertrag beim Anbieter zu hinterlegen und zu überprüfen.

VKSI

Kunde

Rechte

„Ich bekomme ein Brötchen geliefert.“

Pflichten

„Ich muss 50 Cent zahlen.“

Anbieter

Rechte

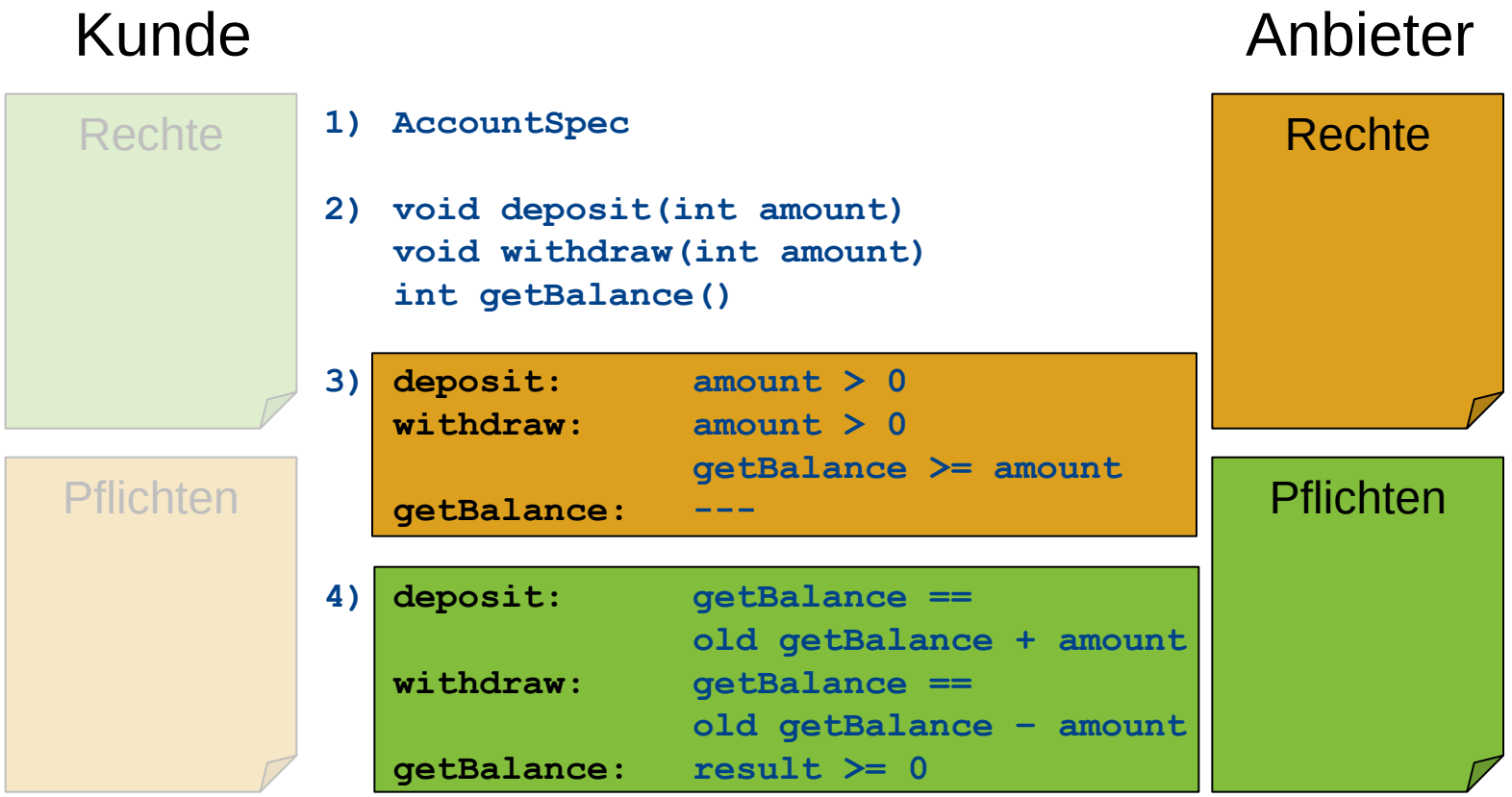
„Ich bekomme 50 Cent bezahlt.“

Pflichten

„Ich muss ein Brötchen liefern.“

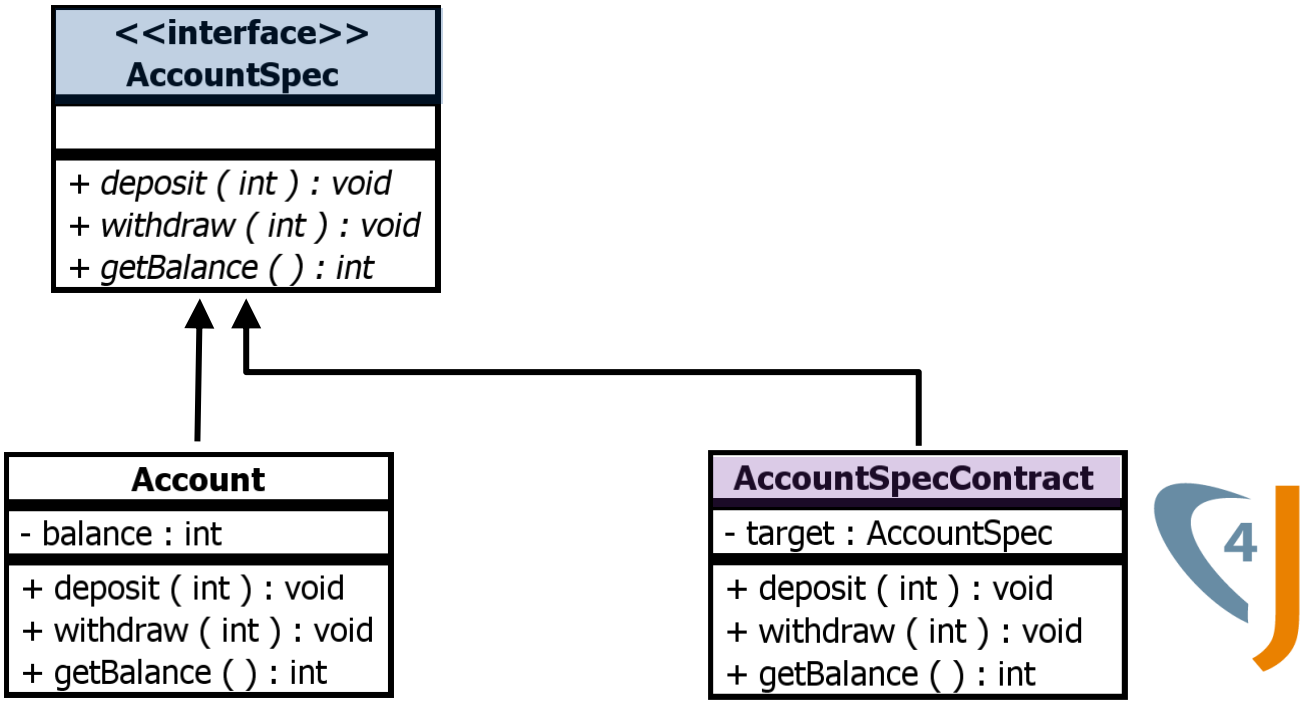
ADT `Account`

Vor- und Nachbedingungen stellen einen Vertrag dar. Es genügt, diesen Vertrag auf Anbieterseite zu definieren und zu prüfen.



Die *Vorbedingungen* definieren die *Rechte des Anbieters* (und damit die *Pflichten des Kunden*).
Die *Nachbedingungen* definieren die *Pflichten des Anbieters* (und damit die *Rechte des Kunden*).

`Account`: Interface, Implementierung – und Vertrag



CODE-BEISPIEL: ACCOUNT

Clean Contract: Wie formuliert man einen vollständigen Vertrag?

CQS: Trenne Abfragen durch `@Pure` von Kommandos

PRE: Prüfe für jede Abfrage und jedes Kommando,
ob eine Vorbedingung erforderlich ist

POST: Erstelle für jedes Kommando eine Nachbedingung

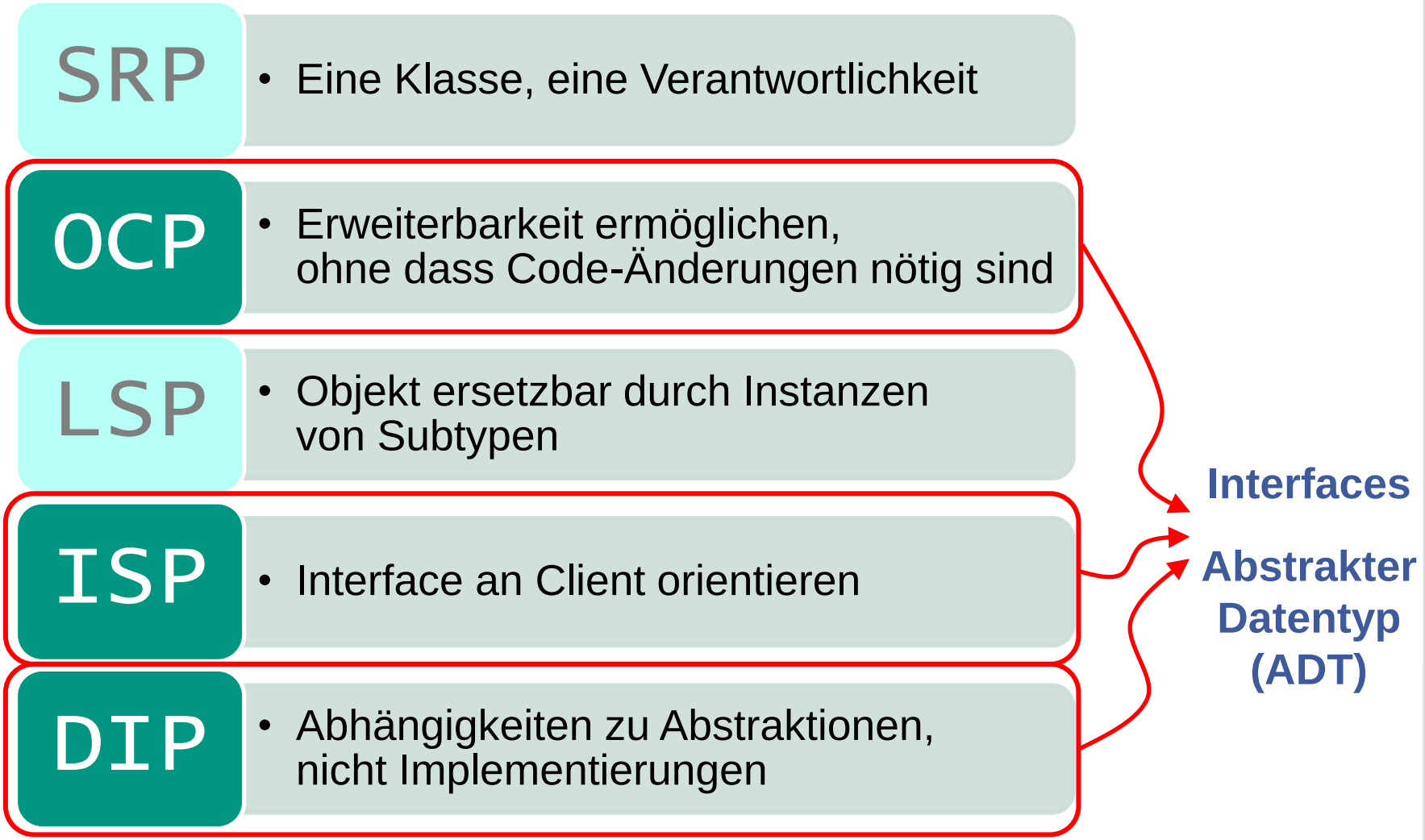
DRY: Formuliere Invarianten
(Nachbedingungen, die für alle Methoden gelten, nur ein Mal formulieren)

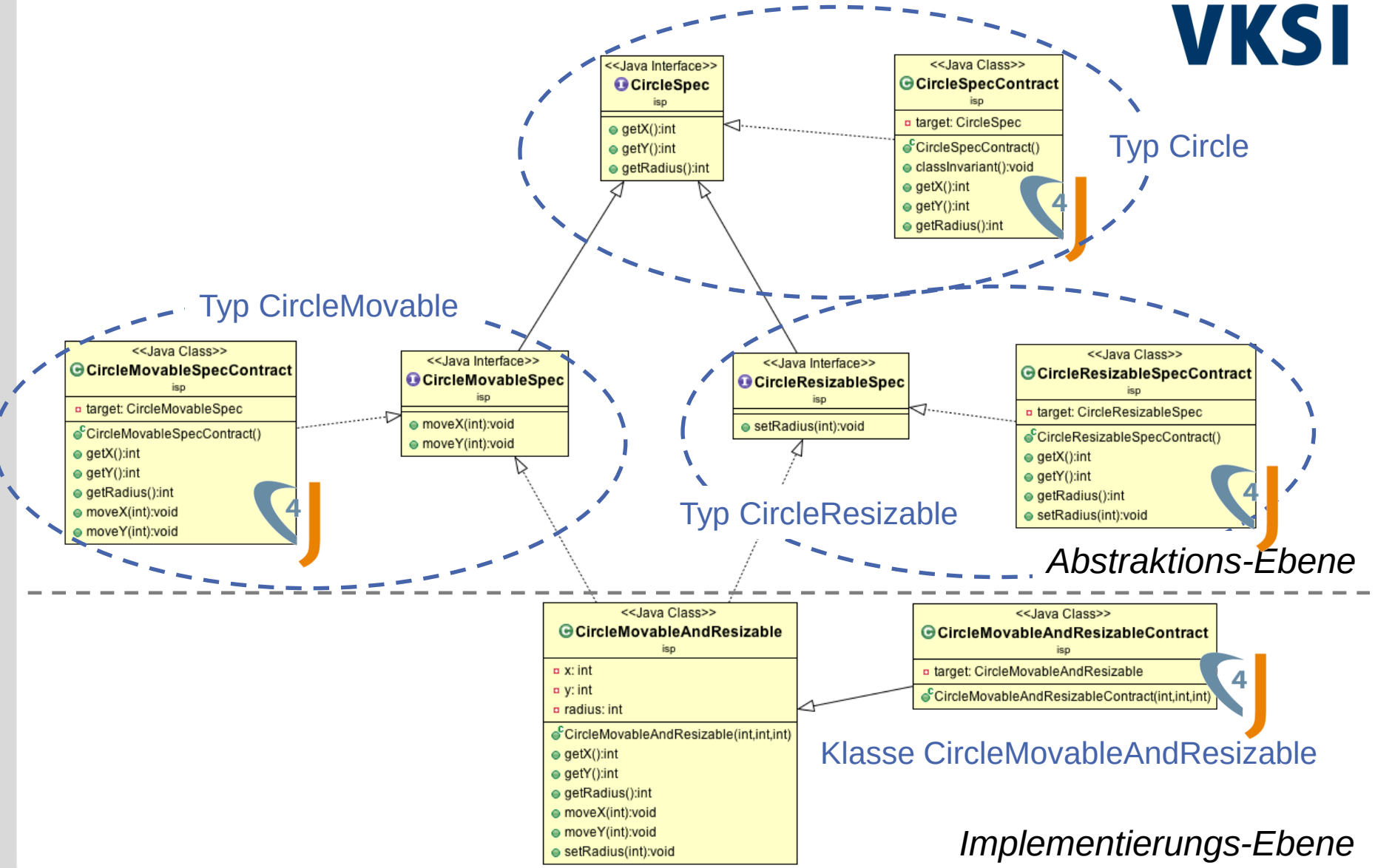
In enger Anlehnung an: Richard Mitchell / Jim McKim: Design by Contract, by Example (2001)

Inhaltsübersicht
Unit Contracts – ein SOLIDer Ansatz

1. Prolog
2. Was ist Clean Code?
3. Was ist ein Unit Contract?
4. Warum sind Unit Contracts ein SOLIDer Ansatz?
5. Epilog

Die 5 SOLID-Prinzipien




<<Java Interface>>
CircleSpec
isp
getX():int
getY():int
getRadius():int
<<Java Class>>
CircleSpecContract
isp
target: CircleSpec
CircleSpecContract()
classInvariant():void
getX():int
getY():int
getRadius():int
Typ Circle
Typ CircleMovable
<<Java Class>>
CircleMovableSpecContract
isp
target: CircleMovableSpec
CircleMovableSpecContract()
getX():int
getY():int
getRadius():int
moveX(int):void
moveY(int):void
<<Java Interface>>
CircleMovableSpec
isp
moveX(int):void
moveY(int):void
<<Java Interface>>
CircleResizableSpec
isp
setRadius(int):void
<<Java Class>>
CircleResizableSpecContract
isp
target: CircleResizableSpec
CircleResizableSpecContract()
getX():int
getY():int
getRadius():int
setRadius(int):void
Typ CircleResizable
Abstraktions-Ebene
<<Java Class>>
CircleMovableAndResizable
isp
x: int
y: int
radius: int
CircleMovableAndResizable(int,int,int)
getX():int
getY():int
getRadius():int
moveX(int):void
moveY(int):void
setRadius(int):void
<<Java Class>>
CircleMovableAndResizableContract
isp
target: CircleMovableAndResizable
CircleMovableAndResizableContract(int,int,int)
Klasse CircleMovableAndResizable
Implementierungs-Ebene

VKSI

- Joshua Bloch spricht in seinem Buch „Effective Java“ (Second Edition, 2008) insgesamt 78 Punkte (Items) an, wie man mit Java programmieren sollte, um hohe Qualität zu erzeugen.
  - Item 19: Use interfaces only to define types

    „When a class implements an interface, the interface serves as a type that can be used to instances of the class. It is inappropriate to define an interface for any other purpose.“
  - Item 40: Design method signatures carefully

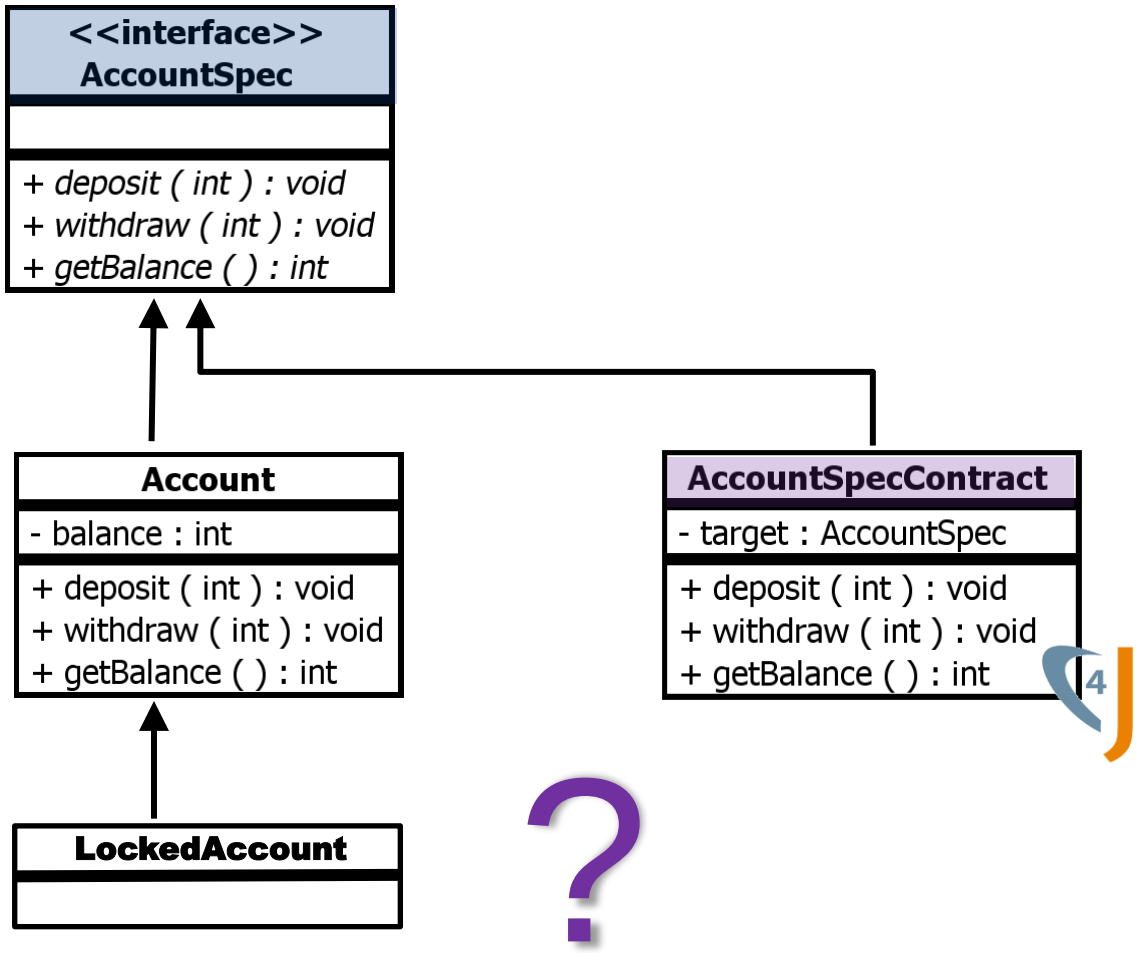
    „Use interfaces rather than classes as parameter types.“
  - Item 52: Refer to objects by their interfaces

    „If appropriate interfaces types exist, then parameters, return values, variables and fields should all be declared using interfaces types. If you get into the habit of using interfaces as types, your program will be much more flexible.“

Die 5 SOLID-Prinzipien

SRP	• Eine Klasse, eine Verantwortlichkeit
OCP	• Erweiterbarkeit ermöglichen, ohne dass Code-Änderungen nötig sind
LSP	• Objekt ersetzbar durch Instanzen von Subtypen
ISP	• Interface an Client orientieren
DIP	• Abhängigkeiten zu Abstraktionen, nicht Implementierungen

Beispiel zu LSP: `LockedAccount`

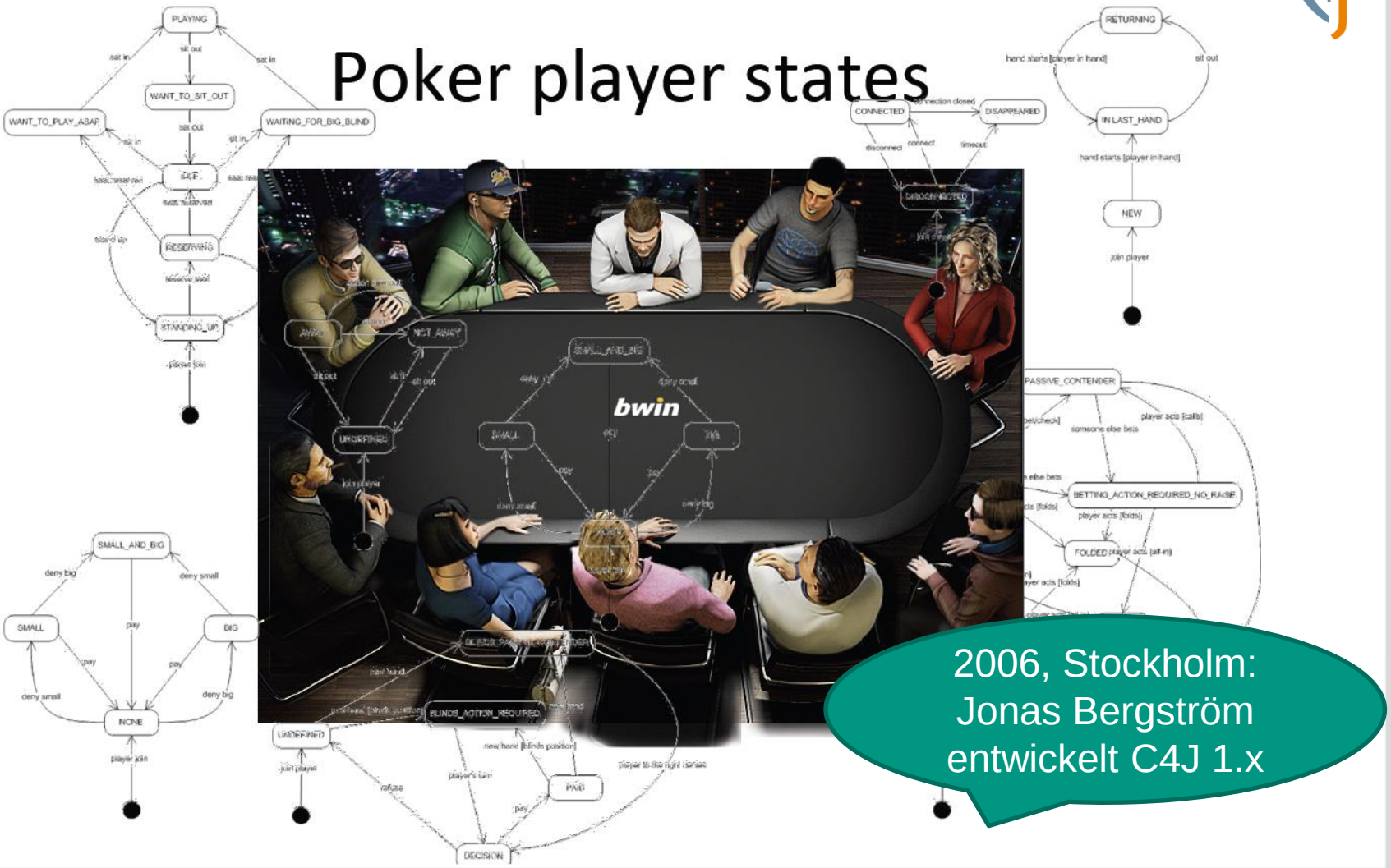


CODE-BEISPIEL: LOCKED ACCOUNT

CODE-BEISPIEL: EQUALS CONTRACT

CONTRACTS IN DER PRAXIS

Unit Contracts mit Test Bots: PokerBots simulieren Spieler und testen die Poker-Engine bei aktivierten Verträgen



2006, Stockholm:
Jonas Bergström
entwickelt C4J 1.x

Wirkung des Einsatzes von Unit Contracts und Test Bots auf die Qualität der bwin PokerEngine

bwin PokerEngine
vor der Sanierung

Fehlermeldungen
pro Betriebsmonat

bwin PokerEngine
mit Unit Contracts

Fehlermeldungen nach
38 Betriebsmonaten

Einsatz von Softwareverträgen seit 2006 bei bwin in Stockholm
Lessons Learned

1. Pareto-Prinzip: 80% der genutzten Funktionalität ist in 20% der Klassen umgesetzt. Es reicht aus, diese Kernklassen mit Softwareverträgen zu schützen.

2. Falsche Annahmen als Hauptfehlerquelle: 80% der bei bwin gefundenen Fehler waren auf Verletzungen von Vorbedingungen (und Klasseninvarianten) zurückzuführen. Der Grund waren falsche bzw. fehlende Annahmen darüber, wie eine Klasse genutzt werden sollte.

3. Autorenschaft schützt nicht vor falschen Annahmen: Falsche Annahmen unterliefen auch den Autoren der Klassen. Nach nur wenigen Wochen Arbeit an anderen Modulen hatten sie die Annahmen vergessen, unter denen sie die Klassen erstellt hatten.

4. Unit Contracts und Test Bots: Die Kombination von Softwareverträgen mit PokerBots – virtuellen Spielern, die über Nacht die aktuelle Version der PokerEngine nutzten – erwies sich als äußerst wirkungsvolles Instrument, um subtile Fehler aufzuspüren (Boundary Conditions, Edge Conditions, Corner Conditions).

5. Macht der Gewohnheit: Trotz dieser Erfolge blieb die Nutzung von Softwareverträgen auf das PokerEngine-Team beschränkt. Andere Teams zogen nicht nach.

Quelle: Interview mit Jonas Bergström, Sept. 2012, Stockholm

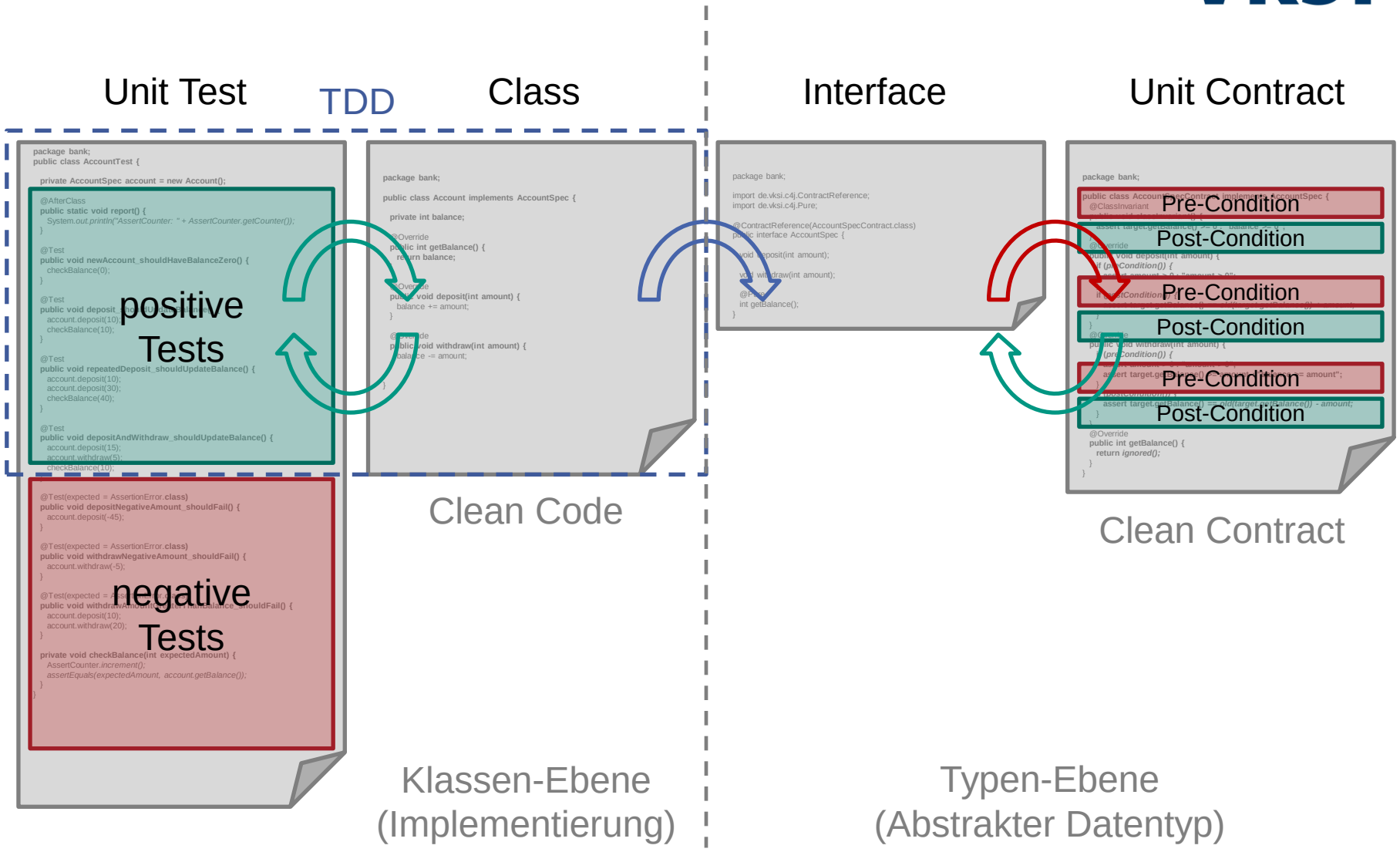
C4J: Zweite Generation

C4J 6.0 (http://c4j.vksi.de)

- Entwickelt im Rahmen der VKSI SIG C4J (2012)
- Verträge von außen anhängen (ohne Produktivcode-Änderung)
  - dadurch auch für externe Bibliotheken/JDK verwendbar!
- Automatische Überprüfung der Pure-Eigenschaft
- Refactorings/TDD mit Unit Contracts möglich
- Eclipse-Plugin verfügbar (Unit Contracts als Javadoc)
- Verhalten konfigurierbar
- Lizenz: EPL 1.0

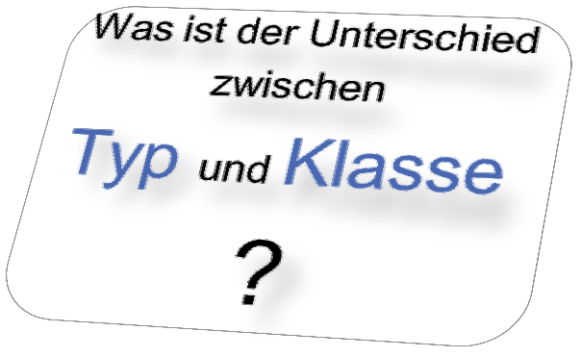
Quelle: Interview mit Jonas Bergström, Sept. 2012, Stockholm

TDD with Unit Contracts
Clean Code & Clean Contract



Inhaltsübersicht
Unit Contracts – ein SOLIDer Ansatz

1. Prolog
2. Was ist Clean Code?
3. Was ist ein Unit Contract?
4. Warum sind Unit Contracts ein SOLIDer Ansatz?
5. Epilog



VKSI

Typ	Definiert einen Datentyp/ADT	Unit Contract
Abstraktions-Ebene		
Klasse	Implementiert einen Typen	Unit Test
Implementierungs-Ebene		
Objekt	Instanz einer Klasse	Zustandsüberwachung durch Unit Contract
Ausführungs-Ebene		