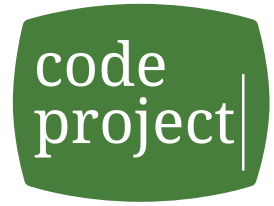


Vorbereitung Programmierworkshop Java

Einführung objektorientierte Programmierung

Oliver Paulus | Dezember 2010 | LUG Frankfurt

Präsentationsteile

- Teil 1: Einführung objektorientierte Programmierung
- Teil 2: OOP-Gundlagen Java
- Teil 3: Beispielprojekt

Präsentationsteile

- Teil 1: Einführung objektorientierte Programmierung
- Teil 2: OOP-Gundlagen Java
- Teil 3: Beispielprojekt

Präsentationsteile

- Teil 1: Einführung objektorientierte Programmierung
- Teil 2: OOP-Gundlagen Java
- Teil 3: Beispielprojekt

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Die Geschichte der Objektorientierung

- Baut auf den Verfahren der strukturierten Programmierung auf (C, PASCAL)
- erste objektorientierte Programmiersprache: Simula-67 (1965) - war Erweiterung von Algol-60 um OOP
- die Programmiersprache Smalltalk baute die OOP-Prinzipien noch weiter aus (Prinzip “Alles ist ein Objekt”)
- Durchbruch der Objektorientierung Mitte/Ende der 1980er Jahre (C++)
- heute mehr als 100 objektorientierte Programmiersprachen (“reine” und hybride objektorientierte Programmiersprachen)
- Aspektorientierte Programmierung (**AOP**) erweitert klassisch objektorientierte Sprachen, unterstützt OOP Prinzipien, meist für Querschnittsfunktionen

Die Geschichte der Objektorientierung

- Baut auf den Verfahren der strukturierten Programmierung auf (C, PASCAL)
- erste objektorientierte Programmiersprache: Simula-67 (1965) - war Erweiterung von Algol-60 um OOP
- die Programmiersprache Smalltalk baute die OOP-Prinzipien noch weiter aus (Prinzip “Alles ist ein Objekt”)
- Durchbruch der Objektorientierung Mitte/Ende der 1980er Jahre (C++)
- heute mehr als 100 objektorientierte Programmiersprachen (“reine” und hybride objektorientierte Programmiersprachen)
- Aspektorientierte Programmierung (**AOP**) erweitert klassisch objektorientierte Sprachen, unterstützt OOP Prinzipien, meist für Querschnittsfunktionen

Die Basis

Unterschiede zur strukturierten Programmierung

- Daten und Routinen sind **nicht** voneinander getrennt
- Daten gehören explizit einem Objekt - nur das Objekt hat das alleinige Recht diese zu lesen und zu ändern
- Aufrufer müssen sich über eine klar definierte Schnittstelle an das Objekt wenden und eine Änderung der Daten anfordern

Die Basis

Unterschiede zur strukturierten Programmierung

- Daten und Routinen sind **nicht** voneinander getrennt
- Daten gehören explizit einem Objekt - nur das Objekt hat das alleinige Recht diese zu lesen und zu ändern
- Aufrufer müssen sich über eine klar definierte Schnittstelle an das Objekt wenden und eine Änderung der Daten anfordern

Die Basis

Unterschiede zur strukturierten Programmierung

- Daten und Routinen sind **nicht** voneinander getrennt
- Daten gehören explizit einem Objekt - nur das Objekt hat das alleinige Recht diese zu lesen und zu ändern
- Aufrufer müssen sich über eine klar definierte Schnittstelle an das Objekt wenden und eine Änderung der Daten anfordern

Die Basis

Begriffe der objektorientierten Programmierung

- Objekte und Klassen
- Attribute und Methoden

Die Basis

Objekte und Klassen

Objekt

- enthält Daten und Funktionalität
- ist ein Exemplar/Instanz einer Klasse

Klasse

- ist ein Modellierungsmittel
- ist die Schablone/der Bauplan für Objekte
- beschreibt Eigenschaften und Verhaltensweisen

Die Basis

Attribute und Methoden

Attribute

- Objekte besitzen verschiedene Eigenschaften (Attribute)

Methoden

- Die einer Klasse von Objekten zugeordneten Algorithmen bezeichnet man auch als Methoden
- Das Verhalten eines Objekts wird mit der Methode beschrieben

Die Basis

Die Säulen der objektorientierten Programmierung

- Abstraktion
- Kapselung
- Modularität
- Polymorphie
- Vererbung

Die Basis

Die Säulen der objektorientierten Programmierung

- Abstraktion
- Kapselung
- Modularität
- Polymorphie
- Vererbung

Die Basis

Die Säulen der objektorientierten Programmierung

- Abstraktion
- Kapselung
- Modularität
- Polymorphie
- Vererbung

Die Basis

Die Säulen der objektorientierten Programmierung

- Abstraktion
- Kapselung
- Modularität
- Polymorphie
- Vererbung

Die Basis

Die Säulen der objektorientierten Programmierung

- Abstraktion
- Kapselung
- Modularität
- Polymorphie
- Vererbung

Die Basis

Abstraktion

Definition

- Mechanismus zum Ausdrücken von relevanten und zum Unterdrücken von irrelevanten Details
- Trennung zwischen Konzept und Umsetzung
- **Relevant:** wie benutze ich die Abstraktion
  Irrelevant: deren Implementierung

Die Basis

Kapselung (Geheimnisprinzip)

Definition

- Kapselung bezeichnet in der Programmierung das Verbergen von Implementierungsdetails
- Kein Teil eines komplexen Systems soll von internen Details eines anderen abhängen

Die Basis

Modularität

Definition

- Programmteile sollten isoliert betrachtet werden, um Komplexität in den Griff zu bekommen (Prinzip “Teile und Herrsche”)
- Das Programm wird in Module zerlegt. Jedem Modul werden spezielle Aufgaben zugeordnet
- Module sollen unabhängig von anderen geändert und wiederverwendet werden können
- Prinzipien für Module:
  - hohe **Kohäsion**
  - schwache **Kopplung**
  - Demeter-Prinzip (“Sprich nur mit deinen Freunden”)

Die Basis

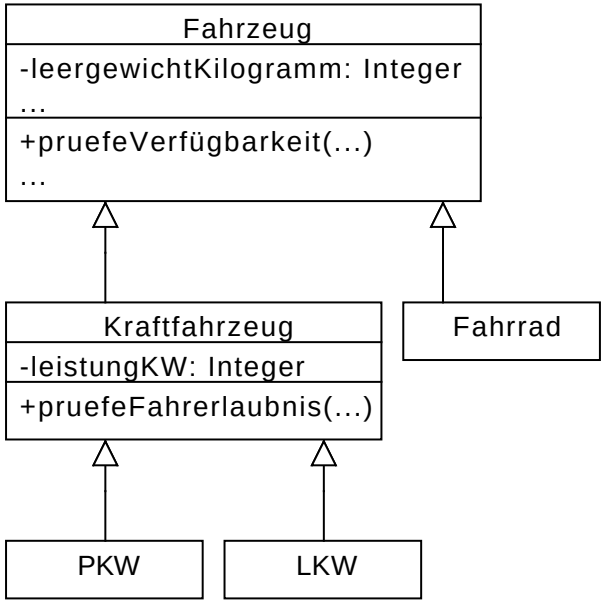
Polymorphie (“Vielgestaltigkeit”)

Definition

Verschiedene Objekte können auf die gleiche Nachricht unterschiedlich reagieren

Die Basis

Vererbung

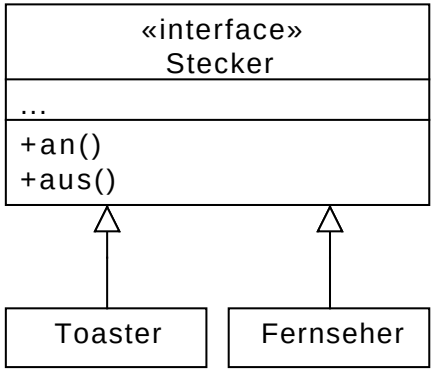


Definition

Vererbung heißt vereinfacht, dass eine abgeleitete Klasse die Methoden und Attribute der Basisklasse ebenfalls besitzt, also “erbt”

Die Basis

Vererbung - Vererbung der Spezifikation (“Schnittstellenvererbung”)

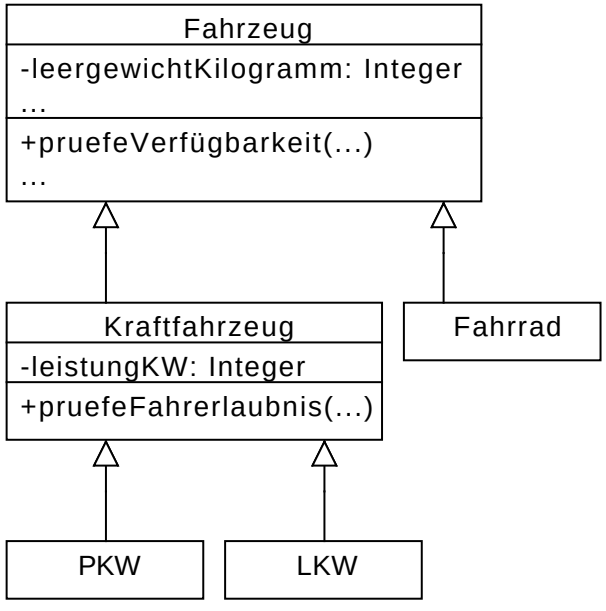


Definition

Eine Unterklasse erbt grundsätzlich die Spezifikation ihrer Oberklasse. Die Unterklasse übernimmt damit alle Verpflichtungen und Zusicherungen der Oberklasse

Die Basis

Vererbung - Erben der Implementierung (“Implementierungsvererbung”)



Definition

Die abgeleitete Klasse übernimmt die Attribute und Funktionalität der Basisklasse und wandelt diese gegebenenfalls ab oder ergänzt diese um weitere für diese Spezialisierung zusätzlich relevante Eigenschaften

Prinzipien
Übersicht

- SOLID-Prinzipien
- weitere Prinzipien

Wichtig

Diese Prinzipien führen zu gutem objektorientierten Design. Entwurfsmuster basieren auf diesen

Prinzipien
Übersicht

- SOLID-Prinzipien
- weitere Prinzipien

Wichtig

Diese Prinzipien führen zu gutem objektorientierten Design.
Entwurfsmuster basieren auf diesen

Prinzipien
Übersicht

- SOLID-Prinzipien
- weitere Prinzipien

Wichtig

Diese Prinzipien führen zu gutem objektorientierten Design. Entwurfsmuster basieren auf diesen

SOLID-Prinzipien[1]

Übersicht

- **S**ingle Responsibility Principle
- **O**pen/Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

[1] von Robert C. Martin

SOLID-Prinzipien

Single Responsibility Prinziple[2]

Definition

Es sollte nie mehr als einen Grund geben eine Klasse zu ändern

[2] von Robert C. Martin

SOLID-Prinzipien

Open/Closed Prinziple[3]

Definition

Module sollten sowohl offen (für Erweiterungen), als auch geschlossen (für Modifikationen) sein

[3] von Bertrand Meyer

SOLID-Prinzipien

Liskov **S**ubstitution **P**rinciple (Ersetzbarkeitsprinzip)[4]

Definition

- Eine Unterklasse muss stets alle Eigenschaften der Oberklasse erfüllen und immer als Objekt der Oberklasse verwendbar sein
- Abgeleitete Klassen müssen durch die Schnittstelle der Basisklasse benutzbar sein, ohne dass der Benutzer den Unterschied zu kennen braucht

[4] von Barbara H. Liskov, Jeannette M. Wing

SOLID-Prinzipien

Interface **S**egregation **P**rinciple[5]

Definition

Clients sollten nicht dazu gezwungen werden, von Interfaces abzuhängen, die sie nicht verwenden

[5] von Robert C. Martin

SOLID-Prinzipien

Dependency Inversion Principle[6]

Definition

- **A.** Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen
- **B.** Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen

[6] von Robert C. Martin

weitere Prinzipien

Separation **o**f **C**oncerns

Definition

Verschiedene Elemente der Aufgabe sollten möglichst in verschiedenen Elementen der Lösung repräsentiert werden

Weitere Prinzipien

Don't **R**epeat **Y**ourself ("Once and Only Once")[7]

Definition

Wiederhole dich nicht ("Einmal und nur einmal")

[7] von Andy Hunts, Dave Thomas

Weitere Prinzipien

Law **o**f **D**emeter[8] (“Sprich nur mit deinen Freunden”)

Definition

Eine Methode, die zu einem Objekt gehört, darf Operationen auf anderen Objekten nur aufrufen, wenn diese auf einem Objekt aus der folgenden Liste aufgerufen werden:

- das Objekt selbst
- ein Objekt, das vom Objekt selbst referenziert wird
- ein Objekt, das als Parameterwert an die Methode übergeben wurde
- ein Objekt, das innerhalb der Methode selbst angelegt wurde

[8] von Karl J. Lieberherr, Ian Holland

Quellen

- Robert C. Martin - “Clean Code: A Handbook of Agile Software Craftsmanship”
- Robert C. Martin - “Agile Software Development, Principles, Patterns, and Practices”
- Steve MacConnell - “Code Complete. A Practical Handbook of Software Construction”
- Martin Fowler - “Patterns of Enterprise Application Architecture”
- Bernhard Lahres, Gregor Rayman - “Objektorientierte Programmierung”

Inhaltsverzeichnis

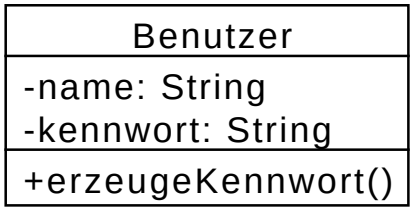
Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Klassen und Objekte

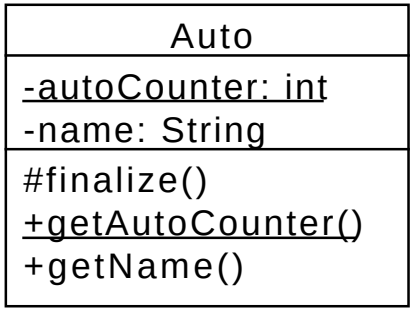
Einfache Definition




```
1  class Benutzer {
2    private String name;
3    private String kennwort;
4
5    public Benutzer(String name) {
6      this.name = name;
7    }
8
9    public void erzeugeKennwort() {
10     /* ... */
11   }
12 }
13
14 class Main {
15   public static void main() {
16     Benutzer hans = new Benutzer("Hans");
17     Benutzer fred = new Benutzer("Fred");
18   }
19 }
```

Klassen und Objekte

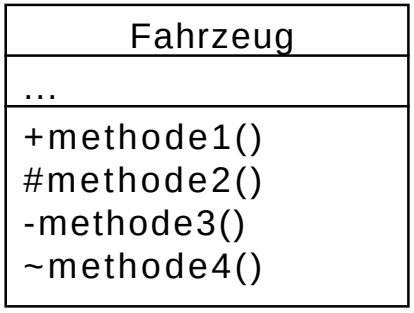
Instanz- und Klassenelemente




```
1  class Auto {
2    static private int autoCounter = 0;
3    private String name;
4
5    public Auto(String name) {
6      ++autoCounter;
7    }
8    @Override
9    protected void finalize() throws Throwable {
10     --autoCounter;
11   }
12   public static int getAutoCounter() {
13     return autoCounter;
14   }
15
16   public String getName() {
17     return this.getName();
18   }
19 }
20
21 class Main {
22   public static void main() {
23     Auto auto1 = new Auto("VW Golf");
24     System.out.println(Auto.getAutoCounter());
25     Auto auto2 = new Auto("Audi A3");
26     System.out.println(Auto.getAutoCounter());
27     System.out.println(auto2.getName());
28   }
29 }
```

Klassen und Objekte

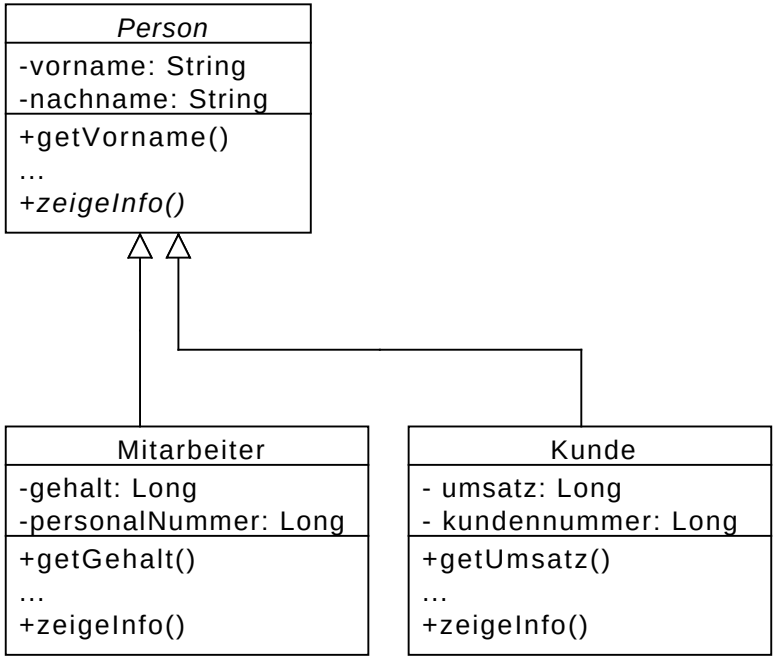
Sichtbarkeit von Attributen und Methoden




```
1  class Fahrzeug {
2    public void methode1() {
3      /* ... */
4    }
5
6    protected void methode2() {
7      /* ... */
8    }
9
10   private void methode3() {
11     /* ... */
12   }
13
14   void methode4() {
15     /* ... */
16   }
17 }
```

Klassen und Objekte

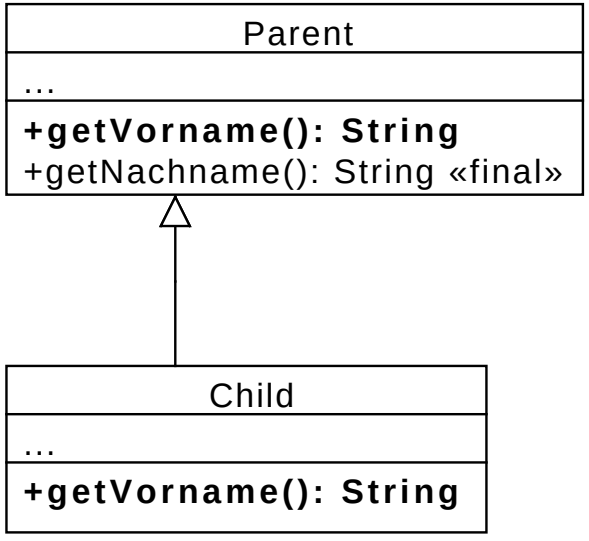
Abstrakte Klassen, Vererbung




```
1  abstract class Person {
2    private String vorname;
3    private String nachname;
4
5    public String getVorname() {
6      return vorname;
7    }
8
9    /* ... */
10
11   public abstract void zeigeInfo();
12 }
13
14 class Mitarbeiter extends Person {
15   private Long gehalt;
16   private Long personalNummer;
17
18   public Long getGehalt() {
19     return gehalt;
20   }
21
22   @Override
23   public void zeigeInfo() {
24     /* ... */
25   }
26 }
27
28 /* ... */
```

Klassen und Objekte

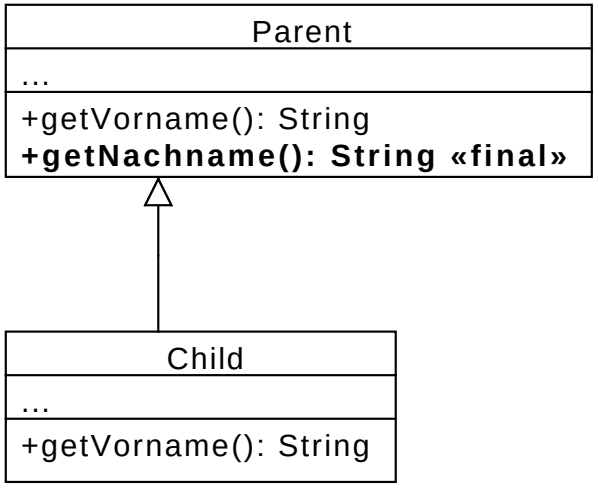
Vererbung - Methoden überschreiben




```
1  class Parent {
2    public String getVorname() {
3      return "parent.vorname";
4    }
5
6    public final String getNachname() {
7      return "parent.nachname";
8    }
9  }
10
11 final class Master {
12   /* ... */
13 }
14
15 class Child extends Parent {
16   @Override
17   public String getVorname() {
18     return "child.vorname";
19   }
20 }
```

Klassen und Objekte

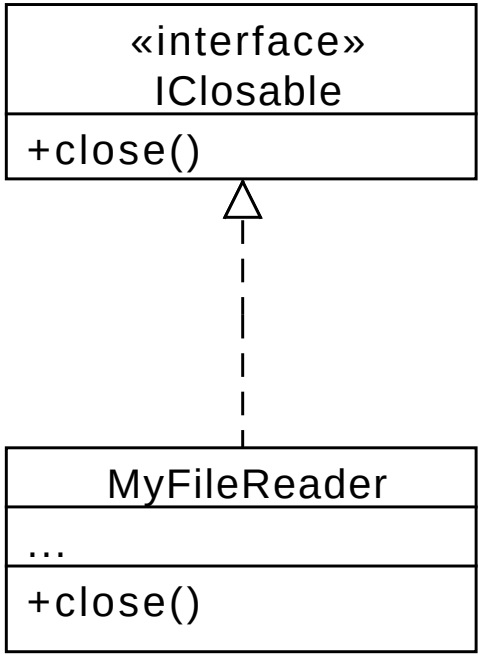
Finale Klasse oder Methode




```
1  class Parent {
2    public String getVorname() {
3      return "parent.vorname";
4    }
5
6    public final String getNachname() {
7      return "parent.nachname";
8    }
9  }
10
11 final class Master {
12   /* ... */
13 }
14
15 class Child extends Parent {
16   @Override
17   public String getVorname() {
18     return "child.vorname";
19   }
20 }
```

Klassen und Objekte

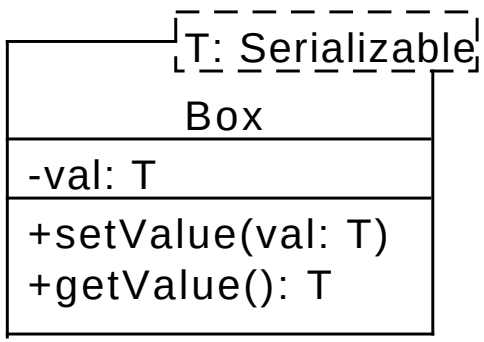
Schnittstelle




```
1  interface IClosable {
2    public void close();
3  }
4
5  class MyFileReader implements IClosable {
6    @Override
7    public void close() {
8      /* ... */
9    }
10 }
```

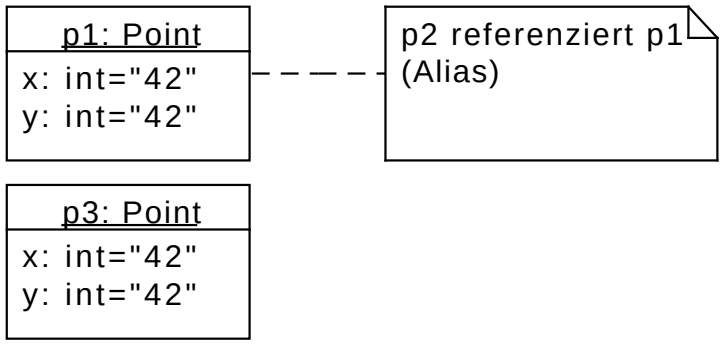
Klassen und Objekte

Generische Datentypen



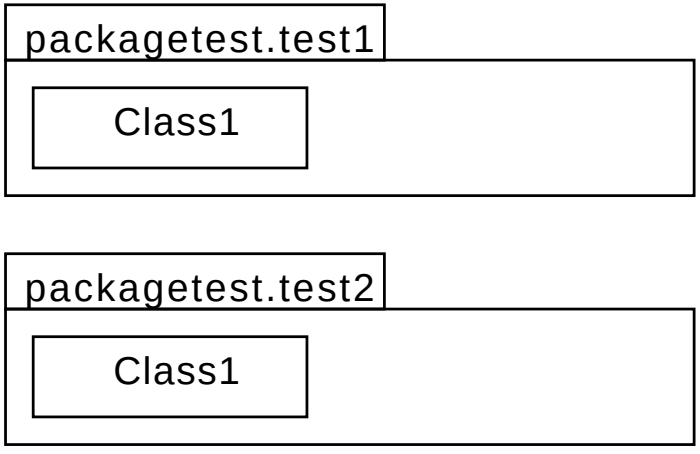

```
1  import java.io.Serializable;
2
3  class Box<T extends Serializable> {
4    private T val;
5
6    void setValue(T val) {
7      this.val = val;
8    }
9
10   T getValue() {
11     return val;
12   }
13 }
14
15 class Main {
16   public static void main() {
17     Box intBox = new Box<Integer>();
18     Box stringBox = new Box<String>();
19   }
20 }
```

Referenzen, Identität und Gleichheit




```
1  import java.awt.Point;
2  import static java.lang.System.out;
3
4  class Main {
5    public static void main() {
6      Point p1 = new Point(42, 42);
7      Point p2 = p1;
8      Point p3 = new Point(42, 42);
9
10     out.println(p1 == p2);
11     out.println(p1 == p3);
12     out.println(p1.equals(p3));
13   }
14 }
```

Pakete




```java
package packagetest.test1;

public class Class1 {
  /* ... */
}
```

```java
package packagetest.test2;

public class Class1 {
  /* ... */
}
```

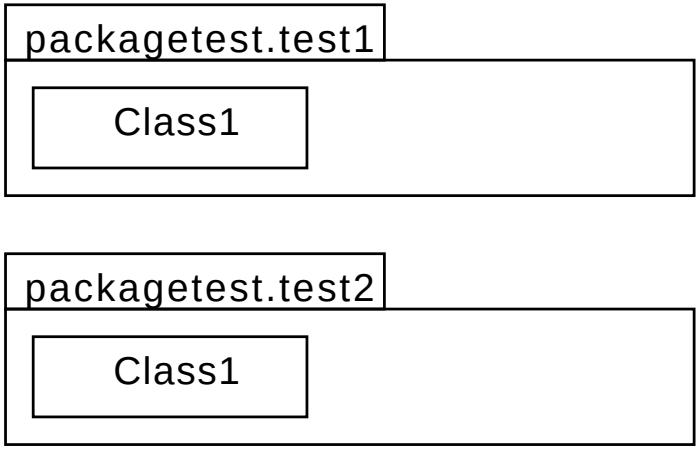
```java
package packagetest.usage;

import packagetest.test1.*;

public class PackageUsage {
  Class1 class1p1 = new Class1();

  packagetest.test2.Class1 class1p2 = new
      packagetest.test2.Class1();
}
```

Pakete




```java
package packagetest.test1;

public class Class1 {
  /* ... */
}
```

```java
package packagetest.test2;

public class Class1 {
  /* ... */
}
```

```java
package packagetest.usage;

import packagetest.test1.*;

public class PackageUsage {
  Class1 class1p1 = new Class1();

  packagetest.test2.Class1 class1p2 = new
        packagetest.test2.Class1();
}
```

Annotationen

```java
public class TestClass1 {
  @Deprecated
  public void oldMethod() {
    /* ... */
  }

  @SuppressWarnings(value = { "deprecation", "unchecked" })
  public void newMethod() {
    oldMethod();
    /* ... */
  }
}
```

```java
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface MyAnnotation {
  int meinIntParameter();

  String meinStringParameter();

  String meinDefaultParameter() default "xyz";

  String[] meinArrayParameter();
}
```

Annotationen

```java
public class TestClass1 {
  @Deprecated
  public void oldMethod() {
    /* ... */
  }

  @SuppressWarnings(value = { "deprecation", "unchecked" })
  public void newMethod() {
    oldMethod();
    /* ... */
  }
}
```

```java
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface MyAnnotation {
  int meinIntParameter();

  String meinStringParameter();

  String meinDefaultParameter() default "xyz";

  String[] meinArrayParameter();
}
```

Quellen

Christian Ullenboom, “Java ist auch eine Insel”

DZone UML Refcard

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Aufgabenbeschreibung

Übersicht

Definition

Erstelle eine “Blog”-Applikation. Man kann Blogbeiträge schreiben und es können Kommentare zu einem Blogartikel abgegeben werden

Komponenten

- Zugriff auf HSQLDB mit Java Database Connectivity (JDBC)
- JUnit-Tests

Aufgabenbeschreibung

Übersicht

Definition

Erstelle eine “Blog”-Applikation. Man kann Blogbeiträge schreiben und es können Kommentare zu einem Blogartikel abgegeben werden

Komponenten

- Zugriff auf HSQLDB mit Java Database Connectivity (JDBC)
- JUnit-Tests

Verwendete Frameworks

HyperSQL (HSQLDB)

- 100% Java (relationale) Datenbank
- leichtgewichtig
- Modi
  - eingebettete Datenbank
  - Server- und Standalone-Betrieb
- unterstützt große Teile der SQL-Standards 92, 92 Advanced Level, 99, 2003, “2008 core” und Teile von “2008 optional features”

Verwendete Frameworks

JUnit[9]

- Framework zum Testen von Java-Programmen
- automatisierte Unit-Tests
- White-Box-Testing
- zwei mögliche Ergebnisse: Test gelingt (“grün”) oder misslingt (“rot”)
- Test-driven software development (TDD)

[9] von Erich Gamma und Kent Beck

JUnit

Beispiel 1

```
1  import java.io.IOException;
2
3  import org.junit.After;
4  import org.junit.Before;
5  import org.junit.Test;
6
7  import static org.junit.Assert.*;
8
9  public class SampleTest1 {
10   @Before
11   public void setUp() {
12     /* ... */
13   }
14   @Test
15   public void test1() {
16     /* ... */
17     assertFalse("check", 1 == 2);
18   }
19   @Test(expected = IOException.class)
20   public void test2() {
21     /* ... */
22   }
23   @After
24   public void tearDown() {
25     /* ... */
26   }
27 }
```

JUnit

Beispiel 2

```java
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class SampleTest2 {
  @BeforeClass
  public static void setUp() {
    /* ... */
  }
  @Test
  public void test1() {
    /* ... */
  }
  @AfterClass
  public static void tearDown() {
    /* ... */
  }
}
```

```java
import org.junit.runners.Suite.SuiteClasses;

@SuiteClasses({ SampleTest1.class, SampleTest2.class })
public class TestSuite {

}
```

Start der Realisierung

Phase 1

Viel Erfolg!

Präsentation der Lösung

Phase 1

Wer möchte?

Verwendete Frameworks

Hibernate

Definition

- ist ein O/R-Mapper
- überwindet “objekt-relationale Unverträglichkeit”-Problem
- bietet objektorientierte Sicht auf Tabellen und Beziehungen in RDBMS
- statt mit SQL-Statements wird mit Objekten operiert (HQL, JPQL, Criteria API, SQL)
- abstrahiert vom jeweiligen RDBMS (verschiedene Dialekte), unterstützt mehr als 25 Datenbank-Systeme
- Mapping mit XML-Konfiguration, XDoclet oder Annotationen (u.a. **J**ava **P**ersistence **A**PI)

Hibernate

Beispiel Konfiguration JPA-Annotationen - Teil 1

```
1  import java.util.HashSet;
2  import java.util.Set;
3
4  import javax.persistence.CascadeType;
5  import javax.persistence.Entity;
6  import javax.persistence.FetchType;
7  import javax.persistence.JoinColumn;
8  import javax.persistence.OneToMany;
9
10 @Entity
11 public class Post extends DomainBase {
12   private Set<Comment> comments = new HashSet<Comment>();
13
14   @OneToMany(fetch = FetchType.LAZY, cascade = { CascadeType.ALL })
15   @JoinColumn(name="POST_ID")
16   public Set<Comment> getComments() {
17     return comments;
18   }
19
20   public void setComments(Set<Comment> comments) {
21     this.comments = comments;
22   }
23 }
```

Hibernate

Beispiel Konfiguration JPA-Annotationen - Teil 2

```
1  import javax.persistence.Column;
2  import javax.persistence.GeneratedValue;
3  import javax.persistence.GenerationType;
4  import javax.persistence.Id;
5  import javax.persistence.MappedSuperclass;
6
7  @MappedSuperclass
8  public abstract class DomainBase {
9    private Long id;
10
11   @Id
12   @GeneratedValue(strategy = GenerationType.IDENTITY)
13   @Column(unique = true, nullable = false)
14   public Long getId() {
15     return id;
16   }
17
18   public void setId(Long id) {
19     this.id = id;
20   }
21 }
```

Hibernate

Beispiel Konfiguration JPA-Annotationen - Teil 3

```
1  import javax.persistence.Column;
2  import javax.persistence.Entity;
3
4  @Entity
5  public class Comment extends DomainBase {
6    private String content;
7
8    public Comment() {
9
10   }
11
12   @Column(columnDefinition = "LONGVARCHAR")
13   public String getContent() {
14     return content;
15   }
16   public void setContent(String content) {
17     this.content = content;
18   }
19 }
```

Start der Realisierung

Phase 2

Viel Erfolg!

Präsentation der Lösung

Phase 2

Wer möchte?

Verwendete Frameworks

Maven

Definition

- Build-Management-Programm
- komplettes Konfigurationsmanagement von Softwareentwicklungsprojekten

Verwendete Frameworks

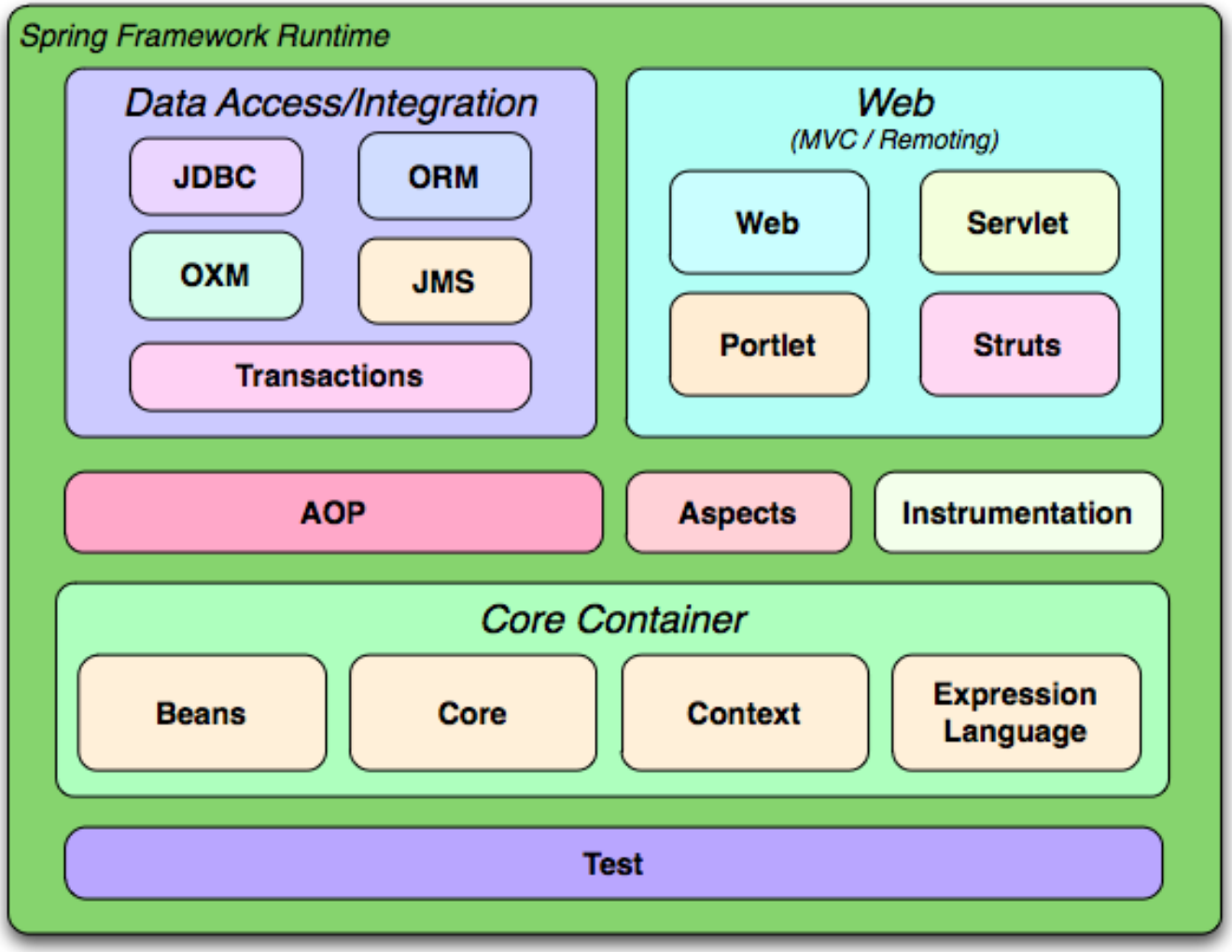
Spring

Definition

- will die Entwicklung mit Java/Java EE vereinfachen
- gute Programmierpraktiken fördern
- fördert die Testbarkeit von Programmen
- die Entkopplung der Applikationskomponenten steht im Vordergrund
- Vereinfacht die Verwendung/Integration verschiedenster Frameworks
- verwendet “**P**lain **O**ld **J**ava **O**bjects”

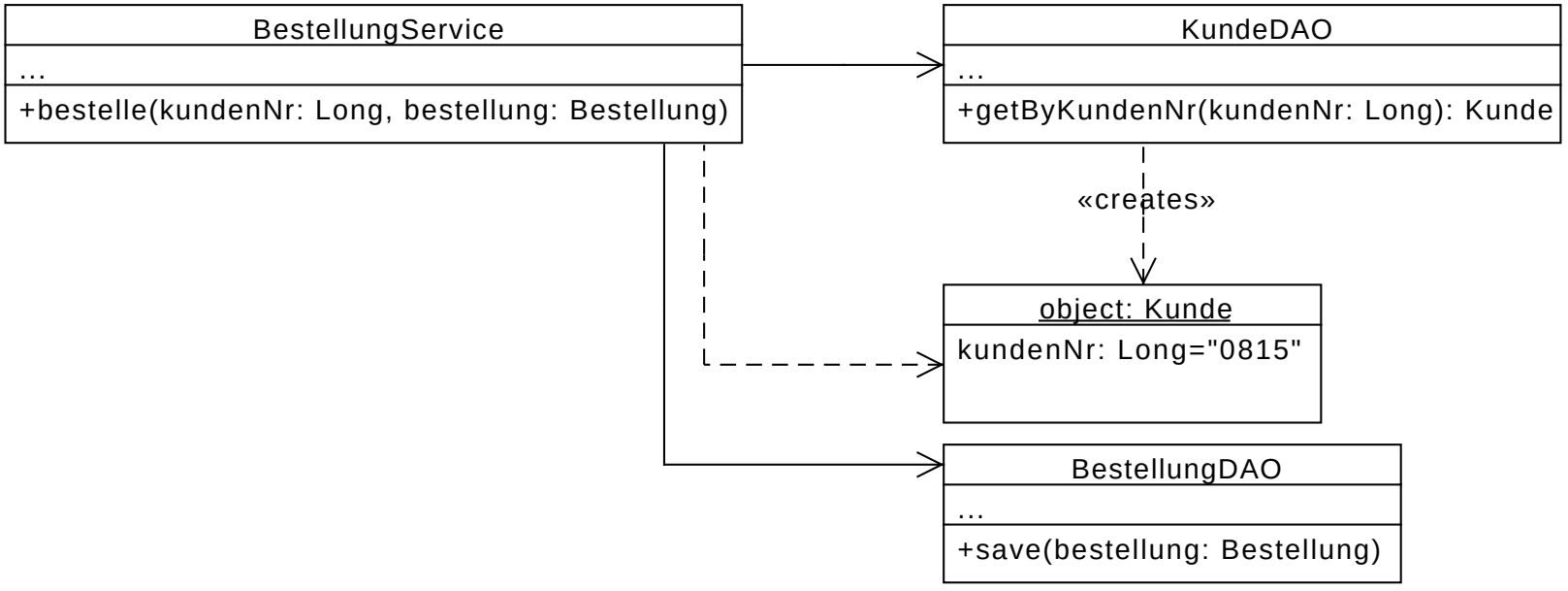
Spring

Module



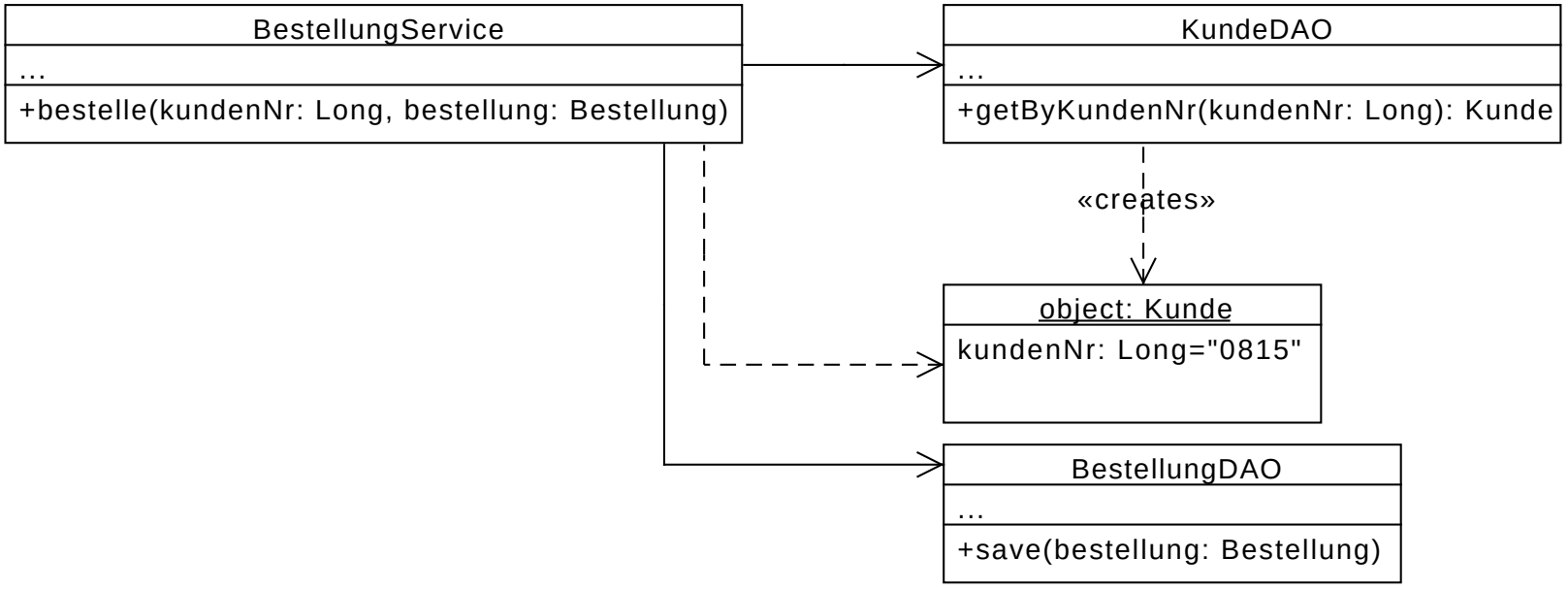
Spring

Beispiel Dependency Injection (DIP)



Spring

Beispiel Dependency Injection (DIP)



Quellen


Apache Maven Homepage


Spring Framework 3 Referenz


Hibernate Quickstart


HyperSQL Documentation


JUnit Cookbook

Fragen?

Fragen?