Tutorial zum Einzelbeispiel
SE&PM PR

Thomas Artner
thomas.artner@inso.tuwien.ac.at

23.2.2015

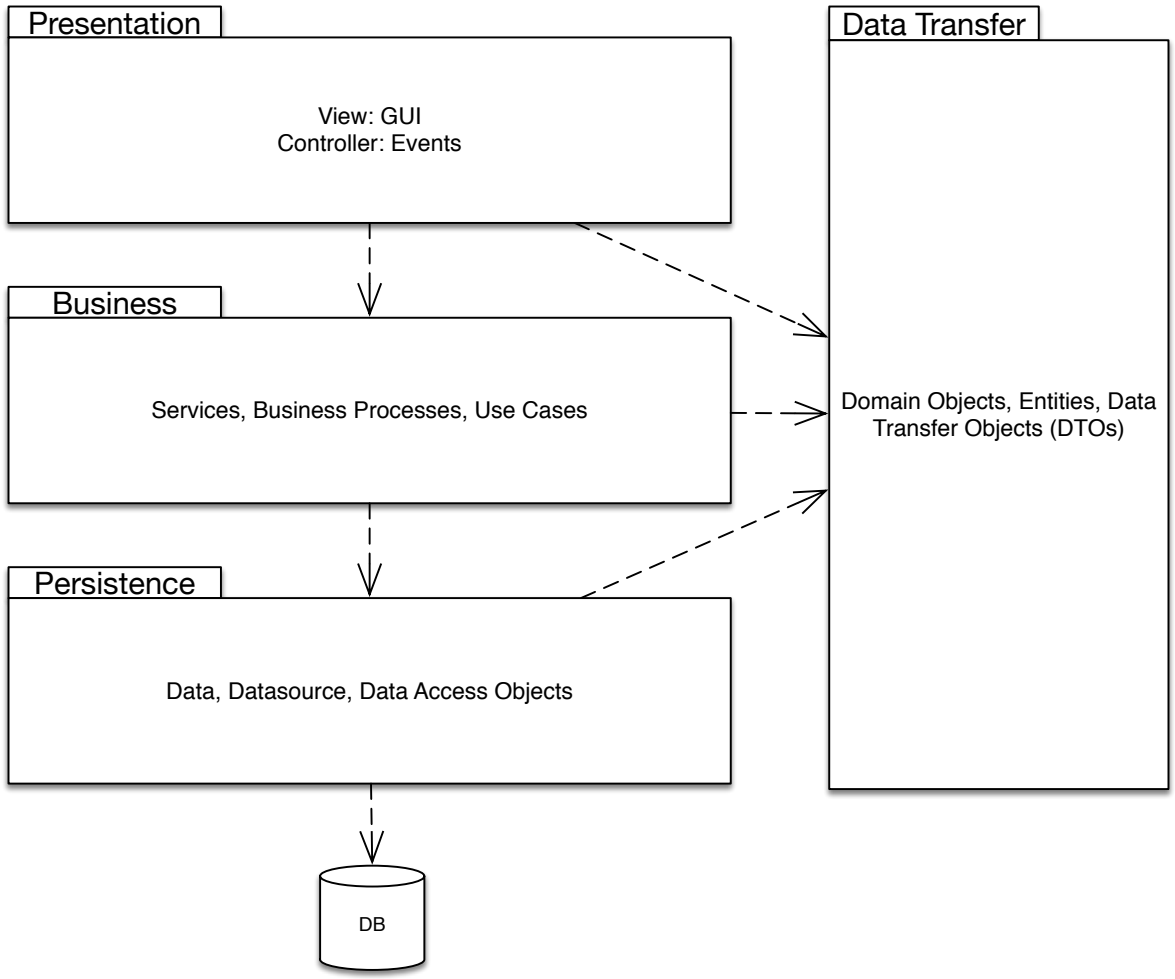
Einzelbeispiel *Wendys Rennpferde*

- Software zur Verwaltung und Durchführung von Pferde-Rennsimulationen
- Verwaltung der Pferde und Jockeys
- Statistische Auswertungen
- Details siehe Angabe und Leitfaden

Werkzeuge und Tools

- Java 8 SDK
- JavaFX 8 als GUI Toolkit
- JavaFX Scene Builder 2
- H2 Datenbank
- Datenbankzugriff via JDBC
- IntelliJ IDEA, Log4j2, JUnit 4
- SVN (Subversion)

Architektur



Entity Objects

- Repräsentieren die in der Datenbank gespeicherten Objekte
- Domänen-, Modelobjekte
- enthalten nur Daten, keine Business Logik
  - private Membervariablen
  - public Getter und Setter

Entity Objects II

```
public class Ware {
    private String name;
    private int price;

    public Ware(String name,
                int price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }
}
```

Domänenmodell

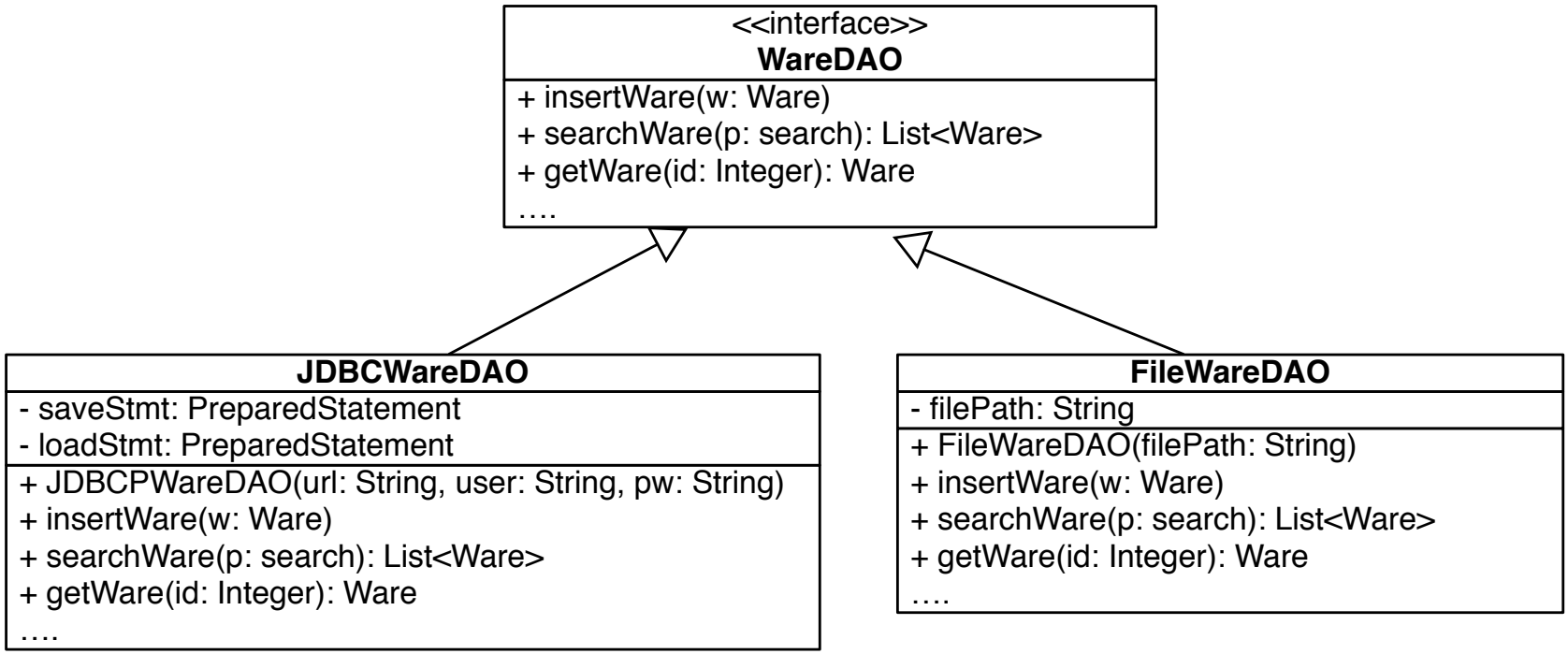
- Abbildung von Objekten der realen Welt
- unabhängig von der konkreten Implementierung

- weiters gefordert:
  - UML-Klassendiagramm
  - EER Diagramm

Data Access Object

- Stellt Methoden für Zugriff auf die Daten in der Datenbank zur Verfügung
- Basic *CRUD* Methoden
  - **C**reate
  - **R**ead
  - **U**pdate
  - **D**elete
- DAO $\leftrightarrow$ Tabelle, nicht zwingend 1:1

Data Access Object II



Löschstrategien

- Anforderung 1: Pferde müssen aus dem System gelöscht werden können.
- Anforderung 2: Rennergebnisse dürfen sich nicht verändern.
- Problem: Was passiert wenn ein Pferd gelöscht wird, das in einem Rennergebnis vorkommt?

Löschstrategien II

- Lösungsansatz 1: Löschen von Pferden mit Rennergebnissen verbieten
  - Konflikt mit Anforderung 1!
- Lösungsansatz 2: Redundante Speicherung
  - kompliziert
- Lösungsansatz 3: isDeleted Flag
  - empfohlen!
- Lösungsansatz 4: ?

Speichern von Bildern

- Lösungsansatz 1: Speichern des Pfades
- Lösungsansatz 2: Bild in ein spezielles Verzeichnis verschieben und Dateinamen speichern
- Lösungsansatz 3: als BLOB in der Datenbank
- Lösungsansatz 4: ?

IntelliJ IDEA

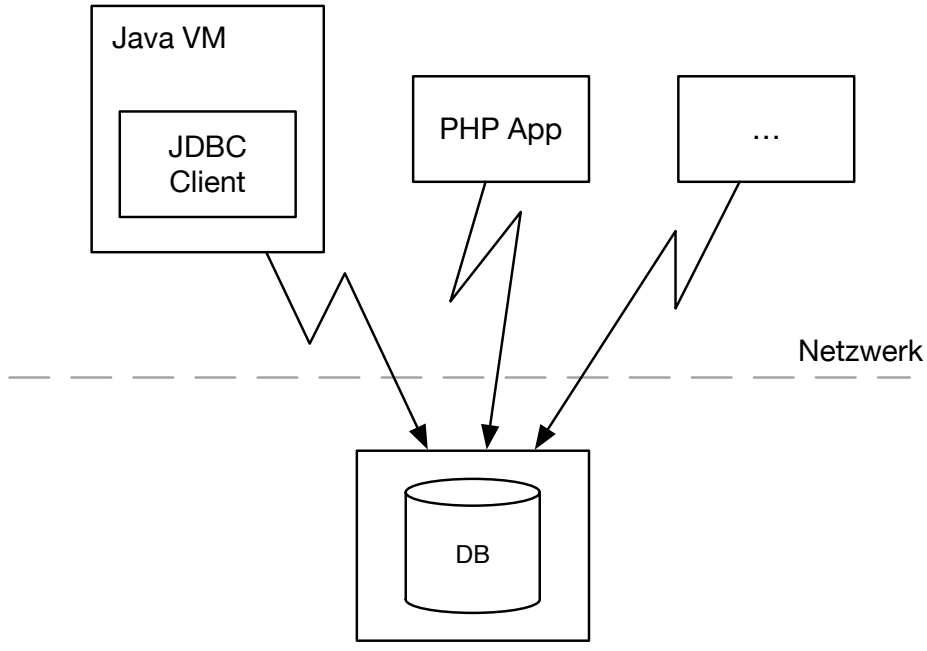
- Integrated Development Environment
- Community Edition
  - Free and Open Source
- Ultimate Edition
  - Edu-Lizenzen werden in der Gruppenphase zur Verfügung gestellt

H2

- relationale Datenbank
- pure Java
- sehr einfach in der Handhabung
- Server- und Embedded Mode
- nur ca. 1MB groß

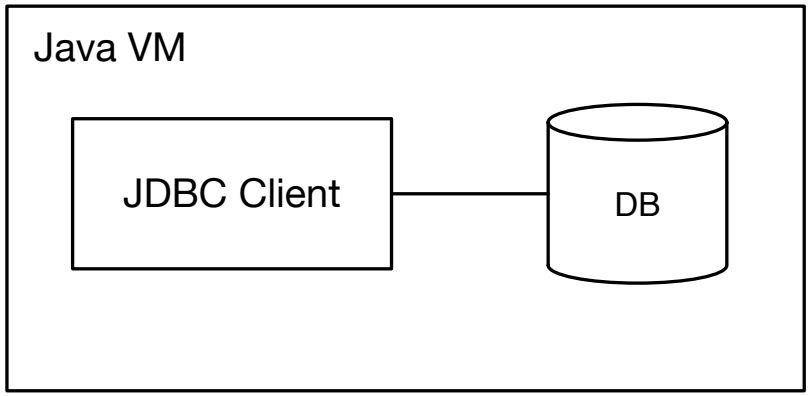
Server Mode

- H2 läuft in einer eigenen Java VM
- horcht auf Verbindungen über Netzwerk
- mehrere Clients können gleichzeitig bedient werden



Embedded Mode

- läuft als Teil der Java Applikation
- läuft in gleicher VM
- Daten werden nicht übers Netzwerk gesendet
- wird mit der Applikation gestartet und beendet



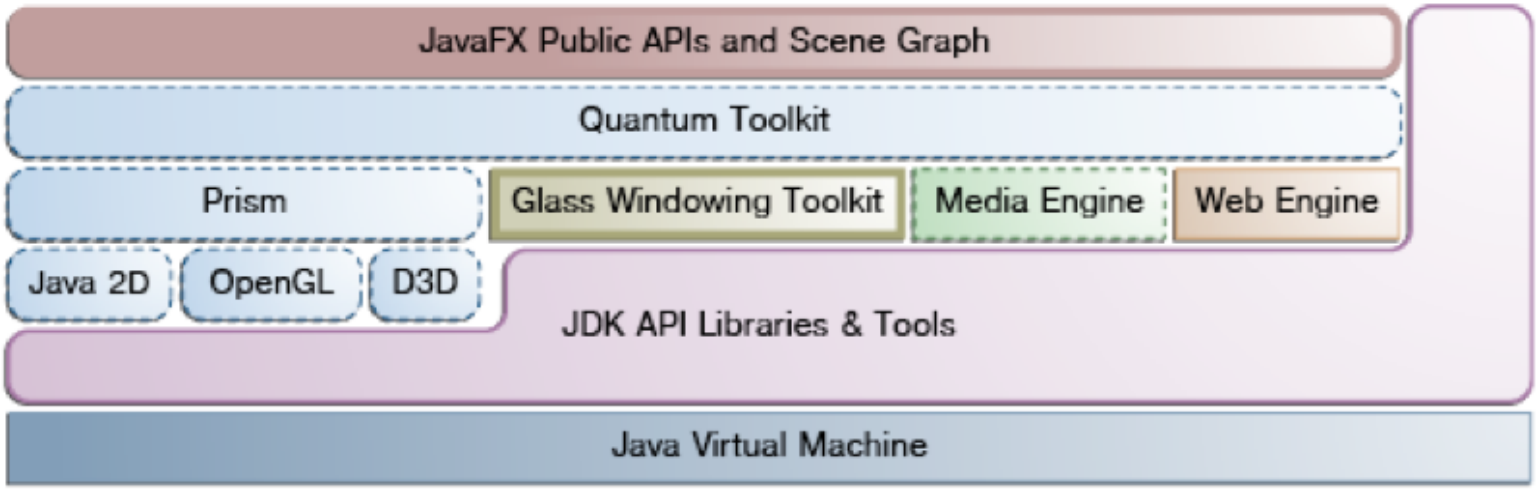
JavaFX I

- plattformübergreifendes UI Toolkit (ab Java 7u6)
- Nachfolger von Swing
- viele Verbesserungen ab Java 8
- designed für Rich Client Applikationen

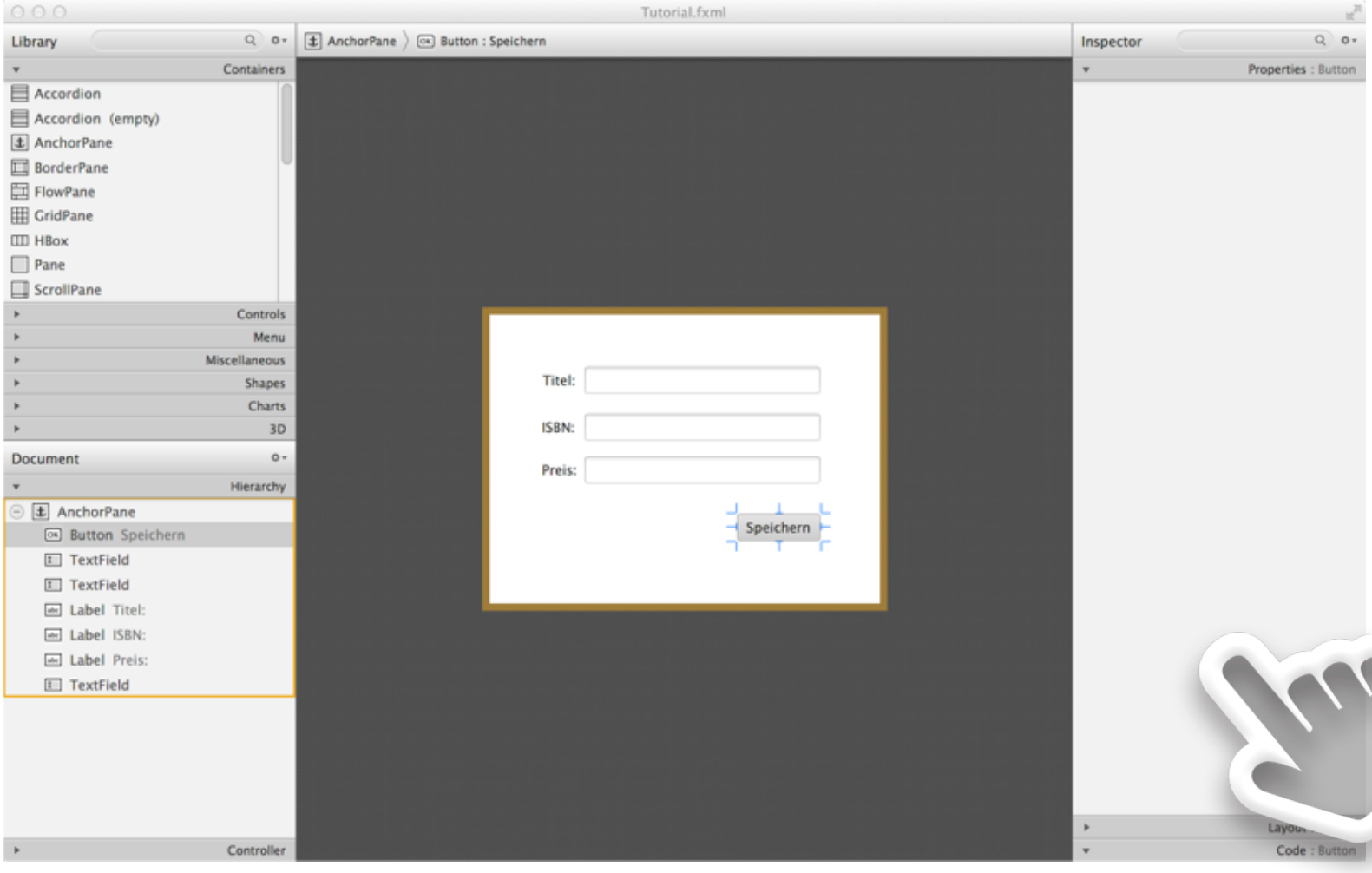
JavaFX II

- FXML und Scene Builder
- Design mittels CSS
- Multitouch Support
- hardwarebeschleunigte Grafikpipeline

JavaFX Architektur



JavaFX Scene Builder



FXML I

```
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
    Infinity" minWidth="-Infinity" prefHeight="277.0" prefWidth="
    374.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
    javafx.com/fxml/1" fx:controller="application.TutorialController
    ">
  <children>
    <Button fx:id="btn_ok" layoutX="237.0" layoutY="191.0"
        mnemonicParsing="false" onAction="#speichernButtonClicked"
         text="Speichern" />
    <TextField fx:id="tf_titel" layoutX="91.0" layoutY="50.0"
        prefHeight="26.0" prefWidth="226.0" />
    <TextField fx:id="tf_isbn" layoutX="91.0" layoutY="95.0"
        prefHeight="26.0" prefWidth="226.0" />
    <Label layoutX="51.0" layoutY="55.0" text="Titel:" />
    <Label layoutX="50.0" layoutY="100.0" text="ISBN:" />
    <Label layoutX="50.0" layoutY="141.0" text="Preis:" />
    <TextField id="tf_preis" layoutX="91.0" layoutY="136.
        prefHeight="26.0" prefWidth="226.0" />
  </children>
</AnchorPane>
```

FXML II

```
...
<TextField fx:id="tf_titel" ... />
<TextField fx:id="tf_isbn" ... />
<TextField fx:id="tf_preis" ... />

<Button onAction="#speichernButtonClicked"
text="Speichern" ... />
...
```

FXML Binding

MainFrame.fxml

```
<TextField fx:id="tf_titel" ... />
<TextField fx:id="tf_isbn" ... />
<TextField fx:id="tf_preis" ... />

<Button
onAction="#speichernButtonClicked"
text="Speichern" ... />
```

MainFrameController.java

```
@FXML
private TextField tf_titel;
@FXML
private TextField tf_isbn;
@FXML
private TextField tf_preis;

@FXML
public void speichernButtonClicked(){
  logger.debug(tf_titel.getText());
  logger.debug(tf_isbn.getText());
  logger.debug(tf_preis.getText());
}
```

JavaFX UI Struktur

- Stage
  - Top-Level Container für eine *Scene*
  - Definiert Titel, Höhe, Breite, Platzierung, ...
- Scene
  - Canvas auf dem der *Scene Graph* gezeichnet wird
- Scene Graph
  - gerichteter azyklischer Graph
  - repräsentiert UI Elemente

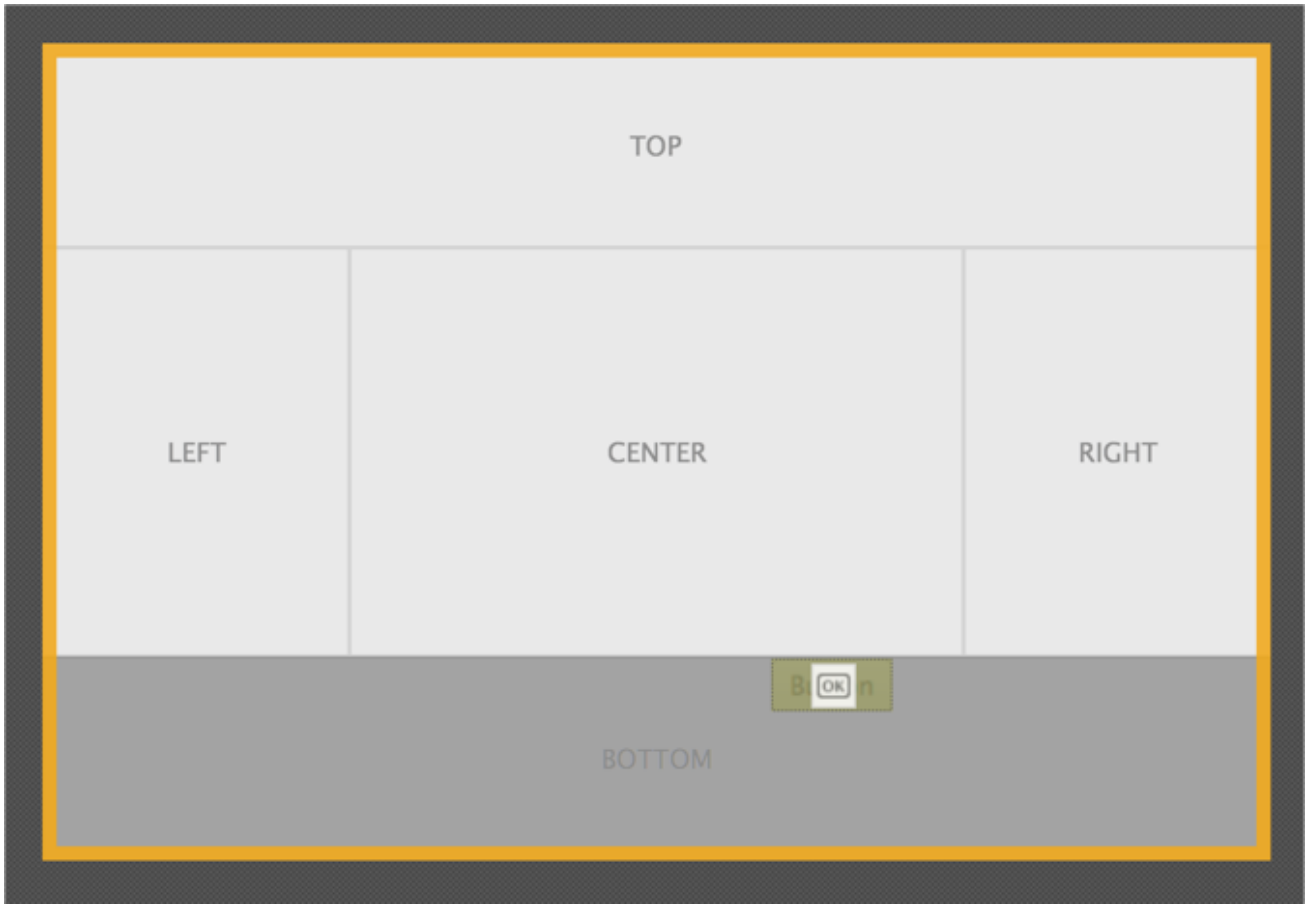
Layout Management I

- absolute Positionierung
  - pixelgenau
  - passt sich an Größenveränderungen des übergeordneten Containers nicht an
- Positionierung mittels Layout Panes
  - Regelwerk entscheidet die Anordnung der Komponenten
  - Komponenten passen sich Größenveränderungen von deren Container an

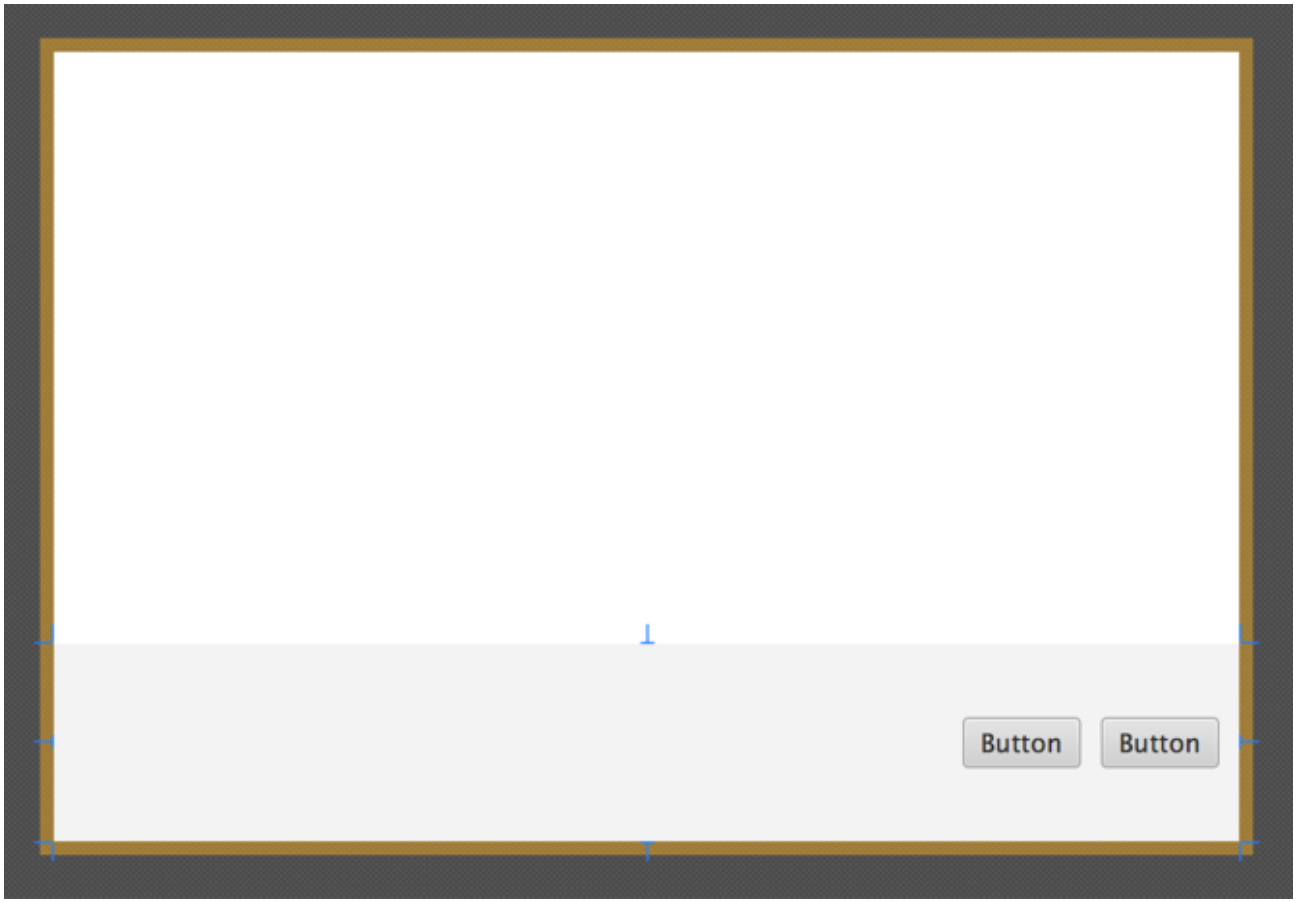
Layout Management II - Anchor Pane

```
<Button layoutX="189.0" layoutY="159.0"
    onAction="#speichernButtonClicked"
    text="Speichern" ...
/>
```

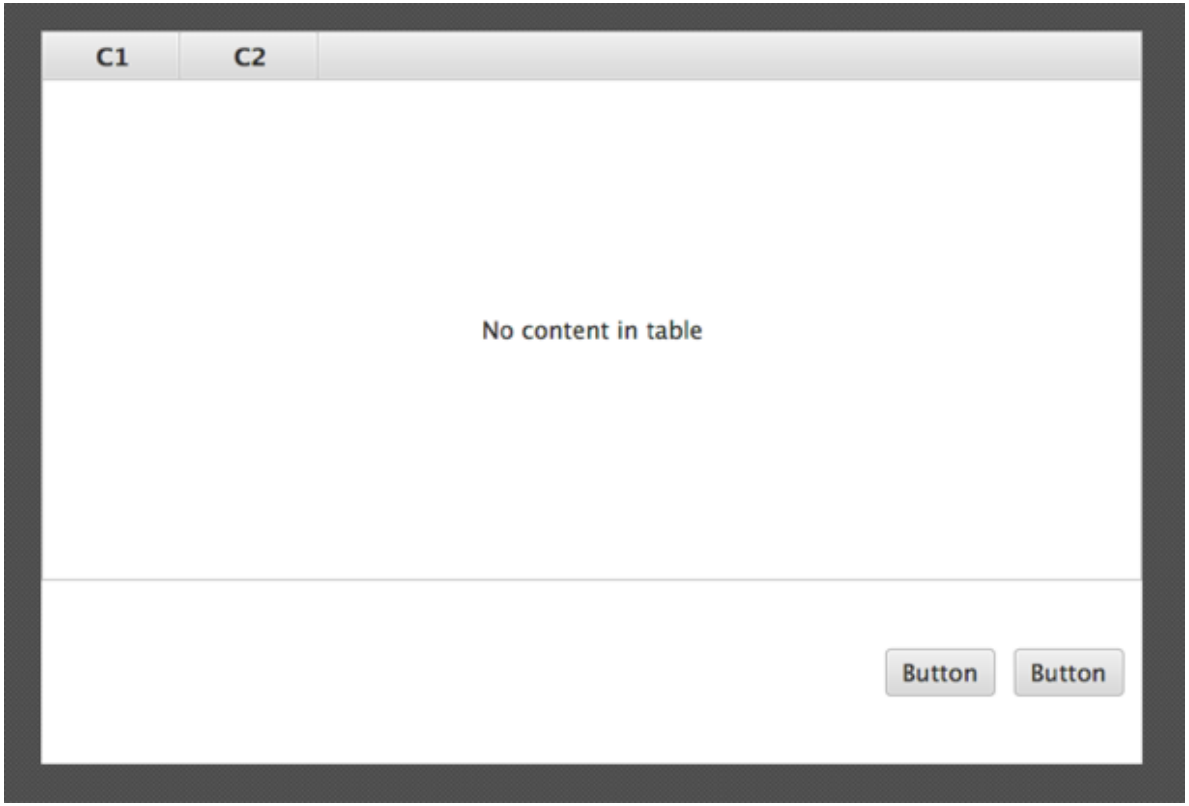
Layout Management III - Border Pane



Layout Management IV - Buttons in HBox in Border Pane (Bottom)



Layout Management V - TableView in Border Pane (Center)

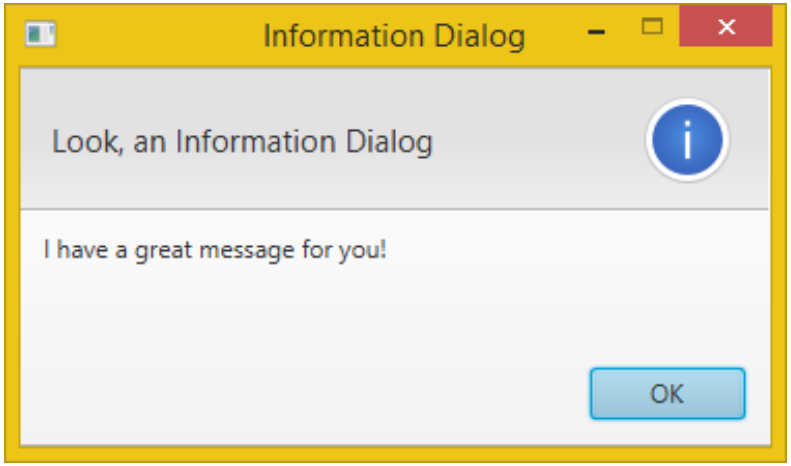


Weitere Layout Panes

- VBox: vertikale Ausrichtung
- GridPane: flexibles Grid
- ...
- `http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm`

Dialoge I

- Erst ab Java 8u40
- Kompatible Bibliothek für frühere Versionen
- In ControlsFX inkludiert *openjfx-dialogs-1.x.x.jar*
- Dokumentation mit Beispielen in TUWEL



Dialoge II

```
Alert alert = new Alert(AlertType.INFORMATION);
alert.setTitle("Information Dialog");
alert.setHeaderText("Look, a info-Dialog");
alert.setContentText("Message for you.");

alert.showAndWait();
```

Logging

- „Low-tech“ Debugging Ansatz
- liefert Feedback zum Systemzustand und über das Systemverhalten
- Log Output wird persistiert
- Log Levels zur Unterscheidung der Relevanz von Log Einträgen

Log4j

- Logging Library für Java
- open source, Verfügbar für Ruby, C#, C++, ...
- klassenbasiertes Logging
- Instanzen des Loggers können überall im Code aufgerufen werden
- `TRACE < DEBUG < INFO < WARN < ERROR < FATAL`

log4j2.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5
                level %logger{36} - %msg%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Console" />
        </Root>
        <Root level="info">
            <AppenderRef ref="Console" />
        </Root>
        <Root level="debug">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>
```

Logging - Beispiel

Java:

```
logger.info("Entering Application");
    ...
    logger.info("Initialise StudentDAO");
    ...
    logger.info("Get Student ID =" + stud.getId());
    ...
    logger.debug("Returning Student" + stud.getFullName());
    ...
logger.info("Exiting Application");
```

Konsolenausgabe:

```
2010-03-07 05:32:30,508 [main] INFO  Entering application.
2010-03-07 05:32:31,780 [main] INFO  at.ac.tuwien.ifs.bpse.basic.dao.JdbcObjectStudentDAO
- Initialise StudentDAO
2010-03-07 05:32:32,477 [main] INFO  at.ac.tuwien.ifs.bpse.basic.dao.JdbcObjectStudentDAO
- Get Student ID = 0
2010-03-07 05:32:33,565 [main] DEBUG at.ac.tuwien.ifs.bpse.basic.dao.JdbcObjectStudentDAO
- Returning Student "Fritz Haber"
2010-03-07 05:32:34,529 [main] INFO  BPSE-Basic - Exiting application
```

Live Demo!

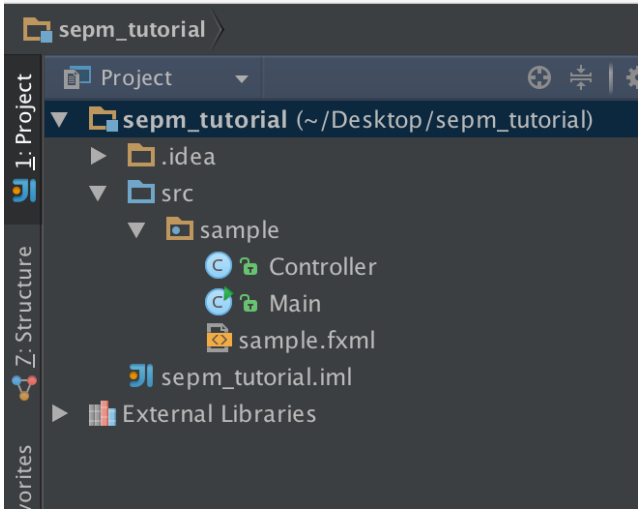
Demo: Installation

- JDK 8 download und Installation
- IntelliJ download und Installation
- JavaFX Scene Builder download und Installation

Demo: Create New Project I

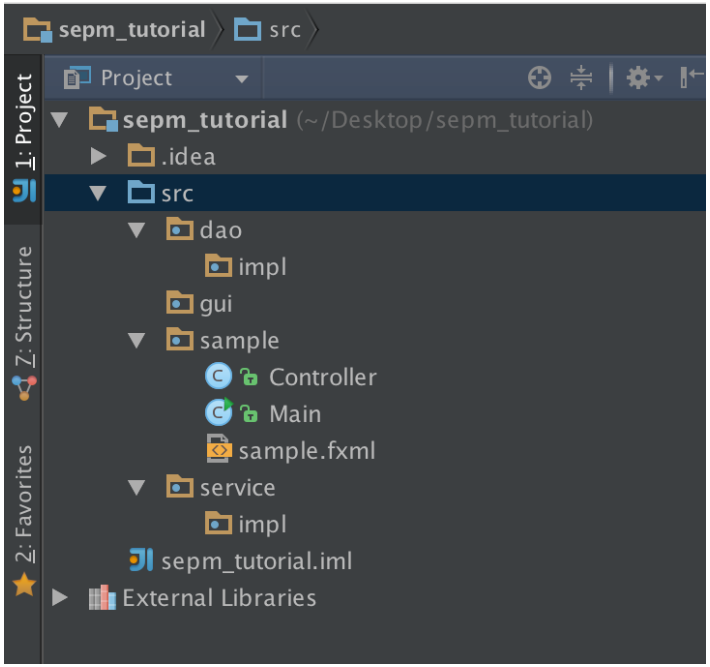
Demo: Create New Project II

- Create New Project → JavaFX → JavaFX Application → Next
- enter Project name → Finish



Demo: Packages anlegen

- dao
  - impl
- gui
- service
  - impl



Demo: Logger initialisieren

- rechte Maustaste auf Projektname → New → Directory
  - neues Directory *lib* anlegen
- *log4j-api-2.x.jar* und *log4j-core-2.x.jar* in das lib directory ziehen
- rechte Maustaste auf die beiden jars → Add as Library → OK
- log4j2.xml in das src Directory kopieren

Demo: Logger testen

```
...
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Main extends Application {

    private static final Logger logger = LogManager.getLogger(Main.
        class);

    @Override
    public void start(Stage primaryStage) throws Exception{...}

    public static void main(String[] args) {
        logger.info("Hello World! :-)");
        launch(args);
    }
}
```

Konsolenausgabe:

```
14:38:49.513 [main] INFO  sample.Main - Hello World! :-)
```

Demo: Entität anlegen

```java
package entity;

public class Buch {

    private String titel;
    private String isbn;
    private Integer preis;

    public String getTitel() {
        return titel;
    }

    public void setTitel(String titel) {
        this.titel = titel;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public Integer getPreis() {
        return preis;
    }

    public void setPreis(Integer preis) {
        this.preis = preis;
    }
}
```

Demo: BuchService Interface anlegen

```java
package service;

import java.util.List;

import entity.Buch;

public interface BuchService {
    public Buch create(Buch b);
    public List<Buch> find(Buch b);
    public void update(Buch b);
    public void delete(Buch b);
}
```

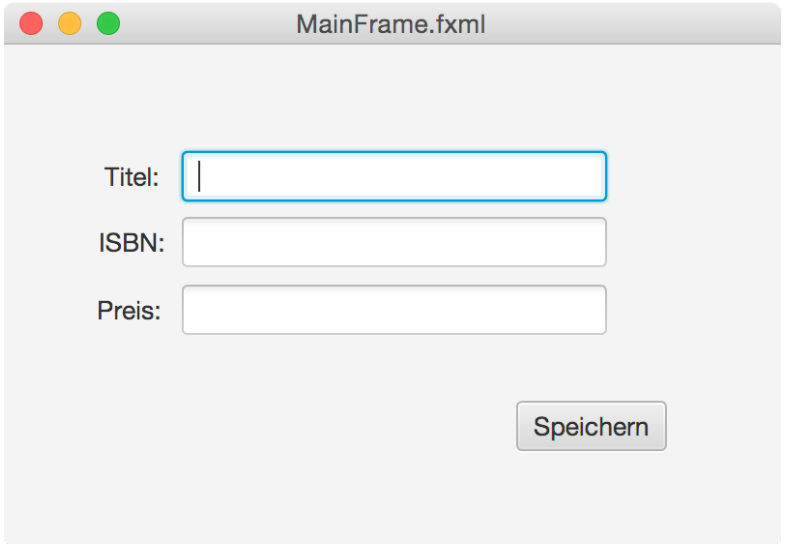
Demo: Dummy BuchServiceImpl anlegen

```
public class BuchServiceImpl implements BuchService {

    private static final Logger logger = LogManager.getLogger
        (BuchServiceImpl.class);

    @Override
    public Buch create(Buch b) {

    logger.info("Das Buch mit dem Titel " + b.getTitel() +
       " soll an dieser Stelle gespeichert werden!");

    return null;
    }
...
}
```

Demo: UI mit Scene Builder erstellen I

- Inhalt vom sample Package in das ui Package verschieben
- rechte Maustaste auf sample.fxml → Refactor → Rename
  - sample.fxml auf MainFrame.fxml umbenennen
- rechte Maustaste auf MainFrame.fxml → Open In SceneBuilder
  - vor erster Verwendung evtl. weitere Konfiguration unter Preferences → Languages and Frameworks → JavaFX nötig!
- UI in SceneBuilder erstellen!

Demo: UI mit Scene Builder erstellen II




```
<TextField fx:id="tf_titel" ... />
<TextField fx:id="tf_isbn" ... />
<TextField fx:id="tf_preis" ... />
<Button
    onAction="#speichernButtonClicked"
  ... />
```

Demo: Controller Bindings

```
public class Controller {

    private static final Logger logger = LogManager.getLogger(
        Controller.class);

    @FXML
    private TextField tf_titel;
    @FXML
    private TextField tf_isbn;
    @FXML
    private TextField tf_preis;

    @FXML
    private void speichernButtonClicked(){
        logger.info("Speichern Button clicked!");
    }

}
```

Run It!

Versionierung

- Dateien über die Zeit verfolgen
  - Backups
  - Distribution
- gleiche Datei, mehrere Versionen
- zentrale vs. verteilte Versionierung

Subversion (SVN)

- wird zur Verfügung gestellt
  - ein paar Tage nach dem Einstiegstest
- ein Repository pro Projekt
- zentrale Versionierung

Testing

- Systemtest
  - z.B. über das UI mit Datenbank
  - Abnahmetests, Test des Gesamtsystems
- Integrationstest
  - testet das Zusammenspiel zwischen Komponenten
- Unit Tests
  - Tests für Klassen oder einzelnen Komponenten

Testautomatisierung

- Automatisierung essentiell
- oftmalige Testdurchführung nach Änderungen im Code bzw. vor dem SVN CheckIn
- Refactoring (Testfallbibliothek)
- Tests sind lebende Dokumentation
- Tests zeigen wie eine Klasse oder eine Komponente verwendet werden soll

Testklassen mit JUnit

- eine JUnit Testfixture ist eine Klasse, die eine oder mehrere Testmethoden enthält (`@Test`)
- zusätzlich enthält eine Testfixture Methoden, die mit `@Before` bzw. `@After` annotiert sind. Diese werden vor bzw. nach jeder Testmethode ausgeführt.
- um vor oder nach allen Tests Code auszuführen gibt es die `@BeforeClass` und die `@AfterClass` Annotation

Testfixture

```java
public class TestBookManager {

    @Before
    public void setUp() throws Exception {
        // Stelle Vorbedingungen fuer Testfaelle her
        // (z.B.: Testdaten einfuegen)
    }
    @After
    public void tearDown() throws Exception {
        // Stelle Ausgangszustand wieder her
        // (z.B.: Testdaten loeschen)
    }
    @Test
    public void saveBook_shouldSaveBookToDB() {
        // Testfall z.B.: speichere Buch, lese Buch
        // ueberpruefe ob Buch korrekt gespeichert
        // wurde (assert!)
    }
    @Test
    public void searchBook_shouldReturnSEBook(){
        //...
    }
}
```

Ablauf

1,4

3,6

2 (5)

5 (2)

JUnit 4 Asserts

- soll/ist Vergleich
- weicht der erwartete Wert vom aktuellen Wert ab, schlägt der Test fehl
- Hamcrest Matcher
- `assertThat(actual value, is(expected value));`
- z.B.: `assertThat(book.getISBN(), is("123"));`
- `assertThat(value, is(not(value)));`
- `assertThat(value, is(notNullValue()));`
- `assertThat(value, is(greaterThanOrEqualTo(0)));`

Normalfall, Fehlerfall

- Normalfall
  - Entspricht das Verhalten den Erwartungen bei korrekter Eingabe?
  - berechnetes Ergebnis prüfen
- Fehlerfall
  - Entspricht das Verhalten den Erwartungen bei fehlerhafter Eingabe?
  - Exceptions prüfen

Qualitative Tests I

- Asserts und/oder ExpectedException
- aussagekräftige Tests
  - `assertThat(actual value, is(expected value));`
- möglichst kurz
- aussagekräftige Namen

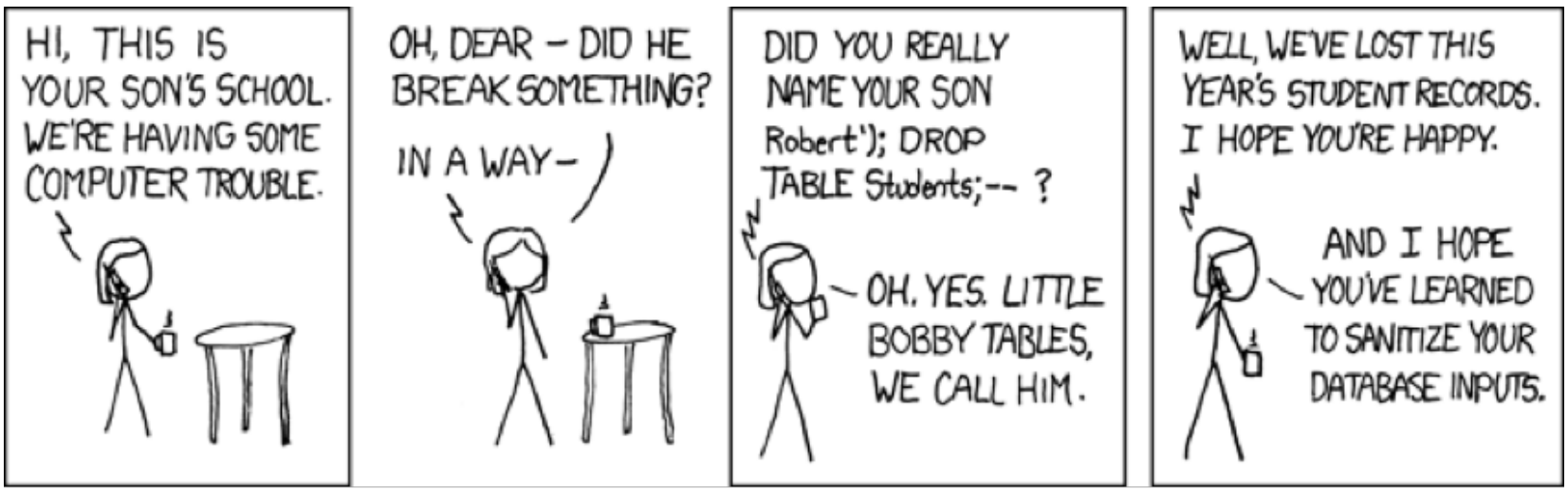
Qualitative Tests II

- keine Abhängigkeiten zwischen Tests
- verschiedene Methoden unabhängig testen
- Komplexität gering halten
  - vermeiden von Schleifen
  - vermeiden von Conditions (if, switch, ...)
  - vermeiden von Exception Handling (try/catch)

]Ausgeklammert[

- SE Prozess
- Security

- Transactions



Quelle: xkcd.org

Fragen/Probleme?

- Leitfaden
- Google
- Stackoverflow
- Kolleginnen und Kollegen
- TUWEL Forum
- Laborstunden

Links I

- Java(FX)
  - http://docs.oracle.com/javase/8/docs/api/
  - http://docs.oracle.com/javafx/
  - http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html
- H2
  - http://www.h2database.com/
  - http://www.h2database.com/html/cheatSheet.html
- Log4j
  - http://logging.apache.org/log4j/2.x/

Links II

- DAO Pattern
  - `http://bpse.ifs.tuwien.ac.at/patterns/dao.html`
- MVC Pattern
  - `http://bpse.ifs.tuwien.ac.at/patterns/mvc.html`
- Designpattern
  - `http://sourcemaking.com/`