Anwendungsentwicklung II

Sebastian Scholz, M.Sc.

FHDW
Fachhochschule der Wirtschaft

10. Oktober 2017

Verwendungshinweise

- Dieses Skript ist **nur** für den internen Gebrauch an der FHDW bestimmt
- Es darf nicht anderweitig zur Verfügung gestellt werden (gilt besonders für die Verbreitung im Internet!)

Rahmenbedingungen

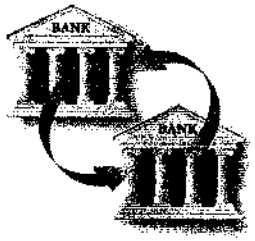
ECTS	6 Punkte
Kontaktstunden	44 Stunden
Selbststudium	136 Stunden
Teilnahmevoraussetzungen	Objektorientierte Programmierung Software Engineering Algorithmen und Datenstrukturen Datenmodel. u. SQL-Programmierung Projekt- und Teammanagement Anwendungsentwicklung I
Art der Prüfungsleistung	Projektarbeit
Kontakt	sebastian.scholz@fhdw.de

Literatur - REST

Titel REST und HTTP
Autoren Stefan Tilkov
Verlag dpunkt.verlag

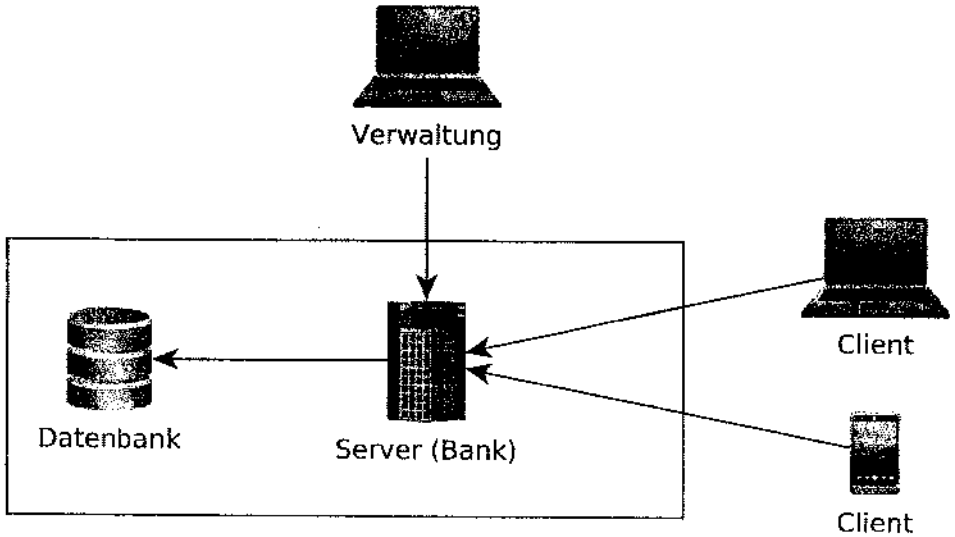
Projektarbeit - Thema

Konzeption, Realisierung und Dokumentation einer Client-Server-Anwendung zur Durchführung und Verwaltung von Banktransaktionen.



FHDW

Projektarbeit - Architektur

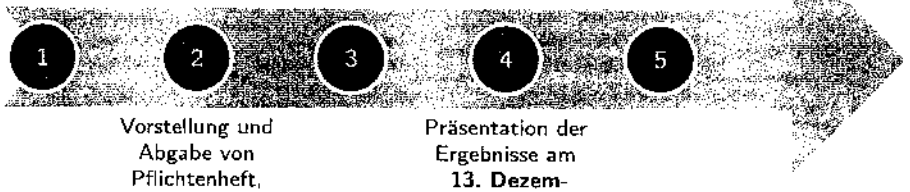


Projektarbeit - Ablauf

Verbindliche Gruppeneinteilung an sebastian.scholz@fhdw.de bis zum **11. Oktober 2017**

Besprechung der Projektfortschritte pro Gruppe jede Veranstaltung bzw. nach Bedarf

Abgabe der gesamten Ausarbeitung in Papierform bis spätestens **08. Januar 2018** im Sekretariat



Vorstellung und Abgabe von Pflichtenheft, Fachkonzept und Projektplanung am **25. Oktober 2017**

Präsentation der Ergebnisse am **13. Dezember 2017**

FHDW

Agenda

1. Server
   - Application Integration Pattern
   - Hypertext Transfer Protocol
   - RESTful Web-Services
   - Logging
   - Build und Deployment
2. Java-Client
   - Java-Client GUIs
   - JavaFX
   - Web-Service Client-Schnittstelle
3. Android-Client
   - Überblick
   - Entwicklung
   - Web-Service Client-Schnittstelle
4. Web-Client
   - Einführung Web-Entwicklung
   - Angular 2
   - Web-Service Client-Schnittstelle

FHDW

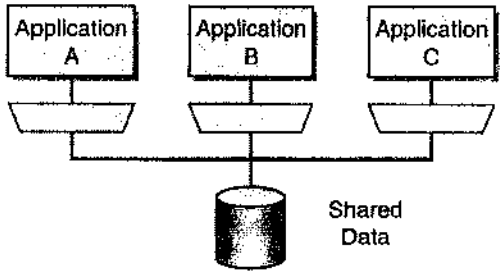
Agenda

Überblick

Möglichkeiten der Anwendungsintegration

- Shared Database
- Remote Procedure Invocation
- Messaging
- File Transfer

Shared Database



- Verschiedene Anwendungen nutzen direkt eine gemeinsame Datenbank zur Integration.
- Das Schema der Datenbank orientiert sich an den Anforderungen aller Anwendungen.

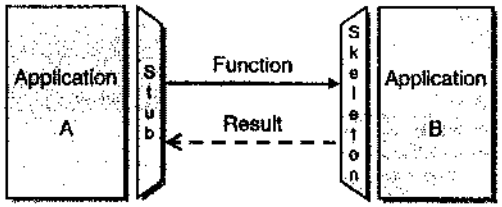
Shared Database - Vorteile

- Änderungen sind sofort für alle Anwendungen sichtbar.
- SQL als Schnittstelle für relationale Datenbanken steht für alle gängigen Plattformen und Programmiersprachen zur Verfügung.
- (Relationale) Datenbanken liefern Transaktionshandling und Konsistenzgarantien.
- Datenbanktreiber führen transparent Typ- und Encoding-Konvertierungen durch.
- Datenbankzugriffe sind für die meisten Entwickler vertrautes Terrain.
- Durch die frühe/tiefe Integration werden Schema-Inkompatibilitäten schnell sichtbar.

Shared Database - Nachteile

- Es ist schwer, ein einheitliches Schema zu entwickeln, das für alle Anwendungen gut nutzbar ist.
- Standard-Anwendungen können schlecht über Shared Databases integriert werden, da ggf. das Schema geändert werden kann.
- Änderungen am Schema betreffen potentiell alle Anwendungen
- Zentralisierte Datenbanken sind meist ungeeignet für räumlich verteile (WAN) Anwendungen.
- Gemeinsam genutzte verteilte Datenbanken "can easily become a performance nightmare."

Remote Procedure Invocation



- Jede Anwendung *kapselt* ihre Daten und Funktionen
- Über definierte Schnittstellen können andere Anwendungen mit der laufenden Anwendung synchron kommunizieren.

Remote Procedure Invocation - Vorteile

- Jede Anwendung kann die Integrität ihrer Daten sicherstellen.
- Jede Anwendung kann ihr internes Datenformat frei verändern
- RPI erlaubt neben der Integration auf Datenebene eine Integration auf Verhaltensebene.
- Verteilung ist für den Entwickler transparent.

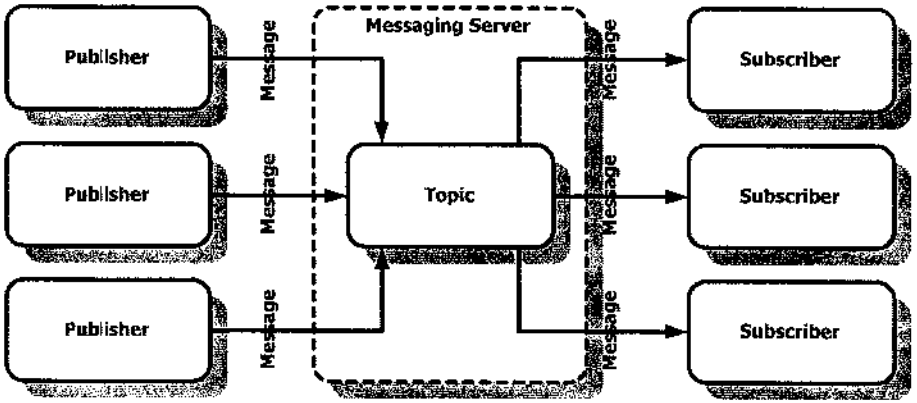
Remote Procedure Invocation - Nachteile

- Verteilung ist für den Entwickler transparent.
- RPI führt zu unzuverlässigen, unperformanten Systemen, wenn die unterscheide zwischen local und remote invocations vernachlässigt werden.
- Anwendungen werden eng miteinander verknüpft.

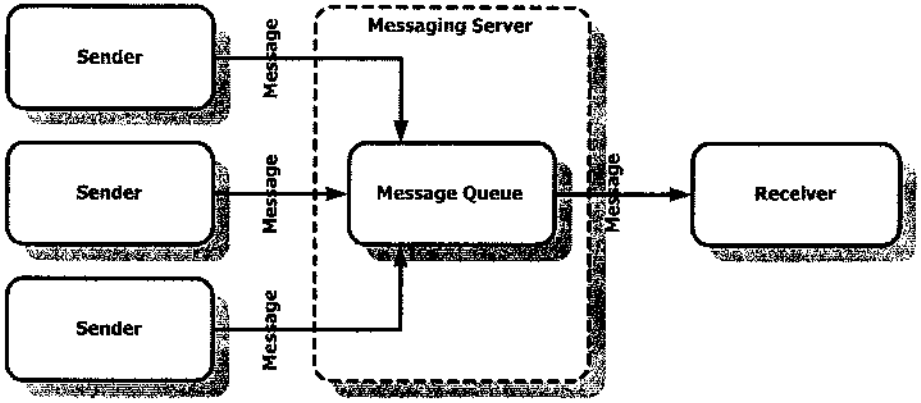
Messaging

- Basiert auf der *asynchronen* Übertragung von Nachrichten (Messages).
- Nachrichtenformat nicht festgelegt, häufig aber XML
- Typische Messaging-Modelle sind
  - Publish-Subscribe Messaging
  - Point-To-Point Messaging

Publish-Subscribe Messaging



Point-To-Point Messaging



Vor- und Nachteile

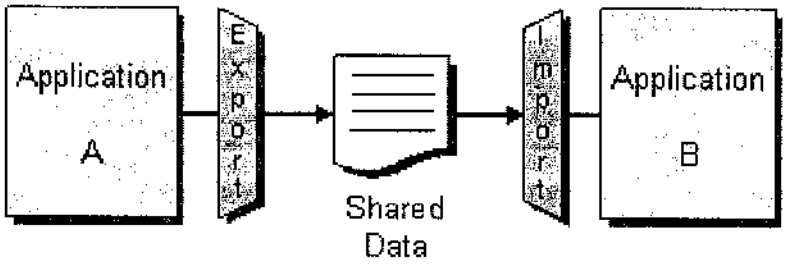
Vorteile

- Messaging-Server übernimmt Weiterleitung an Clients bzw. Server
- Nur noch eine Verbindung zu verwalten
- Caching von Nachrichten
- Lose Kopplung

Nachteile

- Durch Nebenläufigkeit schwieriger zu debuggen
- Ausfall des Messaging-Server führt zu Ausfall des gesamten Systems (Single Point of Failure)

File Transfer Integration



- Jede Anwendung erzeugt Dateien, welche die andere Anwendung konsumiert
- Sehr populär in Form von Webservices

Vor- und Nachteile

Vorteile

- In jeder Programmiersprache / Umgebung umsetzbar
- Erfordert kein Wissen über Interna der beteiligten Anwendung, d.h. lose Kopplung
- Interne Änderungen bei gleichbleibendem Dateiformat sind problemlos möglich
- Keine zusätzlichen Tools/Libraries notwendig
- Dateiaustausch über etablierte Protokolle möglich (Filesystem, NFS, FTP, HTTP, etc.)

Nachteile

- Erfordert mehr Programmieraufwand für die Umsetzung
- Dateiaustausch passiert üblicherweise in größeren Abständen, daher ggf. Probleme mit der Synchronität

Austauschformate

Typische Austauschformate sind:

- Fixed Format File
- Comma-Separated Values (CSV)
- JavaScript Object Notation (JSON)
- Extensible Markup Language (XML)

Austauschformate - Fixed Format File

```
0815 100      Sparkasse
4711 500      Volksbank
5577 2000     ING-DIBA
```

Austauschformate - Comma-Separated Values (CSV)

```
0815, 100, Sparkasse
4711, 500, Volksbank
5577, 2000, ING-DIBA
```

Extensible Markup Language (XML)

```
<konten>
  <konto>
    <nr>0815</nr>
    <saldo>100</saldo>
    <bank>Sparkasse</bank>
  </konto>
  <konto>
    <nr>4711</nr>
    <saldo>500</saldo>
    <bank>Volksbank</bank>
  </konto>
  <konto>
    <nr>2000</nr>
    <saldo>100</saldo>
    <bank>ING-DIBA</bank>
  </konto>
</konten>
```

Austauschformate - JavaScript Object Notation (JSON)

```
{
  "konten" : [
    {
      "nr" : "0815",
      "saldo" : "100",
      "bank" : "Sparkasse"
    }, {
      "nr" : "4711",
      "saldo" : "500",
      "bank" : "Volksbank"
    }, {
      "nr" : "5577",
      "saldo" : "2000",
      "bank" : "ING-DIBA"
    }
  ]
}
```

Agenda

Hypertext Transfer Protocol (HTTP)

- HTTP ist ein Protokoll zur Übertragung von Daten über ein Netzwerk.
- HTTP wurde zusammen mit URL und HTML ab 1989 von Roy Fielding, Tim Berners-Lee et al. am CERN entwickelt
- HTTP ist ein *zustandsloses* Protokoll
- Zur Kommunikation wird üblicherweise TCP als Transportprotokoll eingesetzt
- Durch Erweiterung seiner Anfragemethoden, Header-Informationen und Statuscodes ist HTTP nicht auf Hypertext beschränkt

Kommunikationsbestandteile

- In HTTP werden zwischen Client und Server zwei unterschiedliche Arten von Nachrichten ausgetauscht: Auf die Anfrage (*Request*) vom Client an den Server folgt die Antwort (*Response*) des Servers an den Client.
- Jede Nachricht besteht aus *Header* und *Body*. Der Header enthält Informationen über den Body, wie etwa verwendete Kodierungen oder den Inhaltstyp und ermöglicht dem Empfänger die korrekte Intepretation der im Body enthaltenen Nutzdaten.

Kommunikationsablauf - Beipiel

Request

```
GET /infotext.html HTTP/1.1
Host: www.example.net
```

Response

```
HTTP/1.1 200 OK
Server: Apache/1.3.29 (Unix) PHP/4.3.4
Content-Length: 123
Content-Language: de
Content-Type: text/html
```

(Inhalt von infotext.html)

HTTP-Request-Methoden

GET	Fordert eine Ressource (z. B. eine Datei) unter Angabe eines URL vom Server an.
POST	Schickt (unbegrenzte) Mengen an Daten zur weiteren Verarbeitung zum Server. Beispielsweise als Key-Value-Paare aus einem HTML-Formular.
PUT	Dient dazu eine Ressource (z. B. eine Datei) unter Angabe des Ziel-URLs auf einen Webserver hochzuladen.
DELETE	Löscht die angegebene Ressource auf dem Server.
HEAD	Anfrage an den Server, die gleichen HTTP-Header wie bei GET auszuliefern, nicht jedoch den eigentlichen Dokumentinhalt (Body) zu senden.

Parameterübertragung

Häufig will der Besucher einer Website spezielle Informationen senden. Dazu stellt HTTP prinzipiell zwei Möglichkeiten zur Verfügung:

HTTP-GET Die Daten sind Teil der URL und bleiben deshalb beim Speichern oder der Weitergabe des Links erhalten.

HTTP-POST Übertragung der Daten mit einer speziell dazu vorgesehenen Anfrageart im HTTP-Body, so dass sie in der URL nicht sichtbar sind.

Beispiel (HTTP-GET Query-Parameter):
http://www.wikipedia.org/Search?search=FHDW&language=de

HTTP-Statuscodes

Ein HTTP-Statuscode wird von einem Server auf jede HTTP-Anfrage als Antwort geliefert. Es werden folgende Kategorien von Statuscodes unterschieden:

- 1xx Informationen
- 2xx Erfolgreiche Operation
- 3xx Umleitung
- 4xx Client-Fehler
- 5xx Server-Fehler

Bekannte HTTP-Statuscodes

200 **OK** - Die Anfrage wurde erfolgreich bearbeitet und das Ergebnis der Anfrage wird in der Antwort übertragen.

301 **Moved Permanently** - Die angeforderte Ressource steht ab sofort unter der im Location-Header-Feld angegebenen Adresse bereit. Die alte Adresse ist nicht länger gültig.

400 **Bad Request** - Der Request war fehlerhaft aufgebaut.

404 **Not Found** - Die angeforderte Ressource wurde nicht gefunden.

500 **Internal Server Error** - Sammel-Statuscode für unerwartete Serverfehler.

Agenda

REST (Representational State Transfer) Grundprinzipien

- Ressourcen mit eindeutiger Identifikation
- Verknüpfungen
- Standardmethoden
- Unterschiedliche Repräsentationen
- Statuslose Kommunikation

FHDW

Eindeutige Identifikation

- URI (Uniform Resource Identifier) als ID für Ressourcen (Webseiten, Web-Services, Dateien etc.)
- Bilden einen globalen Namensraum (weltweit eindeutig)
- Beispiel: http://example.com/products/

FHDW

Verknüpfungen

- Über Links werden Ressourcen miteinander verknüpft
- Verknüpfungen funktionieren anwendungsübergreifend (etwa auf einen anderen Server)
- Ermöglicht Steuerung der Anwendung, indem neue Aktionen über Links bereitgestellt werden (z. B. eine Bestellung durchführen oder abbrechen)

FHDW

Standardmethoden

GET	Stellt Repräsentation einer Ressource zur Verfügung
PUT	Erzeugt oder aktualisiert eine Ressource
POST	Erzeugt neue Ressource oder stößt eine Verarbeitung an
DELETE	Löscht die angegebene Ressource

FHDW

Unterschiedliche Repräsentationen

- Nicht jeder Client kann mit jedem Datenformat umgehen
- Daher ist es sinnvoll, für jede Ressource unterschiedliche Datenformate (Repräsentationen) bereit zu stellen (z. B. html, xml)
- Der Client kann bei der Ressourcenanfrage ein gewünschtes Datenformat mitschicken

```
//HTTP Content Negotiation
GET /products
Host: example.com
Accept: application /xml
```


JSON

- JSON (JavaScript Object Notation) ist ein sehr einfach gehaltenes Repräsentationsformat für Datenstrukturen
- Stammt aus dem JavaScript Umfeld und kann daher auch mit dem JavaScript Schlüsselwort eval() zu einem JavaScript Objekt umgewandelt werden
- Ist aber grundsätzlich unabhängig von der Programmiersprache, da auch in vielen anderen Sprachen Parser für JSON existieren
- Spart im Vergleich zu XML in der Regel Speicher

JSON Datentypen

- Objekt ({ })
  - Enthält Attribute die aus Key (Zeichenkette) und Value (beliebiger unterstützter Datentyp) bestehen
  - Key und Value werden durch einen ":"getrennt
- Array ([ ])
  - Enthält eine Liste beliebiger Werte unterstützter Datentypen
- Zahl
- Zeichenkette (" ")
- Boolescher Wert (true/false)
- Null (null)

JSON Beispiel

```
{
    "orderDate": "2013.01.01",
    "total": 349.99,
    "canceled": false,
    "orderItems": [
        {
            "productName": "Thinkpad T60",
            "price": 149.99
        },
        {
            "productName": "Thinkpad X60",
            "price": 200
        }
    ]
}
```

REST mit Jersey und Jetty

- Jetty dient als Webserver und Container für den REST-Service
- Der eigentliche REST-Service wird von Jersey bereitgestellt

REST mit Jersey und Jetty - Konfiguration

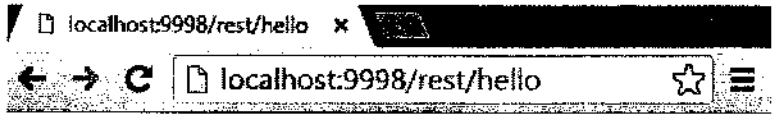
```
//Jetty zusammen mit Jersey starten
Server server = new Server(9998);
ResourceConfig rc = new PackagesResourceConfig("de.rest.resources");
ServletContextHandler sh = new ServletContextHandler();
sh.setContextPath("/rest");
sh.addServlet(new ServletHolder(new ServletContainer(rc)), "/*");
server.setHandler(sh);
server.start();
```


REST mit Jersey und Jetty - Hello World

```
//REST-Service mit Jersey erzeugen
package de.rest.resources;
 [...]
@Path("/")
@Singleton
public class RestResource {
    @GET
    @Path("/hello")
    @Produces({ MediaType.TEXT_PLAIN + "; charset=utf-8" })
    public String hello(){
        Response.ok("Hello World").build();
    }
}
```


REST mit Jersey und Jetty - Hello World

Der REST-Service kann nun im Browser wie folgt aufgerufen werden:



REST mit Jersey und Jetty - Pfadparameter

```
@GET
@Path("/hello/{name}")
@Produces({ MediaType.TEXT_PLAIN + "; charset=utf-8" })
//Aufruf mit Parameter
public Response helloName(@PathParam("name") String name) {
    return Response.ok("Hello " + name).build();
}
```


REST mit Jersey und Jetty - Pfadparameter

Objekt als JSON zurückgeben

Jersey kann Objekte direkt als JSON zurückgeben, wenn folgende Voraussetzungen erfüllt sind:

- @Produces der REST-Methode wird mit dem Medientyp JSON konfiguriert
- Die Klasse des Objekts wird mit @XmlRootElement annotiert
- Außerdem enthält die Klasse Getter/Setter für die relevanten Attribute sowie einen Defaultkonstruktor (also ohne Parameter)
- Optional: Attribute der Klasse können mit der Annotation @XmlTransient an der entsprechenden get-Methode ignoriert werden

Übungsaufgabe

- Erstellen Sie basierend auf dem Beispielprojekt einen REST-Service mit Jetty und Jersey
- Dieser soll eine GET-Methode enthalten, welche den Inhalt eines Ordners auf dem Dateisystem im JSON-Format zurückgibt
- URL zum Testen des bestehenden REST-Service im Beispielprojekt: **http://localhost:9998/rest/hello**

Agenda

1. Server
   - Application Integration Options
   - Hypertext Transfer Protocol
   - REST ful Web-Services
   - Logging
   - Build und Deployment
2. Java-Client
   - Java-Client GUI
   - JavaFX
   - Web-Service Client-Schnittstelle
3. Android-Client
   - Überblick
   - Entwicklung
   - Web-Service Client Schnittstelle
4. Web-Client
   - Einführung Web-Entwicklung
   - Angular 2
   - Web-Service Client-Schnittstelle

Allgemein

- Protokolliert das Ablaufverhalten eines Programms
- Lässt sich an einer zentralen Stelle konfigurieren
- Für die verteilten Log-Nachrichten im Programm kann so an einer Stelle definiert werden, welches Layout, Ausgabeort (Konsole, Datei, Datenbank etc.) und Art von Log-Nachrichten angezeigt werden sollen
- Kann ein Debuggen des Codes ersparen, um Fehler zu finden

Logging-Level

- Beschreibt die Priorität der Log-Nachrichten
- Wird für jede Log-Nachricht einzeln festgelegt
- Vor der Ausführung des Programms kann festgelegt werden, ab welcher Priorität Log-Ausgaben zur Laufzeit erzeugt werden sollen
- Durch diese Filterung kann je nach Einsatzzweck eine niedrige Priorität bei der Fehlersuche oder eine hohe Priorität im Produktivbetrieb (zur Ressourcenschonung) verwendet werden

Logging-Level

Häufig verwendete Logging-Level:

Fatal	Ein Fehler, der die weitere Ausführung verhindert, ist aufgetreten (Inkonsistenz bei Daten)
Error	Fehler aufgetreten (SqlException)
Warning	Unerwartete Situation (Eingabewert hat falsches Format)
Info	Grobe Informationen zum Programmablauf (Verarbeitung abgeschlossen)
Debug	Informationen zum Programmablauf (Teilaufgaben einer Verarbeitung)
Trace	Detaillierte Informationen zum Programmablauf inklusive Werten von Variablen (Methode mit Parameter a,b mit Werten x,y aufgerufen)

Beispielimplementierung Log4j

```
//Konfiguration
//=======
Logger logger = Logger.getRootLogger();
SimpleLayout layout = new SimpleLayout();
ConsoleAppender appender = new ConsoleAppender(layout);
logger.addAppender(appender);
logger.setLevel(Level.ALL);

//Anwendung
//======
Logger logger = Logger.getLogger(getClass());
logger.info("Server gestartet");
```


Agenda

Build und Deployment

Build- und Deployment-Prozess

- Kompilieren der Anwendung
- Automatisierte Tests (JUnit)
- Berechnung von Qualitätsmetriken
- Paketierung zu einer lauffähigen Anwendung (ggf. für unterschiedliche Plattformen)
- Bereitstellung der Anwendung

Um Fehler zu vermeiden, die Entwicklungsgeschwindigkeit zu erhöhen und Vertretbarkeit sicherzustellen, sollten diese Schritte soweit wie möglich **automatisiert** erfolgen.

Ant Buildscript Beispiel

```
<project>
    //Source kompilieren
    <target name="compile">
        <javac srcdir="src" destdir="bin" />
    </target>

    //Jar erstellen
    <target name="buildServer" depends="compile">
        <jar destfile="server.jar">
            //Kompilierte Klassen ins Jar packen
            <fileset dir="bin" />
            //Alle Libraries in das Jar extrahieren
            <zipfileset src="lib/derby.jar" />
            [...]
        </jar>
    </target>
</project>
```

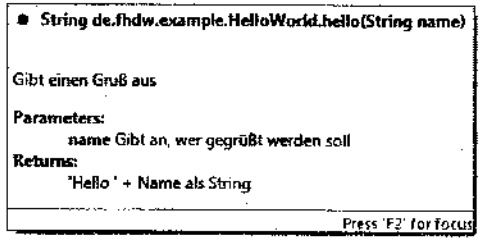
JavaDoc Beispiel

```java
/**
 * Dient als Beispielklasse für das JavaDoc
 * @author Sebastian Scholz
 */
public class HelloWorld {
    /**
     * Gibt einen Gruß aus
     * @param name Gibt an, wer gegrüßt werden soll
     * @return 'Hello ' + Name als String
     */
    public String hello(String name){
        return "Hello " + name;
    }
}
```


JavaDoc Beispiel

Die Methode besitzt nun folgenden Tooltip:

```java
public static void main(String[] args) {
    new HelloWorld().hello("Hugo");
}
```

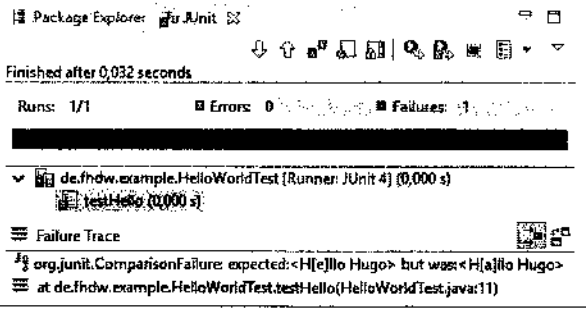


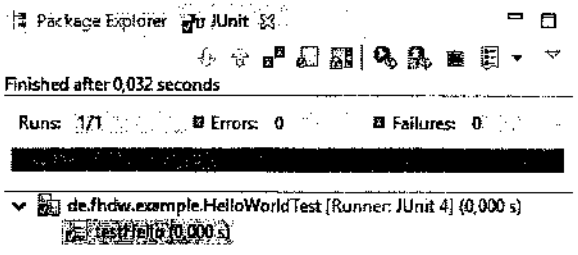
Tipp (Eclipse): Mauscursor auf Methoden/Klassennamen setzen und ALT+SHIFT+J drücken, um automatisch den JavaDoc-Rumpf zu erstellen.

JUnit Beispiel

```java
import org.junit.Test;
import org.junit.Assert;

public class HelloWorldTest {
    @Test
    public void testHello(){
        String name = "Hugo";
        String result = new HelloWorld().hello(name);
        //Vergleich von Erwartung und tatsächlicher Rückgabe
        Assert.assertEquals("Hallo " + name, result);
    }
}
```


JUnit Beispiel - Ergebnis





Agenda

Java-Client GUI-Bibliotheken

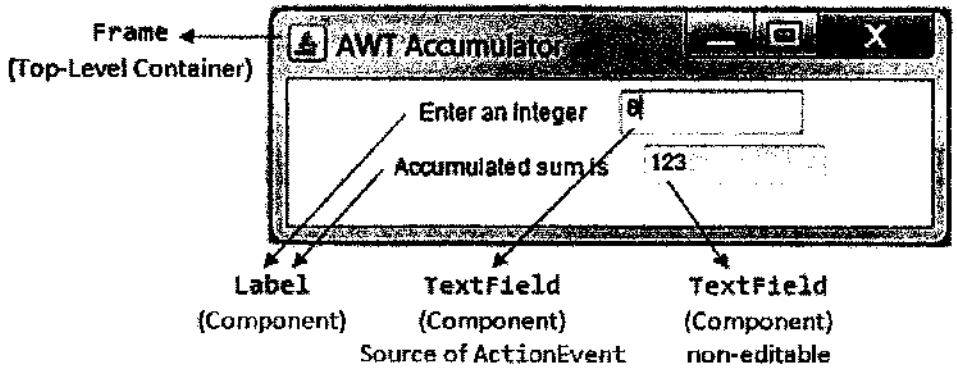
Standard-GUI-Bibliotheken für Java

- Abstract Window Toolkit (AWT)
- Swing
- Standard Widget Toolkit (SWT)
- JavaFX

Abstract Window Toolkit (AWT)

- Veröffentlicht 1995
- Standard-API zur Erzeugung und Darstellung *plattformabhängiger* GUIs
- *Heavyweight*-Framework, d.h. nutzt native GUI-Komponenten
- Hohe Performance
- Geringer Umfang an Funktionen und Widgets
- Teil der Java Standard Edition

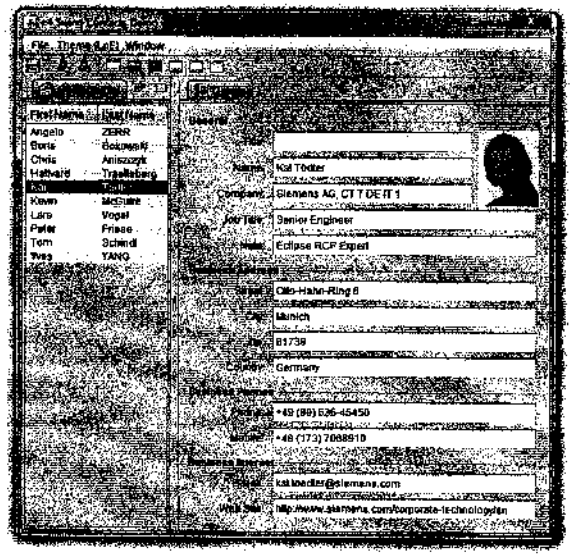
Abstract Window Toolkit (AWT) - Beispiel



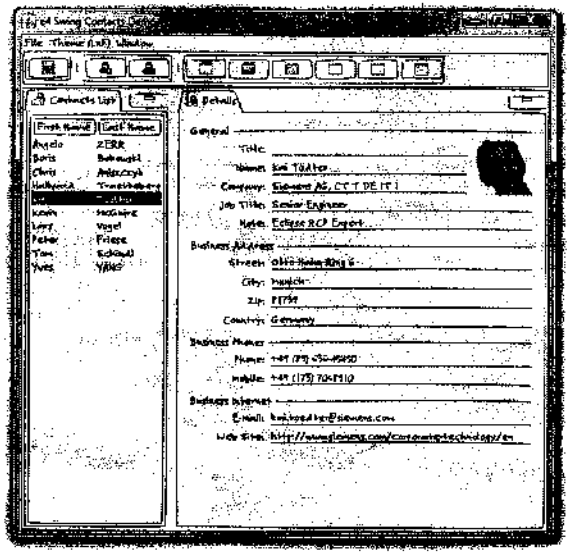
Swing

- Veröffentlicht 1997
- Basiert grundsätzlich auf AWT
- *Lightweight*-Framework, d.h. Komponenten werden direkt von Java gerendert und nicht vom Betriebssystem
- Oberfläche verhält sich auf allen Plattformen gleich
- Hoher Funktionsumfang
- Look-and-Feel nähert sich dem einer nativen Oberfläche an

Swing - Beispiel Nimbus Look & Feel



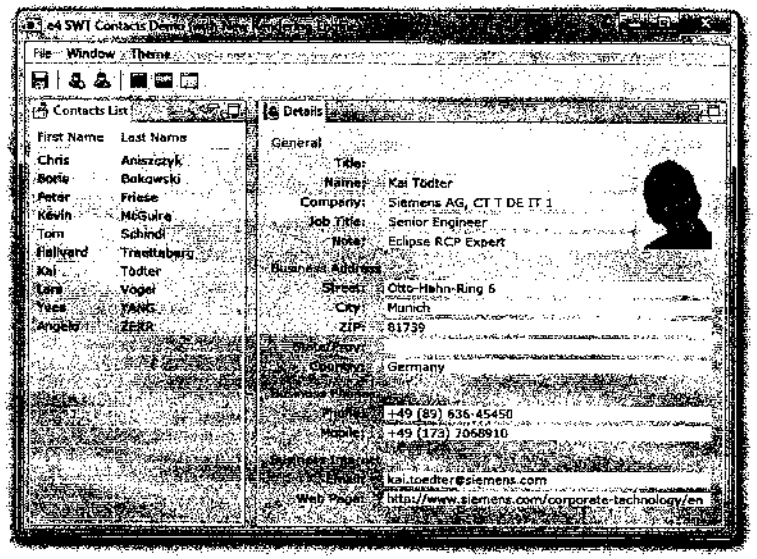
Swing - Beispiel Napkin Look & Feel



Standard Widget Toolkit (SWT)

- Veröffentlicht 2001 von IBM
- *Heavyweight*-Framework, d.h. nutzt native GUI-Komponenten
- Plattformabhängig, d.h. verschiedene Bibliotheken für Windows, Linux, OS X, usw.
- GUI-Toolkit von Eclipse
- Natives Look-and-Feel

Standard Widget Toolkit (SWT) - Beispiel

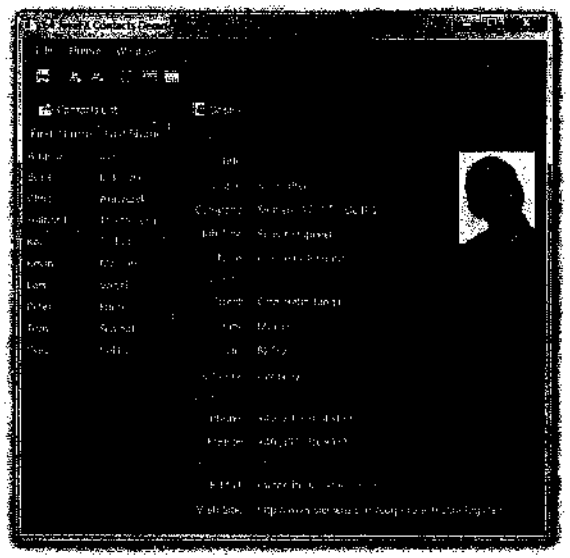


JavaFX

- Veröffentlicht 2008 von SUN
- *Lightweight*-Framework
- Plattformunabhängig
- Trennung von Oberflächendesign und Logik in (F)XML und Java-Code
- Einsatz von CSS zur Erstellung eigener Themes

FHDW Fachhochschule der Wirtschaft

JavaFX - Beispiel



FHDW Fachhochschule der Wirtschaft

Agenda

FHDW Fachhochschule der Wirtschaft

Aufbau Oberfläche

Die Oberfläche einer JavaFX-Anwendung setzt sich wie folgt zusammen

- *Stage*-Container enthält die komplette Anwendung
- *Scene*-Container wird innerhalb einer Stage erzeugt und stellt eine komplette Bildschirmmaske dar
- *Layout*-Container stellt das Basislayout innerhalb einer Scene bereit (z.B. Gridpane)
- *Controls* werden innerhalb eines Layout-Containers erstellt (Buttons, Textfelder etc.)

FHDW Fachhochschule der Wirtschaft

Aufbau Oberfläche

Stage

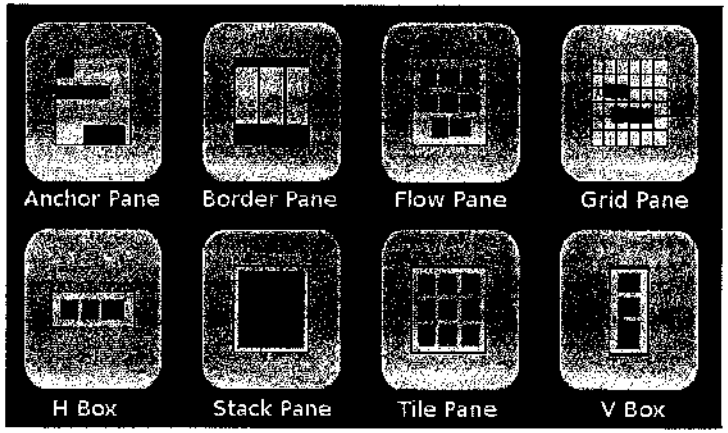
Scene

Layout-Container

Controls

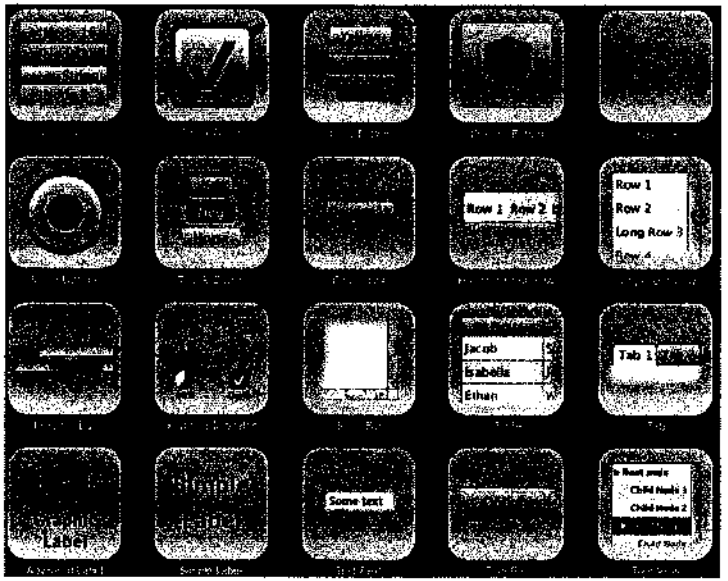
FHDW Fachhochschule der Wirtschaft

Layout-Container



FHDW Fachhochschule der Wirtschaft

Controls



Aufbau Projekt

Eine JavaFX-Anwendung besteht aus den folgenden Komponenten

- *Application*-Klasse zur Initialisierung der Anwendung
- *FXML*-Datei zur Beschreibung der GUI-Komponenten
- *Controller*-Klassen, um die GUI-Elemente mit Logik zu versehen
- *CSS*-Datei für die Erstellung eines Themes (optional)

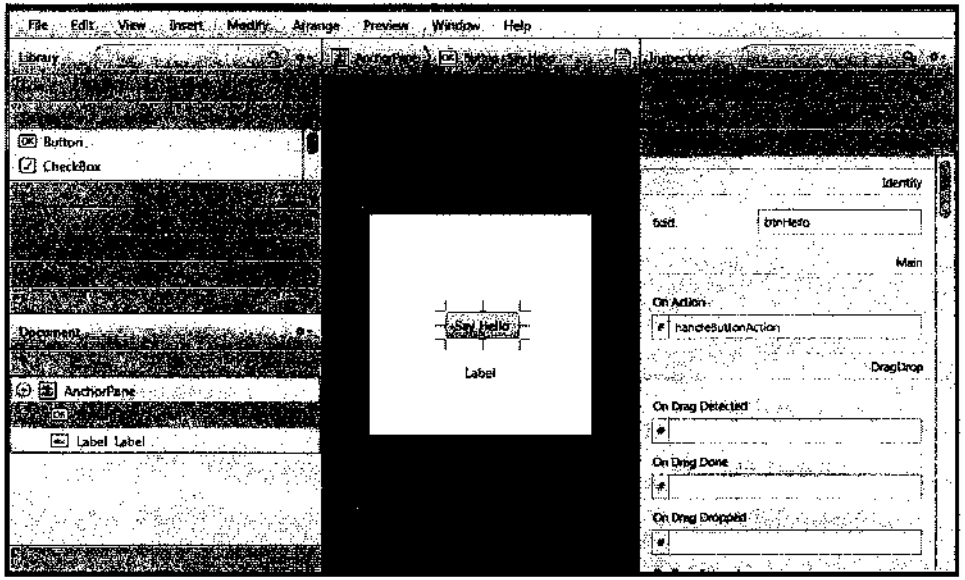
Application - Initialisierung der Anwendung

```
public class Main extends Application {
    public void start(Stage primaryStage) {
        try {
            AnchorPane root = (AnchorPane)FXMLLoader
                .load(getClass().getResource("Client.fxml"));
            Scene scene = new Scene(root);
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch(Exception e) { e.printStackTrace(); }
    }

    public static void main(String[] args) {launch(args);}
}
```


FXML - Oberflächendesign

```
<AnchorPane prefHeight="200.0" fx:controller="MyController">
    <children>

        <Button fx:id="btnHello" text="Say Hello"
          AnchorPane.topAnchor="88.0"
          onAction="#handleButtonAction" />

        <Label fx:id="lblResult" text="Label"
          AnchorPane.topAnchor="135.0" />

    </children>
</AnchorPane>
```

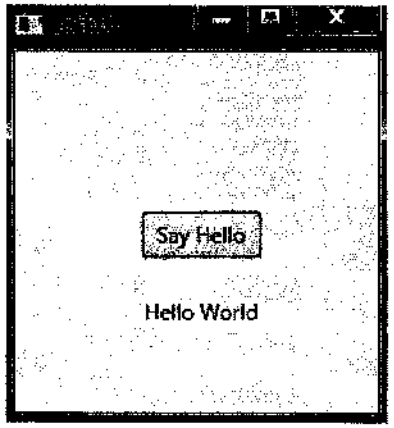

FXML - Scene Builder



Controller - Anwendungslogik

```
public class MyController {
    /* Controls werden durch die FXML-Annotation initialisiert,
    da die FXML-Datei diesen Controller referenziert */
    @FXML
    private Button btnHello;
    @FXML
    private Label lblResult;

    @FXML
    public void handleButtonAction(ActionEvent event) {
        if (btnHello.equals(event.getSource())) {
            lblResult.setText("Hello World");
        }
    }
}
```


Ergebnis



Agenda

Zugriff auf den REST-Service

1. Absetzen einer Anfrage (HTTP-Request) an den REST-Service
2. Die folgende Antwort (HTTP-Response) verarbeiten
   - 2.1 HTTP-Statuscode prüfen
   - 2.2 Auslesen des JSON-Strings
   - 2.3 Konvertierung des JSON-Strings ins Java-Datenmodell
3. Java-Objekt verwenden

Apache HttpClient

```
HttpClient httpClient = HttpClients.createDefault();
HttpGet get = new HttpGet("http://localhost:9998/rest/data");
HttpResponse response = httpClient.execute(get);

if (HttpStatus.SC_OK == response.getStatusLine().getStatusCode()) {
    String respString = EntityUtils.toString(response.getEntity());
    Gson gson = new GsonBuilder().create();
    RestData restData = gson.fromJson(respString, RestData.class);
    System.out.println(restData.getInfo());
}
```


Übungsaufgabe

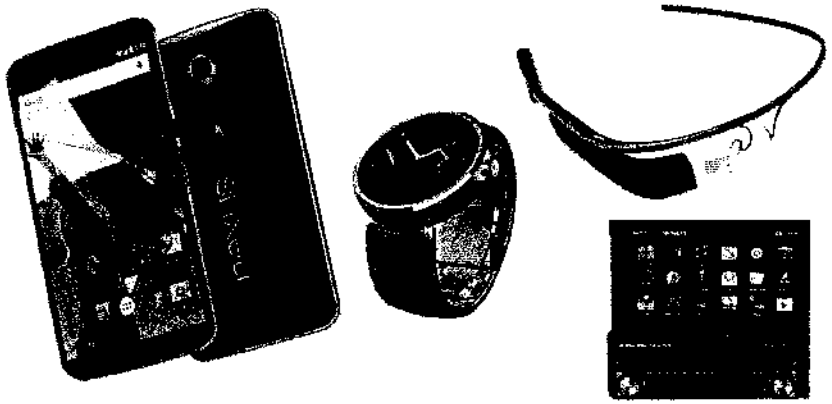
- Erweitern Sie den Beispiel REST-Service um eine GET-Methode, die die Addition zweier Zahlen als Ergebnis zurückgibt.
- Lassen Sie den Taschenrechner aus dem JavaFX-Beispiel den REST-Service für die Addition nutzen
- Jede Addition soll auf dem Server per log4j geloggt werden
- Geben Sie das Log auf der Konsole und in einer Logdatei aus
- Hinweis: Verwenden Sie für diese Übung die (von mir bereitgestellten) zusätzlichen Libraries (GSON, Apache HTTP)

Agenda

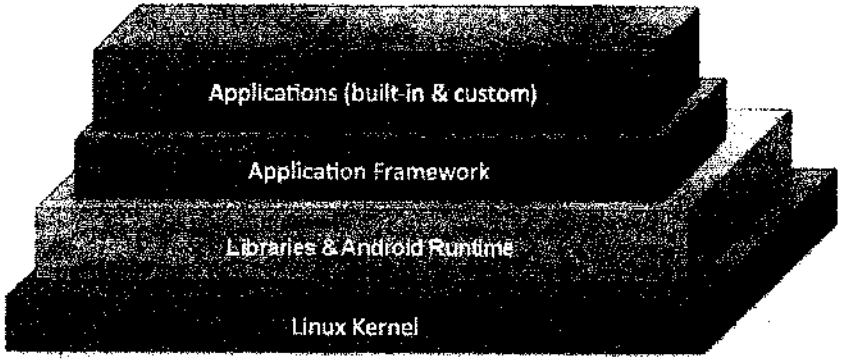
Android

- Betriebssystem von Google für mobile Geräte
- 2008 wurde die erste kommerzielle Version 1.0 veröffentlicht
- Darf von Geräteherstellern angepasst und kostenlos verwendet werden
- Besitzt den größten Marktanteil bei Mobilgeräten weltweit
- Verteilung und Monetarisierung von Apps über den Google PlayStore
- Entwicklung auf verschiedenen Plattformen möglich (Windows, Linux, MacOS)

Geräte



Architektur

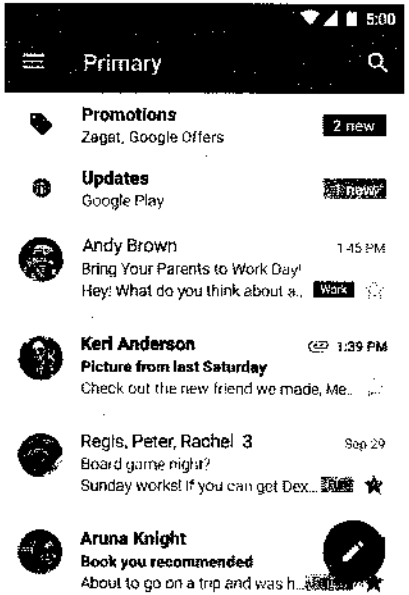


Agenda

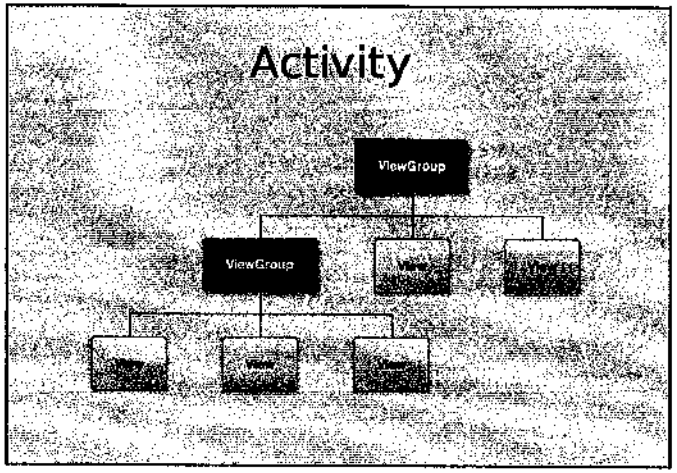
Activity

- Stellt Inhalte dar und nimmt Benutzereingaben entgegen
- Es wird immer nur jeweils eine Activity im Vordergrund angezeigt
- Mehrere können miteinander zu einer komplexeren Anwendung verknüpft werden
- Kann mit anderen Activities (auch fremder Apps) über sogenannte Intents kommunizieren
- Besitzt einen Lebenszyklus von der Erstellung bis zur Zerstörung
- Logik wird in Java geschrieben und Layout in XML

Activity Beispiel

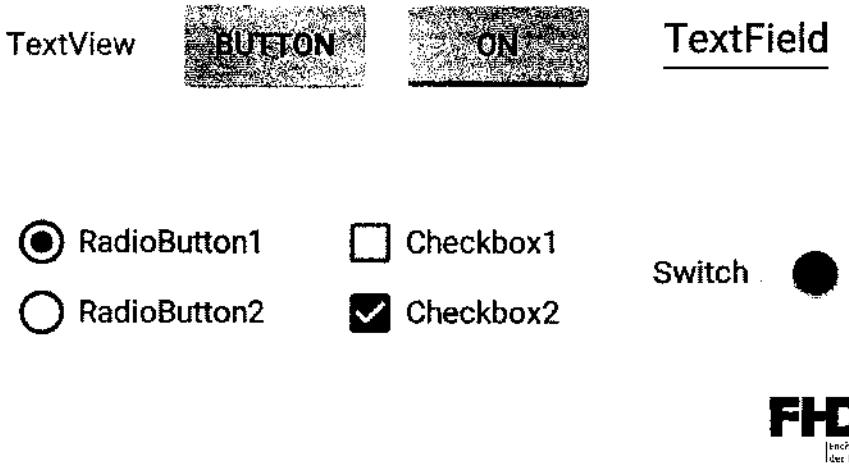


Activity Aufbau



ViewGroup

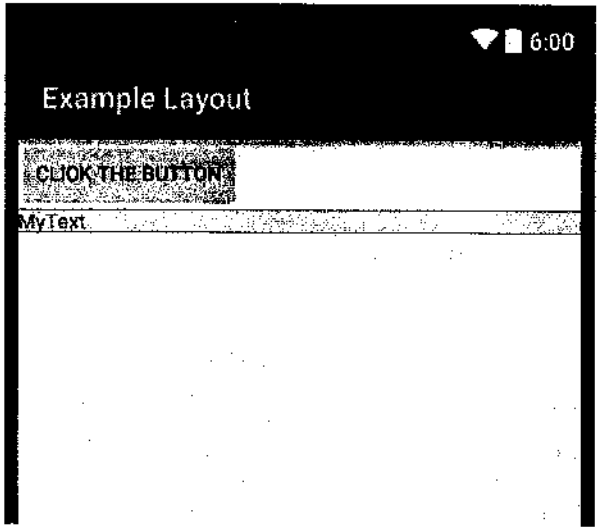
View



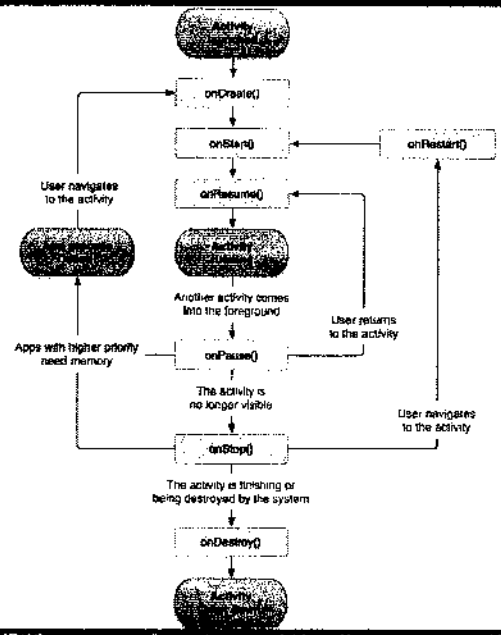
Activity Layout in XML

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:text="Click the Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:text="MyText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```


Activity Layout Ergebnis



Activity Lifecycle



Activity Lifecycle nutzen

```
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("startup", "Activity created");
    }
}
```

Berechtigungen

- Jede Android App läuft in einer eigenen Sandbox
- Diese Sandbox hat nur sehr limitierten Zugriff auf das System/Hardware
- Um weiteren Zugriff zu Erlangen, muss die App Berechtigungen verwenden
- Der Benutzer sieht bei der Installation einer App, welche Berechtigungen verlangt werden
- Seit Android 6.0 kann der Benutzer der App auch nur eine Teilmenge der Berechtigungen vergeben
- Beispiele für Berechtigungen: Kamera, Netzwerk, SMS, Speicher

Projektstruktur

- **manifests/AndroidManifest.xml:** Metadaten des Projekts (App-Name, Activities, Berechtigungen, etc.)
- **java/:** sämtlicher Java-Code (Activities, Datenmodell)
- **res/mipmaps:** App-Icon
- **res/drawable:** Bilder für die gesamte App
- **res/layout:** XML-Beschreibung der Activity-Layouts
- **res/values:** diverse weitere XML-Beschreibungen wie z.B. Strings und deren Übersetzung
- **Gradle Scripts:** Zum Bau der Anwendung benötigte Informationen (Abhängigkeiten etc.)

Android Debug Bridge (ADB)

- Command-Line Tool zur Kommunikation mit Android-Geräten und dem Android-Emulator
- Befindet sich im SDK-Ordner im Bereich *platform-tools*
- Auf Android-Geräten muss ADB in den Entwicklereinstellungen erst aktiviert werden
- Erlaubt z.B. Dateiübertragungen, App-Installation und Zugriff auf die Android-Shell
- Aufrufbeispiel: *adb install Name.apk* (Installiert die angegebene Android-APK auf dem Gerät / Emulator)
  (Die APK Ihres Projektes befindet unter *app/build/outputs/apk*)

Agenda

Zugriff auf den REST-Service

1. Absetzen einer Anfrage (HTTP-Request) an den REST-Service
2. Die folgende Antwort (HTTP-Response) verarbeiten
   2.1 HTTP-Statuscode prüfen
   2.2 Auslesen des JSON-Strings
3. Konvertierung des JSON-Strings ins Java-Datenmodell
4. Activity mit Java-Objekt aktualisieren

Netzwerkzugriff mit AsyncTask

```
new AsyncTask<String, Void, Pair<String, Integer>>() {
    protected HttpResponse doInBackground(String... params) {
      (1), (2)
    }
    protected void onPostExecute(Pair<String, Integer> result) {
      (3), (4)
    }
    //10.0.2.2 entspricht localhost auf dem Android Emulator
}.execute("http://10.0.2.2:9998/rest/data");
```


HTTP-Request im Hintergrund

```
protected Pair<String, Integer> doInBackground(String... params) {
    try {
        HttpClient httpClient = new DefaultHttpClient();
        HttpGet get = new HttpGet(params[0]);
        HttpResponse httpResponse = httpClient.execute(get);

        if(response == null) return null;
        int responseCode = response.getStatusLine().getStatusCode();
        if(responseCode == HttpStatus.SC_OK){
            String json = EntityUtils.toString(response.getEntity());
            return Pair.create(json, responseCode);
        }
        else {
            return Pair.create(null, responseCode);
        }
    } catch (IOException e) {
        return null;
    }
}
```


HTTP-Response verarbeiten

```
protected void onPostExecute(Pair<String, Integer> result) {
    if(result != null && result.first != null) {
        Gson gson = new GsonBuilder().create();
        RestData data = gson.fromJson(result.first, RestData.class);
        myTextView.setText(data.getInfo());
    }
    else {
        //Fehlermeldung anzeigen
    }
}
```

Übungsaufgabe

- Erweitern Sie die MainActivity um zwei Textfelder und einen Button
- Erstellen Sie zudem eine weitere Activity, auf der zwei Zahlen und deren Summe angezeigt wird
- Nutzen die von Ihnen hinzugefügte Additionsmethode aus dem REST-Service, um die Addition zweier Zahlen aus den zwei Textfeldern der MainActivity zu berechnen und schicken sie das Ergebnis und die verwendeten Zahlen an die neue Activity

Agenda

Hypertext Markup Language (HTML)

- Die Hypertext Markup Language ist eine textbasierte Auszeichnungssprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten
- HTML-Dokumente sind die Grundlage des World Wide Web und werden von einem Webbrowser dargestellt
- HTML wurde zusammen mit URL und HTTP ab 1989 von RoyFielding, Tim Berners-Lee et al. am CERN entwickelt

HTML-Syntax

- Dem Text wird durch Auszeichnungen (*markup*) von Textteilen eine Struktur verliehen
- Die Auszeichnung erfolgt üblicherweise über ein *Tag*-Paar, d.h. Starttag und Endtag, welche zusammen ein HTML-Element bilden
- Ein Starttag beginnt immer mit einem <, gefolgt vom Elementnamen
- Es folgt eine optionale Liste von Attributen
- Das Starttag wird durch ein > abgeschlossen
- Anschließend wird der Inhalt des HTML-Elements angegeben
- Mit dem (in Ausnahmefällen optionalen) Endtag wird das HTML-Element abgeschlossen. Es ist wie das Starttag aufgebaut, hat jedoch keine Attribute und wird durch einen / vor dem Elementnamen vom Startrag abgegrenzt
- Beispiel:

```
<p class="alert">Textabsatz und <em>Betonung</em></p>
```

HTML-Struktur

Ein HTML-Dokument besteht aus drei Bereichen:

- **Doctype** Gibt die verwendete Dokumenttypdefinition an
- **HEAD** Enthält überwiegend technische oder dokumentarische Informationen, die üblicherweise nicht vom Browser dargestellt werden
- **BODY** Enthält die Informationen, die gewöhnlicherweise im Anzeigebereich des Browsers zu sehen sind

HTML-Struktur - Beispiel

```
<!DOCTYPE html>
<html>
    <head>
        <title>Titel der Webseite</title>
        <!-- weitere Kopfinformationen -->
    </head>
    <body>
        <p>Inhalt der Webseite</p>
    </body>
</html>
```


Einige HTML-Elemente

Überschriften

```
<h1>Text</h1>
```

Links

```
<a href="URL">Verweistext</a>
```

Bilder

```
<img src="URL" alt="Text" width="Breite" height="Hoehe">
```

Listen

```
<ul>
    <li>Listeneintrag</li>
    <li>Listeneintrag</li>
</ul>
```

Eingabefelder

```
<input type="text" placeholder="Vorname">
```


HTML-Elemente Resultat

Text

Verweistext

- Listeneintrag
- Listeneintrag

Vorname

Cascading Style Sheets (CSS)

- Cascading Style Sheets sind eine deklarative Sprache für Stilvorlagen von strukturierten Dokumenten
- Basieren auf der Grundidee, inhaltliche Repräsentation (HTML, XML) und konkrete Darstellung (Farben, Layout, usw.) zu trennen
- Erlaubt die Festlegung von Darstellungsattributen an zentraler Stelle
- CSS ermöglicht es, für verschiedene Ausgabemedien (Bildschirm, Papier, Sprache, ...) unterschiedliche Darstellungen zu definieren
- CSS gilt heute als die Standard-Stylesheetsprache für Webseiten

CSS-Syntax

```
Selektor [, Selektor2, ...]
  {
    Eigenschaft 1: Wert 1 [Wert2 Wert3 ...];
    Eigenschaft 2: Wert 1 [Wert2 Wert3 ...];
    Eigenschaft 3: Wert 1 [Wert2 Wert3 ...];
    ...
  }
```


CSS-Beispiel

HTML

```
<link rel="stylesheet" href="style.css">
<p>Einfacher Text</p>
<p class="alert">Warnung</p>
```

CSS

```
.alert {
    background-color: red;
    border: 3px solid black;
    color: white
}
```

Ausgabe:

Einfacher Text

Warnung

CSS-Framework Bootstrap

- Von Twitter entwickeltes OpenSource HTML-, CSS- und Javascript-Framework
- Enthält vorgefertigte CSS-Styles für viele HTML-Elemente mit einheitlichem Design
- Unterstützt Reponsive-Design (dynamische Anpassung an Display-Größe)
- Gestaltung des Layouts über ein Grid-System

Bootstrap Beispiel

```
<link rel="stylesheet" href="bootstrap.css">
<ul class="list-group">
    <li class="list-group-item">Listeneintrag</li>
    <li class="list-group-item">Listeneintrag</li>
</ul>
<input class="form-control" type="text" placeholder="Vorname">
<button class="btn btn-primary">Button</button>
```


Bootstrap Resultat

Ohne Bootstrap

- Listeneintrag
- Listeneintrag

Vorname Button

Mit Bootstrap

Listeneintrag

Listeneintrag

Vorname

Button

Strukturierung mit dem HTML-Element DIV

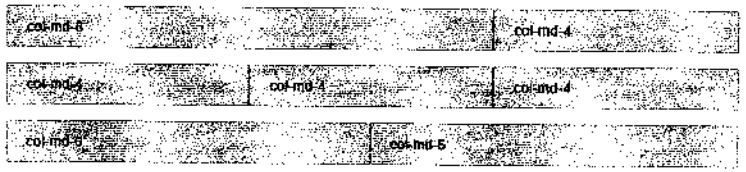
DIV-Element

Mit dem DIV-Element lassen sich verschiedene HTML-Elemente zu einem Bereich gruppieren. Alle HTML-Elemente in diesem Bereich können anschließend gemeinsam mit Hilfe von CSS formatiert werden.

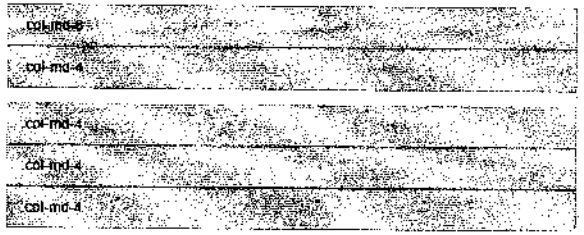
```
<div>
  <Element><Element/>
  <Element><Element/>
  ...
</div>
<div>
  <Element><Element/>
</div>
```


Bootstrap Grid I

Desktop



Mobile



Bootstrap Grid II

```
<div class="container- fluid">
    <div class="row">
        <div class="col-md-8">.col-md-8</div>
        <div class="col-md-4">.col-md-4</div>
    </div>
    <div class="row">
        <div class="col-md-4">.col-md-4</div>
        <div class="col-md-4">.col-md-4</div>
        <div class="col-md-4">.col-md-4</div>
    </div>
    <div class="row">
        <div class="col-md-6">.col-md-6</div>
        <div class="col-md-6">.col-md-6</div>
    </div>
</div>
```


Agenda

Überblick

- Von Google entwickeltes OpenSource-Framework zur Erstellung von Webanwendungen
- Nachfolger von AngularJS
- Hält Daten und HTML-Ansicht konsistent
- Erweitert dazu das HTML-Vokabular um eigene Elemente
- Unterstützt **TypeScript**, JavaScript und Dart
- Erste finale Version am 15.09.2016 veröffentlicht

Typescript

```
import {Component} from '@angular/core';

class Hello {
    helloName: string;
    constructor(name: string) {
        this.helloName = name;
    }

    public sayHello(): string {
        let greeting: string = "Hello " + this.helloName;
        return greeting;
    }
}

class Main {
    public helloWorld() {
        let helloInstance: Hello = new Hello("Hugo");
        console.log(helloInstance.sayHello());
    }
}
```

Bestandteile

- Component
- Service
- Directive
- Pipe

Component

- Kapselt eine komplette, wiederverwendbare Ansicht
- Erzeugt ein eigenes HTML-Element (*selector*)
- Unterstützt Eingangs- und Ausgangsparameter
- Jede Angular2-Anwendung benötigt mindestens eine Root-Component
- Durch Komposition vieler Components entstehen komplexe Anwendungen
- Wird mittels @Component Decorator an einer Klasse definiert
- Bestimmt ihr Aussehen über ein HTML-Template

Component Beispiel

```
import {Component, Input, Output, EventEmitter} from '@angular/core';

@Component({
    selector : 'my-component',
    template: '<h2>Hello World</h2>'
})
export class MyComponent {
    @Input() myInput: string;
    @Output() myOutput = new EventEmitter<string>();

    greet (): string {
      let greeting: string = "Hello " + myInput;
        this.myOutput.emit(greeting);
        return greeting;
    }
}
```


Component Verwendung I

```
<my-component
    [myInput]="Hugo" (myOutput)="console.log($event);" #myVar>
</my-component>
<p>{{myVar.greet()}}</p>
```

- <**my-component**> bezieht sich auf den im *selector* (siehe Component Decorator) angegebenen Component-Namen
- **(Event-Name)** gibt an, was bei Auftreten des Events passieren soll
- **$event** enthält einen optional vorhandenen Rückgabewert des Events

Component Verwendung II

```
<my-component
    [myInput]="Hugo" (myOutput)="console.log($event);" #myVar>
</my-component>
<p>{{myVar.myInput}}</p>
```

- **[Attribut-Name]** setzt ein Attribut eines HTML-Elements (hier der Input-Parameter) auf den gegebenen Wert
- **#VariablenName** erzeugt eine lokal verfügbare Variable des HTML-Elements
- {{**Ausdruck**}} Angular2 Interpolation: Gibt das Ergebnis eines Ausdrucks als Text aus

Component: Externe HTML-Templates

- Die Erstellung von HTML-Templates innerhalb von Typescript ist nicht komfortabel
- So fehlt etwa das Highlighting und Code-Completion
- Daher lassen sich komplexere HTML-Templates in externe Dateien auslagern
- Template-Datei und Typescript-Datei müssen in @Component miteinander verknüpft werden

Externes HTML-Template Beispiel

Inhalt app.template.html

```
<h1>{{myTitle}}</h1>
<ul>
    <li>1</li>
    <li>2</li>
</ul>
```

Einbindung des Templates

```
@Component({
    selector : 'my-component',
    templateUrl: 'app/app.template.html'
})
export class AppComponent {
    myTitle: string = "My App Title"
}
```


Service

- Stellt gemeinsam genutzte Methoden und Daten zur Verfügung
- Eine Instanz kann von mehreren Components genutzt werden
- Wird Häufig für Kapselung von Zugriff auf externe Daten verwendet
- Definition durch @Injectable Decorator an einer Klasse
- Eine Referenz für den Zugriff auf den Service wird im Konstruktor einer Klasse erstellt

Service Beispiel

```
import { Injectable } from '@angular/core';

@Injectable()
export class HelloService {
    sayHello(name: string){
        let helloString : string = "Hello " + name;
        console.log( helloString );
        return helloString ;
    }
}
```


Service Verwendung

```
import {HelloService} from './hello-service.component';

@Component({
    providers : [ HelloService ]
})
export class MyComponent {
    constructor( private helloService : HelloService ) { }

    greet(){
        this . helloService . sayHello("Hugo");
    }
}
```


Directive

- Manipuliert bestehende HTML-Elemente
- Besitzt daher kein eigenes HTML-Template
- Wird über @Directive an einer Klasse definiert
- Es gibt diverse vorgefertigte Directives (z.B. *ngFor* zur Wiederholung von HTML-Elementen)

Directive Verwendung

```
<ul>
    <li *ngFor="let item of itemList">{{item.name}}</li>
</ul>
```

Erzeugt zur Laufzeit folgendes HTML:

```
<ul>
    <li>Hugo</li>
    <li>Hans</li>
    <li>Horst</li>
    ...
</ul>
```

Pipe

- Transformiert und filtert Daten
- Kann innerhalb von HTML-Templates (Interpolation) verwendet werden
- Erstellung durch Dekoration einer Klasse mit @Pipe
- Verwendung mittels Pipe-Operator |
- Angular2 bietet verschiedene direkt verfügbare Pipes (z.B. *DatePipe* zur Formatierung des Datums)

Pipe Verwendung

```
@Component({
  template:
      <p>My birthday is {{ birthday}}</p>
      <p>My birthday is {{ birthday | date }}</p>
      <p>My birthday is {{ birthday | date | uppercase }}</p>
  '
})
export class Birthday {
    birthday: Date = new Date(1911,11,11);
}
```

Ausgabe im Browser:

My birthday is Mon Dec 11 1911 00:00:00 GMT+0100 *//ohne Pipe*

My birthday is Dec 11, 1911 *//mit DatePipe*

My birthday is DEC 11, 1911 *//mit DatePipe und UppercasePipe*

Databinding

- Verknüpft Daten aus den Typescript-Dateien mit der HTML-Darstellung
- Hält somit HTML-Repräsentation stets aktuell
- Funktioniert bidirektional
- Die Verbindung wird durch [(ngModel)] erzeugt

Databinding Verwendung

```
@Component({
    template:
      <h2>{{myText}}</h2>
      <input [(ngModel)]="myText">
      '
})
export class Birthday { myText: string = "Init"; }
```

Initiale Darstellung:

Init

Init

Darstellung nach Eingabe:

Hugo

Hugo

Agenda

Zugriff auf den REST-Service

1. Per Angular2-Service eine Anfrage (HTTP-Request) an den REST-Service stellen
2. Die Antwort (HTTP-Response) vom Service in ein JSON-Objekt konvertieren lassen
3. Das JSON-Objekt verarbeiten, sobald es vom Angular2-Service zur Verfügung gestellt wird (subscribe)

REST: Angular2-Service

```
import { Injectable } from '@angular/core';
import {Http, Headers, Response} from '@angular/http';
import 'rxjs/add/operator/map';

@Injectable()
export class MyService {
    constructor(private http: Http) { }

    getData() {
        return this.http.get('http://localhost:9998/rest/data')
            .map((response: Response) => response.json()
            );
    }
}
```


REST: Angular2-Component

```
import {Component} from '@angular/core';
import {Response} from '@angular/http';
import {MyService} from "./my-service.component";

@Component({
    providers: [MyService]
})

export class MyComponent {
    constructor(private myService: MyService) { }

    getData() {
        this.myService.getData().subscribe(
            (data: RestData) => console.log(data.info),
            (error: Response) => console.log(error.status),
            () => console.log("Request completed!")
        );
    }
}
```

Übungsaufgabe

- Erstellen Sie eine komplett neue Webanwendung
- Der Benutzer soll zwei Zahlen addieren und das Ergebnis nach Klick auf einen Button sehen können
- Führen Sie die Addition zunächst lokal durch
- Fügen Sie anschließend einen weiteren Button hinzu, der die Berechnung mit Hilfe der von Ihnen hinzugefügten Additionsmethode aus dem REST-Service durchführt

FHDW
Fachhochschule der Wirtschaft