

# ZUSAMMENFASSUNG

## Lernziele

Entwurf und Realisierung von User-Interfaces für Mobil- und Desktop Anwendungen

## Bücher / Unterlagen

- Reine Vorlesungsfolien
- Weiterführende Links aus dem Internet

## Lerninhalte

- **Erster Teil: Android**
  - o Aufbau und Funktionsweise von Android-Apps erklären
  - o Konzepte anhand von Problemstellungen umsetzen
  - o Entwickeln einer eigenen Mobil-App im Miniprojekt
- **Zweiter Teil: WPF (ab Seite 61 in dieser Zusammenfassung)**
  - o Grundlagen der Desktop-App-Entwicklung mit WPF und C# erklären
  - o Desktop UI-Konzepte anhand von problemspezifischen Situationen anwenden
  - o Unterschiede zur Mobil-App-Entwicklung erläutern
  - o Entwickeln einer eigenen Desktop-App im Miniprojekt

## Erlaube Hilfsmittel an der Prüfung

- Pro Teil 10 Seiten Zusammenfassung (5 A4 Seiten)

<b>KONZEPTE DER ANDROID-ENTWICKLUNG.....</b>	<b>11</b>
ACTIVITIES UND LEBENSZYKLUS .....	12
<i>Unterschied zu anderen Modellen.....</i>	12
Android Activities.....	12
Activity aus Entwicklersicht.....	13
Activity Lebenszyklus .....	13
ACTIVITY STACK UND TASKS .....	14
Activities vs. Tasks.....	14
Activity Launch Modes.....	14
Activities und Tasks aus Systemsicht .....	15
APK - Android Application Package.....	15
INTENTS.....	15
Intents Erzeugen .....	15
Intents Starten Activities.....	16
Intents Parameterübergabe.....	16
Weitere Komponenten: Services, Broadcasts .....	16
VIEWS.....	16
Activity-View-Intent Zusammenfassung.....	17
<b>IDE, SDK, API UND TOOLS.....</b>	<b>17</b>
ANDROID SDK MANAGER.....	17
API LEVEL.....	17
WAHL DES API LEVELS.....	17
MANIFEST .....	17
SDK-Versionen im Manifest.....	18
APPLICATION .....	18
<b>GRUNDLAGEN DER GUI-PROGRAMMIERUNG .....</b>	<b>18</b>
GUI ERSTELLEN .....	18
Deklarativ ( <i>eine Beschreibung in XML</i> ) .....	19
Imperativ.....	19
<b>VIEWS .....</b>	<b>19</b>
VIEWGROUP .....	19
EIGENES VIEWS .....	19
<b>WIDGETS .....</b>	<b>19</b>
EINGABE-WIDGETS.....	19
BUTTONS.....	19
EINGABEFELDER .....	19
<b>LAYOUTS.....</b>	<b>20</b>
LAYOUT PARAMETER.....	20
LINEAR LAYOUT.....	20
RELATIVE LAYOUT.....	20
WEITERE LAYOUTS.....	20
AYOUTS UND ACTIVITIES .....	21
<b>@REFERENZEN UND @+IDS .....</b>	<b>21</b>

DIE KLASSE R .....	21
VALUES RESSOURCEN .....	21
DIMENSIONEN .....	21
VARIANTEN VON RESSOURCEN .....	21
<b>EVENTS.....</b>	<b>22</b>
LISTENERS.....	22
GUI-OBJEKT FINDEN.....	22
LISTENER REGISTRIEREN .....	22
EVENTS ZUR TEXTEINGABE.....	22
INPUTVALIDIERUNG .....	23
<b>STRUKTURIERUNG UND NAVIGATION .....</b>	<b>23</b>
VORGEHEN BEIM ENTWURF .....	23
VORGEHEN SCHRITT-FÜR-SCHRITT .....	24
UP- UND HOME-NAVIGATION .....	24
FRAGMENTS .....	24
<i>Fragments definieren .....</i>	25
<i>Fragment statisch einfügen.....</i>	25
<i>Fragment dynamisch einfügen .....</i>	25
ACTIVITY FRAGMENT KOMMUNIKATION.....	25
MASTER-DETAIL NAVIGATION.....	26
WANN FRAGMENTS VERWENDEN? .....	26
MENUS.....	26
<i>Options Menu .....</i>	26
<i>Fragment Menu Contributions.....</i>	27
<i>Context und Popup Menus.....</i>	27
<i>Action Bar.....</i>	27
<i>Toolbar – Die neue Action Bar .....</i>	27
<i>Navigation Drawer .....</i>	28
TOASTS.....	28
SNACKBAR .....	28
<b>LISTEN .....</b>	<b>29</b>
EINFACHE LISTVIEW.....	29
ADAPTER DESIGN PATTERN .....	29
ADAPTERVIEW .....	29
EIGENER ARRAYADAPTER .....	29
BEISPIEL – LISTVIEW MIT CHECKBOXEN .....	30
LISTVIEW PERFORMANCE .....	30
<i>View Tages.....</i>	30
<i>Optimierte getView.....</i>	30
RECYCLER VIEW .....	30
<i>Komponenten .....</i>	31
<i>Schnittstelle.....</i>	31
<i>Setup .....</i>	31
LEERE LISTEN .....	31
<b>PERSISTENZ.....</b>	<b>31</b>
DATEN PERSISTIEREN.....	31

VIEW-DATEN .....	32
APP-DATEN .....	32
SHARED-PREFERENCES.....	32
FILE STORAGE .....	32
SQLITE STORAGE .....	32
3RD PARTY ANGEBOTE .....	33
<b>HINTERGRUNDTASKS.....</b>	<b>33</b>
PROBLEMATIK .....	33
HINTERGRUNDAKTIONEN .....	33
RÜCKBLICK – EVENT LOOP .....	33
HINTERGRUNDAKTIONEN MIT THREADS.....	34
ASYNC TASK .....	34
<i>AsyncTask Fortschrittsanzeige</i> .....	34
CALLBACKS IN LIBRARYSERVICE .....	35
<b>MATERIAL DESIGN.....</b>	<b>35</b>
DESIGN LANGUAGE .....	35
HUMAN INTERFACE GUIDELINES .....	35
MATERIAL DESIGN .....	35
<i>Prinzipien</i> .....	35
<i>Materialien und Schatten</i> .....	36
<i>Style Guide</i> .....	36
<i>Layout</i> .....	36
<i>Style</i> .....	36
<i>Animation</i> .....	36
<i>Components</i> .....	37
<i>Patterns</i> .....	37
<i>Usability</i> .....	37
<i>Zusammenfassung</i> .....	37
MATERIAL DESIGN UMSETZUNG .....	38
<i>Styling</i> .....	38
<i>Themes</i> .....	38
<i>Vordefinierte Styles und Themes</i> .....	39
<i>Theme Editor in Android Studio</i> .....	39
<i>Material Theme</i> .....	39
<i>Theme Overlays</i> .....	40
<i>Material Design Components</i> .....	40
<b>DESIGN UND ARCHITEKTUR PATTERNS.....</b>	<b>43</b>
SOFTWARE ARCHITECTURE .....	43
MULTITIER ARCHITECTURE .....	43
MGE & MULTITIER ARCHITECTURE.....	43
KEINE ZYKLEN? .....	44
OBSERVER PATTERN .....	44
<i>Beispiel</i> .....	44
<i>Observer Pattern in Java</i> .....	45
ALTERNATIVE – SWIPE-REFRESH.....	45
MODEL-VIEW-CONTROLLER.....	46
MVC IN ANDROID .....	46

**WEITERE ANDROID-KOMPONENTEN .....**

DIE CONTEXT-KLASSE .....	47
SERVICES .....	47
<i>Deklaration</i> .....	47
<i>Starten oder Binden</i> .....	47
<i>Started Services</i> .....	48
<i>Bound Services</i> .....	49
<i>Services Lifecycle</i> .....	50
BROADCAST RECEIVER .....	50
<i>Meldungen</i> .....	51
<i>Registrieren</i> .....	51
<i>Implementation</i> .....	51
<i>Versenden</i> .....	51
<i>Lokal versenden</i> .....	52
CONTENT PROVIDER .....	52
<i>Client</i> .....	52
<b>CONTENT PROVIDER</b> .....	<b>53</b>
SERVER .....	53
SERVER IMPLEMENTATION .....	53
<b>SENSOREN</b> .....	<b>54</b>
VERFÜGBARKEIT VON SENSOREN .....	54
ANDROID SENSOR FRAMEWORK .....	54
MINIMALES SENSOREN-BEISPIEL .....	54
<b>DEPENDENCY INJECTION</b> .....	<b>55</b>
DEPENDENCIES VON KLASSEN .....	55
DEPENDENCIES VON AUSSERHALB .....	55
DEPENDENCIES ÜBERGEBEN .....	55
DEPENDENCY INJECTION MIT DAGGER 2 .....	56
LIBRARY SERVICE INJECTION .....	56
<i>Modul und Komponente</i> .....	56
<i>Weitere Implementation</i> .....	56
LOHNT SICH DEPENDENCY INJECTION? .....	57
VIEW INJECTION? .....	57
BUTTER KNIFE! .....	57
<i>Binding</i> .....	57
<b>DATA BINDING</b> .....	<b>58</b>
DATA BINDING GRUNDLAGEN .....	58
BINDING VON LISTENERN .....	58
OBSERVABLES .....	59
NACHTEILE VON DATA BINDING .....	59
<b>JAVA 8</b> .....	<b>59</b>
JAVA ANDROID COMPILER KIT (JACK) .....	59
UNTERSTÜTZTE FEATURES .....	59
<b>EINFÜHRUNG WPF &amp; XAML</b> .....	<b>61</b>
DESKTOP APPS – TECHNOLOGIE VON GESTERN? .....	61

<i>Typische Applikations-Architektur: N-Tier.....</i>	61
WPF QUICK INTRO.....	61
<i>Grundprinzip .....</i>	61
<i>XAML.....</i>	61
<i>XAML → CLR.....</i>	62
<i>XAML vs. C#.....</i>	62
<i>Device Independet Pixels/Units.....</i>	62
<i>„Logical Tree“ und „Visual Tree“.....</i>	63
<i>„Route Events“.....</i>	63
<i>Der „UI Thread“.....</i>	64
<i>XAML.....</i>	64
<i>„Attribute Syntax“ vs. „Property Element Syntax“ .....</i>	64
<i>Weitere XAML Konzepte.....</i>	64
<i>Beyond XAML .....</i>	65
<i>C# vs. JAVA .....</i>	65
<i>C# Keywords .....</i>	66
<i>C# Data Types.....</i>	67
<i>C# Features.....</i>	67
<i>C# SPEZIALITÄTEN.....</i>	68
<i>„Properties“.....</i>	68
<i>„Fields“ .....</i>	68
<i>„Delegates“.....</i>	68
<i>Vordefinierte Delegate-Typen.....</i>	69
<i>Lambda Expressions“.....</i>	69
<i>„Events“.....</i>	70
<i>Formartierte Strings .....</i>	70
<i>EventHandler&lt;T&gt;.....</i>	70
<i>„Extension Methods“.....</i>	71
<i>„Linq“ – Language Integrated Query.....</i>	71
<i>VISUAL STUDIO .....</i>	72
<i>NEUES WPF – PROJEKT .....</i>	72
<i>Projekt-Layout.....</i>	72
<i>App.config – Beispiel.....</i>	73
<i>App.xaml – Beispiel.....</i>	73
<i>App.xaml.cs – Beispiel .....</i>	73
<i>MainWindow.xaml – Beispiel .....</i>	73
<i>MainWindow.xaml.cs – Beispiel.....</i>	74
<b>GUI-ENTWURF MIT WPF.....</b>	<b>75</b>
<i>INTRO.....</i>	75
<i>App &amp; Window .....</i>	75
<i>WPF Controls Overview.....</i>	75
<i>CONTAINERS CONTROLS .....</i>	77
<i>Container Controls mit Layout.....</i>	77
<i>Container Controls ohne Layout.....</i>	79
<i>EVENTS UND COMMANDS .....</i>	81
<i>Event Handling.....</i>	81
<i>Commands (Alternative zu Events) .....</i>	81
<i>Events vs. Commands.....</i>	82
<i>NUGET PACKAGE MANAGER.....</i>	82

AUTOMATED UI TESTING .....	82
<i>Testing</i> .....	82
Übersicht von <i>Testing</i> .....	82
<i>TestStack.White</i> .....	83
<b>GUI-DESIGN MIT WPF.....</b>	<b>85</b>
RESOURCES.....	85
<i>Vorteile von zentralen Ressourcen</i> .....	85
<i>Physischen Ressourcen in WPF</i> .....	85
„Resource“.....	86
„ResourceDictionary“.....	86
<i>Zugriff auf Resource</i> .....	87
<i>Zugriff auf System-Ressourcen</i> .....	87
<i>Zugriff auf Ressourcen mit C#</i> .....	88
<i>StaticResource vs. DynamicResource</i> .....	88
<i>Markup Extension</i> .....	88
<i>Externe Ressourcen</i> .....	88
STYLES.....	89
<i>Situation</i> .....	89
<i>Explizite Styles</i> .....	90
<i>Styles definieren</i> .....	90
<i>Typspezifische Styles</i> .....	90
<i>Styles kombinieren</i> .....	90
CONTROL TEMPLATES.....	91
TRIGGER .....	91
VISUAL STATE MANAGER.....	91
TRANSFORMATIONEN .....	92
<i>In den Styles</i> .....	92
SKINS UND THEMES.....	93
RESOURCES & STYLES.....	93
<b>GUI DESIGN PRINCIPLES (DESKTOP APPS).....</b>	<b>93</b>
WINDOWS 7 .....	93
WINDOWS 8 / 10.....	93
AKTUELLE DESIGN TRENDS .....	94
<i>Flat Design</i> .....	94
<i>Minimalism</i> .....	94
<i>Typography</i> .....	94
<i>Icons beat Text =&gt; a picture is worth a thousand words.</i> .....	94
<i>Personalization</i> .....	95
<i>Dynamic Layout (= Responsive Design in Web/App Development)</i> .....	95
DYNAMIC LAYOUT IN WPF.....	95
<i>Strategie 1 – Ändern der Position</i> .....	95
<i>Strategie 2 – Ändern der Grösse</i> .....	96
<i>Strategie 3 – Neuanordnen</i> .....	96
<i>Strategie 4 – Einblenden</i> .....	96
<i>Strategie 5 – Ersetzen (Wechsel der UI-Elemente)</i> .....	96
<i>Strategie 6 – Ändern der Architektur</i> .....	97
<i>Umsetzbarkeit in WPF</i> .....	97
<b>UMGANG MIT DATEN.....</b>	<b>98</b>

MVVM (PREVIEW).....	98
MARKUP EXTENSIONS.....	98
ÜBERBLICK DATA BINDING.....	98
<i>Nützliche Properties (Appendix mit Detailbeschrieb vorhanden)</i> .....	98
BINDING.....	98
<i>Source vs. RelativeSource vs. ElementName</i> .....	98
<i>DataContext</i> .....	99
<i>StringFormat</i> .....	101
<i>Path</i> .....	101
<i>Mode</i> .....	101
IVALUECONVERTER.....	102
<i>Beispiel BooleanToVisibilityConverter</i> .....	102
IMULTIVALECONVERTER.....	102
<i>Beispiel RgbToColorConverter</i> .....	103
NUTZUNGSHINWEISE IVALUECONVERTER & IMULTIVALECONVERTER.....	103
<i>Value Converter anwenden</i> .....	103
<i>Eigene Value Converters Implementieren?</i> .....	104
<b>BINDING AUF EIGENE OBJEKTE .....</b>	<b>104</b>
INOTIFYPROPERTYCHANGED.....	105
<i>Implementieren V1</i> .....	105
<i>Implementieren V2</i> .....	105
<i>Implementieren V3</i> .....	105
<i>Berechnete Properties</i> .....	106
<i>Beispiel</i> .....	106
<i>Automatisch implementieren?</i> .....	106
INOTIFYPROPERTYCHANGED V4 (PROPERTYCHANGED.FODY) .....	106
OBSERVABLECOLLECTION<T> .....	107
OBJECTDATAPROVIDER.....	107
DEBUGGING DATA BINDING.....	107
<b>NÜTZLICHE UI ELEMENTE .....</b>	<b>108</b>
ITEMSCONTROL.....	108
<i>Variante 1</i> .....	108
<i>Variante 2 (Besser)</i> .....	108
DATAGRID .....	109
DATENQUELLEN .....	110
SONSTIGES .....	110
<b>PROPERTY KONZEPT C#.....</b>	<b>111</b>
OOP: PROPERTIES .....	111
JAVA PROPERTIES.....	111
C# PROPERTIES.....	111
<b>REVIEW C# EVENTS.....</b>	<b>112</b>
EVENTARGS .....	112
PUBLISH .....	112
SUBSCRIBE.....	112
<i>Beispiel</i> .....	112
<i>Konventionen</i> .....	113
SUBSCRIBE VIA LAMBDA-SYNTAX (KÜRZER).....	113

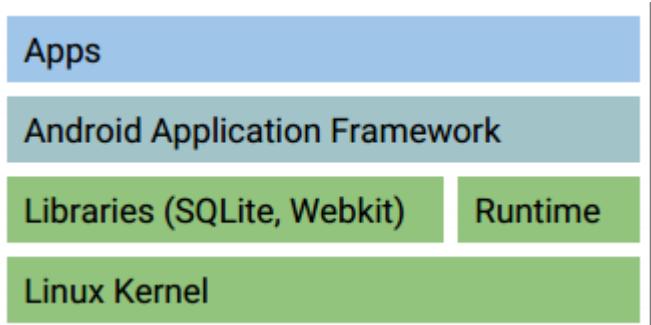
VOLLSTÄNDIGES BEISPIEL «CLOCK» (BASIEREND AUF LÖSUNGSVORSCHLAG Ü4) .....	113
WPF .....	113
C# EVENTS IN WPF.....	114
C# EVENTS VS. VIRTUAL EVENT HANDLERS.....	114
<b>DER WPF APP LIFECYCLE.....</b>	<b>115</b>
APP STARTUP .....	115
<i>Zur Erinnerung Java.....</i>	115
<i>C#(.NET) generell .....</i>	115
<i>WPF App.....</i>	115
APP STARTUP BEISPIELE .....	116
<i>Variante «StartupUri».....</i>	116
<i>Variante «OnStartUp» im Code Behind .....</i>	116
WINDOW CLOSE .....	116
APP SHUTDOWN .....	117
<b>DETAILS DER BENUTZERINTERAKTION.....</b>	<b>117</b>
REVIEW.....	117
<i>Logical und Visual Tree.....</i>	117
“ROUTES EVENTS” .....	117
<i>Beispiel Button.....</i>	118
<i>Beispiel (TextBox).....</i>	118
<i>Routed Events behandeln.....</i>	118
INTERESSANTE ROUTED EVENTS .....	119
EIN GENAUER BLICK – EVENTARGS .....	119
<i>RoutedEventArgs.....</i>	119
<i>EventArgs der verschiedenen Event-Gruppen.....</i>	119
EIN GENAUER BLICK – KEYBOARD-EVENTS.....	120
SONSTIGE EVENTS .....	120
ACHTUNG! ROUTED EVENTS!.....	120
<i>Beispiel Problemstellung.....</i>	120
<i>Event Handling vs. Data Binding.....</i>	121
MULTITOUCH .....	121
DRAG & DROP .....	122
<b>HINTERGRUND-OPERATIONEN .....</b>	<b>122</b>
REVIEW.....	122
IM HINTERGRUND DATEN TRANSFERIEREN.....	122
BACKGROUND OPERATION .....	123
<i>Schritt 1 – visuelles Feedback geben.....</i>	123
<i>Schritt 2 – Starten eines Background-Threads.....</i>	123
<i>Schritt 3 – UI-Thread benachrichtigen.....</i>	123
<b>WPF-ARCHITEKTUR .....</b>	<b>124</b>
INTERNATIONALIZATION (I18N) .....	124
<i>Möglichkeiten zur Internationalisierung.....</i>	124
<i>Standard .NET-Mechanismen (Embedded Resources).....</i>	124
<i>Unterstützung zur Übersetzung.....</i>	125
<i>WPF-Spezifisch.....</i>	125
<i>Standard .NET-Mechanismus in WPF verwenden .....</i>	126
<i>Parameter in String?.....</i>	126

<i>Die Sprache zur Laufzeit ändern .....</i>	126
<i>Nativer WPF-Mechanismus für i18n.....</i>	127
<i>Fazit.....</i>	127
<b>MVVM.....</b>	<b>127</b>
<i>Von MVC zu MVVM.....</i>	127
<i>MVVM im Detail .....</i>	127
<i>N-Tier Architektur .....</i>	128
<i>N-Tier Architektur (WPF).....</i>	128
<i>Eine typische WPF-App .....</i>	128
<i>Zusammenfassung WPF-Architektur.....</i>	134
<i>MVVM Testing.....</i>	134
<i>MVVM &amp; andere UI Technologien .....</i>	135
<i>MVVM in WPF – Technologieübersicht.....</i>	135
<b>PROJEKT-LAYOUT .....</b>	<b>135</b>
<i>Kleine(re) Projekte.....</i>	135
<i>Mittlere Projekte .....</i>	135
<i>Grosse Projekte.....</i>	135

# Konzepte der Android-Entwicklung

## Was ist Android?

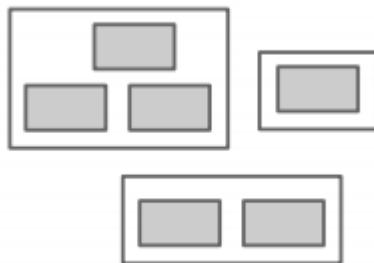
Es ist ein Betriebssystem basierend auf Linux. Es bildet eine Plattform für die (mobile) Softwareentwicklung. Es ist Open Source und steht unter der Apache License. Dabei ist zu beachten, dass nur das Basissystem ist. Es gibt nur sehr wenige Open Source Apps. Die Programmiersprache ist Java und C++. Primär nur Java 7. Java 8 mit Einschränkungen teilweise nutzbar. Es gibt keine Einstiegshürden.



## Chronologie

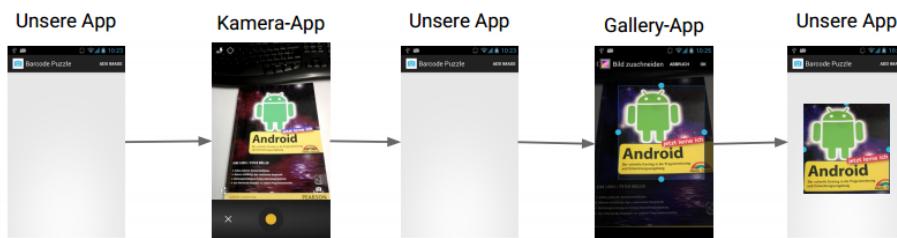
Es wurde 2003 von Andy Rubin gestartet und 2005 von Google übernommen. 2008 ging Android an die Öffentlichkeit. Marktanteil von etwa 86% im Q2 16. Die Anteile sind je nach Land unterschiedlich.

## Kernprinzipien von Android



Apps bestehen aus lose gekoppelten, wiederverwendbaren Komponenten. Die Komponenten sind Activities (etwa Screens) und Services, Content Provider, Broadcast Receivers. Das System hat eine feste Kontrolle über die Applikationen. Es verwaltet den Lebenszyklus, die Kommunikation zwischen den Komponenten und die Apps werden automatisch geschlossen um Speicher zu sparen. Die Komponenten anderer Apps können aufgerufen und Systemkomponenten ersetzt werden (z.B. andere Browser-App).

## Beispiel mit Komponenten



## Mobile and GUI Engineering

### Activities und Lebenszyklus

Activities sind die Hauptbausteine in der App-Entwicklung. Eine Activity ist ein Screen / Bildschirmseite mit GUI Elementen. Eine Activity interagiert mit dem User. Er erhält Touch und andere Events.



### Unterschied zu anderen Modellen

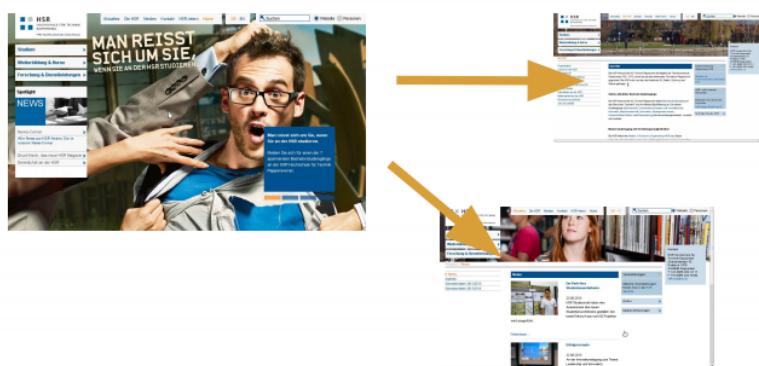
#### Windows



Dabei handelt es sich um die SDI (Single Document Interface) oder MDI (Multiple Document Interface). Der Platzbedarf ist sehr hoch. Das Window-Management findet im Window statt. Ein und dasselbe Hauptfenster in dem alle Aktionen ablaufen. Das User Interface ist auf die Arbeit mit Dateien ausgelegt.

#### Web

Einzelne Seiten, die mit Links verbunden sind.



### Android Activities

Eine oder mehrere Seiten, die einer Aufgabe oder Aktivität gewidmet sind.



## Activity aus Entwicklersicht

Activity = GUI + Code. Es wird implementiert als Klasse die von Activity abgeleitet ist. Das Grundgerüst sieht folgendermassen aus:

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Hier unser Code
    }
}
```

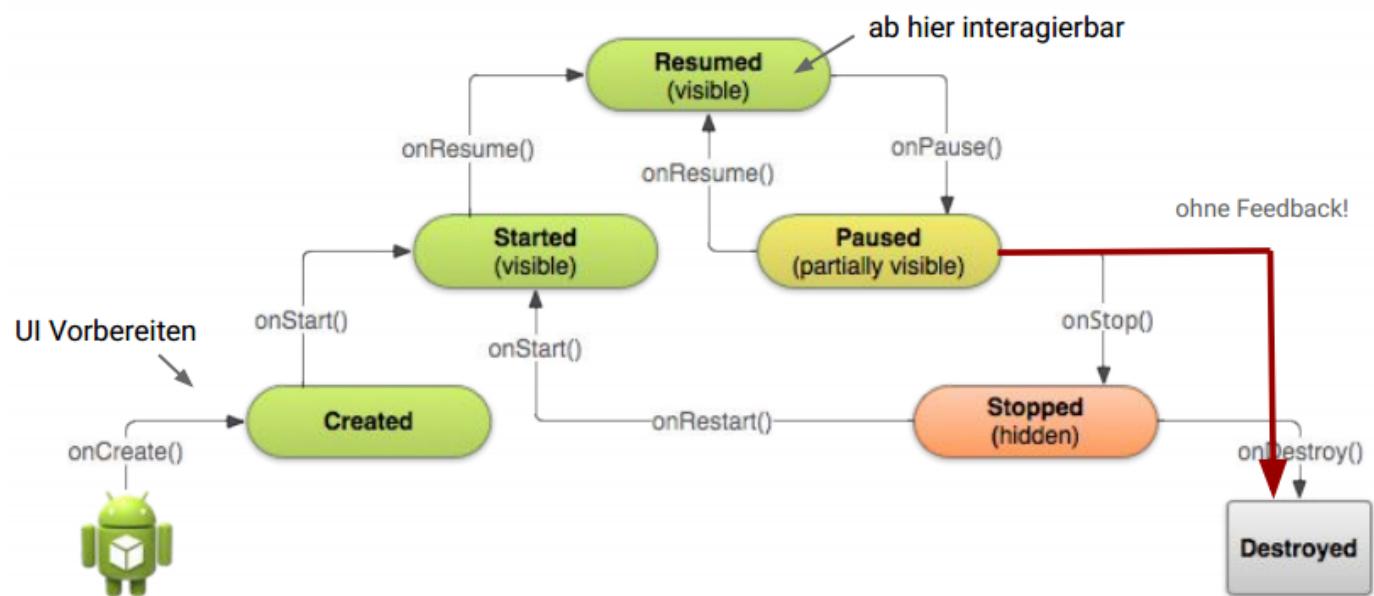
Es gibt kein main oder new MainActivity. Dies wird vom System instanziert.

## Activity Lebenszyklus

Eine Activity kann sich während ihrer Lebenszeit in den verschiedenen Zuständen befinden, zum Beispiel:

- Wird gerade gestartet
- Ist im Vordergrund aktiv
- Wird gleich in den Hintergrund gehen.

Das System ruft bei Zustandswechsel jeweilige Methode der Activity-Klasse auf. Der Entwickler überschreibt die Methoden die ihn interessieren.



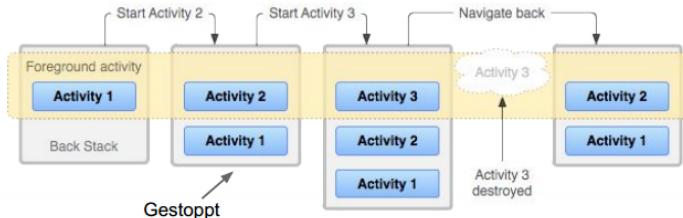
Wird die Activity von einer anderen Activity überdeckt, wird erstere pausiert. Kommt sie dann wieder in den Vordergrund wird `onResume` aufgerufen. Kommt die Activity wieder in den Vordergrund, z.B. weil der User die Applikation nochmal startet oder mit dem Back-Button zurückkommt, dann wird `onRestart` aufgerufen. `Destroyed` wird die App vom System oder wenn es der User explizit schliesst. Dies kann auch direkt aus dem `Paused`-Zustand geschehen!.

Braucht Android den Speicher wird, dann wird die Applikation gekillt. Dies kann aus verschiedenen Zuständen geschehen: `onPause`, `onStop` oder `onDestroy`. Bei einer Konfigurationsänderungen



**Tipp** Daten in onPause sichern, da wir nicht darauf vertrauen können, dass onStop oder onDestroy aufgerufen werden!

## Activity Stack und Tasks



Die Activities werden in einem Stack verwaltet. Activities eines Stacks können zu verschiedenen Apps gehören.

## Activities vs. Tasks

Eine Gruppe von Activities in einem Stack nennt man auch Task. Mehrere Tasks können gleichzeitig existieren. Die Tasks lassen sich über den Overview Screen anzeigen.

## Analogie zum Browser

*Browser:* Tabs mit je eigener History beliebiger Seiten

*Android:* Tasks mit eigenem Stack von Activities.

## Activity Launch Modes

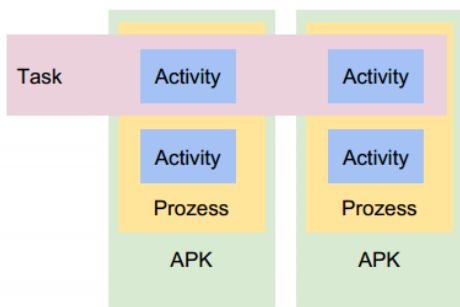
Android verwaltet Tasks und Activities für uns. Das Defaultverhalten ist wie folgt:

- Start der App vom Launcher oder Homescreen startet neuen Task
- Back geht zum vorherigen Screen zurück (pop)
- für den Benutzer vielleicht sogar gewohnt und erwartet

Das Verhalten kann geändert werden, sollte man aber nur in aussergewöhnlichen Umständen machen. Den Back-Button in einem Game unterbinden ist OK, sonst nur mühsam. Launch-Modes können dieses Verhalten ändern.

- Buchstaben stehen für Activities, Farben für Tasks:

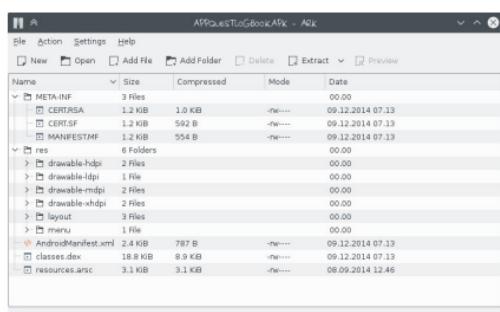
<b>standard</b>	A B + C => A B C
<b>singleTop</b>	A B C + C => A B C
<b>singleTask</b>	A B + C => C      C + D => C D               Browser, Maps
<b>singleInstance</b>	A B + C => C      C + D => D               Home Screen



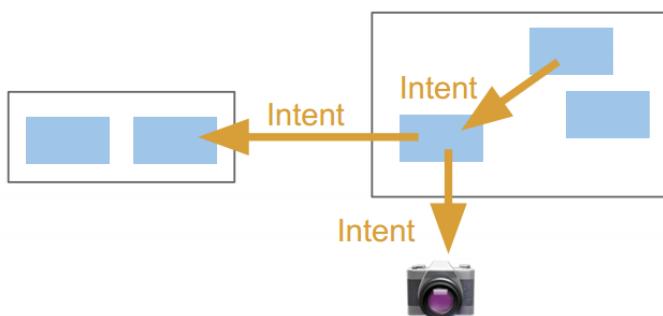
Activities (und Ressourcen, etc.) werden in ein APK gepackt und installiert. Wird eine Activity aktiv, wird pro APK ein Linux Prozess mit einem Thread gestartet. Dieser Prozess führt alle Activities die in diesem APK enthalten sind aus. Jedes APK wird unter einem eigenen Linux User installiert. (isolation).

## APK – Android Application Package

MIME-Type application/vnd.android.package-archive. Es sind jar's mit der Erweiterung .apk. JARs wiederum sind ZIP-Dateien. Ein APK enthält Libraris, Ressourcen, Assets und Metadaten. Die kompilierten Java-Klassen sind im DEX-Format. Die APKs können direkt installiert werden und somit den Play-Store umgehen. Dies muss aber vom User erlaubt sein. Das APK wird vom Build-System erstellt.



## Intents



Die Kommunikation zwischen den Komponenten erfolgt über Intents (Absicht, Vorhaben). Ein Intent beschreibt, was genau gemacht werden sollen. Das System entscheidet wer zuständig ist. Dies ermöglicht die lose Koppelung von Activities und gewährleistet die Austauschbarkeit. Die Apps können selbst wiederum Activities zur Verfügung stellen und so auch bestehende Applikationen ersetzen.

## Intents Erzeugen

Andere Activities können explizit oder implizit aufgerufen werden.

- Explizit mit der **Klasse**:

```
new Intent(this, CalculateActivity.class)
```

- Implizit mit einer **Aktion**:

```
new Intent(MediaStore.ACTION_IMAGE_CAPTURE)
```

Typischerweise werden interne Activities explizit aufgerufen, wohingegen generische Aktionen implizit angefordert werden (Bild erstellen, E-Mail senden).

## Intents Starten Activities

- Mit startActivity um Kontrolle zu übergeben

```
startActivity(intent);
```

- oder startActivityForResult um Rückgabewert zu erhalten

```
startActivityForResult(intent, SOME_ID);
```

```
@Override
protected void onActivityResult(int request, int result, Intent data) {
    if (result == Activity.RESULT_OK && request == SOME_ID) {
        // Resultat verarbeiten
    }
}
```



## Intents Parameterübergabe

Die Intent-Klasse hat eine setData Methode, welche eine URI entgegennimmt, in einem Fall das Ziel der auszuführenden Aktion. Zum Beispiel Datei, URL oder auch ein Kontakt der angerufen werden soll. Zudem gibt es zusätzliche Parameter, welche an eine Activity übergeben werden können.

```
intent.putExtra(MediaStore.EXTRA_OUTPUT, imageCaptureUri);
```

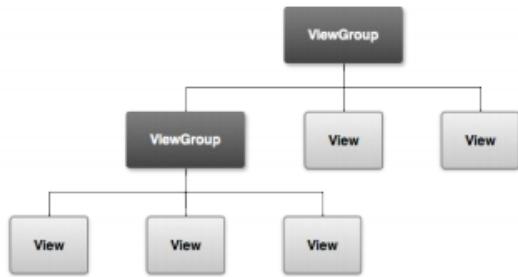
Vorhanden sind Key-Value Paare mit weiteren Daten. Nur für primitive Daten, Strings, sowie solche die serialisierbar sind.

## Weitere Komponenten: Services, Broadcasts

Intents starten nicht nur Activities, sondern werden auf für die Kommunikation mit anderen Komponenten verwendet. Z.B. Um Services zu starten oder um Broadcasts zu versenden. Services sind Hintergrundprozesse die ohne User Interface auskommen. Broadcasts werden über einen Messagebus an alle interessierten Apps im System versendet. Über Broadcasts werden auch Informationen von Android an die Apps weitergegeben, z.B. wenn die Batterie leer ist oder geladen wird. Services und Broadcasts werden in einem separaten Kapitel behandelt.

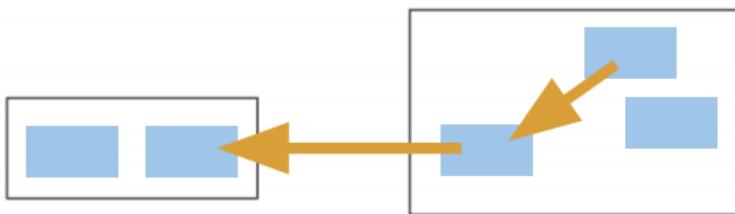
## Views

Graphisches User Interface, alles was der Benutzer sieht und mit dem er interagiert. Das GUI kann deklarativ oder imperativ erstellen werden (Deklarativ: XML, Imperativ: in Java Code). Die Komponenten werden hierarchisch angeordnet. Eine View Group enthält andere Komponenten. Die View ist die Oberklasse aller GUI-Elemente.



## Activity-View-Intent Zusammenfassung

Eine Android App besteht aus einer oder mehreren Activities. Jede Activity hat ihren genau definierten Aufgabenbereich und eine dazugehörige View. Der Wechsel zwischen Activities geschieht über Intents.



## IDE, SDK, API und Tools

Grundsätzlich gibt es nur ein IDE, welches noch zu nutzen ist. Android Studio von Google. Android Developer Tools von Eclipse Andmore ist sozusagen tot.

Das SDK wird im Setup ebenfalls mitinstalliert.

### Android SDK Manager

Er zeigt die installierten SDKs und andere Tools an. Wird mit dem Android Studio mitinstalliert, ist aber standalone. Pro API-Level existiert ein eigenes SDK. Zudem ebenfalls System Images für Emulatoren.

### API Level

API Level sind für die Entwickler relevanten Versionsnummern. Eine Version der Plattform unterstützt immer auch alle älteren APIs. Wenn eine Applikation ein Feature einer neueren Android Version verwendet, dann muss die minimale API entsprechend erhöht werden.

### Wahl des API Levels

Um ein möglichst grosses Publikum zu erreichen, sollte der API Level so niedrig wie möglich angesetzt werden. Allerdings sind bestimmte Funktionen nur in neuen APIs verfügbar. 98% aller Geräte haben Android 4 oder höher. Achtung: Die Statistik umfasst nur die Geräte die den Play Store kontaktiert haben.

### Manifest

Das Manifest umfasst die Komponenten der App, Metadaten, Permissions und Anforderungen an die Geräte API. Sie wird vom System konsultiert um zu erfahren, ob die App installiert werden kann und welche Permissions diese verwendet. Achtung: Android Studio erweitert das Manifest beim Build um zusätzliche Tags!

## Mobile and GUI Engineering

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.misto.myapplication"
    android:versionCode="1" android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="21"
        android:targetSdkVersion="22" />
    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Referenz auf Ressource  
Activity kann gestartet werden  
Wird in der App-Liste aufgeführt

## SDK-Versionen im Manifest

minSdkVersion gibt an, welche Version das Gerät mindestens haben muss. Die targetSdkVersion ist die höchste Version, mit der getestet wurde.

```
<uses-sdk
    android:minSdkVersion="21"
    android:targetSdkVersion="22" />
```

## Application

Parent unserer Activities ist Application. Application ist auch eine Klasse die den globalen Zustand unserer App hält. Es kann auch durch eine eigene Applications-Subklasse ersetzt werden.

```
<manifest >
    <application android:name="MyApplication">
        <activity>
        </activity>
    </application>
</manifest>
```

Aus Activities kann mit getApplication auf die Instanz zugegriffen werden. Sie verfügt über Lifecycle-Methoden: onCreate, onLowMemory und onConfigurationChanged.

## Grundlagen der GUI-Programmierung

In den 70er Jahren sind erstmals GUI verwendet worden, allerdings nur aus Zeichen und nicht Pixeln. Dies waren Systeme wie Xerox Star oder der Apple Lisa.

### WIMP

Windows-Icons-Menus-Pointer Prinzip beherrschte dann die GUI's für einige Jahrzehnte. Es war basierend auf Büro Metaphern. Es wurde allerdings schnell unübersichtlich, da Metaphern irgendwann nicht mehr passen.

### Post-WIMP (Wie ist es heute?)

Die Windows werden von FullScreen-Applikationen verdrängt, Menus verschwinden und Pointer werden durch Gesten ersetzt. Files verschwinden zudem immer mehr (Cloud). WIMP ist aber nicht tot, Windows 10 hat mehr verglichen mit Windows 8.

## GUI Erstellen

GUI's können entweder deklarativ oder imperativ erstellt werden.

## Deklarativ (eine Beschreibung in XML)

Für dieses Art bietet Android Studio einen GUI-Builder an. Dies bildet eine Trennung zwischen UI-Definitionen und Programmlogik. XML-GUI's sind sogenannte Ressourcen.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal">
</LinearLayout>
```

## Imperativ

Dabei wird das GUI über Java Code erstellt.

## Views

Die Basisklassen um UI in Android zu bauen ist View. Eine View belegt einen rechteckigen Bereich auf dem Screen, ist zuständig dessen Inhalt zu zeichnen und Events zu behandeln. Untergruppen sind Widgets und ViewGroups.

### ViewGroup

Bildet eine Unterklasse von View. Es beinhaltet andere Views mit einer Parent-Child Beziehung. Sie kann die beinhalteten Views anordnen. In diesem Fall spricht man dann von einem Layout.

### Eigenes Views

Eigene Views können einfach eingebaut werden. Der Tag entspricht dem Klassennamen. Daher gilt es nur eine passende Klasse zu erstellen.

## Widgets

Ist ein Sammelbegriff für alle fix-fertigen Komponenten des UI. Dazu zählen Buttons, Images, Checkboxen. Keine klaren Definitionen für Android sowie keine Klasse dieses Namens.

### Eingabe-Widgets

Interaktive Widgets, um die Benutzereingaben entgegen zu nehmen (Buttons, Checkboxen, Toogles, ...). Die Aktionen des Benutzer lösen dann Events aus.

### Buttons

Er kann eine Kombination aus Text und Bild anzeigen. Dieser löst einen Click-Event aus, auf die unsere Applikation reagieren kann.

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Alarm"
  android:id="@+id/alarmButton" />
<ImageButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/imageButton"
  android:src="@mipmap/ic_launcher" />
```

Elemente haben eine ID, also ein String mit @Präfix

Strings mit @Präfix sind Referenzen auf Ressourcen

### Eingabefelder

Für Strings und Zahlen. Abhängig des Typs wird auch eine spezifische Tastatur gewählt. Kombinationen sind möglich. Wird mit dem Attribut **inputType** angegeben.

# Layouts

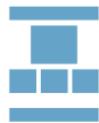
Es sind ViewGroups und beschreiben die visuelle Struktur des UI's. Die Sammlung ist nicht vollständig und es können auch eigene Layouts erstellt werden.



Linear Layouts



Grid Layout



Relative Layout



Table Layout

## Layout Parameter

Sie geben an wie ein Layout seine Kinder anordnet. Dies wird über Attribute bestimmt. Auch die Kinder werden attributiert. In XML sind es diese mit dem layout\_ Präfix.

Für alle gemeinsam sind die Parameter height und width. Zulässige Werte sind android:layout\_width="match\_parent" und „wrap\_content“. Match Parent heisst so gross wie möglich, also wie es der Parent erlaubt. Wrap Content so klein wie möglich, also wie die Kinder erlauben.

## Linear Layout

Elemente werden entweder vertikal oder horizontal angeordnet. Gleich viel Platz für alle Elemente ist selten sinnvoll, daher kann zusätzlich mit layout\_weight ein Gewicht vergeben werden. Ohne weight wird nur der minimale Platz verwendet.



## Relative Layout

Das vielseitige Layout, welches Kinder relativ zueinander anordnet.

```
<RelativeLayout xmlns:android="...">
    <TextView
        android:text="1. Platz"
        android:id="@+id/first"
        android:layout_centerHorizontal="true"/>
    <TextView
        android:text="2. Platz"
        android:id="@+id/textView2"
        android:layout_below="@id/first"
        android:layout_toStartOf="@id/first" />
    <TextView
        android:text="3. Platz"
        android:id="@+id/textView3"
        android:layout_below="@id/first"
        android:layout_toEndOf="@id/first" />
</RelativeLayout>
```



## Weitere Layouts

**Framelayout** kann Kinder übereinander anordnen, (Live-Kamerabild, Kartenansicht)

**ConstraintLayout** Konzeptuell mit RelativeLayout verwandt.

**WebView** Um HTML anzuzeigen, JS kann auch aktiviert werden und mit Java angesprochen

## Layouts und Activities

Android instanziert die im XML deklarierten Views automatisch. Die generierte Klasse R enthält Konstanten für die Layout-Datei. Mit setContentView wird das Layout in der onCreate() Methode der Activity erzeugt.

## @Referenzen und @+IDs

Sind dazu da um GUI-Elemente zu referenzieren. IDs sind Stings, die mit @ beginnen. Wenn wir einen neuen ID einführen, wird diese mit @+ geschrieben. Bei erneuter Verwendung reicht dann @ aus.

Das Android-Buildsystem sammelt alle diese IDs als Konstanten in der automatisch generierten Klasse R. Ermöglicht uns, im Code auf die XML deklarierten Elemente zuzugreifen.

## Ressourcen

### Die Klasse R

Auch für alle Ressourcen enthält die Klasse R eine Konstante. Bis jetzt kennen wir den Typ Layout. Eine Layoutdatei activity\_main.xml im Ordner layout wird durch die Konstante R.layout.activity\_main repräsentiert.

Als Ressourcen gibt es nicht nur Layouts, sondern auch noch drawables (Bilder), menu (Menus), mipmap(Launcher-Icon) und values (Strings und andere Konstanten). Wie erwartet wird pro Ordner eine innere Klasse in R generiert, mit der Ausnahme der Values.

### Values Ressourcen

Sind Strings, welche externalisiert werden. Zudem Farben und Dimensionen in den jeweiligen XML-Dateien. Zugriff darauf erlangt man mit getString(R.string.app\_name).

```
<resources>
    <string name="app_name">My Application</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

### Dimensionen

Konstanten in dimens.xml werden für Größen in den Layouts benutzt

```
<RelativeLayout xmlns:android="..." xmlns:tools="..."
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
```

Inhalt von dimens.xml

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

<dimen name="activity\_horizontal\_margin">16dp</dimen>
<dimen name="activity\_vertical\_margin">16dp</dimen>

Größenangaben von Views erfolgen in density-independent pixels: **dp** oder **dip**

	mdpi-Screen	hdpi-Screen ("Retina")
Länge in Inch	1.5in	1.5in
Dots per Inch (DPI)	160	240
Dots oder Pixels	1.5in * 160dots/in = 240dots	1.5in * 240dots/in = 360dots
Density (als Faktor von 160)	1	1.5
dp	240	240

Quelle: <http://blog.edwinenvans.me/?p=131>

Bei Schriften verwendet man scale-independent pixels: **sp**

### Varianten von Ressourcen

Einerseits dient die Auslagerung zur sauberen Trennung zwischen der Strukturdefinition und dem Aussehen. Anderseits aber auch für Bilder in verschiedenen Auflösungen, Texte in anderen Sprachen und verschiedene Farben. Mehr dazu später.

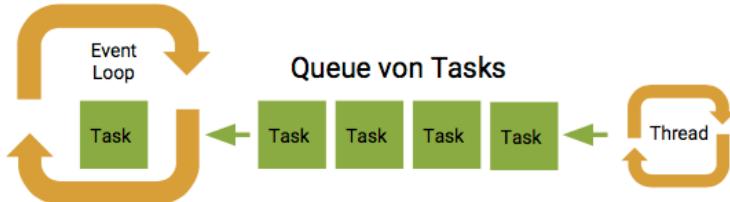
**Beispiel:** übersetzte Strings (Pro Sprache ein Ordner auf dem Filesystem).

**Nebenbei:** Es müssen nicht alle möglichen Selektoren auswendig gelernt werden. Hat Wizard dafür

## Events

Sobald die App läuft (Activity-Lifecycle Methoden wurden aufgerufen) hat unsere App keine aktive Kontrolle mehr. Das Android Framework hat dafür einen sogenannten Event-Loop. Der Loop wartet bis ein Ereignis passiert und bearbeitet dies dann.

Nur der Main-Thread darf das GUI verändern.



Die Events werden durch den Benutzer (Tasten, Geste) oder von Sensoren ausgelöst. Um darauf reagieren zu können, müssen wir diese zuerst empfangen. Dies geschieht durch sogenannte Event Listener. Listener müssen bei dem entsprechenden Objekt registriert werden.

## Listeners

Für unterschiedliche Typen von Events gibt es unterschiedliche Listener Interfaces. Das App implementiert ein Interface und registriert es für die gewünschten Events. Es gibt zudem spezialisierte Interfaces für bestimmte Widgets z.B: TextWatcher für Eingabefelder.

OnTouchListener: wenn Touch Events wie z.B. Gesten ausgelöst werden

OnClickListener: wenn eine View "angeklickt" wird

OnLongClickListener: touch-and-hold einer View

OnKeyListener: wenn Hardware Tasten ausgelöst werden

## GUI-Objekt finden

Um einen Listener zu registrieren, wird zuerst die Objektinstanz benötigt. Mit der ID können wir diese finden. Button button = (Button) findViewById(R.id.button). Die vorherige Methode sucht im aktuellen Layout. Der Rückgabetyp ist immer die Oberklasse View, das Resultat muss also noch gecastet werden.

## Listener Registrieren

```

button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        ...
    }
});

```

An einem View-Objekt können wir dann die entsprechenden Listener registrieren. Am besten sollte aber die Activity oder das Fragment das Interface implementieren.

onClick Listener können auch direkt im XML angeben werden. Die Activity-Klasse muss dazu eine Methode mit der Signatur public void onButtonClicked(View view) implementieren.

Ein Listener kann natürlich auch für mehrere Views verwendet werden. Dazu braucht es dann innerhalb der Methode eine entsprechende Fallunterscheidung.

## Events zur Texteingabe

Bei der Verarbeitung von Texteingaben haben wir mehrere Möglichkeiten auf Events zu reagieren. Das Interface ist Textwatcher und umfasst 3 Methoden:

**beforeTextChanged**

wird aufgerufen bevor der Text geändert wird, also noch alter Text

**onTextChanged**

wird aufgerufen, sobald der Text geändert hat (Einzelnes Zeichen)

**afterTextChanged**

nachdem der Text geändert wurde, Achtung vor Endlosschleife!

```
editText.addTextChangedListener(new TextWatcher() {

    public void onTextChanged(CharSequence s, int start, int before, int count){

    }
    public void beforeTextChanged(CharSequence s, int start, int count, int after){

    }
    public void afterTextChanged(Editable s) {

    }
});

});
```

**Inputvalidierung**

EditTexts bieten eine einfache Möglichkeit Inputvalidierung durchzuführen. Mit der setError Methode wird die Nachricht gesetzt, bei jeder Änderung wird diese wieder resetted.

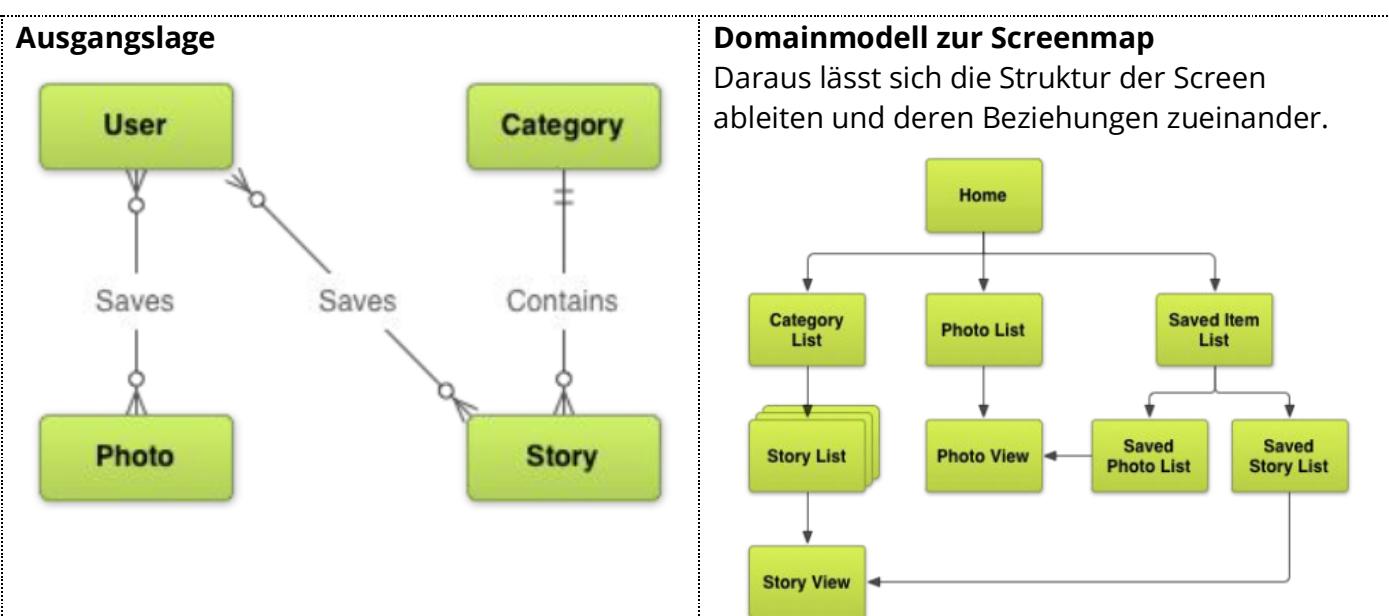
```
final EditText password = (EditText) findViewById(R.id.password);
password.addTextChangedListener(new TextWatcher() {
    @Override
    public void afterTextChanged(Editable s) {
        String pw = s.toString();
        if (s.length() < 8) {
            password.setError("Passwort muss mindestens 8 Zeichen lang sein.");
        }
    }
});
```

**Strukturierung und Navigation**

Die Benutzbarkeit einer App hat heute einen sehr hohen Stellenwert. Die Konkurrenz ist gross und die Benutzer sind schnell wieder weg. Kleine Screen sowie mobile Benutzung stellen erhöhte Anforderungen. → Thema im Modul HCID.

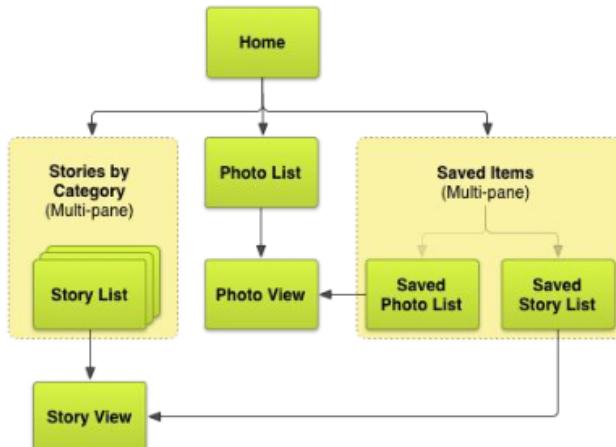
**Vorgehen beim Entwurf**

Aus einer Domainanalyse ergibt sich das Domainmodell (auch Domain Information Model). Als Beispiel nehmen wir eine App zum Merken von Fotos und Stories.



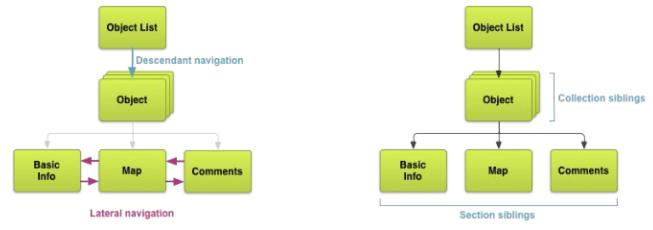
## Screens Gruppieren

überlegen, ob die Screens alle einzeln sind oder auf einem Tablet, auch gleichzeitig.



## Navigation zwischen Screens

Mit den Beziehungen und den Screens kann als nächstes die Navigation festgelegt werden.



## Vorgehen Schritt-für-Schritt

- Domainmodell entwerfen
- Screens ableiten
- Screens in Beziehung bringen und gruppieren
- Navigation zwischen den Screens festlegen
- Wireframe/Storyboard für die Gesamtübersicht erstellen
- Usability Test mit einem Paper-Prototypen des Wireframes

## Up- und Home-Navigation

Es gilt nicht zu vergessen: der Anwender muss auch wieder zurückkönnen. Zurück kann bedeuten:

- Zum hierarchischen Parent (Ancestral Navigation)
- Zum Vorherigen Element (Temporal Navigation)

Eine Ancestral Navigation geschieht über den Up oder Home Button, während die Temporal Navigation über den Back Button funktioniert.

## Fragments

Activities sind die Screens unserer App. Je nach Formfaktor wäre es aber nützlich mehrere Screens anzuzeigen. (Tablet). Mit Activities ist dies grundsätzlich nicht so einfach machbar. Daher gibt es nun Fragments.

Ein **Fragment** ist ein Modularer Teil einer Activity mit eigenen Lebenszyklus. Es wird in einen Activity-Layout eingebunden. Das Fragment kann in mehreren Activities eingebunden werden.

Der Lebenszyklus ist ähnlich, aber noch umfangreicher. Zusätzlich sind folgende Methoden:

**onAttach:** Fragment wird einer Activity hinzugefügt

**onCreateView:** UI des Fragments erstellen

**onActivityCreated:** wenn Activity onCreate fertig ist

**onDestroyView:** Gegenstück zu onCreateView

**onDetach:** Gegenstück zu onAttach

Fragments erben von Fragment. Der LayoutInflator nimmt XML und instanziert die View-Klassen. Der ViewGroup container ist das Layout in der Parent-Activity.

```
public class MainActivityFragment extends Fragment {
    public MainActivityFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_main, container, false);
    }
}
```

false bedeutet, dass die View nicht in den Container eingefügt werden soll, da das bereits das Fragment macht

## Fragment statisch einfügen

Statisch bedeutet, dass wir ein Fragment fix einbinden.

### Layout der Activity

```
<LinearLayout ...>
<fragment
    android:id="@+id/fragment"
    android:name="com.example.myfragmentapplication.MainActivityFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout="@layout/fragment_main" />
</LinearLayout>
```

### Code der Activity

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

## Fragment dynamisch einfügen

Dynamisch eingebundene Fragmente lassen sich austauschen.

### Layout der Activity

```
<LinearLayout ...>
<FrameLayout
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
</LinearLayout>
```

### Code der Activity

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
            fragmentManager.beginTransaction();

        MainActivityFragment fragment = new MainActivityFragment();
        fragmentTransaction.add(R.id.fragment_container, fragment);
        fragmentTransaction.commit();
    }
}
```

Animat  
einba

## Activity Fragment Kommunikation

Fragments sind wiederverwendbare Komponenten und werden daher in verschiedenen Activities eingebunden. Sie dürfen also keine Abhängigkeiten auf Activity haben. **Best Practices:** Ein Fragment definiert ein Interface zur Kommunikation, welches die Parent Activity implementieren muss.

```
public class MainActivityFragment extends Fragment {
    public interface OnItemSelectedListener {
        void onItemSelected(String item);
    }
    OnItemSelectedListener parentActivity;

    @Override
    public void onAttach(Context activity) {
        super.onAttach(activity);
        if (!(activity instanceof OnItemSelectedListener)) {
            throw new AssertionError(
                "Activity must implement View.OnClickListener!");
        }
        parentActivity = (OnItemSelectedListener) activity;
    }
}
```

Activity muss Interface implementieren

Auf einem Tablet wurde die Detailansicht direkt nebendran angezeigt, während es auf dem Smartphone ein eigener Screen wäre. Die Unterscheidung zwischen den Anzeigevarianten wird über Layouts getroffen. Das Default-Layout der ItemListActivity beinhaltet nur das ItemListFragment, während das Tablet-Layout ein Linear-Layout mit dem ItemListFragment sowie einem Platzhalter für das ItemDetailFragment enthält.

```
public class ItemListActivity
    extends Activity
    implements ItemListFragment.Callbacks {

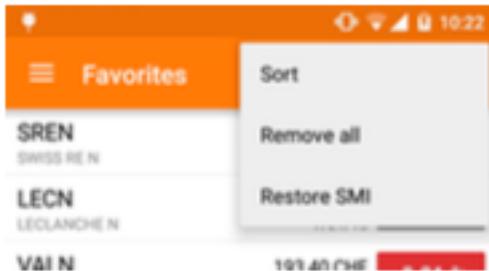
    private boolean twoPane; ← Wir merken uns, ob wir im
                                Tablet-Modus sind oder nicht

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_item_list);
        if (findViewById(R.id.item_detail_container) != null) {
            twoPane = true;
        }
    }
    ...
}

...
@Override
public void onItemSelected(String id) { ← Parameter für anzulegendes Item
    if (twoPane) {
        Bundle arguments = new Bundle();
        arguments.putString(ItemDetailFragment.ARG_ITEM_ID, id);
        ItemDetailFragment fragment = new ItemDetailFragment();
        fragment.setArguments(arguments);
        getSupportFragmentManager()
            .beginTransaction()
            .replace(R.id.item_detail_container, fragment) ← Fragment austauschen oder
            .commit();
    } else {
        Intent detailIntent = new Intent(this, ItemDetailActivity.class);
        detailIntent.putExtra(ItemDetailFragment.ARG_ITEM_ID, id);
        startActivity(detailIntent);
    }
}
```

## Wann Fragments verwenden?

Eine Activity stellt immer auch einen möglichen Eintrittspunkt in die ein App dar (einen Task). Wir dies gebraucht, dann auf jeden Fall eine Activity ansonsten reichen Fragments aus.



## Menus

## Options Menu

Es ist Teil der ActionBar und war früher über die Menu-Taste erreichbar. Es enthält Actions die generell für die App/Activity gedacht sind. Apps mit einem NavigationDrawer haben teilweise kein Options Menu mehr.

```
Options Menu (Code)
```

```
public boolean onCreateOptionsMenu(Menu menu) {  
    menu.add(0, START_MENU_ITEM, 0, "Start");  
    menu.add(0, SUBMIT_MENU_ITEM, 0, "Submit");  
    return true;  
}  
  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case START_MENU_ITEM:  
            // handle start  
            return true;  
        case SUBMIT_MENU_ITEM:  
            // handle submit  
            return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

frei wählbare Konstanten

true bedeutet behandelt

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" tools:context=".MainActivity">

    <item android:id="@+id/action_search"
        android:title="@string/action_search"
        android:icon="@drawable/ic_action_search"
        android:orderInCategory="100"
        android:showAsAction="never" />

    <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        android:showAsAction="never" />
</menu>

public class MainActivity extends Activity {

    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_search:
                // handle start
                break;
        }
    }
}
```

Auch Fragments können Einträge dem Menu der Activity hinzufügen. Die Behandlung erfolgt analog entweder im Fragment oder in der Activity.

```
public class MainActivityFragment extends Fragment {
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        inflater.inflate(R.menu.main, menu);
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }
}
```

Hier ein Parameter  
Wichtig!

## Context und Popup Menus

Context Menu für Actions, die die selektierte View betreffen. Dies ist immer seltener, meistens ändert sich bei Selektion die Actionbar..

Popup Menus lassen sich an einen beliebigen Button binden. Vergleichbar zu einem Spinner/Combobox.

Context und Popup Menus funktionieren ähnlich wie Options Menu.

## Action Bar



1. App Icon und ev. Up- / Home-Navigation
2. Name der App oder View-Switcher
3. Actions (Teil des Options Menu)
4. Action-Overflow mit dem Rest des Menus

Welche Actions in der Action Bar angezeigt werden hängt einerseits vom verfügbaren Platz aber auch von der Menu-Item Konfiguration ab. collapseActionView ist ein Flag für Actions die direkt in der ActionBar angezeigt werden, sogenannte Actions View.



Die ActionBar hat die Möglichkeit, die Icons und Actions unten am Screen anzuzeigen (das hängt aber vom Gerät und Theme ab). Zudem gibt es eine Leiste für Tabs (werden später noch detaillierter behandelt). Seit 5.0 ist aber die Action Bar deprecated.

## Toolbar – Die neue Action Bar

ActionBar ist zu unflexibel, deshalb gibt es seit Lollipop die neue Toolbar. Die ActionBar ist allerdings immer noch in den API Guides vorhanden. Die Toolbar hat einen ActionBar-Modus um z.B. das Options Menu nachzubauen.

```
<RelativeLayout xmlns:android="..." xmlns:app="..." xmlns:tools="..."
    android:layout_width="match_parent" android:layout_height="match_parent"
    tools:context=".MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </android.support.v7.widget.Toolbar>

    <fragment
        ...
        android:layout_below="@+id/toolbar"
        tools:layout="@layout/fragment_main" />

</RelativeLayout>
```

Damit die Toolbar als ActionBar funktioniert, müssen wir diese konfigurieren. Die Toolbar übernimmt dann auch die Actions und Action Overflow.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

## Navigation Drawer

Die Hauptnavigation der App und Wechsel zwischen den Activities oder Fragments. Teilweise auch Settings. Er ist platzsparend, allerdings wird der Hamburger auch sehr kritisch gesehen.

Um überhaupt zu sehen, was wir machen können, müssen wir daran denken das Menu zu öffnen.

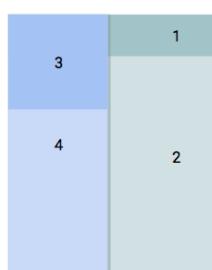
### Implementierung

Layout der Activity (1+2)

- DrawerLayout
  - FrameLayout
    - Toolbar (1)
    - Eigentlicher Inhalt der Activity (2)
  - NavigationView

Die NavigationView benötigt eine

Layoutdefinition (3) sowie eine  
Menudeklaration (4)



### Implementierung

```
<android.support.v4.widget.DrawerLayout xmlns:android="..." xmlns:app="..." xmlns:tools="..."
    android:id="@+id/drawer_layout"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">

    <FrameLayout
        android:id="@+id/content"
        android:orientation="vertical">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"/>

        <TextView ... />
    </FrameLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/navigation_view"
        android:layout_gravity="start"
        app:headerLayout="@layout/drawer_header"
        app:menu="@menu/drawer" />
</android.support.v4.widget.DrawerLayout>
```

### Activity

In der Activity kann dann ein Listener registriert werden, welcher auf die Menupunkte des Drawers reagiert

```
NavigationView view = (NavigationView) findViewById(R.id.navigation_view);
view.setNavigationItemSelectedListener(
    new NavigationView.OnNavigationItemSelectedListener() {

        @Override
        public boolean onNavigationItemSelected(MenuItem menuItem) {
            menuItem.setChecked(true);
            drawerLayout.closeDrawers();
            return true;
        }
});
```

### Support

Die NavigationView ist noch nicht Teil von Android

Aber über die Design-Support Library erhältlich

- Design-Support rüstet Material-Design Komponenten nach (mehr in L5)  
build.gradle der App anpassen mit

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.+'
    compile 'com.android.support:design:24.+'
}
```

## Toasts

Sind kleine Feedback-Nachrichten.

```
Toast toast = Toast.makeText(getApplicationContext(), "Hello MGE!", Toast.LENGTH_SHORT);
toast.show();
```

Toast.LENGTH\_SHORT oder Toast.LENGTH\_LONG

Die Position ist änderbar.

```
toast.setGravity(Gravity.TOP| Gravity.LEFT, 0, 0);
```



## Snackbar

Die Snackbar löst den Toast für Feedback-Nachrichten ab. Sie ist flexibler, da auch eine Action angegeben werden kann.

```
private void mkSnack() {
    Snackbar snackbar = Snackbar.make(content, "Hello MGE!", Snackbar.LENGTH_LONG);
    snackbar.setAction("Again!", new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mkSnack();
        }
    });
    snackbar.show();
}
```



## Listen

### Einfache ListView

Ist ein sehr häufiges UI-Element. Es gibt die ListView und ExpandableListView Klassen. Einzelne Einträge können komplexere Layouts sein. Beliebige Java-Objekte können dargestellt werden.

```
<ListView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/listView"/>
```

### Adapter Design Pattern

„Passe die Schnittstelle einer Klasse an eine andere von ihren Klienten erwartete Schnittstellen an.“

Die ListView muss mit allen Java-Klassen umgehen können, gleichzeitig soll unser Datenmodell unabhängig von der ListView bleiben. Daher soll ein Adapter zwischen der Daten-Klassen und ListView implementiert werden.

### AdapterView

List- und GridViews sind AdapterViews. Android.R.layout.simple\_list\_item\_1 ist ein vordefiniertes Layout. Achtung android.R. Package und nicht die R Klasse des Projekts.

```
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(
        this,
        R.layout.rowlayout,
        R.id.label);
```

Id einer TextView im Layout

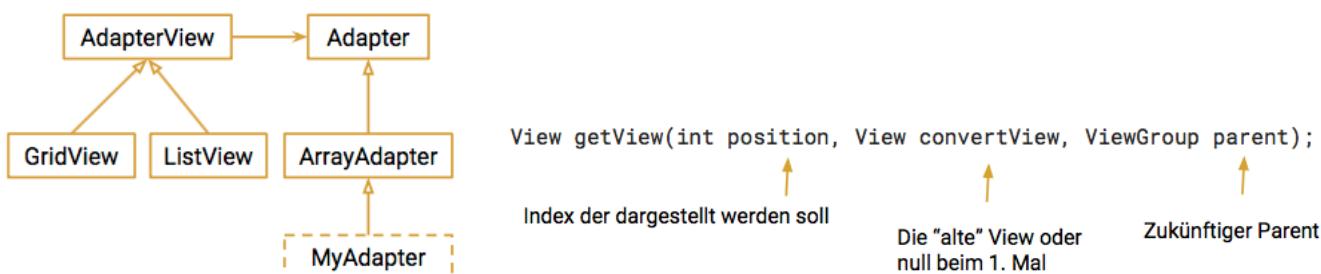
```
Eigenes Layout
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >
    <ImageView
        android:id="@+id/icon"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:src="@android:drawable/btn_star" />
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Platzhalter" />
</LinearLayout>
```

```
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(
        this,
        android.R.layout.simple_list_item_1,
        myStringArray);

listView.setAdapter(adapter);
```

### Eigener ArrayAdapter

Noch mehr Kontrolle erhält man mit einem eigenen ArrayAdapter. getView Methode werden überschrieben um zu kontrollieren, wie genau ein Eintrag aussieht.

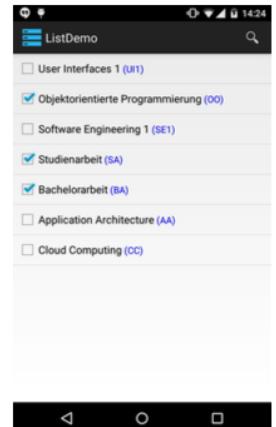


Die Alte View wird aus Performancegründen mitgegeben, damit nicht immer eine neue Instanz der Layouts erstellt werden sondern nur die Daten aktualisiert werden können.

## Beispiel – ListView mit Checkboxen

Die getView hat die folgenden Aufgaben

1. Layout erstellen, falls es nicht schon existiert
2. Anzuzeigendes Modul finden
3. Moduldaten in CheckBox und TextView anzeigen.



```

public View getView(int position, View convertView, ViewGroup parent) {
    final Module module = modulList.get(position);           ← Zugriff auf Domaindaten

    if (convertView == null) {
        LayoutInflater layoutInflater = (...) getSystemService(Context.LAYOUT_INFLATER_SERVICE)
        convertView = layoutInflater.inflate(R.layout.rowlayout, null);
    }                                         ← Layout instanziieren beim 1. Mal

    TextView textView = (TextView) convertView.findViewById(R.id.textView);
    CheckBox checkBox = (CheckBox) convertView.findViewById(R.id.checkBox);

    textView.setText(" (" + module.getCode() + ")");
    checkBox.setText(module.getName());          ← Kinder im Layout finden
    checkBox.setChecked(module.isSelected());      ← Viewdaten setzen

    return convertView;
}

```

public View getView(int position, View convertView, ViewGroup parent) {
final Module module = modulList.get(position);
...
checkbox.setOnClickListener(new View.OnClickListener() {
 public void onClick(View v) {
 CheckBox cb = (CheckBox) v;
 module.setSelected(cb.isChecked());
 }
}); ← Listener falls die Checkbox geändert wurde.

 ...
} ← Domaindaten ändern

■ Code auf [github.com/HSR-MGE/W04-CustomArrayAdapterDemo](https://github.com/HSR-MGE/W04-CustomArrayAdapterDemo)

## ListView Performance

Erste Performanceoptimierung bereits implementiert (alte View in getView wiederverwenden). Die convertView.findViewById ist eine weitere teure Operation. Dazu muss die ganze Objekthierarchie abgesucht werden. ➔ Lösung: gefundene Views zwischenspeichern

Neues Problem: Wo speichern wir solche View spezifischen Daten?

## View Tags

Views können getaggt werden. Zwei Varianten. Einmal mit einem Key-Value Paar oder einfach mit einem Object. Wir können also beliebige Daten an eine View anhängen!

## Optimierte getView

```

if (convertView == null) {
    ...

    TextView textView = (TextView) convertView.findViewById(R.id.textView);
    CheckBox checkBox = (CheckBox) convertView.findViewById(R.id.checkBox);

    Pair<TextView, CheckBox> views = new Pair<>(textView, checkBox);
    convertView.setTag(views);
}

Pair<TextView, CheckBox> views = (Pair<TextView, CheckBox>) convertView.getTag();
TextView textView = views.first;
CheckBox checkBox = views.second;

```

↓ Oder eine eigene Klasse  
(auch ViewHolder genannt)

## Recycler View

Ist der Nachfolger von List- und GridView. Dabei wurden folgende Verbesserungen gemacht:

- Mehrere Layoutmanager
- Animationen für Hinzufügen/Entfernen von Einträgen
- Recycling von Elementen fest eingebaut.

## Komponenten

### Adapter

- Analog zu eigenem ArrayAdapter, leicht anderes API

### ViewHolder

- Klasse, die die Views zwischenspeichert (wie die Pair-Klasse)

### LayoutManager (optional)

- Defaults für übliche Layouts

### ItemDecoration (optional)

- Trennlinien oder andere Dekorationen um die Elemente

### ItemAnimator (optional)

- Animiert hinzufügen, entfernen und scrolling von Elementen

## Schnittstelle

```
public class MyAdapter extends RecyclerView.Adapter<ViewHolder> {
    private ArrayList<Module> dataset;

    public MyAdapter(ArrayList<Module> modules) { dataset = modules; }

    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) { ... }

    public void onBindViewHolder(ViewHolder holder, int position) { ... }

    public int getItemCount() {
        return dataset.size();
    }
}
```

[github.com/HSR-MGE/W04-RecyclerViewDemo](https://github.com/HSR-MGE/W04-RecyclerViewDemo)

getView aufgeteilt in zwei Methoden

## Setup

```
recyclerView = (RecyclerView)                         Suchen
findViewById(R.id.recyclerView);

layoutManager = new LinearLayoutManager(getActivity());   Layoutmanager
recyclerView.setLayoutManager(layoutManager);

adapter = new ...                                     Adapter für die
recyclerView.setAdapter(adapter);                      Daten
```

## Leere Listen

Leere Views (Listen, Grids) sollten unbedingt vermieden werden. Besser ist ein Platzhalter und noch besser ist, wenn man sagt wie ein neuer Eintrag in die Liste kommt.

## Persistenz

### Daten Persistieren

Apps werden vom System beendet, ohne dass dem Benutzer bewusst ist. Sichern der Daten ist also Aufgabe der App! Zwei unterschiedliche Arten von Daten sind vorhanden. Einerseits die Zustandsdaten der Views (aktuelle Eingabewerte, ...) und andererseits die Anwendungsdaten unserer Domain-Klassen.

## View-Daten

onCreate und onSaveInstanceState erhalten ein ein Bundle Objekt. Ein Bundle ist wie eine Map oder PropertyList mit Assoziationen vom String-Key zu Value. Wichtig: Super Aufruf nicht vergessen, dieser speichert alle Views die eine ID haben!.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
    }
}
```

## App-Daten

Die oben genannte onSaveInstanceState Methode wird nicht immer ausgeführt. Beispielsweise wenn die App gekillt wird oder über den Back-Button verlassen wird.

→ Daher sollte die Daten immer in onPause bereit gesetzt werden.

## Shared-Preferences

Mit Key-Value Paaren für wenige Daten (privat in der App).

Key-Value Paare mit Value von Typ

```
boolean, float, int, long, String, Set<String>
String Konstante
Alternativ:
MODE_MULTI_PROCESS
aus mehreren Activities
SharedPreference settings = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);
SharedPreference.Editor editor = settings.edit();
editor.putBoolean("disabled", false);
boolean isEnabled = settings.getBoolean("disabled", false);
editor.commit();
Wichtig! (WICHTIG)
Defaultwert wenn
nicht gesetzt
```

Nützlich: settings.registerOnSharedPreferenceChangeListener

## File Storage

Für private oder Daten die mit anderen Programmen geteilt werden (eher für Binärdaten).

Interner, privater Speicher:

```
FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write("File Content".getBytes());
fos.close();
```

Externer Speicher

- Muss nicht, kann aber removable sein
- Andere Apps können auf diese Daten zugreifen
- Konstanten für verschiedene vordefinierte Verzeichnisse



```
File path = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
File file = new File(path, "HSR_Cat.png");
Lesen/Schreiben wie gewohnt
```

## SQLite Storage

Für strukturierte Daten, die in einer relationalen Datenbank abgelegt werden können (kleine Datenbank, einfache Implementation).

```

public class DBHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;

    DBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE ...");
    }

    @Override
        Abstand kann auch sehr gross sein.
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        } wird ausgeführt, wenn eine neue Version vorhanden ist.
    }

    DBHelper helper = new DBHelper(this);
    SQLiteDatabase db = helper.getReadableDatabase();
    db.execSQL("SELECT * FROM ...");
}

```

## 3rd Party Angebote

Neben diesen built-in Möglichkeiten gibt es auch weitere Anbieter von Persistenzlösungen:

<b>Realm.io</b>	ORM-ähnliches Interface, Domainklassen erben von RealmObject
<b>Firebase</b>	NoSQL (JSON) Cloud-Storage und weitere Dienste
<b>Parse.com</b>	Facebook acqui-hired und geschlossen (Code für self-hosting)

## Hintergrundtasks

### Problematik

```

protected void onCreate(Bundle state) {
    super.onCreate(state);

    final ImageView imageView = (ImageView) findViewById(R.id.imageView);

    Button button = (Button) findViewById(R.id.button);
    button.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            Bitmap bitmap = download("http://slow.hsr.ch/hsr_cat.bmp");
            imageView.setImageBitmap(bitmap); App wird in der Zwischenzeit blockiert.
            ist eingeforen und hängt.
        }
    });
}

```

### Hintergrundaktionen

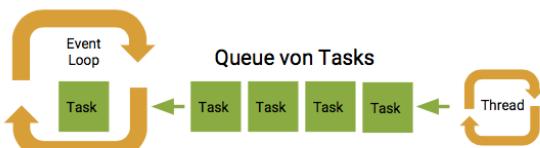
Die Activity läuft in einem Prozess mit einem Thread. Dieser Thread ist der Main oder auch UI Thread. Er koordiniert Events, zeichnet das GUI und führt unseren Code aus.

### Problem

Operationen die lange laufen (Berechnungen, Downloads, ...) blockieren alle anderen Tätigkeiten die unsere Applikation erledigen sollte. Die App reagiert nicht mehr auf den Userinput!

### Rückblick – Event Loop

Event-Loop wird oft auch GUI- oder Main-Thread genannt. Ereignisse werden in einer Message-Queue abgelegt und dann vom Main Thread abgearbeitet.



**Ein erster Lösungsansatz:**

```
public void onClick(View v) {
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            Bitmap bitmap = download("http://slow.hsr.ch/hsr_cat.bmp");
            imageView.setImageBitmap(bitmap);
        }
    };
    Thread thread = new Thread(runnable);
    thread.start();
}
```

**Problem:** Die ImageView und andere GUI Elemente dürfen nur aus dem Main-Thread verändert werden.

**Besserer Ansatz**, ok für einfache Tasks, ein besseres Modell bietet AsyncTask

```
public void onClick(View v) {
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            final Bitmap bitmap = download("http://slow.hsr.ch/hsr_cat.bmp");

            Runnable command = new Runnable() {
                @Override
                public void run() {
                    imageView.setImageBitmap(bitmap);
                }
            };
            imageView.post(command); ← command wird im UI-Thread ausgeführt
        }
    };
    Thread thread = new Thread(runnable);
    thread.start();
}
```

**AsyncTask**

Typischerweise starten wir eine Aufgabe die in einem eigenen Thread ablaufen soll, um dann am Ende wieder etwas auf dem Main-Thread zu tun:

- Vorbereitungsschritt (z.B. im GUI die Aktion anzeigen)
- Berechnung oder Netzwerkzugriff in eigenem Thread
- Resultat im GUI anzeigen

```
class DownloadBitmapTask extends AsyncTask<String, Void, Bitmap> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute(); ← Wird im UI-Thread ausgeführt
    }

    @Override
    protected Bitmap doInBackground(String... params) {
        return download(params[0]); ← Wird in einem eigenen Thread ausgeführt
    }

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        imageView.setImageBitmap(bitmap);
    }
}

new DownloadBitmapTask().execute("http://slow.hsr.ch/hsr_cat.bmp");
```

**AsyncTask Fortschrittsanzeige**

- Anstelle von Void String oder Integer einsetzen für eine Fortschrittsanzeige

```
class DownloadBitmapTask extends AsyncTask<String, Integer, Bitmap> {

    @Override
    protected Bitmap doInBackground(String... params) {
        publishProgress(10);
        return download(params[0]);
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        progressBar.setProgress(values[0]);
    }
}
```

Nun ist auch klar, weshalb wir im LibraryService mit Callbacks arbeiten müssen. LibraryService benutzt die Request-Klasse.

```
class Request<T> extends AsyncTask<Void, Void, Pair<String, T>> {
    protected Pair<String, T> doInBackground(Void... unused) {
        AsyncHttpClient.BoundRequestBuilder request = ...;
        Response response = request.execute().get();
        ...
        return new Pair<>(null, response);
    }

    protected void onPostExecute(Pair<String, T> result) {
        if (result.first != null) {
            callback.onError(result.first);
        } else {
            callback.onCompletion(result.second);
        }
    }
}
```

## Material Design

### Design Language

Definition gemäss Wikipedia: „A design language or design vocabulary is an overarching scheme or style that guides the design of a complement of products or architectural settings.“. Eine Design Language ist also eine Hilfestellung für den Designprozess. Sie beschreibt, wie die Teile einer Applikation aussehen und sich verhalten sollen.

### In Software

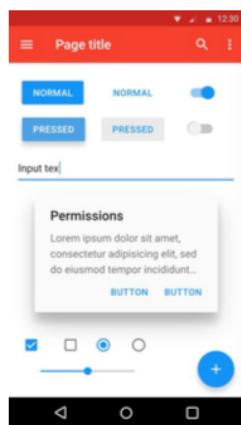
Teileweise sind es klare Regeln, aber auch Empfehlungen und Beispiele. Dazu zählen Farbschemas, Icons, Screenaufbau und auch Buttons. Ziel ist es, ein konsistentes und benutzbares Look-and-Fell zu erreichen. Dies nicht nur innerhalb einer Applikation oder eine Produktfamilie sondern auch im ganzen Betriebssystem. Geklappt hat es nicht immer. Zum Beispiel iTunes auf Windows.

### Human Interface Guidelines

Eine Auswahl davon ist auf Wikipedia zu finden.

<a href="#">Android Material Design</a>	<a href="#">KDE Human Interface Guidelines</a>
<a href="#">Apple Watch Human Interface Guidelines</a>	<a href="#">OS X Human Interface Guidelines</a>
<a href="#">Design library for Windows Phone</a>	<a href="#">Ubuntu App Design Guides</a>
<a href="#">Eclipse User Interface Guidelines</a>	<a href="#">UX guidelines for Windows Store</a>
<a href="#">GNOME Human Interface Guidelines</a>	<a href="#">Apps (Windows 8 and Windows RT)</a>
<a href="#">iOS Human Interface Guidelines</a>	<a href="#">Windows User Experience</a>
<a href="#">Java Look and Feel Design Guidelines, and Advanced Topics</a>	<a href="#">Interaction Guidelines</a>

### Material Design

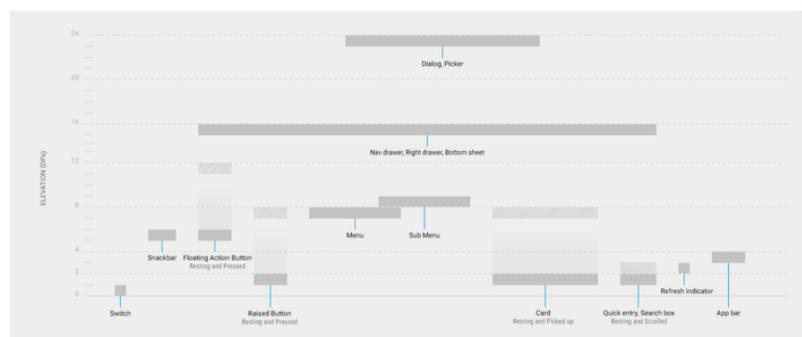


Googles aktuelle Design-Guidelines werden Material Design genannt. An der Google I/O 2014 vorgestellt, sind sie inzwischen in den meisten (nicht nur Google) Apps eingeführt. Je länger, je mehr werden Sie auch für die Google Webapplikationen sowie 3rd Parties übernommen.

### Prinzipien

Material ist die Metapher für einen 3D-Raum mit Licht und Schatten. Es ist angelehnt an die physikalischen Begebenheiten. Bei wurde extra eine eigene Font „Roboto“ entwickelt. Die Ausrichtung findet wie in Printmedien an einem Grid statt. Die Bewegung gibt Erklärung als Reaktion auf den User-Input. Dies aber nur wenn angemessen.

Materialien sind geometrische Formen (bzw. Schnipsel aus Papier) mit einer Dicke von exakt 1dp. Durch die unterschiedliche Anordnung auf der z-Achse entsteht Schatten. Den UI-Elementen sind unterschiedliche Höhen zugewiesen.

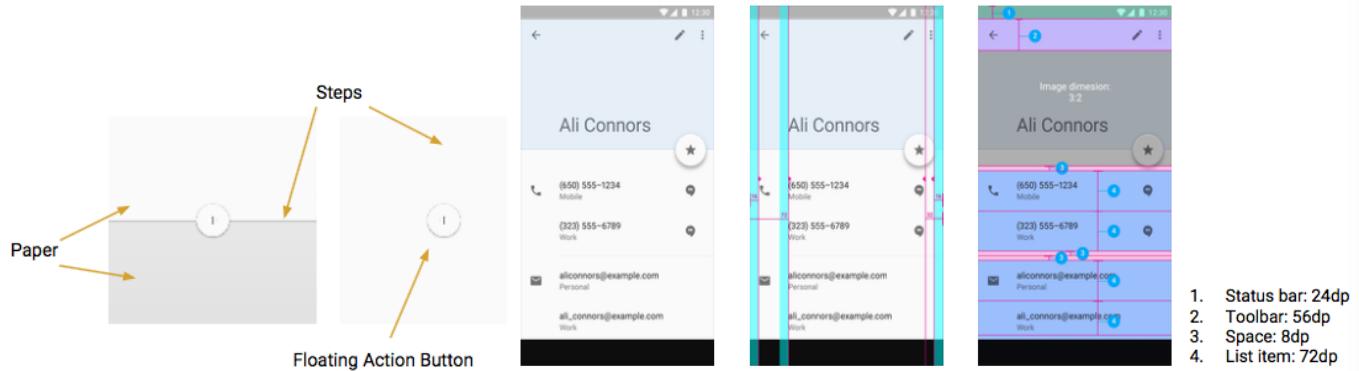


## Style Guide

Dieses Konzept von Material hat Einfluss auf die verschiedensten Aspekte. Die Material Design Styleguide umfasst deshalb Layout, Style, Animation, Components, Patterns und Usability.

## Layout

Das Grundelement ist das Papier, das an- und übereinandergelegt werden kann. Ein Floating Action Button bietet eine Aktionsmöglichkeit für das Papier. Alle Materialien werden an einem 8dp Grid ausgerichtet.



## Style

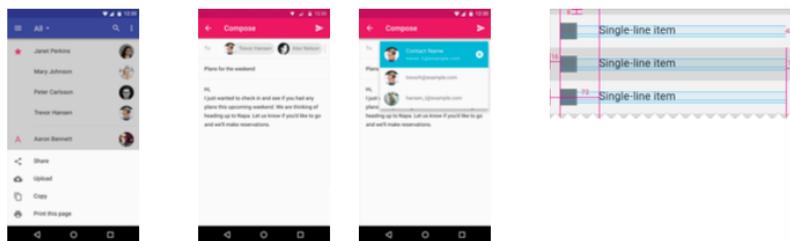
Zur Farbauswahl ist online eine grosse Palette mit Farben und Abstufungen vorhanden. Grundsätzlich ist es empfohlen 3 Farbtöne der Primärpalette und eine Akzentfarbe aus einer zweiten Palette auszuwählen.



## Animation

Die Interaktion mit dem Smartphone/Tablet geschieht direkt mit dem Finder, und nicht indirekt über eine Maus mit Cursor. Die Animationen helfen die Illusion aufrecht zu erhalten, dass wir die Materialien auf dem Screen (durch eine Glasscheibe) direkt manipulieren. Es soll visuelles Feedback auf den Input geben.

Die Bekannten UI-Controls wie Listen, Buttons, aber auch komplexere wie das Bottom Sheet oder die Chips sind mit den Layout-Guidelines definiert.



## Patterns

Patterns geben Hilfestellung bei oft auftretenden Problemen. Zum Beispiel:

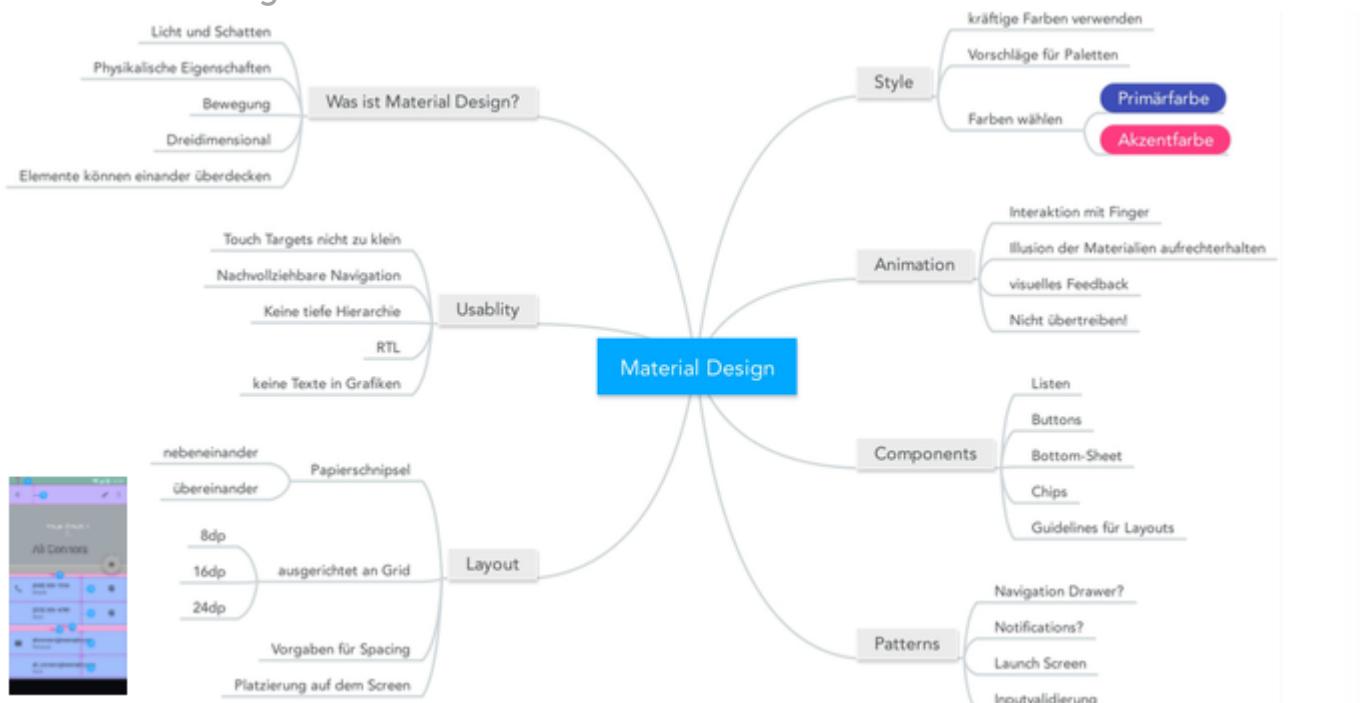
- Was gehört in den Navigation Drawer?
- Wofür sollte man Notifications einsetzen? Mit welchen Actions?
- Brauche ich einen Launch Screen?
- Wie gibt man Feedback zu falschen Usereingaben?

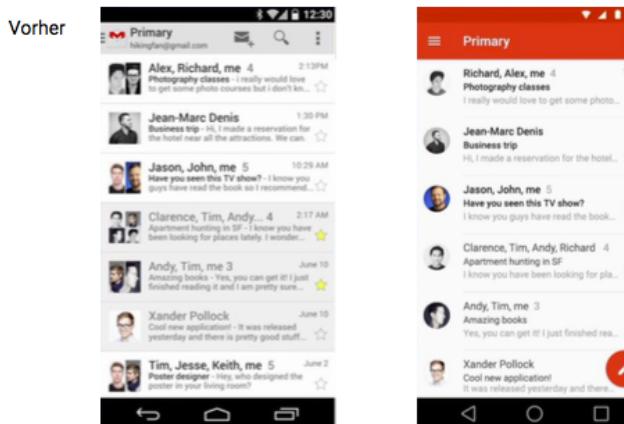
## Usability

Der Usability-Teil in den Guidelines verweist mehrheitlich auf Googles Accessibiliilty Webseite. Wichtige Punkte sind zu Beispiel:

- Touchtargets nicht zu klein machen
- Navigation soll nachvollziehbar und nicht zu komplex sein
- Fontgrößen, Farben
- RTL-Layouts werden (wenn richtig gemacht automatisch) gespiegelt
- Texte in Grafiken sind problematisch.

## Zusammenfassung





## Styling

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button2"
    android:layout_gravity="center"
    android:background="#ff85e1"
    android:height="36dp"
    android:minWidth="64dp"
    android:padding="8dp" />
```

Views können direkt im XML-Layout gestyled werden. Als Verbesserung sollten die Größenangaben in eigene Ressourcen ausgelagert werden. Noch besser ist die Angabe eines Style mit **style="@style/..."**.

Styles werden in **res/values/styles.xml** ausgelagert und in der View mit style referenziert. Wie alle Ressourcen können auch Style für unterschiedliche Geräte, Versionen, etc. spezifiziert werden. Gelten immer für genau diese View. Es findet keine Vererbung statt.

```
<style name="MyButtonStyle">
    <item name="android:background">#ff85e1</item>
    <item name="android:height">36dp</item>
    <item name="android:minWidth">64dp</item>
    <item name="android:padding">8dp</item>
</style>
```

## Themes

Themes sind Styledefinitionen die für eine Activity oder App gelten werden. Achtung, der GUI-Builder könnte zur Anzeige auch anderen Style verwenden.

```
<application
    ...
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:theme="@style/AppTheme" >
```

Die entsprechende Definition in styles.xml:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <!-- Hier können wir Theme-Einstellungen überschreiben -->
</style>
```

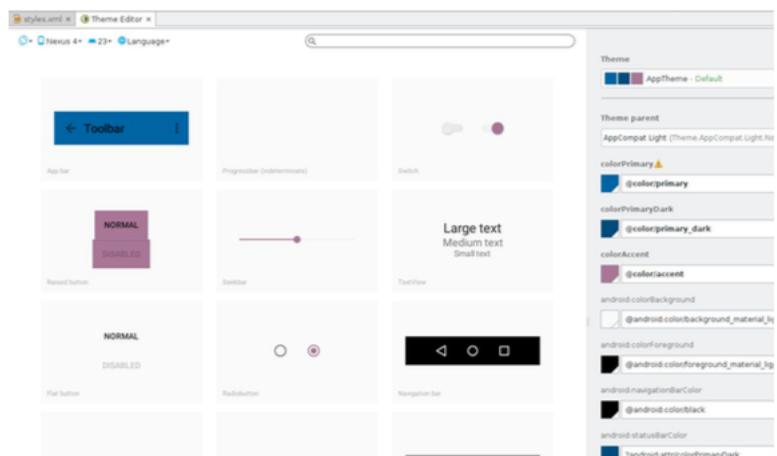
Alle Styles können von einem Parent-Style erben

Styles in Themes gelten für alle Views der Activity.

Im Theme definierte Styles können in anderen Ressourcen referenziert werden. Anstelle eines @ wird ein ? verwendet.

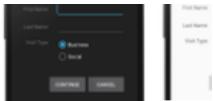
Android hat eine ganze Menge an vordefinierten Styles und Themes. Die Dokumentation ist eher schwach. Am besten sollten direkt die Style-XML's konsultiert werden. Oder im Android Studio im GUI-Builder umschalten.

## Theme Editor in Android Studio



## Material Theme

Das Material Theme ist definiert als:

`@android:style/Theme.Material (dark)`  
`@android:style/Theme.Material.Light`   
`@android:style/Theme.Material.Light.DarkActionBar`

Dies gilt allerdings erst ab API Level 21. Für niedrigere APIs gibt es das Theme in der Kompatibilitätslibrary.

`@android:style/Theme.AppCompat (dark)`  
`@android:style/Theme.AppCompat.Light`  
`@android:style/Theme.AppCompat.Light.DarkActionBar`

Die Design-Guide schlägt vor, eine Primär- und Akzentfarbe zu wählen. Damit diese auch im Theme verwendet werden, müssen wir die entsprechende Option im Theme überschreiben.

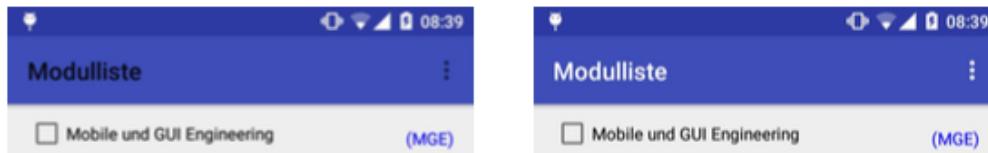
```
<style name="AppTheme" parent="Theme.Material.Light">
    <item name="colorPrimary">@color/primary</item>
    <!-- für die Status-Bar eine dunklere Farbe -->
    <item name="colorPrimaryDark">@color/primary_dark</item>
    <item name="colorAccent">@color/secondary</item>
</style>
```

Die Farben sind wiederum definiert als

```
<item name="primary" type="color">#3F51B5</item>
<item name="primary_dark" type="color">#303F9F</item>
<item name="accent" type="color">#FF4081</item>
```

Achtung! Android: sollte bei AppCompat weggelassen werden.

Die Kombination von Light-Theme und dunkler Toolbar führt dazu, dass die Schrift in der Toolbar schwarz ist.



Views können keine eigenen Themes haben, aber Theme Overlays, die einen Teil der Theme-Attribute überschreiben (hier die Farbe).

```
<android.support.v7.widget.Toolbar
    ...
    app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Menu soll dann aber wieder Light sein

## Material Design Components

### FAB – Floating Action Button

Primäre Aktion für die darunterliegende View. Falls die primäre Aktion nicht eindeutig ist, lieber keinen FAB nehmen, da dieser doch sehr penetrant ist und Inhalte überdeckt?

```
<android.support.design.widget.FloatingActionButton
    android:src="@drawable/plus"
    android:onClick="onPlusClicked" />
```

### TextInputLayout

EditText, bei dem der Hinweistext beim editieren nicht verschwindet.

```
<android.support.design.widget.TextInputLayout
    android:id="@+id/username_text_input_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center">

    <EditText
        android:id="@+id/username_edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Your Username"
        android:layout_gravity="center" />

```

### Scrollende Toolbar

```
<android.support.design.widget.CoordinatorLayout xmlns:android="..." xmlns:app="..."
    android:layout_width="match_parent" android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:layout_scrollFlags="scroll|enterAlways" />

```

Diese View wird gescrolled



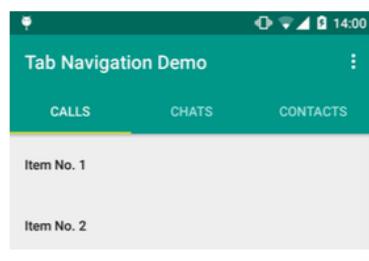
```
<android.support.design.widget.AppBarLayout>
    ...
    <android.support.design.widget.CollapsingToolbarLayout
        android:id="@+id/collapsing_toolbar"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:contentScrim="?attr/colorPrimary"
        app:expandedTitleMarginEnd="64dp"
        app:expandedTitleMarginStart="48dp"
        app:layout_scrollFlags="scroll|exitUntilCollapsed">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:layout_collapseMode="pin" />
```

## Tab Navigation

Kein neues Feature, aber mit CoordinatorLayout und AppBarLayout integriert. Tabs sind unterschiedliche Screens (implementiert mit Fragments). Swipe oder Touch wechselt zwischen Tabs.

```
<CoordinatorLayout>
    <AppBarLayout>
        <Toolbar/>
        <TabLayout/>
    </AppBarLayout>
    <ViewPager/>
</CoordinatorLayout>
```



## TabLayout und ViewPager

TabLayout zeigt die Liste der Tabs. ViewPager zeigt den Inhalt der Tabs an, wechselt also zwischen Fragments. Kann aber auch ohne Tabs verwendet werden.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...

    ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager);
    viewPager.setAdapter(adapter);

    TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
    tabLayout.setupWithViewPager(viewPager);
```



## Tab Navigation – Toolbar Verhalten

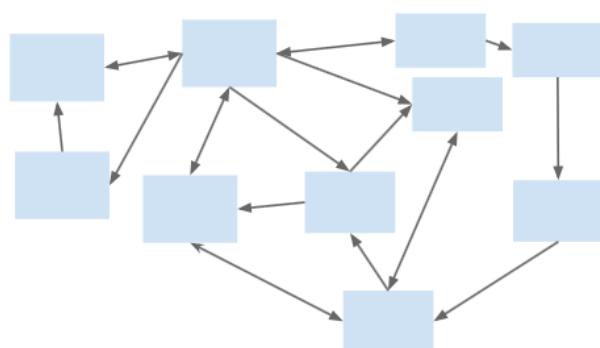
Auch hier lässt sich die Toolbar beim Scrollen verstecken. Achtung, dies klappt nur wenn die Fragments Recyclers oder NestedScrollViews sind und nicht normale List- oder ScrollViews.

```
<android.support.design.widget.CoordinatorLayout ...>
    <android.support.design.widget.AppBarLayout ...>
        <android.support.v7.widget.Toolbar ...
            app:layout_scrollFlags="scroll|enterAlways" /> ← Die Toolbar soll wegscrollen, lies: sich auch auf TabLayout anwenden
        <android.support.design.widget.TabLayout ... /> ← diese View scrollt
    </android.support.design.widget.AppBarLayout>
    <android.support.v4.view.ViewPager ...
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />
</android.support.design.widget.CoordinatorLayout>
```

Für viele Components und Umsetzungen von Patterns gibt es eigene Views. Sie sind nicht im Standard-SDK enthalten, aber in der Android Design Support Library.

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:24.+'  
    compile 'com.android.support:design:24.+'  
}
```

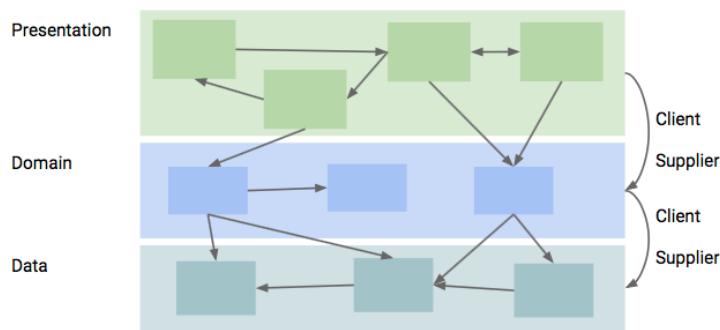
## Software Architektur



Darin stellt sich die Frage, wie ein grosses Software-System in mehrere Teile zerlegt werden kann um die Wartbarkeit und Verständlichkeit zu verbessern.

## Multitier Architecture

Dies ist ein typisches Modell. Darin sind 3 Layer aufgeteilt mit klar unterschiedlichen Aufgaben. Die Abhängigkeiten darin sind klar gekennzeichnet. Keinen Zyklen.



### Presentation

Ist Zuständig für die Darstellung und Interaktion mit dem Benutzer. Dadurch ist dieser Layer sehr stark an ein UI-Toolkit gebunden. Dieser Layer hat Zugriff auf die Domain.

### Domain

Darin sind die Businesslogik und die Domainklassen ohne UI Funktionalität. Der Layer hat wenig externe Abhängigkeiten und ist daher einfach zu testen.

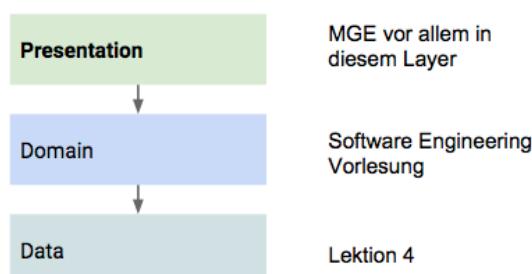
### Data (auch Persistence, Datenhaltung)

Implementiert die Speicherung der Daten, stellt diese Dienste der Domain zur Verfügung (zum Beispiel eine DB oder eine Cloud).

Es sind zudem diverse Variationen dieser Unterteilung denkbar. Der Domain-Layer kann in Businesslayer unterteilt werden, welche die Use Cases implementieren. So werden Layer auch strukturiert in MVC, MVP oder MVVM. Larman hat sogar 6 Schichten.

Wichtig ist, dass keinen zyklischen Abhängigkeiten entstehen. Die Domain soll zum Beispiel nicht abhängig von Android werden → Einfachere Testbarkeit. An den Schnittstellen der Layer wird gerne mit Interfaces und Fassaden gearbeitet um diese zu entkoppeln. Alles weitere in anderen Modulen.

## MGE & Multitier Architecture



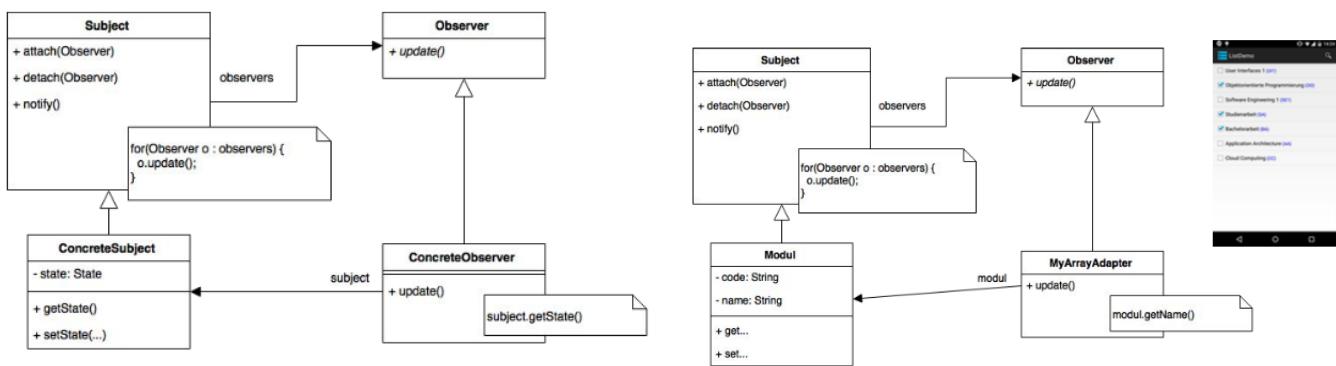
## Keine Zyklen?

Wenn keine Zyklen erlaubt sind, wie kann die Domain dann die Presentation benachrichtigen, wenn der Zustand eines Domainobjektes geändert wird.

Dazu ist das Oberserver Pattern da. Folgende Definition: „Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so dass die Änderung des Zustands eines Objekts dazu führt, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.“

## Observer Pattern

Im Observer Pattern gibt es zwei Rollen. Subject, welches beobachtet wird (z.B. Model). Observer welcher beobachtet wird (z.B. View). Die Observer melden sich beim Subject an, damit das Subject den Observer doch bitte informiert, sobald was geschehen ist. Sobald der Observer benachrichtigt wird, holt er sich vom Subject den aktuellen Zustand. Observer kennt Subject ziemlich gut, aber das Subject muss nichts über den konkreten Observer wissen.



Das Pattern spielt in Desktop-GUIs eine grössere Rolle da es dort öfters auch mehrere Views auf dieselben Daten gibt. Bei grösseren Modellen und vielen Views kann schnell unübersichtlich werden, wer wen observiert (An- und Abmeldungen müssen korrekt gemacht werden). Das Observer Pattern ist oft Teil des Model-View-Controller Patterns.

## Beispiel

ObserverDemo			
Apple	120.94	17:02:33	
Amazon	530.54	17:02:33	
Yahoo	37.20	17:02:32	
Microsoft	46.45	17:02:33	
Oracle	39.78	17:02:32	
Red Hat	80.02	17:02:34	
Last Trade: 17:02:34			

Die App zeigt eine Liste von Aktienkursen sowie das neuste Handelsdatum.  
Die Kurse ändern sich alle paar Sekunden

Obserable → domain.Stock

Observer → presentation.MainActivity & presentation.StocksAdapter

**Oberservable**

```

public class Stock extends Observable {
    private String name;
    private double price;
    private Date date;

    // Konstruktor, Getter
    private void setNewRandomPrice() {
        date = new Date();
        price += new Random().nextDouble() - 0.49;
        setChanged();
        notifyObservers();
    }
}
  
```

wird periodisch aufgerufen

nicht vergessen!

**Oberserver**

```

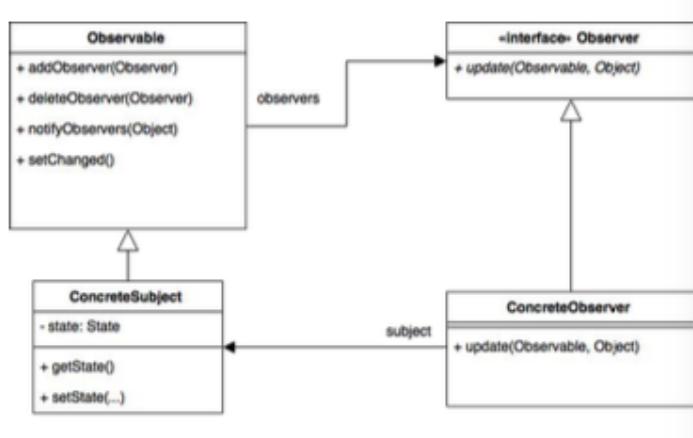
public class StocksAdapter extends RecyclerView.Adapter<...> implements Observer {
    private ArrayList<Stock> dataset;
    public StocksAdapter(ArrayList<Stock> stocks) {
        dataset = stocks;
        for (Stock stock : stocks) {
            stock.addObserver(this);
        }
    }
    @Override
    public void update(Observable observable, Object data) {
        int position = dataset.indexOf(observable);
        notifyDataSetChanged();
    }
}
  
```

anmelden um benachrichtigt zu werden

das aufrufende Stock-Objekt

Achtung, auch wieder abmelden!





Subject = Klasse Observable

Observer = Interface Observer

Notify finden in zwei Schritten statt (setChanged und notifyObservers).

updateMethode enthält als Parameter das Subject sowie den Parameter der beim Aufruf von notifyObservers mitgegeben wurde.

## Alternative – Swipe-Refresh

Für diese App liese sich alternativ (oder in Kombination) auch ein Swipe-Refresh einbauen.



```

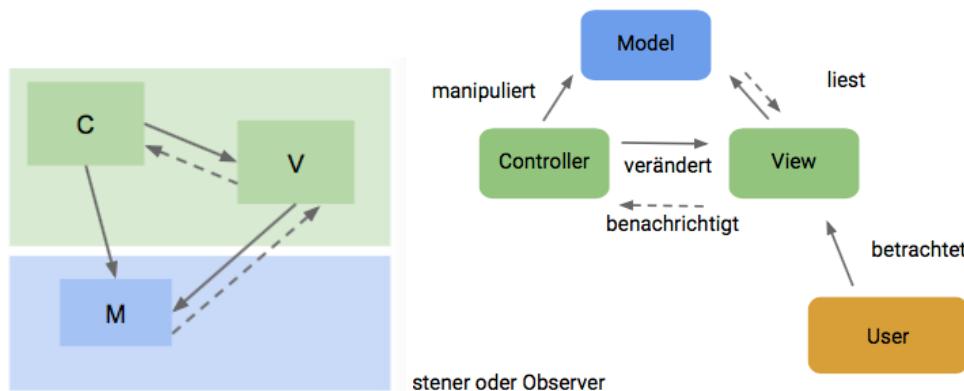
<android.support.v4.widget.SwipeRefreshLayout>
    <android.support.v7.widget.RecyclerView/>
</android.support.v4.widget.SwipeRefreshLayout>
  
```

```

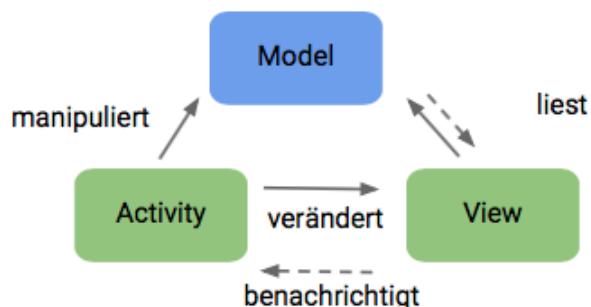
final SwipeRefreshLayout swipeRefreshLayout =
    (SwipeRefreshLayout) findViewById(R.id.swipeRefreshLayout);

swipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
    public void onRefresh() {
        ...
        swipeRefreshLayout.setRefreshing(false);
    }
}); 
```

Es ist ein Pattern für die Organisation der Presentation. Das Model beinhaltet die Daten, die View liest die Daten des Modells und zeigt diese an und der Controller erhält Events der View und manipuliert das Model. Entstanden in den ersten Smalltalk GUIs, heute oft auch Grundlage für die meisten Web-Frameworks. Es gibt diverse Varianten, Umsetzung je nach Framework leicht unterschiedlich. MVP und MVVM sind Verwandte.



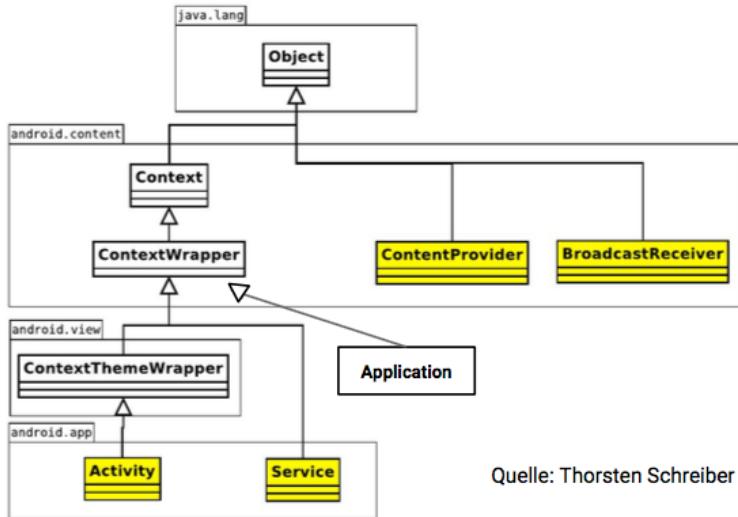
## MVC in Android



Android basiert lose auf dem MVC-Pattern. Was geschieht, wenn wir Controller durch Activity ersetzen?

View liest Model? Ja, wenn wir unter View nicht die View-Klasse, sondern beispielsweise einen ArrayAdapter verstehen.

# Weitere Android-Komponenten



Quelle: Thorsten Schreiber

## Die Context-Klasse

```

public class MyFragment extends Fragment {

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        ...
    }
}
  
```

Mit einer Context-Instantz (also typischerweise die Activity) können wir neue Views erstellen (Kontext muss als Parameter mitgegeben werden), auf System-Services zugreifen, die Applikationsinstanz erhalten, neue Activities starten (mittels Intent), Preferences lesen und schreiben sowie Services starten.

## Services

Für Aufgaben, die im Hintergrund ablaufen sollen oder deren Abarbeitung das UI zu lange blockieren würde. (z.B. Abspielen von Musik, Laden von Daten über das Netzwerk oder rechenintensive Aufgaben).

Services haben kein UI. Sie können gestartet oder gebunden werden. Gearbeitet wird im UI-Thread der Applikation, es sind also keine eigenen Threads. Die Services müssen in der Manifest-Datei deklariert werden.

## Deklaration

Wie Activities müssen auch Services-Komponenten im Manifest deklariert werden.

```

<application>
    <service
        android:name=".ExampleService"
        android:exported="false" />
</application>
  
```

→ Dieser Service steht nur der eigenen App zur Verfügung

## Starten oder Binden

Services unterstützen zwei unterschiedliche Szenarien. Einerseits die Erledigung einmaliger Aufgaben (Download) anderseits Client-Server ähnliche Kommunikation über eine längere Zeitspanne.

Im ersten Fall spricht man von einem gestarteten Service, in letzterem von einem gebundenen Service.

Ein Service wird mit startService(Intent) gestartet. Der Service läuft im Hintergrund und wird nicht gestoppt (solange das System genügend Ressourcen hat), auch wenn der Anwender die App wechselt oder der startende Context zerstört wird. Im Hintergrund meint nicht in einem anderen Thread, sondern dass der Service auch dann weiterläuft, wenn der User die App wechselt!. Der Service sollte sich beenden, wenn die Arbeit getan ist.

<b>AsyncTask</b>	Aufgabe von Main-Thread entkoppeln
<b>Service</b>	Aufgabe von Context enkoppeln

### Starten

Gestartet werden Sie mit:

```
Intent intent = new Intent(this, SimpleService.class);
startService(intent);
```

Es ruft ggf. onCreate() und anschliessend onStartCommand() des Services auf. Per se gibt es keine Kommunikation zwischen Service und Activity.

Service wird im Service gestoppt.

```
stopSelf();
```

Der Service muss irgendwann gestoppt werden! Es könnte auch mit stopService() aus Activity gestoppt werden, aber wann?

### IntentService

Stellt ein Worker-Thread zur Verfügung. Verwendet eine Queue, um Intents zwischenzuspeichern. Intents werden sequentiell in onHandleIntent() abgearbeitet. Er stoppt den Service nachdem alle Intents abgearbeitet werden.

```
public class HelloIntentService extends IntentService {
  public HelloIntentService() {
    super("HelloIntentService");
  }
  @Override
  protected void onHandleIntent(Intent intent) {
  }
}
```

### Resultat?

Wie erhält die Activity ein Resultat vom Service?

- onHandleIntent() erhält keine Referenz auf die aufrufende Activity
- Lösung 1
  - o Service verschickt Intent mit Resultsdaten als Broadcast
  - o Activity stellt einen Broadcast Receiver zur Verfügung, der Intent empfängt.
- Lösung 2
  - o PendingIntent – bei Bedarf im Internet nachlesen

## Bound Services

Ein Service wird mit bindService(Intent, ...) gebunden. Der Service liefert ein Interface, das dem Client erlaubt, Methoden des Service aufzurufen. Es ist möglich, dass mehrere Clients gebunden sind. Wenn alle Clients unbindService(...) aufgerufen haben, wird der Service zerstört.



Client-Server Modell eines Service. Die Clients können Methoden eines Services aufrufen. Die App kann Funktionalität anderen Apps zur Verfügung stellen.

Bei Aufruf von bindService() erhält der Client ein Interface, um mit dem Service zu kommunizieren. Es ist möglich, dass mehrere Clients gleichzeitig gebunden werden.

### Server

```

public class LocalService extends Service {
    private final IBinder binder = new LocalBinder();

    public class LocalBinder extends Binder {
        LocalService getService() {
            return LocalService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    public int getRandomNumber() {
        return new Random().nextInt(100);
    }
}
  
```

### Client

Ein Bound-Service liefert bei onBind() ein Interface vom Typ IBinder- Es ist ein Binder als Rückgabewert vorhanden.

Wenn der Service nur innerhalb der App und im gleichen Prozess arbeitet, sollte ein Binder als Interface geliefert werden. Clients rufen die Methoden des Binders oder des Services auf.

```

Intent intent = new Intent(this, LocalService.class);
bindService(intent, connection, Context.BIND_AUTO_CREATE);
  
```

Der Aufruf von bindService() liefert nicht den onBind() Binder. Der Callback onServiceConnected() wird aufgerufen.

```
private ServiceConnection connection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName className,
                                   IBinder binder) {
        LocalService.LocalBinder myBinder = (LocalService.LocalBinder) binder;
        LocalService myService = myBinder.getService();
        int random = myService.getRandomNumber();
    }

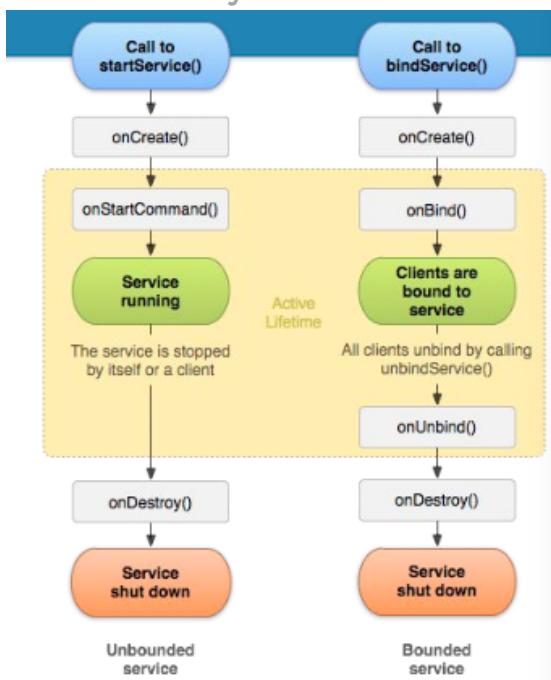
    @Override
    public void onServiceDisconnected(ComponentName name) { }
};

}
```

## Messenger

Ein Messenger kann eingesetzt für prozessübergreifende Services. Der Service erhält einen Handler zur Verfügung, um Meldungen zu erhalten. Der Messenger verarbeitet die Meldungen sequentiell. Für Rückgabewerte des Servers muss der Client seinerseits einen Messenger zur Verfügung stellen.

## Services Lifecycle



### Callbacks ähnlich wie Activity

- `onCreate()`
- `onStartCommand()`
- `onBind()`
- `onUnbind()`
- `onDestroy()`

### Callbacks überschreiben

## Broadcast Receiver

Das System versendet Meldungen per Broadcast. Die Meldungen werden als Intent verschickt. Die Action bestimmt den Ereignistyp. Eine Meldung erhält Informationen über das bestimmte Ereignis (SMS empfangen, System wurde gerebootet, Akku schwach). Die Broadcast Receiver können registriert werden, um bestimmte Meldungen zu erhalten. Die Apps können auch Meldungen per Broadcast verschicken.



Ereignistyp	Beschreibung
ACTION_BOOT_COMPLETED	Boot completed. Requires the RECEIVE_BOOT_COMPLETED permission.
ACTION_POWER_CONNECTED	Power got connected to the device.
ACTION_POWER_DISCONNECTED	Power got disconnected to the device.
ACTION_BATTERY_LOW	Triggered on low battery. Typically used to reduce activities in your app which consume power.
ACTION_BATTERY_OKAY	Battery status good again.

## Registrieren

Die Registration eines Broadcast Receivers erfolgt entweder statisch über die Manifest-Datei oder dynamisch mit context.registerReceiver().

### Beispiel der statischen Registration

```
<receiver android:name=".TimeChangedReceiver">
    <intent-filter>
        <action android:name="android.intent.action.TIME_SET" />
    </intent-filter>
</receiver>
```

### Beispiel der dynamischen Registration (z.B. in onResume)

```
LocalBroadcastManager lbm =
    LocalBroadcastManager.getInstance(getApplicationContext());
IntentFilter filter = new IntentFilter(Intent.ACTION_BOOT_COMPLETED);
MyBroadcastReceiver receiver = new MyBroadcastReceiver(this);
lmb.registerReceiver(receiver, filter);
```

### Beispiel – Receiver abmelden mit in onPause()

```
LocalBroadcastManager lbm =
    LocalBroadcastManager.getInstance(this);
lmb.unregisterReceiver(receiver);
```

## Implementation

Dabei wird von der Klasse BroadcastReceiver abgeleitet und die Methode onReceive() überschrieben. Wenn er statisch registriert wurden, wird der Receiver bei jeder Meldung neu instanziert und ist an keine Activity gebunden.

```
private class MyBroadcastReceiver extends BroadcastReceiver {
    public MyBroadcastReceiver(MainActivity activity) {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
    }
}
```

## Versenden

Mit sendBroadcast(intent) können Broadcast Meldungen versendet werden. Die Action im angegeben Intent definiert den Ereignistyp. Die Namesgebung der Action ist global, bei eigenen Actions Name mit

## Lokal versenden

Meldungen können auch nur im selben Prozess versendet werden. Dabei wird der LocalBroadcastManager eingesetzt. Die Meldungen verlassen den Prozess nicht. Andere Prozesse können unserer App keine Meldung schicken. Effizienter und sicher als globaler Broadcast.

## Content Provider

### Schnittstelle

Der Contentprovider stellt in der Regel Daten zur Verfügung die in einer Datenbank Tabellen abgelegt sind. Die Schnittstelle ähnelt stark der SQL-Syntax. Operationen für query, insert, update und delete von Daten.

### Security

Der Content Provider kann Permissions setzen, um den Zugriff auf die Daten einzuschränken oder zu schützen. Der Client muss die erforderlichen Permissions im Mainfest angeben.

```
<uses-permission android:name="android.permission.READ_USER_DICTIONARY" />
```

### Client

ContentResolver stellt eine Schnittstelle zur Verfügung. Die Tabelle des Content-Provider wird als URI angegeben.

```
content://user_dictionary/words
```

```
Cursor cursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,
    mProjection,
    mSelectionClause,
    mSelectionArgs,
    mSortOrder);

int index = cursor.getColumnIndex(UserDictionary.Words.WORD);
while (cursor.moveToNext()) {
    String word = cursor.getString(index);
}
```

## Server

Der Einsatz ist nur erforderlich, wenn wir anderen Apps strukturierte Daten oder Dateien zur Verfügung stehen sollen oder andere Apps die Möglichkeit haben sollen, Daten zu durchsuchen.

Ein Provider kann Daten als Dateien oder „strukturiert“ anbieten. Dateien werden im Internal Storage abgelegt. Der Client erhält einen Handle, um auf eine Datei zuzugreifen. Strukturierte Daten werden in der Regel in einer Datenbank abgelegt. Der ContentResolver hat gezeigt, dass das Prinzip von Tabellen mit Spalten und Reihen verwendet werden sollte.

Der Content-Provider muss thread-safe programmiert werden.

## Server Implementation

- Festlegen der Art, wie Daten persistent gespeichert werden soll
- Content-URIs festlegen
  - o Der Client verwendet diese URI um zum Beispiel eine Tabelle referenzieren.
- Implementation der Klasse ContentProvider
  - o Die Klasse muss die Methoden query(), insert(), etc. implementieren.
  - o Die Methodensignatur gleich wie beim ContentResolver.
- Implementation einer Contract Klasse
  - o Die Klasse enthält alle Bezeichner (Konstanten), die ein Client benötigt
  - o Die Klasse muss in den Applikationscode des Client kopiert werden
- Festlegen der Permissions in der Manifest-Datei.

## Sensoren

Im Gegensatz zu einem Laptop/Desktop Computer hat ein Smartphone diverse Sensoren. Die Unterstützung ist allerdings von Gerät zu Gerät verschieden. Zudem ist die Qualität der Sensordaten sehr unterschiedlich (z.B. Kompass). Es gibt 3 Kategorien von Sensoren:

- Bewegungssensoren (Beschleunigung) des Smartphones
- Umgebungssensoren wie Temperatur, Beleuchtung
- Lagesensoren für die Bestimmung der Neigung und des Nordpols

## Verfügbarkeit von Sensoren

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
<code>TYPE_ACCELEROMETER</code>	Yes	Yes	Yes	Yes
<code>TYPE_AMBIENT_TEMPERATURE</code>	Yes	n/a	n/a	n/a
<code>TYPE_GRAVITY</code>	Yes	Yes	n/a	n/a
<code>TYPE_GYROSCOPE</code>	Yes	Yes	n/a <sup>1</sup>	n/a <sup>1</sup>
<code>TYPE_LIGHT</code>	Yes	Yes	Yes	Yes
<code>TYPE_LINEAR_ACCELERATION</code>	Yes	Yes	n/a	n/a
<code>TYPE_MAGNETIC_FIELD</code>	Yes	Yes	Yes	Yes
<code>TYPE_ORIENTATION</code>	Yes <sup>2</sup>	Yes <sup>2</sup>	Yes <sup>2</sup>	Yes
<code>TYPE_PRESSURE</code>	Yes	Yes	n/a <sup>1</sup>	n/a <sup>1</sup>
<code>TYPE_PROXIMITY</code>	Yes	Yes	Yes	Yes
<code>TYPE_RELATIVE_HUMIDITY</code>	Yes	n/a	n/a	n/a
<code>TYPE_ROTATION_VECTOR</code>	Yes	Yes	n/a	n/a
<code>TYPE_TEMPERATURE</code>	Yes <sup>2</sup>	Yes	Yes	Yes

## Android Sensor Framework

Für die Arbeit mit einem Sensor brauchen wir:

- Einen SensorManager als Einstiegspunkt für die verfügbaren Sensoren
- Einen Sensor als Repräsentant für einen konkreten Sensor
- Einen SensorEventListener um sich für Updates von Sensordaten zu registrieren
- Einen SensorEvent um die Sensordaten (Werte, Zeitpunkt) auszulesen.

Ein Sensor liefert und dann nur die Rohdaten (float[]), diese sich je nach Sensor unterschiedlich zu interpretieren.

## Minimales Sensoren-Beispiel

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {
    private TextView textView;
    private SensorManager sensorManager;
    private Sensor lightSensor;                                Implementieren für onSensorChanged
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        textView = (TextView) findViewById(R.id.textView);
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        lightSensor = sensorManager.getSensorList(Sensor.TYPE_LIGHT).get(0);      ← Besser: prüfen ob
    }                                                       Sensor vorhanden!
    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(this, lightSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }
    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }
    @Override
    public void onSensorChanged(SensorEvent event) {           Je nach Sensor unterschiedliche Daten
        textView.setText(String.format("Helligkeit: %.0f", event.values[0]));
    }
}
```

# Dependency Injection

Die meisten Klassen sind abhängig von anderen Klassen.

```
public class GadgeothekActivity extends AppCompatActivity {

    LibraryService service = new LibraryService();

    protected void onCreate(Bundle savedInstanceState) {

        service.setServerAddress("http://mge1...");

    }
}
```

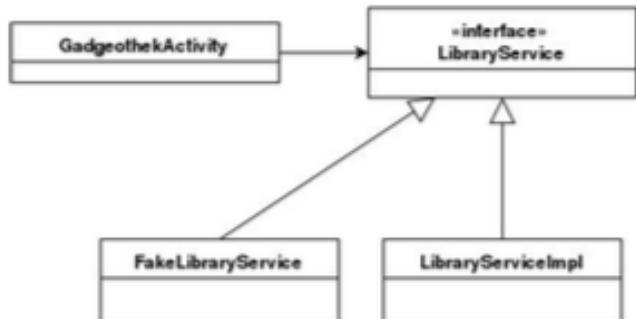
Die Activity instanziert also den LibraryService und konfiguriert die Serveradresse. Ist das eine gute Lösung?

## Dependencies von Klassen

Die Gadgeothe muss die Server-Adresse kennen, diese sollte aber änderbar sein. Die Activity instanziert den Service, es gibt also keine Chance diesen auszutauschen (z.B. um mit einem Fake-Server zu testen).



## Dependencies von Ausserhalb



Eine flexiblere Lösung würde den Service ausserhalb der Activity erstellen und dieser übergeben.

Noch besser wäre es ein Interface von LibraryService zu extrahieren und im Test durch einen Fake-Server zu ersetzen.

Die Klasse erstellt seine Dependencies also nicht selbst, stattdessen werden diese injiziert oder injected. Die Frage stellt sich hier nach der Implementation.

## Dependencies Übergeben

- **Die einfachste Lösung**

Ein Konstruktor mit Parametern und final-Attributen. Dies stellt sicher, dass die Klasse vollständig initialisiert wurde.

- **Die weniger schöne Alternative**

Setter Methoden. Da muss der Benutzer aber aufpassen, dass er diese alle aufrufen.

- **Das Builder-Pattern**

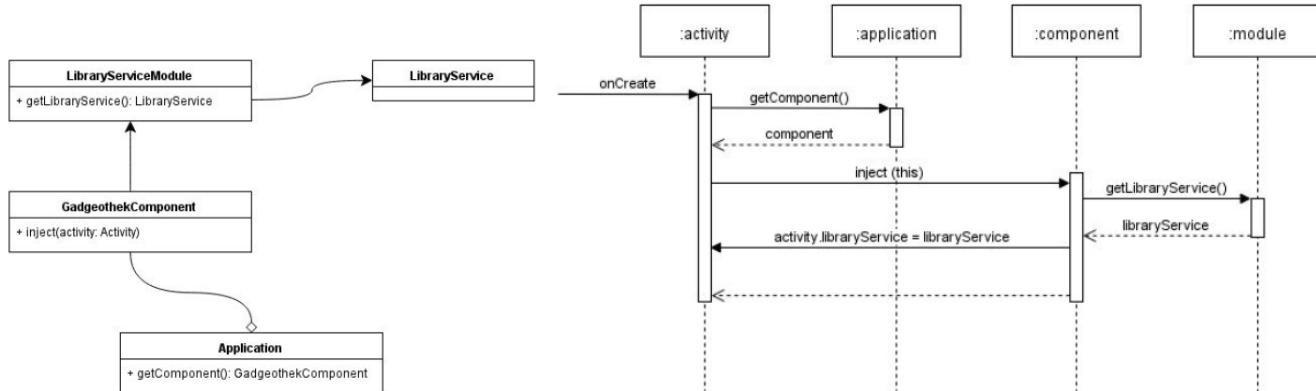
Bleibt nur noch, alle die Konstruktoren, bzw. Setter aufzurufen und so das Objekt zu konstruieren. In Android erstellen wie aber viele zentrale Objekte gar nicht selbst.

## Dependency Injection mit Dagger 2

Ein Dependency-Injection Framework wie Dagger übernimmt das Erstellen des Objektgraphen für uns. Es besteht grundlegend aus drei Teilen. Modul, Komponente und Injection.

Ein **Modul** instanziert unsere Klasse (z.B. LibraryService) die wir injecten wollen. Eine **Komponente** fasst Module zusammen und ist zuständig für die Injection. Eine Klasse lässt sich über die Komponenten ihrer Abhängigkeiten injecten.

### Library Service Injection



### Modul und Komponente

```

@Module
public class LibraryServiceModule {
    @Provides
    @Singleton
    public LibraryService getLibraryService() {
        return new LibraryService();
    }
}
  
```

Instanziert den LibraryService

```

@Singleton
@Component(modules = {LibraryServiceModule.class})
public interface GadgeothekComponent {
    void inject(GadgeothekActivity activity);
}
  
```

Für jede Klasse, in die wir den LibraryService injizieren wollen

### Weitere Implementation

```

public class Application extends android.app.Application {
    GadgeothekComponent component;
    In der Application erstellen wir die Komponente
    public void onCreate() {
        component = DaggerGadgeothekComponent.builder().build();
    }

    public GadgeothekComponent getComponent() { return component; }
}
  
```

```

public class GadgeothekActivity extends AppCompatActivity {
    @Inject
    LibraryService libraryService;
    Mit @Inject annotierte Felder werden von der
    Komponente gesetzt
    @Override
    protected void onCreate(Bundle savedInstanceState) {
    ...
    ((Application) getApplication()).getComponent().inject(this);
}
  
```

## Vorteile

- Keine statischen Methoden mehr
- Zentrale Konfiguration in Modul und Komponente
- Einfache Testbarkeit: Modul oder Applikation austauschen

## Nachteile

- Nicht unbedingt weniger Schreibaufwand bei kleinen Projekten
- Braucht Tests: bei einer Fehlkonfiguration drohen NullPointerException.

## View Injection?

Viel Code in den Activities sieht folgendermassen aus. Kann dies vereinfacht werden?

```
emailView = (EditText) findViewById(R.id.email);
passwordView = (EditText) findViewById(R.id.password);
findViewById(R.id.signInButton).setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        attemptLogin();
    }
});
```

## Butter Knife!

Es ist kein eigentliches Dependency Injection. „A butter knife is like a dagger only infinitely less sharp“. Es verwendet aber ähnliche Techniken für die Implementierung. Dabei findet eine Attribut- und Methodenbindung für Views, Listener und Ressourcen statt.

### Binding

```
class ExampleActivity extends Activity {
    @Bind(R.id.username) EditText username;
    @Bind(R.id.password) EditText password;

    @BindString(R.string.login_error)
    String loginErrorMessage;

    @OnClick(R.id.submit)
    void submit() {
        ...
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.simple_activity);
        ButterKnife.bind(this);
        ...
    }
}
```

## Data Binding

Das View Bindung erspart uns schon eine gewisse Tipparbeit. Noch toller wäre es, im XML direkt auf Objekte zugreifen zu können. Die onClick-Angabe direkt im Layout kennen wir ja bereits.

Optimal wäre natürlich, dass sich das GUI selbst aktualisiert, sobald sich Objekte ändern. Sozusagen das XML-Layout als Observer Pattern.

## Data Binding Grundlagen

```
public class User {
    public String firstName;
    public String lastName;

    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}

<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="user" type="ch.hsr.mge.databindingdemo.User"/>
    </data>
    <RelativeLayout
        ... >
        <TextView android:text="@{user.firstName}" ... />
        <TextView android:text="@{user.lastName}" ... />
    </RelativeLayout>
</layout>
```

Parameterdeklaration für Layout

Java-ähnliche Expression Language

Die Expression Language unterstützt fast alle Java Expressions, aber keine Statements (Deklarationen, Loops, kein new, this oder super).

```
        android:text="@{String.valueOf(index + 1)}"
        android:visibility="@{age < 13 ? View.GONE : View.VISIBLE}"
        android:transitionName='@{"image_" + id}'"

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityMainBinding binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
        ----- Generierte Klasse anhand des Layouts -----
        User user = new User("Mirko", "Stocker");
        binding.setUser(user);
    }
}
```

## Binding von Listenern

Neben den Properties können auch Events gebunden werden. Natürlich sind nicht nur onClick möglich, sondern auch z.B. TextWatcher.

```
<Button
    android:text="Save"
    android:onClick="@{controller.onButtonSaveClicked}" />

<EditText
    android:text="@{user.lastName}"
    android:addOnTextChangedListener="@{user.lastNameWatcher}" />
```

## Observables

Was geschieht aber, wenn die Daten zum Beispiel mit einem TextWatcher aktualisiert werden? Um die Daten aus gebundenen Objekten in der View zu aktualisieren benötigen wir wieder das Observer-Pattern. Die View observiert das gebundene Objekt und das Objekt ist das Observable.

Um unser Objekt observable zu machen müssen wir observable Fields verwenden.

```
public class User {
    public ObservableField<String> firstName = new ObservableField<>();
    public ObservableField<String> lastName = new ObservableField<>();

    public TextWatcher lastNameWatcher = new TextWatcher() {
        ...
        if (!Objects.equals(lastName.get(), s.toString())) {
            lastName.set(s.toString());
        }
    };
}
```

↳ löst Update, etc. aus

## Nachteile von Data Binding

- Aufpassen, dass nicht zu viel Logik ins XML wandert
- Android generiert noch mehr zusätzlichen Code
- Code ist schwieriger zu lesen und zu debuggen
- Data Binding ist aktuell eine Extra Library und ist nicht der Default.

## Java 8

„Android supports all Java 7 language features and a subset of Java 8 language features that vary by platform version“. - <https://developer.android.com/guide/platform/j8-jack.html> → Wer ist Jack ?

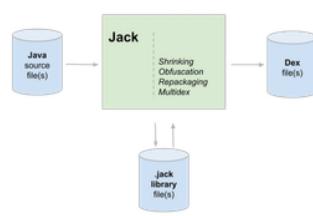
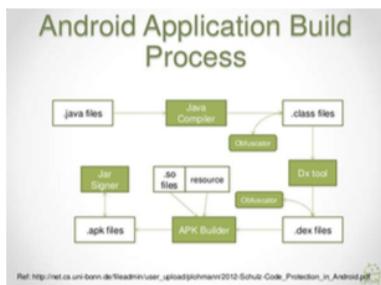
### Java Android Compiler Kit (JACK)

Bisherige Toolchain (Dalvik):

.java -> .class -> .dex

■ Jack Compiler:

.java -> .dex



## Unterstützte Features

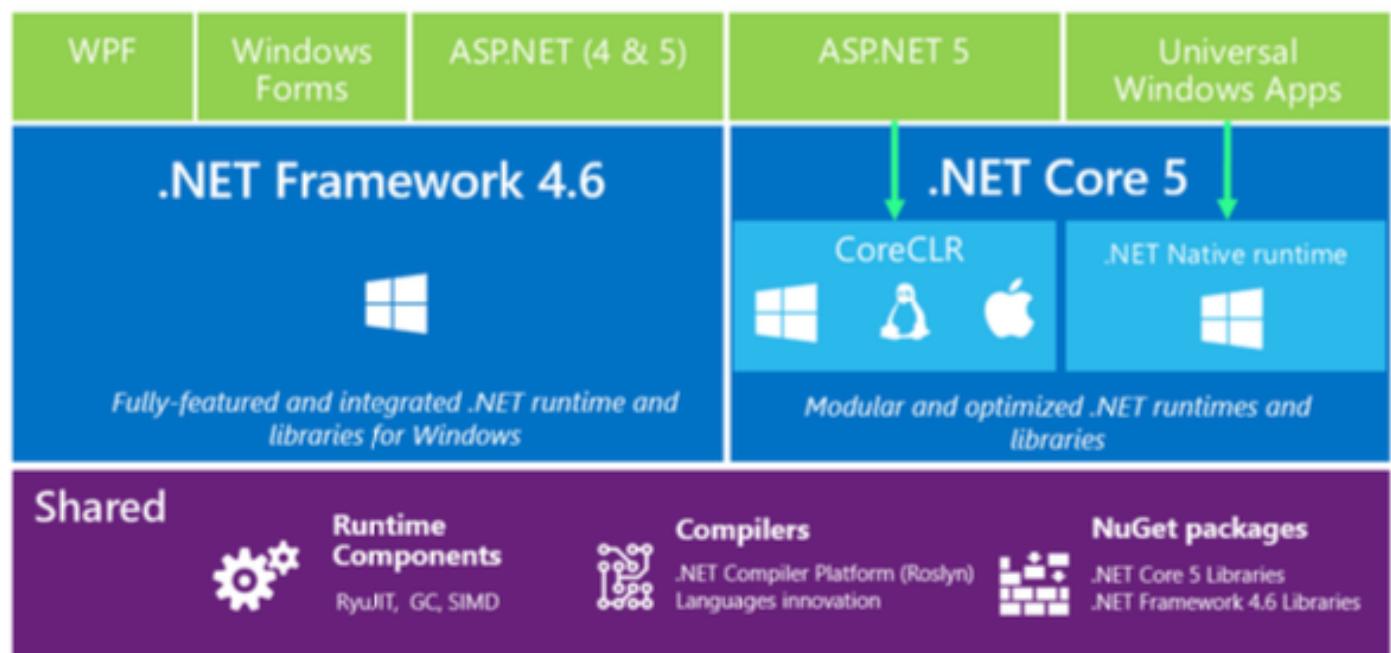
Folgende Features werden mit API Level 24 unterstützt:

- Default and static interface methods
- Lambda expressions (also available on API level 23 and lower)
- Repeatable annotations
- Method References (also available on API level 23 and lower)
- Type Annotations (also available on API level 23 and lower)

Aber Instant Run funktioniert aktuell mit Jack nicht und wird deaktiviert, wenn Jack eingesetzt wird.



# Windows Presentation Foundation



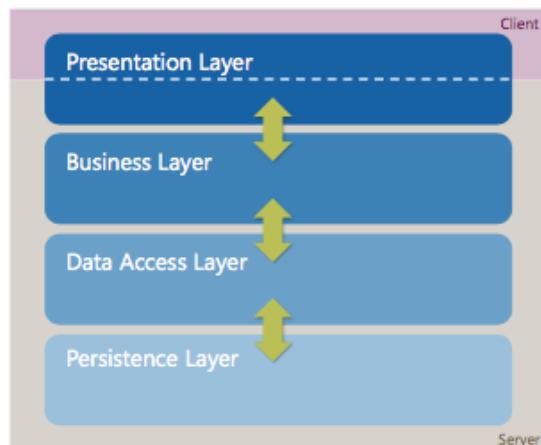
# Einführung WPF & XAML

## Desktop Apps – Technologie von gestern?

Im Jahre 2012 hat man noch vorausgesagt, dass die PC bis 2017 nur noch rund ein viertel ausmachen würden im Vergleich mit Tablets. Statistiken von verkauften Geräten zeigen das die Verkäufe von Smartphones massiv gestiegen sind, dass aber die Verkäufe von Tablets wieder stagnieren.

Wenn man es kumuliert anschaut, zeigt sich sogar dass nur noch 1 Milliarde PC vorhanden sind, auf welchen Microsoft Windows läuft.

## Typische Applikations-Architektur: N-Tier



### Presentation Layer

Mobile UI, Web UI, Desktop UI, Embedded UI

### Business Layer

Business Objects & Rules (Reiner Code mit vielen Tests)

### Data Access Layer

Data Objects & Rules (Wie speichere ich es ab?)

### Persistence Layer

SQL, No-SQL, XML, Text, Binary (Speichern von Bits)

### Desktop App



#### Pro

- Auch Komplexe UIs problemlos darstellbar
- Höhere Rechenkapazität, Performance der Hardware
- Reduzierte Komplexität (kein HTTP nötig)
- Line-Of-Business Apps sind (noch) oft kostengünstiger zu realisieren
- Viel breiterer Technologiemix nutzbar



#### Contra

- Benötigt Desktop-PC oder Notebook
- Platz auf Screen verleiht zu überladenen UIs
- Speicherhierarchien laden zur Verschwendungen ein → «Featuritis»

### Mobile/Tablet App



#### Pro

- Immer dabei («Ubiquitous Computing»)
- Einfach bedienbar, da UI stark vereinfacht
- «Gut genug» Ansatz
- Sehr viele Fehlerquellen und Ansatzpunkte für Sicherheitsprobleme



#### Contra

- Komplexe UIs wegen beschränktem Platzangebot nur bedingt darstellbar
- Große Datenmengen nur bedingt beherrschbar
- Abhängigkeit von der Netzwerkinfrastruktur (Mobiles Netz, Internet)

## WPF Quick Intro

### Grundprinzip

Trennung von Darstellung und Code (ähnlich Android).

XAML als Markup für Design, Elemente und Layout und der „Code Behind“ als Programmcode.

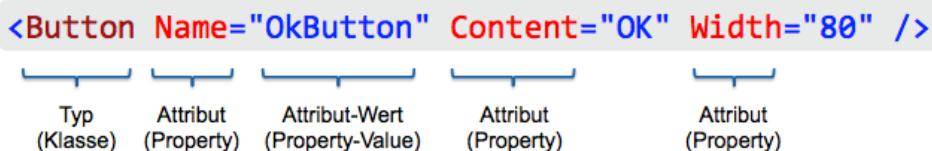
### XAML

XML-basierende Beschreibungssprache für WPF UIs. XAML ist eine sehr mächtige Form der Objekt-Serialisierung. Dabei ist zu bemerken, dass das System für diese Applikation rund 18 MB verbraucht. Dies, da WPF alles selber zeichnet.



## XAML → CLR

XAML-Elemente werden 1:1 zu erzeugten CLR-Objekten zur Laufzeit (Common Language Runtime). Die XML-Attribute werden 1:1 zu Properties und Events der erzeugten CLR-Objekte.



## XAML vs. C#

Grundsätzlich kann alles, was in XAML implementiert werden kann, auch in Code (C#) ausgedrückt werden. So würde der Code für obiges Beispiel aussehen.

### In C#:

```

var btn = new Button();
btn.Name = "OkButton";
btn.Content = "OK";
btn.Width = 80;
  
```

oder

### In C#: (mit Initializer-Syntax)

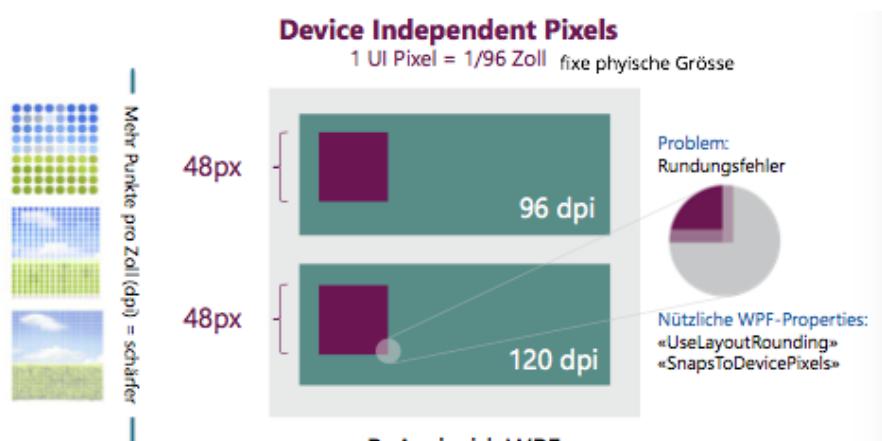
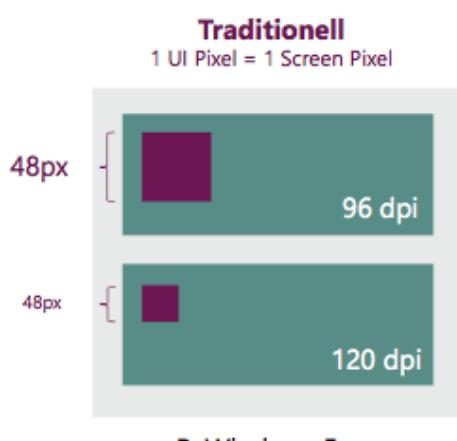
```

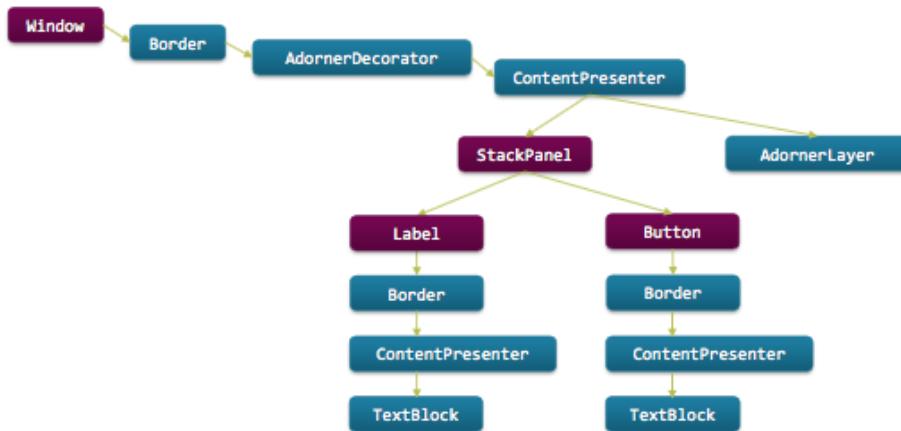
var btn = new Button
{
    Name = "OkButton",
    Content = "OK",
    Width = 80
};
```

Die XAML-Baumstruktur ist wie gesehen auch gleichwertig über C# definierbar. Der XAML-Code ist aber oft leichtgewichtiger bzw. kürzer. Zudem ist XAML auch via Designer definierbar.

## Device Independent Pixels/Units

Ähnliche Implementierung wie unter Android. Nur ist das Ratio zwischen einem UI Pixel und dem Screen Pixel fix auf 1/96 Zoll festgelegt.





### Logical Tree (dunkel)

Entspricht der Struktur der XAML-Elemente. Beschreibt die Beziehungen zwischen den verschiedenen Elementen des UIs. Zuständig für:

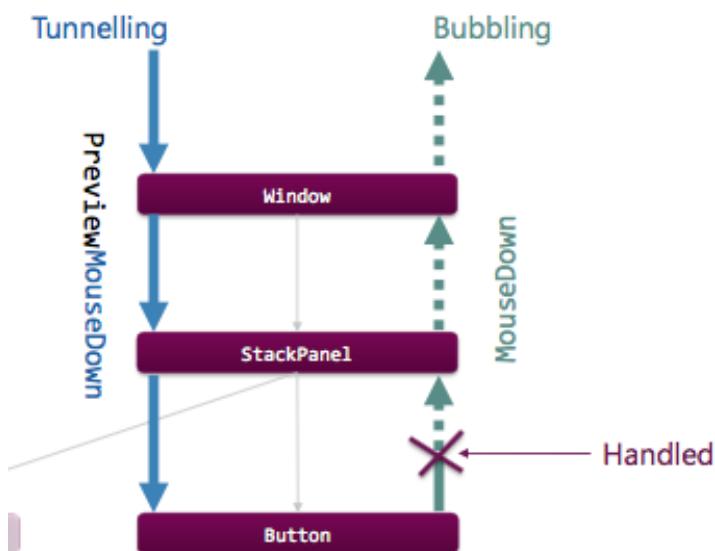
- Dependency Properties erben
- Dynamische Ressourcen-Referenzen auflösen
- Elementnamen für Datenbindung nachschlagen
- Routed Events weiterleiten

### Visual Tree (hell)

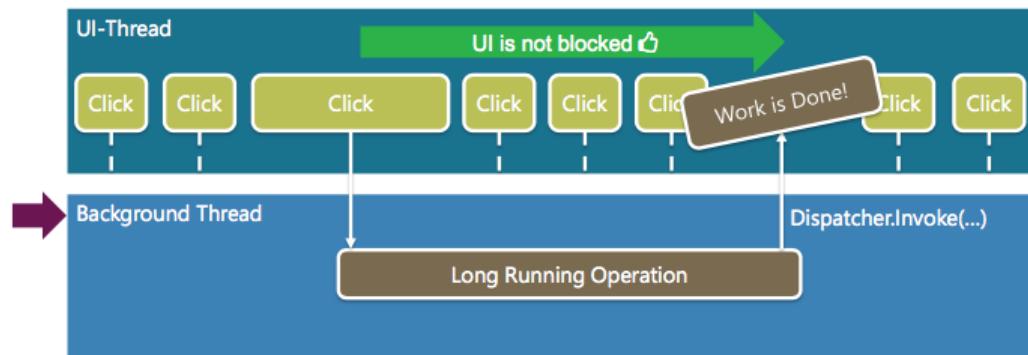
Entspricht der grafischen Repräsentation. Er beinhaltet alle dargestellten Elemente gemäss der Vorlage jedes Controls. Zuständig für:

- Visuelle Darstellung
- Vererben der Transparenzeinstellungen
- Vererben von Transinformationen
- Vererben der IsEnabled-Property
- Hit-Testing

### „Route Events“



Hier gilt das gleiche wie in Android. Alles was im UI Thread ist blockiert die Bedienung. Daher sollten länger andauernde Operationen ausgelagert werden.



## XAML

„Attribute Syntax“ vs. „Property Element Syntax“



```
<Button
    Height="50"
    Width="200"
    Content="Watch Now"
/>
```

Attribute Syntax

```
<Button
    Width="120"
    Height="50">
    <Button.Content>Watch Now</Button.Content>
</Button>
```

Property Element Syntax  
(Property Syntax)

Die Property Element Syntax erlaubt es auch komplexere Sachen innerhalb eines Buttons darzustellen und nicht nur einfach ein Text.

**Syntax** » Zusammengesetzter Inhalt

```
<Button Width="120" Height="50">
    <Button.Content>
        <StackPanel>
            <TextBlock Text="Watch Now" FontSize="20" />
            <TextBlock Text="Duration: 50m"
                FontSize="12" Foreground="#888888"
            />
        </StackPanel>
    </Button.Content>
</Button>
```

## Weitere XAML Konzepte

Folgen in weiteren Kapitel der Zusammenfassung in detaillierter Form.

## Data Binding mit „Dependency Properties“ und INotifyPropertyChanged

Bidirektionale Datenbindung an Eigenschaften von UI-Elementen und Model-Klassen (MVVM). Eine vollautomatische Aktualisierung von UI und/oder Model-Klassen bei Änderungen.

## „Attached Properties“

Das Element setzt Eigenschaften, die das Parent-Element betreffen. Mit statischen Setter-/Getter-Methoden in der Klasse des Parent-Elements gelöst.

```
DockPanel.Dock="Top"
steht für (C#)
DockPanel.SetDock(element, Dock.Top)
```

## „Markup extensions“

Verkürzte Notation um komplexe Ausdrücke unter Verwendung der „Attribute Syntax“ eingeben zu können.

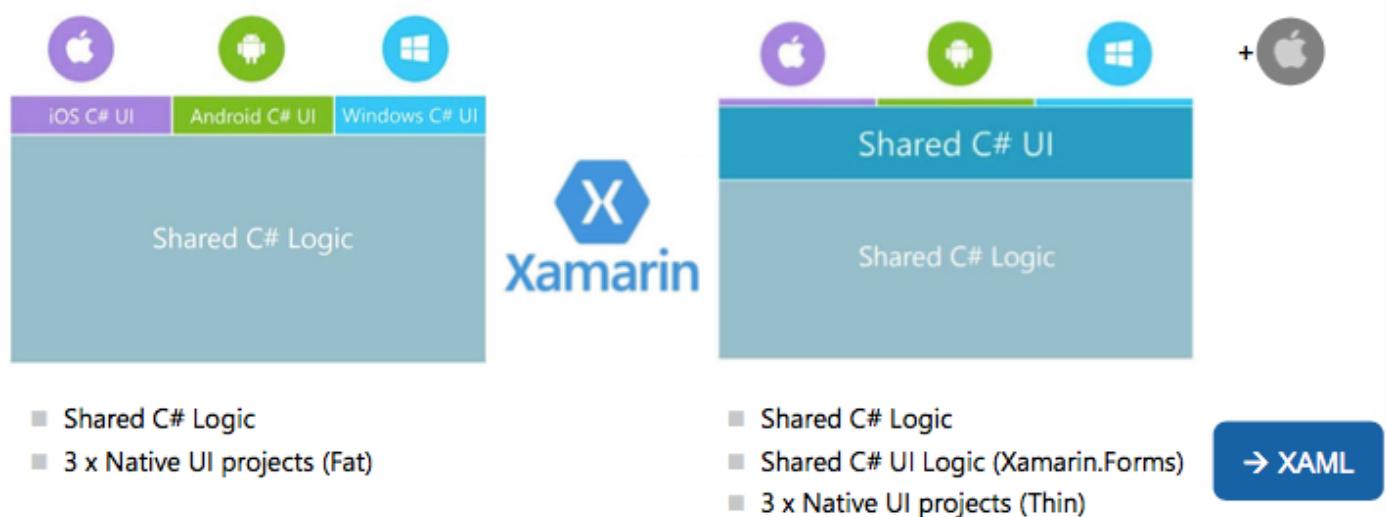
```
<TextBox Text="{Binding Path=FirstName}" />
als Kurzform für:
<TextBox><TextBox.Text><Binding Path="FirstName" /></TextBox.Text></TextBox>
```

## „Type Converts“

Automatisches Konvertieren von (String-)Werten in einen passenden Datentyp.

```
<Button Background="Aqua" />
als Kurzform für:
<Button><Button.Background><SolidColorBrush Color="Aqua" /></Button.Background></Button>
```

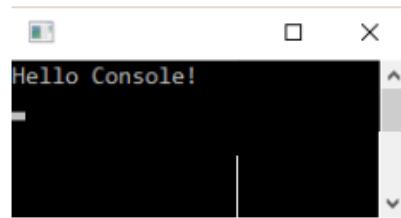
## Beyond XAML



## C# vs. Java

Es ist eine OO-Programmiersprache, welche ähnlich wie Java ist. Ein Cheatsheet für Java-Entwickler ist unter <http://download.microsoft.com/download/D/E/E/DEE91FC0-7AA9-4F6E-9FFA-8658AA0FA080/CSharp%20for%20Java%20Developers%20-%20Cheat%20Sheet.pdf> zu finden.

```
namespace HelloConsole
{
    public class Program
    {
        public static void Main(string[] args)
        {
            System.Console.WriteLine("Hello Console!");
            System.Console.ReadKey();
        }
    }
}
```



Die Änderungen/Unterschiede sind mit Pfeilen markiert.

#### → MORE C# Keywords

Java	C#
native	extern
new	new
null	null
package	namespace
private	private
protected	internal
public	public
return	return
static	static

→	super	base	←
→	switch	switch	←
→	synchronized	lock	←
→	this	this	←
→	throw	throw	←
→	throws	n/a	←
→	transient	[Nonserialized] (attribute)	←
→	true	true	←
→	try	try	←
→	... (varargs)	params	←
→	void	void	←
→	volatile	volatile	←
→	while	while	←

Java	C#
abstract	abstract
assert	Debug.Assert (method)
break	break
case	case
catch	catch
class	class
const	const
continue	continue
default	default
do	do

→	else	else	←
→	enum	enum	←
→	extends	:	←
→	false	false	←
→	final	sealed	←
→	finally	finally	←
→	for	for / foreach	←
	goto	goto	
	if	if	
→	implements	:	←
→	import	using	←
	instanceof	is	
	interface	interface	

→	event	Declares an event	←
	new	Declares a method that hides a method in a base class (method modifier)	

→	operator	Declares an overloaded operator	←
	out	Declares an output parameter (method call)	
→	override	Declares a method that overrides a method in a base class	←

→	readonly	Declares a field to be read- only	←
	ref	Declares a reference to a value type	
	struct	Defines a new value type	
	virtual	Declares a member that can be overridden	

Java	C#	Java Range	C# Range
boolean	bool	true or false	true/false
byte	sbyte	-128 to 127	-128 to 127
char	char	0 to $2^{16}-1$	0 to $2^{16}-1$
double	double	$\pm 5.0 \times 10^{-124}$ to $\pm 1.7 \times 10^{38}$	$\pm 5.0 \times 10^{-124}$ to $\pm 1.7 \times 10^{38}$
float	float	$\pm 1.5 \times 10^{-4} to \pm 3.4 \times 10^9$	$\pm 1.5 \times 10^{-4} to \pm 3.4 \times 10^9$
int	int	$-2^{31}$ to $2^{31}-1$	$-2^{31}$ to $2^{31}-1$
long	long	$-2^{63}$ to $2^{63}-1$	$-2^{63}$ to $2^{63}-1$
short	short	$-2^{15}$ to $2^{15}-1$	$-2^{15}$ to $2^{15}-1$
String	string	n/a	n/a
Object	object	n/a	n/a
Date	DateTime	1st Jan 1970 to <i>implementation dependent</i> value	1st Jan 0001 to 31st Dec 9999

## C# Features

\* = jene, welche nicht in Java verfügbar sind (Der Vergleich bezieht sich auf Java 7 (Android vor N), in Java 8/9 sind dann Features wie z.B. Delegates und Lambdas verfügbar.)

Hell = Gründe, warum viele C# Programmierer sich nicht mit (Android-)Java herummühen wollen.

Dunkel = Gründe, warum ICH als Dozent mich in der Regel nicht mit Java herummühen will.

- Delegates\*
- Events
- Properties
- Indexers
- Lambda Expressions\*
- Partial Classes
- Structures (Value Types)
- Operator Overloading
- Optional Parameters
- LINQ
- Extension Methods
- Anonymous Types
- Expression Trees
- Runtime Generics
- IDisposable & the Using Statement

## „Properties“

```
public class Person
{
    // backing field for property LastName
    private string _lastName;

    // property using explicit getter and setter
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }

    // auto-implemented property (with automatically allocated backing field)
    public string FirstName { get; set; }

    ...
}
```

```
...
// getter/setter body can be anything (e.g. Read-Only computed property)
public string FullName
{
    get
    {
        var fullName = LastName + " " + FirstName
        return fullName;
    }
}

// Lambda-syntax (C# >=6.0)
public string FullNameFirstLast => FirstName + " " + LastName;
}
```

Verfügen über ein Getter/Setter-Paar als natives Sprachefeature. Kontrollierter Zugang zu Eigenschaften eines Objektes. Zudem sind Properties debuggable. Dabei ist aber zu beachten, dass es zu Seiteneffekten kommen kann. Die Properties ermöglichen folgendes:

- Lazy Initialization
- Change Tracking → INotifyPropertyChanged
- Calculated Properties
- Read-Only Properties
- Public-Getter / Private-Setter

## „Fields“

Sind einfache Klassenvariablen. Der Zugriff ist ungeschützt. Werden deshalb in der Regel nur für kontrollierbare Zugriffe innerhalb einer Klasse benutzt. Dazu muss die Sichtbarkeit private oder protected benutzt werden.

## „Delegates“

- 1 Spezieller Datentyp für Funktionssignaturen
- 2 Ermöglichen Funktionsvariablen (Funktionszeiger)

```
namespace DelegateExample
{
    class Program
    {
        delegate int Calculation(int a, int b); ①

        static void Main(string[] args)
        {
            int x = 2;
            int y = 3;
            Calculation add = delegate(int a, int b) { return a + b; }; ②
            int answer = add(x, y);
            System.Console.WriteLine(answer); // output: 5
        }
    }
}
```

### Action<X,Y,Z>

Methode, die einen Wert vom Typ X, Y und Z als Parameter akzeptiert.

```
public delegate void Action();
public delegate void Action<T>(T t);
public delegate void Action<T, U>(T t, U u);
public delegate void Action<T, U, V>(T t, U u, V v);
```

### Func <X,Y,Z>

Methode, die Werte vom Typ X und Y als Parameter akzeptiert und einen Wert vom Typ Z zurückgibt.

```
public delegate TResult Func<TResult>();
public delegate TResult Func<T, TResult>(T t);
public delegate TResult Func<T, U, TResult>(T t, U u);
public delegate TResult Func<T, U, V, TResult>(T t, U u, V v);
public delegate TResult Func<T, U, V, W, TResult>(T t, U u, V v, W w);
```

## „Lambda Expressions“

Kurzform für die Definition von anonymen Delegates. Ermöglicht „Closures“ (Zugriff auf Variablen im Umgebungskontext).

```
namespace LambdaExample
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 2;
            int y = 3;
            Func<int, int, int> add = (a, b) => a + b;
            int answer = add(x, y);
            System.Console.WriteLine(answer); // output: 5
        }
    }
}

// Use delegate method expression.
Func<int, int> func1 = delegate (int x) { return x + 1; };
// Use delegate expression with no parameter list.
Action func2 = delegate { Console.WriteLine(); };

// Use implicitly typed lambda expression.
Func<int, int> func3 = x => x + 1;
// Use lambda expression with statement body.
Func<int, int> func4 = x => { return x + 1; };
// Use formal parameters with expression body.
Func<int, int> func5 = (int x) => x + 1;
// Use parameters with a statement body.
Func<int, int> func6 = (int x) => { return x + 1; };
// Use multiple parameters.
Func<int, int, int> func7 = (x, y) => x * y;
// Use no parameters in a lambda expression.
Action func8 = () => Console.WriteLine();
```

Anwendung des Observer-Patterns (Publish/Subscribe) als natives Sprach-Feature in C#. Zudem gibt es speziell markierte Multicast Delegates. Es ist von vielen Interessenten abonnierbar (Subscribe), aber immer nur innerhalb der Klasse auslösbar (Publish).

```
public class Person
{
    // the event declaration
    public event EventHandler<PersonNameChangedEventArgs> NameChanged;

    // the method which encapsulates raising the event
    protected virtual void OnNameChanged(PersonNameChangedEventArgs e)
    {
        if (NameChanged != null) NameChanged(this, e);
    }
    ...
}

...
// the backing field for the Name property
private string _name;

// the tracked property
public string Name
{
    get { return _name; }
    set
    {
        if (_name == value) return;
        OnNameChanged(new PersonNameChangedEventArgs(_name, value));
        _name = value;
    }
}
```

## Subscribe

```
obj.EventName += ...some delegate...;
```

## Beispiel:

```
// create a new person object
var max = new Person() { Name = "Max" };

// add an anonymous event handler to the NameChanged multicast delegate - Lambdas! :-)
max.NameChanged += (o, a) => Console.WriteLine($"{a.OldName} changed name to {a.NewName}");

// see if this works:
max.Name = "Moritz"; // --> writes the string "Max changed name to Moritz" to the console
```

## Formatierte Strings

C# <= 5.0

```
string.Format("{0} changed name to {1}", a.OldName, a.NewName)
```

## C# ermöglicht neu String Interpolation

vergleichbar mit PHP, Perl, etc. Aber Interpolated Strings sind lediglich Syntactic Sugar und werden zu Compile Time ausgewertet.

```
 $"{a.OldName} changed name to {a.NewName}"
```

## EventHandler<T>

In vielen Fällen wird der generische Delegate EventHandler aus dem .NET Framework verwendet.

```
public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e)
    where TEventArgs : System.EventArgs;
```

Statische Methoden, welche bestehenden Klassen beigemischt werden können. Ermöglichen die Erweiterung des Funktionsumfangs einer Klasse (auch ohne den Source Code zu haben). Der Aufruf ist wie bei Methoden der Klassen (Member) → **Achtung:** Namespace importieren.

```
public static class FileExtensions
{
    public static string ToFileSize(this long size)
    {
        var units = new[] { "B", "KB", "MB", "GB", "TB" };
        var pow = 0;
        while (size >= 1024 && pow < units.Length)
        {
            size /= 1024;
            pow++;
        }
        return $"{size} {units[pow]}";
    }
}
```

```
[TestClass]
public class FileExtensionsTests
{
    [TestMethod]
    public void ToFileSizeTest()
    {
        Assert.AreEqual("0 B", 0L.ToFileSize());
        Assert.AreEqual("1 B", 1L.ToFileSize());
        Assert.AreEqual("567 B", 567L.ToFileSize());
        Assert.AreEqual("1 KB", 1025L.ToFileSize());
        Assert.AreEqual("101 KB", 103424L.ToFileSize());
        Assert.AreEqual("1 MB", 1048576L.ToFileSize());
    }
}
```

## „Linq“ – Language Integrated Query

Sammlung von „Extension Methods“ zur Abfrage und Manipulation von Daten. Es besteht aus einem einheitlichen API für alle möglichen Formen von Daten: Collections, SQL, XML. Dazu nutzt es „Expression Trees“ → Zugriff auf den Abstract Syntax Tree (AST) eines Lambda-Ausdrucks zur Laufzeit. Eine alternative Syntax, Query Syntax, ist verfügbar.

### Query Syntax

```
var q = from s in students
        where s.Grade < 4m
        select s.FirstName + " " + s.LastName;
```

### Method Syntax

```
var q = students
        .Where(s => s.Grade < 4m)
        .Select(s => s.FirstName + " " + s.LastName);
```

Empfehlung:

## Query Syntax

```
// get some sample students
var students = GetSampleStudents();

// build linq expression tree using query syntax:
var q = from s in students
        where s.Grade < 4m
        select s.FirstName + " " + s.LastName;

// enumerate
foreach (var i in q)
{
    Console.WriteLine(i);
}
```

```
// get some sample students
var students = GetSampleStudents();

// build linq expression tree using method syntax:
var q = students
    .Where(s => s.Grade < 4m)
    .Select(s => s.FirstName + " " + s.LastName);

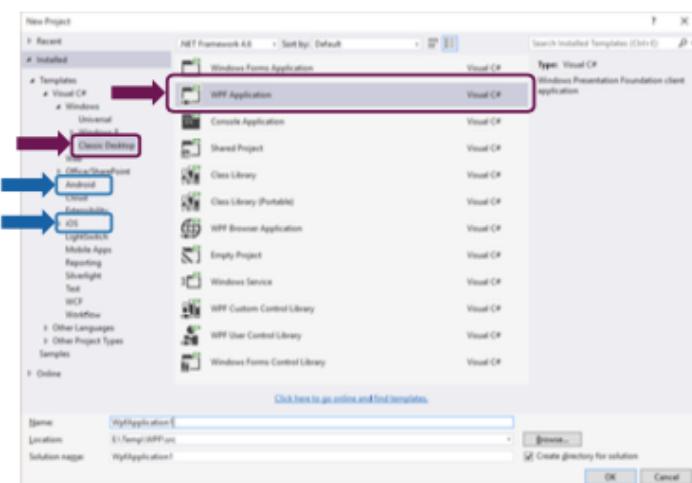
// cache to memory (optional)
var failedOnes = q.ToList();

// enumerate (from memory)
foreach (var i in failedOnes)
{
    Console.WriteLine(i);
}
```

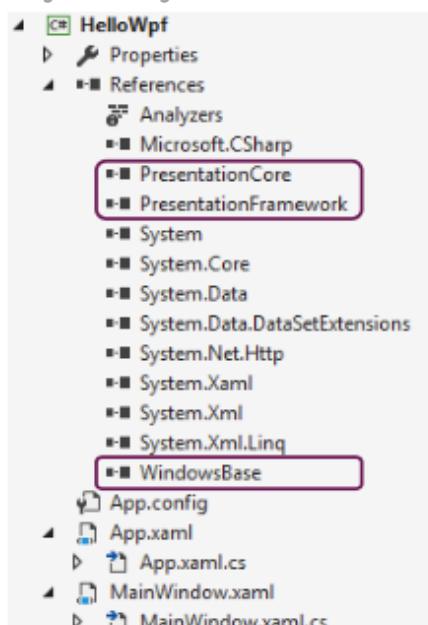
## Visual Studio

### Neues WPF – Projekt

Wird über File | New | Project erstellt.



## Projekt-Layout



### App.config

Config-File (XML) mit App-Einstellungen, Connection Strings, etc.

### App.xaml

Markup der Startup-Klasse

### App.xaml.cs

Code-Behind der Startup-Klasse

### MainWindow.xaml

Markup des Hauptfensters

### MainWindow.xaml.cs

Code-Behind des Hauptfensters

## Mobile and GUI Engineering

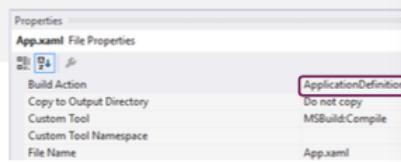
### App.config – Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="upload.folder" value="E:/test/daupload/files"/>
    <add key="costunit" value="1030"/>
    <add key="organisationid" value="1"/>
    <add key="periodid" value="3"/>
  </appSettings>
  <connectionStrings>
    <add name="contents"
      connectionString="server=localhost;user id=hw77gx6;password=...;database=db1;" providerName=" MySql.Data.MySqlClient" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```



### App.xaml – Beispiel

```
<Application x:Class="HelloWpf.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:HelloWpf"
  StartupUri="MainWindow.xaml">
  <Application.Resources>
  </Application.Resources>
</Application>
```



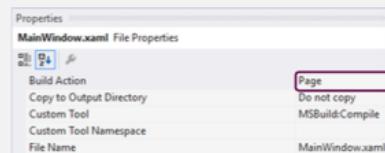
### App.xaml.cs – Beispiel

```
using System.Windows;

namespace HelloWpf
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        // your logic here
    }
}
```

### MainWindow.xaml – Beispiel

```
<Window x:Class="HelloWpf.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:HelloWpf"
  mc:Ignorable="d"
  Title="WPF Quick Intro" Height="350" Width="525">
  <Grid>
    <Label x:Name="HelloWorldLabel"
      Content="Hello WPF!" HorizontalAlignment="Center" VerticalAlignment="Center" />
  </Grid>
</Window>
```



```
using System.Windows;

namespace HelloWpf
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

# GUI-Entwurf mit WPF

## Intro

### App & Window

Viele vordefinierte Properties, Methoden und Events. Einfache Nutzung von SplashScreens. Viele Details dazu sind im Appendix zu Block 2 vorhanden.

### WPF Controls Overview

Einige Controls wie z.B. Line, Path, Polygon, etc. sind standardmäßig nicht sichtbar.

 Pointer	 Ellipse	 PasswordBox	 TextBlock
 Border	 Expander	 ProgressBar	 TextBox
 Button	 Frame	 RadioButton	 ToolBar
 Calendar	 Grid	 Rectangle	 ToolBarPanel
 Canvas	 GridSplitter	 RichTextBox	 ToolBarTray
 CheckBox	 GroupBox	 ScrollBar	 TreeView
 ComboBox	 Image	 ScrollViewer	 Viewbox
 ContentControl	 Label	 Separator	 WebBrowser
 DataGrid	 ListBox	 Slider	 WindowsFormsHost
 DatePicker	 ListView	 StackPanel	 WrapPanel
 DockPanel	 MediaElement	 StatusBar	
 DocumentViewer	 Menu	 TabControl	

### System.Windows.FrameworkElement

#### Größenangaben

Width/ Height-Attribut stammt von der Klasse System.Windows.FrameworkElement. Fixe Größenangaben werden mit Double angegeben. Dabei ist das Device Independent Pixel (1px = 1/96") zu beachten. Auto wäre dabei = Double.NaN, was eine automatische Größe wäre.

#### Qualifizierte Größenangabe

...px	= Device Independent Pixels	1px = 1/96"
...in	= Inches (Zoll)	1in == 96px
...cm	= cm	1cm == (96/2.54)px
...pt	= Points	1pt == 1/72" == (96/72)px

Zusätzlich sind MinWidth/MaxWidth zur Angabe der minimalen/maximalen Größe verfügbar. Priorität bei Platzproblem: 1. MinWidth, 2. MaxWidth, 3. Width. Das ReadOnly Property ActualWidth zur Laufzeit auch verfügbar um die tatsächliche Breite abzurufen.

#### Ausrichtung

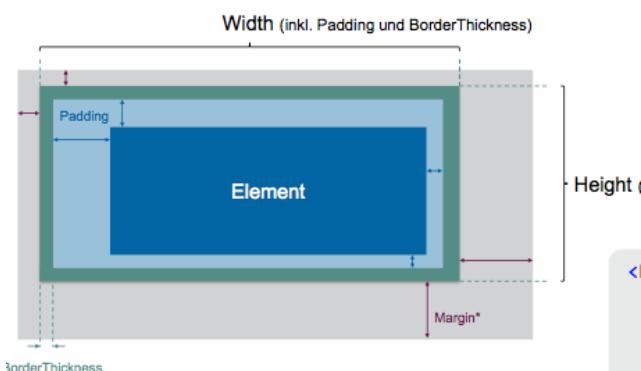
Horizontal- und VerticalAlignment ist verfügbar auf dem Element. Das Bild zeigt das Verhalten der verschiedenen Eigenschaften.



## Ränder und Rahmen

Margin = Aussenabstand  
 Padding = Innenabstand  
 BorderThickness = Rahmenstärke  
 CornerRadius = Radius für abgerundete Ecken

Angaben müssen entweder mit l,t,r,b; mit l,t (Links Und Rechts sowie Oben und Unten) oder mit x (für alles) erfolgen.



## System.Windows.Controls.Control

### Sonstige Eigenschaften UIElement

- IsEnabled
- SnapsToDevicePixel (Runden Angaben)
- Visibility

### Sonstige Eigenschaften FrameworkElement

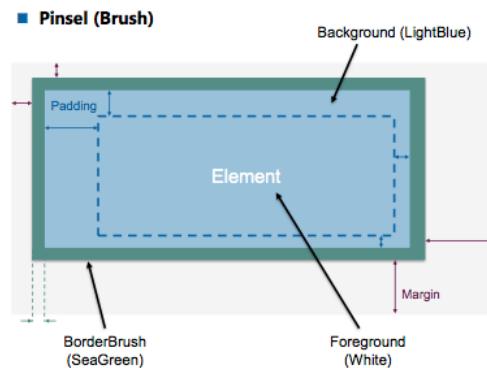
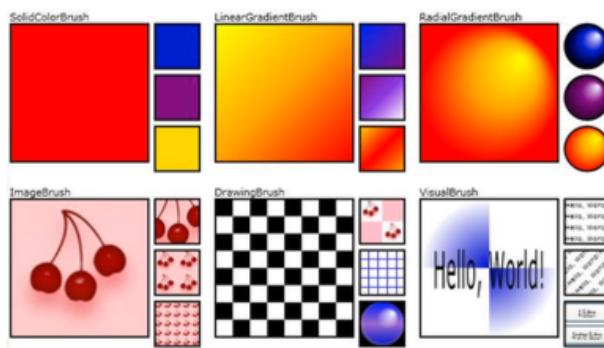
- Name
- Ressource
- Tag
- Tooltip
- UseLayoutRounding

### Sonstige Eigenschaften Control

- Background (Farbe)
- BorderBrush (Farbe)
- Foreground (Farbe)
- FontFamily, FontSize, FontStyle, und FontWeight

### Brushes

Es gibt 6 Pinseltypen. Zudem gibt es diverse vordefinierte Pinseltypen (benannte Farbe). Mit Opacity kann ein Transparenzwert zwischen 0..1 gesetzt werden.



## Clipping

ClipToBounds: Sollen Child Controls an den Rändern des Parent Controls abgeschnitten werden?

Clip: Form, die zum Abschneiden benutzt werden soll (vgl. Canvas & Shapes → Geometrie)

## UI Control Library

Weitere Details zu UI Contorls sind ebenfalls im Appendix zu Block 2 vorhanden. Es wird nicht in der Vorlesung behandelt, gehört aber natürlich zum Stoff.

Verfügbar sind:

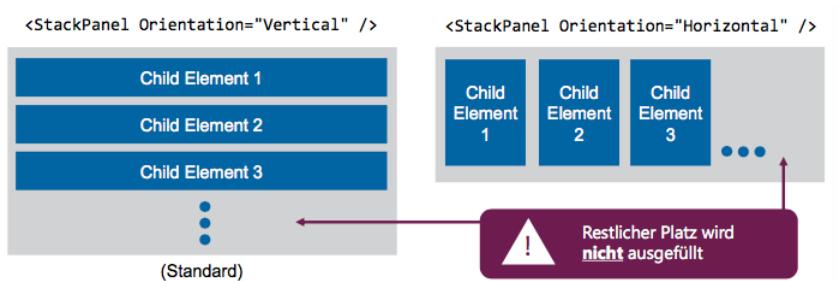
- Input Controls
- List Controls
- Beschriftungen
- Wertebereiche (ProgressBar & Slider)
- Organisation (GroupBox, Expander, TabControl)

## Containers Controls

### Container Controls mit Layout

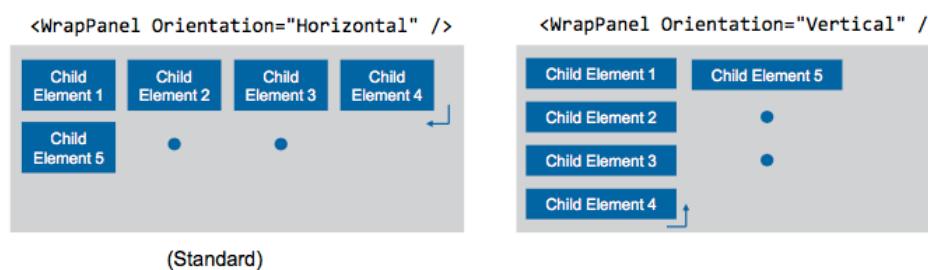
#### StackPanel

Child Elements werden vertikal oder horizontal aufgelistet.

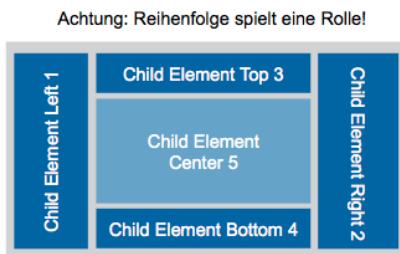
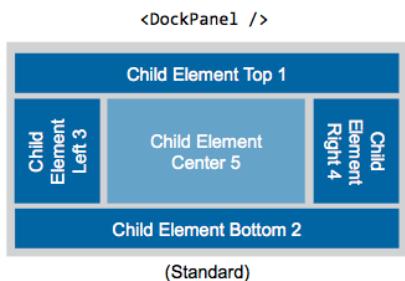


#### WrapPanel

Ist wie das StackPnael aber mit Zeilen-/Spaltenenumbruch. In der Praxis wird es zu Layoutzwecken nur selten anwendbar. (z.B. Tabs, Menus, Auflistung von Elementen).



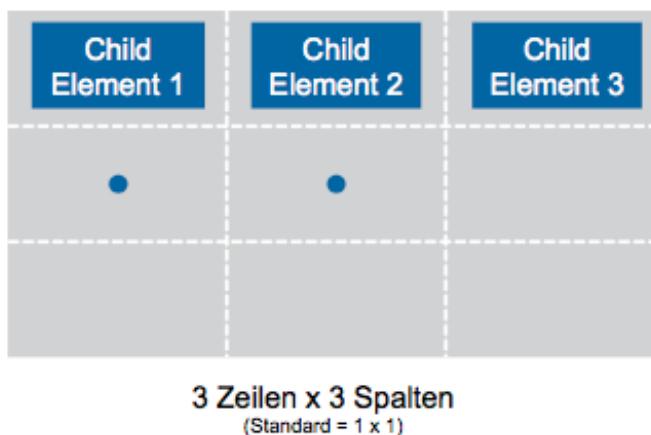
Child Elements werden an einer Seite oder dem Zentrum angedockt.



```
<DockPanel>
<TextBlock Text="Child Element Top 1"
           DockPanel.Dock="Top" />
<TextBlock Text="Child Element Bottom 2"
           DockPanel.Dock="Bottom" />
<TextBlock Text="Child Element Left 3"
           DockPanel.Dock="Left" />
<TextBlock Text="Child Element Right 4"
           DockPanel.Dock="Right" />
<TextBlock Text="Child Element Center 5" />
</DockPanel>
```

## Grid

Child Elements werden den Zellen einer Tabelle zugeordnet. Zeilen und Spalten müssen explizit angegeben werden.

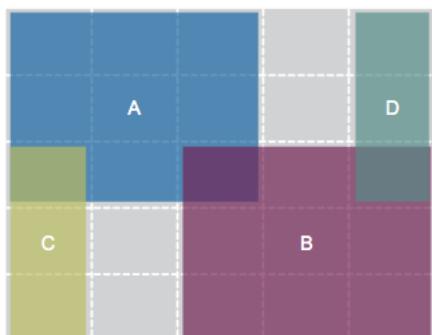


```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
  </Grid.RowDefinitions>
</Grid>
```

Auf den Gridelementen können auch Breiten-/ und Höhenangaben gemacht werden. Entweder als Ganzzahl, mit Auto, mit \* (Nutzt die Ganze Breite/Höhe entsprechend dem verfügbaren Rest) und 2\* Gewichtete Angabe (Verfügbarer Rest wird gewichtetet aufgeteilt).

```
<RowDefinitions>
  <RowDefinition Height="50"/>
  <RowDefinition Height="Auto" MinHeight="10"/>
  <RowDefinition Height="*" />
<RowDefinitions>
```

## RowSpan & ColSpan



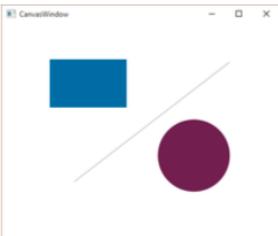
```
<Grid>
  <Grid.ColumnDefinitions>...</Grid.ColumnDefinitions>
  <Grid.RowDefinitions>...</Grid.RowDefinitions>

  <Button Content="A"
         Grid.Column="0" Grid.Row="0"
         Grid.RowSpan="3" Grid.ColumnSpan="3" />
  <Button Content="B"
         Grid.Column="2" Grid.Row="2"
         Grid.RowSpan="3" Grid.ColumnSpan="3" />
  <Button Content="C"
         Grid.Column="0" Grid.Row="2"
         Grid.RowSpan="3" Grid.ColumnSpan="1" />
  <Button Content="D"
         Grid.Column="4" Grid.Row="0"
         Grid.RowSpan="3" Grid.ColumnSpan="1" />
</Grid>
```

## Spezialfall

Bei einem 1x1 Grid , werden die Elemente in der Zelle gestapelt.

## Canvas



```
<Canvas>
  <Rectangle
    Canvas.Left="80" Canvas.Top="60"
    Width="128" Height="80"
    Fill="#006aa6" />
  <Ellipse
    Canvas.Left="260" Canvas.Top="160"
    Width="128" Height="120"
    Fill="#6E1C50" />
  <Path
    Canvas.Left="120" Canvas.Top="64"
    Width="260" Height="200"
    Stroke="DarkGray" Stretch="Fill"
    Data="M1,0 L0,1" />
</Canvas>
```

Absolute Positionierung, keinerlei Layout-Logik

Die Positionierung der Child Controls findet mittels Attached Properties statt.  
In Verbindung mit „Shapes“ gut zum „programmierten Zeichen“ geeignet.

## Canvas und Shapes

Auf Shapes können diverse Eigenschaften angegeben werden:

- Fill (Füllung)
- Stroke (Pen für Rahmen)
- StrokeThickness (Breite des Rahmens)
- StrokeDasharray (Muster für Linien)
- StrokeLineJoin (Art des Eckenübergangs)

**Shapes**

Line	Polyline	Polygon

```
<Line
  X1="50" Y1="50"
  X2="200" Y2="200"
  Stroke="Red"
  StrokeThickness="4" />
```

Figur bleibt offen

```
<Polyline
  Points="50, 100 200, 100 200, 200 300, 30"
  Stroke="DarkGreen"
  StrokeThickness="4"
  Fill="YellowGreen" />
```

Rectangle

```
<Rectangle
  Canvas.Left="80"
  Canvas.Top="60"
  Width="128"
  Height="80"
  Fill="#006aa6" />
```

Ellipse

```
<Ellipse
  Canvas.Left="260"
  Canvas.Top="160"
  Width="128"
  Height="120"
  Fill="#6E1C50" />
```

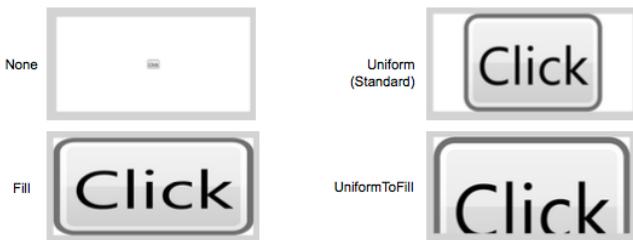
Circle

```
<Ellipse
  Canvas.Left="260"
  Canvas.Top="160"
  Width="100"
  Height="100"
  Fill="#6E1C50" />
```

Shapes verwenden intern leichtgewichtige Infrastruktur-Klassen für Geometrie-Berechnungen. Diese können auch selbst verwendet werden, zum Beispiel für das Definieren einer Clipping-Form.

## ViewBox

Skaliert einzelnes Child Control (inkl. Text) um den verfügbaren Platz auszunutzen. Funktioniert via Transformation.



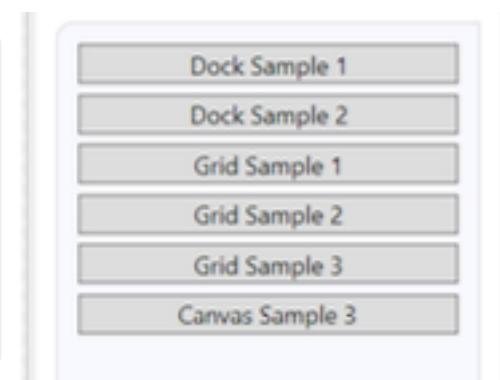
## Image

Attribut „Source“ für Angabe der Datenquelle, Verhalten wie bei einer ViewBox. Bei einer Übergrösse wird Alignment berücksichtigt.



Zeichnet einen Rahmen um ein Child Control. Kann mit Panels kombiniert werden.

```
<Border Background="GhostWhite" CornerRadius="8,0,8,0"
    BorderBrush="#ddd" BorderThickness="1"
    Margin="10"
    Padding="10">
    <StackPanel Orientation="Vertical">
        <Button Margin="0,0,0,5" Content="Dock Sample 1" />
        <Button Margin="0,0,0,5" Content="Dock Sample 2" />
        <Button Margin="0,0,0,5" Content="Grid Sample 1" />
        <Button Margin="0,0,0,5" Content="Grid Sample 2" />
        <Button Margin="0,0,0,5" Content="Grid Sample 3" />
        <Button Margin="0,0,0,5" Content="Canvas Sample 3" />
    </StackPanel>
</Border>
```



## Eigene Controls

### User Control (Komposition)

Wiederverwendbare Zusammenstellung mehrerer Controls als Gruppe. Besteht aus XAML und einem Code-Behind File. Kann nicht mit Styles/Templates umgehen.

### Custom Control (Ableitung)

Erweitert ein bestehendes Control um neue Funktionen. Besteht aus einem Code-File und allenfalls einem Standard-Style. Kann mit Styles/Templates umgehen.

## Dialogfenster

Dienen zum Abruf von Daten zum Beispiel einer Auswahl. Meist Model daher blockierend.

### Vorgehen

Dialogfenster erstellen, Dialogfenster mit Methode ShowDialog (aus Window heraus) anzeigen, Im Dialogfenster die Property DialogResult setzen true = ok, false = abgebrochen. Abrufen allfälliger Daten aus der Dialogfenster-Instanz.

In der Dialogfenster-Implementation - Markup (xaml):

- Button mit IsCancel="true" erstellen → schliesst bei Klick automatisch das Dialogfenster, kein Event-Handler nötig
- Button mit Click-Event-Handler erstellen

```
<Button Content="Abbrechen" IsCancel="true" />
<Button Name="OkButton" Content="OK" IsDefault="true" Click="OkButton_OnClick" />
```

Im Code-Behind des Dialogfensters (xaml.cs):

```
private void OkButton_OnClick(object sender, RoutedEventArgs e)
{
    SelectedCustomer = "Max Muster";
    DialogResult = true; // triggers dialog close as side effect!
}
```

In einem Event-Handler in einer Code-Behind Datei des Aufrufers (xaml.cs):

```
private void OpenDialog_OnClick(object sender, RoutedEventArgs e)
{
    var win = new DialogWindow();
    if (win.ShowDialog() != true)
    {
        Debug.WriteLine("Cancelled :-(");
        return;
    }

    Debug.WriteLine("OK :-)");
    var chosenCustomer = win.SelectedCustomer;
    ...
}
```

## Fenster mit Spezialformen

Der Fensterrahmen kann mittels Clipping verändert werden. Voraussetzung ist das folgende Fenstereigenschaften gesetzt werden: AllowTransparency=True und WindowStyle = None.

### Vorgehen

Form in Windows.Clip mittels einem Geomerty-Element festlegen

```
<Window.Clip>
    <RectangleGeometry RadiusX="8" RadiusY="8" Rect="0,0,800,600" />
</Window.Clip>
```

- z.B. Rechteck mit abgerundeten Rändern
- → Beliebige Formen werden mit PathGeometry ermöglicht (vgl. Block 2)
- → Fenstergröße muss natürlich auf Geometrie abgestimmt sein!

## Event Handling

- „Name“-Attribut der benötigten Elemente im XAML-Code definieren. Dies ermöglicht den Zugriff im Code-Behind.

```
<TextBox Name="Greeting" ... />
<TextBox Name="NameInput" ... />
<Button Name="SayHelloButton" ... Click="SayHelloButton_Click" ... />
```

- Event-Handler im Code-Behind ergänzen (oder durch Visual Studio automatisch erstellen lassen). Dies ermöglicht eine Reaktion auf einen Event.

```
private void SayHelloButton_Click(object sender, RoutedEventArgs e) {
    ...
    Greeting.Text = $"Hello, {NameInput.Text}!";
    ...
}
```

- Namenskonvention:

XAML: [Event] → XML-Attribut mit dem Namen des Events verfügbar z.B. „Click“  
Code-Behind: [Name]\_On[Event] → z.B. „OkButton\_OnClick“

## Commands (Alternative zu Events)

Standardisierte Ausführung eines Befehls, ermöglicht Wiederverwendung derselben Aktion, Verhindert so die Implementierung von x Click Handlern.

### Nötige Schritte vorerst

- Definieren eigener Commands als statische Instanzen von RoutedUICommand
- Nutzung durch Setzen des Command-Attributs auf die statischen Instanzen
- CommandBinding-Elemente auf Ebene Window setzen
- Für Tastenkürzel: KeyBinding-Elemente auf Ebene Window setzen
- Via Command-(Event-) Handler Code ausführen

## Implementierung

<ul style="list-style-type: none"> <li>Eigenen Command erstellen (xaml.cs)           <pre>public static RoutedUICommand MyCutCommand = new RoutedUICommand("Ausschneiden",     "MyCut",     typeof(WindowWithToolbar));</pre> </li> </ul>	<ul style="list-style-type: none"> <li>Binden an Command-(Event-)Handler (xaml)           <pre>&lt;Window.CommandBindings&gt;     &lt;CommandBinding Command="local:WindowWithToolbar.MyCutCommand" Executed="MyCutCommand_Executed" /&gt; &lt;/Window.CommandBindings&gt;</pre> </li> </ul>
<ul style="list-style-type: none"> <li>Nutzung im Markup (.xaml): (im Beispiel bei MenuItem, aber fast überall möglich)           <pre>&lt;MenuItem Header="_Cut" InputGestureText="CTRL + X"     Command="local:WindowWithToolbar.MyCutCommand" /&gt;</pre> </li> </ul>	<ul style="list-style-type: none"> <li>Shortcuts definieren (.xaml)           <pre>&lt;Window.InputBindings&gt;     &lt;KeyBinding Key="X" Modifiers="Control" Command="local:WindowWithToolbar.MyCutCommand" /&gt; &lt;/Window.InputBindings&gt;</pre> </li> </ul>

Header-Text wird bei Menüitem automatisch aus Command-Text abgeleitet, falls nicht angegeben

Für Zugriff auf Assembly (lokaler Namespace)

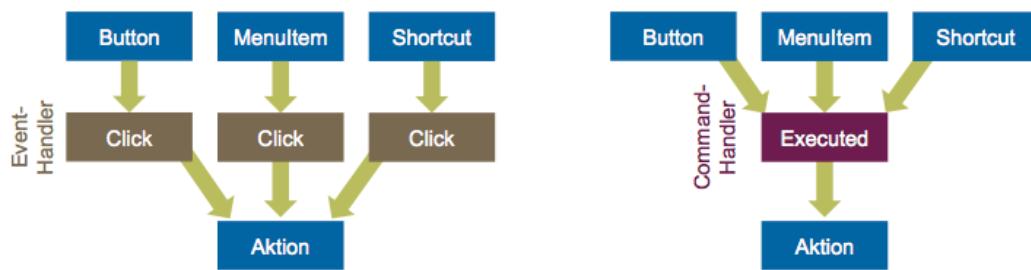
Klasse, in der der Command implementiert ist

Name der Property mit dem Command

### Implementierung Command-(Event-)Handler (.xaml.cs)

```
private void MyCutCommand_Executed(object sender, ExecutedRoutedEventArgs e)
{
    ...
}
```

Fazit: In der Regel ist der Command dem normalen Event-Handling vorzuziehen.



## NuGet Package Manager

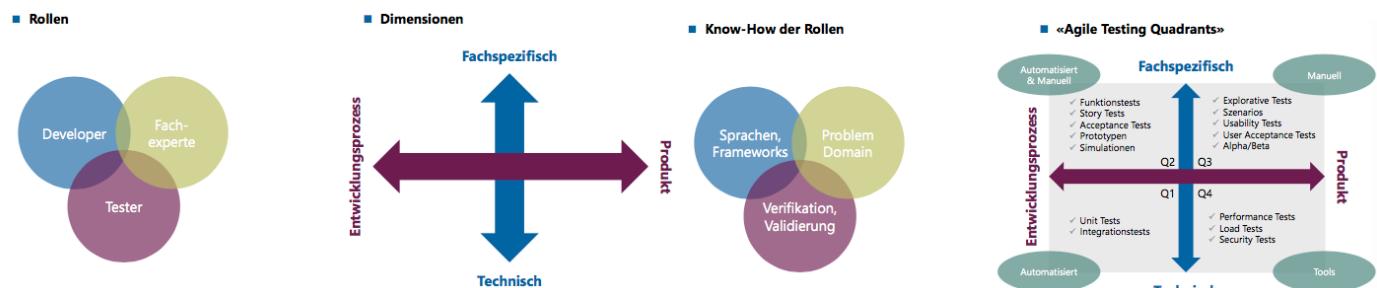
Package Management System für Visual Studio Projekte. Aussprache wie „new get). Es ist Open Source und stellt im Wesentlichen ein Abhängigkeitsmanagement zur Verfügung. Wie npm ist es ein Package Manager mit einem Command Line Interface.

### Beliebte NuGet Packages

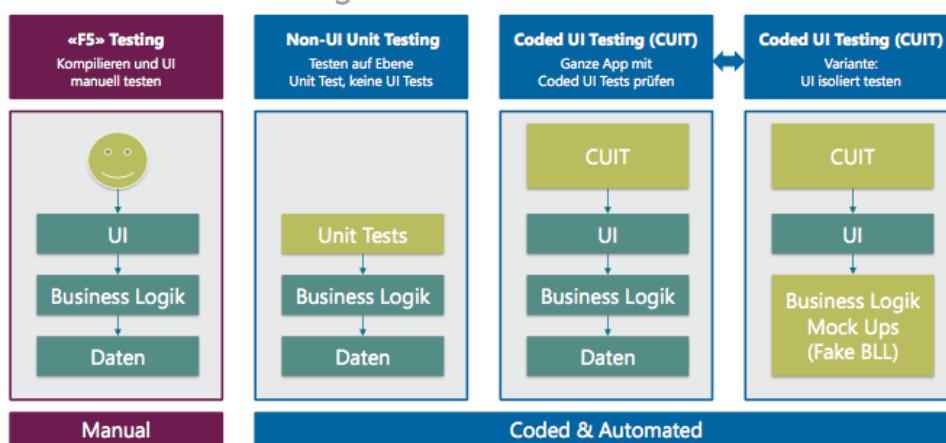
<b>«EntityFramework»</b>	> 22 Mio
■ Microsoft's empfohlene Technologie für Projekte mit Datenzugriff (O/R-Mapping)	
<b>«Json.NET»</b>	> 40 Mio
■ High-Performance Framework für Mapping von .NET-Objekten nach JSON und umgekehrt (vgl. Miniprojekt)	
<b>«Bootstrap CSS»</b>	> 7 Mio
■ Das bekannte Front-End Framework für die Erstellung von Responsive Web Applications	
<b>«jQuery»</b>	> 26 Mio
■ Das bekannte Javascript Framework für die rasche Programmierung von Web Applications	
<b>«Microsoft.AspNet.Mvc»</b>	> 20 Mio
■ Das Web-Anwendungs Framework ASPNET MVC als NuGet Paket	
<b>«AutoMapper»</b>	> 5 Mio
■ Ein Object-to-Object mapper, gut geeignet für die Verwendung in Apps, die den MVVM-Ansatz nutzen	
<b>«log4net»</b>	> 6 Mio
■ Das beliebte Logging Framework als NuGet Package	
<b>«Unity»</b>	> 4 Mio
■ Ein Dependency Injection (DI) Container für die konfigurierbare Erstellung von Objekthierarchien	
<b>«RestSharp»</b>	> 3 Mio
■ Einfacher REST und HTTP API Client (vgl. Miniprojekt)	
<b>«System.Data.SQLite»</b>	> 0.8 Mio
■ X-Plattform .NET Library für den Zugriff auf SQLite DBs, unterstützt das EntityFramework und LINQ	

## Automated UI Testing

### Testing



## Übersicht von Testing



Ist eine Sammlung von Open Source Projekten, um Tests in .NET-Projekten zu automatisieren.

### TestStack.White

CUIT-Framework für WPF, basiert auf Microsoft's UI Automation Framework, ist ein NuGet Package. Es deckt dabei die Quadranten Q1 (Verifikation) und Q2 (Validierung) ab.

## Vorbereitungen

### Voraussetzungen

- Solution mit WPF-Projekt und Unit-Test-Projekt
- Referenzen auf das WPF-Projekt und die .NET-WPF-Libraries gesetzt
- NuGet-Package „TestStack.White“ im Test-Projekt installiert

### Testklasse erstellen

```
[TestClass]
public class MyFirstUITest
{
    [TestMethod]
    public void MyFirstUITestMethod()
    {
    }
}
```

#### Hinweis:

Wird eine neues MSTest-Projekt erstellt, gibt es bereits eine Klasse «UnitTest1»

### Verzeichnis mit der WPF App Assembly festlegen

Aktuelles Verzeichnis und Name der in Ausführung befindlichen Assembly herausfinden. Verzeichnis dann am besten gleich als Read-Only Property in der Testklasse zur Verfügung stetllen.

```
/// <summary>
/// the directory in which the test is running
/// </summary>
public string BaseDir => Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

/// <summary>
/// system under test (wpf app to be tested)
/// </summary>
public string SutPath => Path.Combine(BaseDir, $"{nameof(MenuAndCommands)}.exe");
```

### Schritt 1.5

#### 1. Schritt: WPF App starten

- Neues Application-Objekt aus WPF App Assembly erstellen:

```
var app = Application.Launch(SutPath);
```

- Die Variable `app` enthält nun ein UI-Automatisierungsobjekt des Typs `TestStack.White.Application`

#### 2. Schritt: Fenster abrufen

- Fenster anhand des Fenstertitels abrufen:

```
var window = app.GetWindow("Hauptfenster", InitializeOption.NoCache);
```

- Fenster aus der Liste der Fenster abrufen (mit Linq):

```
var window = app.GetWindows().First();
```

- Fenster anhand der ID (Name-Attribut in WPF) abrufen:

```
var window = app.GetWindow(SearchCriteria.ByAutomationId("Win1"), InitializeOption.NoCache);
```

- Die Variable `window` enthält nun ein UI-Automatisierungsobjekt des Typs `TestStack.White.UIItems.WindowItems.Window`

**3. Schritt: Control abrufen**

- Control anhand der Beschriftung abrufen:

```
var button = window.Get<Button>(SearchCriteria.ByText("Speichern"));
```

- Control anhand der ID (Name-Attribute in WPF) abrufen:

```
var button = window.Get<Button>("SaveButton");
```

- Die Variable button enthält nun ein UI-Automatisierungsobjekt des Typs TestStack.White.UIItems.Button
- Diverse weitere Automatisierungsobjekte verfügbar (vgl. Übungen): TextBox, Label, CheckBox, RadioButton, Slider, Spinner, ProgressBar, ...

**4. Schritt: Aktionen ausführen und Test-Annahmen treffen**

- Beispiel: Effekt eines Klicks simulieren

```
Assert.AreEqual("Speichern", button.Text);
button.Click();
Assert.AreEqual("Gespeichert!", button.Text);
```

- Beispiel: Eingabe in Textfeld simulieren

```
var input = window.Get<TextBox>("NameInput");
Assert.IsTrue(string.IsNullOrEmpty(input.Text));
var newText = "You've been haaaacked!";
input.Text = newText;
Assert.AreEqual(newText, input.Text);
```


**5. Schritt: App beenden**

- Fenster schliessen und App beenden

```
app.Close();
```

- Wird die app nicht beendet, bleibt das Fenster nach Ausführung des CUIT offen!

## Vollständiges Beispiel

```
[TestClass]
public class MyFirstUITest
{
    public string BaseDir => Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    public string SutPath => Path.Combine(BaseDir, $"{nameof(MenuAndCommands)}.exe");

    [TestMethod]
    public void MyFirstUITestMethod()
    {
        var app = Application.Launch(SutPath);
        var window = app.GetWindow("Hauptfenster", InitializeOption.NoCache);
        var button = window.Get<Button>("SaveButton");

        Assert.AreEqual("Speichern", button.Text);
        button.Click();
        Assert.AreEqual("Gespeichert!", button.Text);

        app.Close();
    }
}
```

## Screenshots

Im White-Framework lassen sich Screenshots erstellen.

```
var screenshot = Desktop.CaptureScreenshot();
```

Die Variable screenshot enthält nun ein .NET Bitmap Objekt, welches weiterverarbeitet werden kann... und als .png speichern → normal.NET (System.Drawing.Bitmap).

```
var path = System.IO.Path.Combine(BaseDir, "screenshot.png");
screenshot.Save(path, ImageFormat.Png);
```

Im Zielordner des Tests liegt nun unter dem Namen „screenshot.png“ der Screehot im .png – Format.

## Review

Es ist ein Open Source Automatisierungsframework für .NET zur Entwicklung von Coded UI Tests (CUIT). Abstrahiert UI Bestandteile verschiedener Technologien wie WPF. Ermöglicht die Verwendung in normalen Unit-Testing-Frameworks wie MSTest oder NUnit.

Aber...

Test schreiben = Code Schreiben → Entwicklungsaufwand, ändert sich UI müssen die Tests aktualisiert werden, was wiederum Wartungsaufwand bedeutet. Die Performance bei komplexen UIs ist schlecht oder sogar fehlerhaft. Dies reduziert die Aussagekraft. Zudem wird TestStack.White seit einiger Zeit nicht mehr gross weiterentwickelt. Der Einsatz eines UI Testing Frameworks wie TestStack-White sollte also gut überlegt sein.

## Inhalt

- **Technisch** – Standardisiertes GUI Design mit WPF
  - o Resources
  - o Styles
  - o Control Templates
  - o Trigger
- **Konzeptionell** - GUI Design Principles

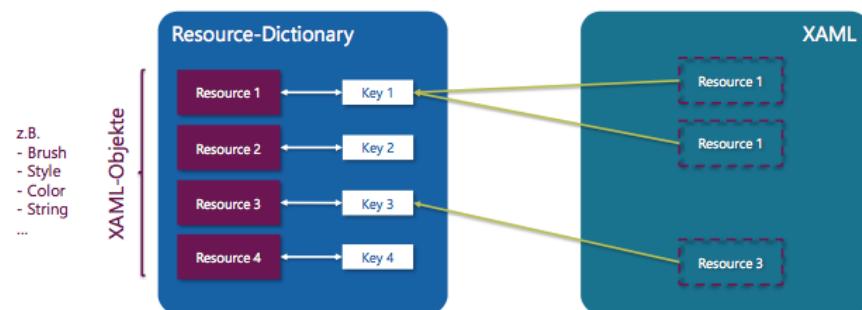
*“I majored in engineering—it’s almost a badge of pride to build something that looks awful.”*

— Unknown

## Resources

### Ziel

Nützliche Objekte, z.B. Pinsel, Styles oder Templates, zentral definieren und überall wiederverwenden.



### Wichtig

Das Konzept der Ressourcen ist in der WPF vollkommen neu definiert worden und unterscheidet sich damit grundlegend von Windows Forms.

### Vorteile von zentralen Ressourcen

#### Effizienz

1x definieren, Nx verwenden, weniger Code und weniger Speicherbedarf

#### Wartbarkeit

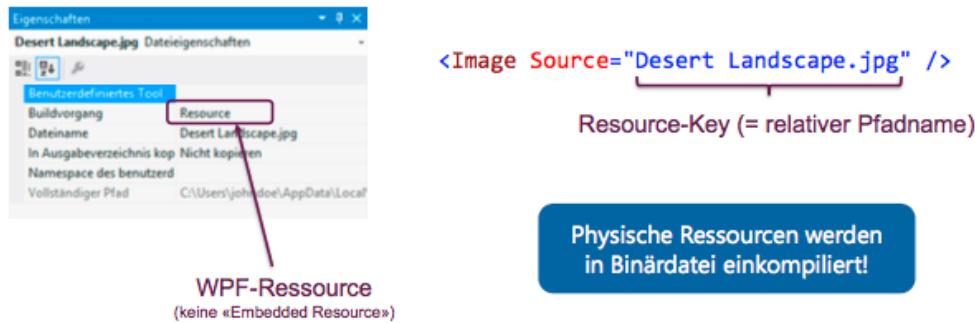
Zentral abgelegte Formatierungsangaben, Bessere Konfigurierbarkeit und bessere Wartbarkeit

#### Adaptierbarkeit

Trennung von Darstellung und Struktur, Bessere Anpassbarkeit an User, System und sonstige Voraussetzungen.

### Physischen Ressourcen in WPF

Zum Beispiel Bilder.



### „Resource“

Ist ein beliebiges Objekt (=Instanz), das in XAML definiert werden kann. Es wird mit dem Key-Attribute aus dem x-Namespace benannt und ist dann über diesen Key ansprechbar. {StaticResource...} wird mittels einer Markup Extension in einen Data Binding Ausdruck umgewandelt.

```
<Application.Resources>
  <SolidColorBrush x:Key="MyButtonBackground" Color="#EEEEEE" />
</Application.Resources>
```

```
<Button Background="{StaticResource MyButtonBackground}" Content="Save" />
```

### „ResourceDictionary“

Ein Behälter, um Ressourcen zu speichern. Indexiert nach dem Ressourcen-Namen (x:Key). In allen von FrameworkElement angeleiteten Elementen out-of-the-box verfügbar.

### Beispiel Application.Resources, Window.Resources, Button.Resources

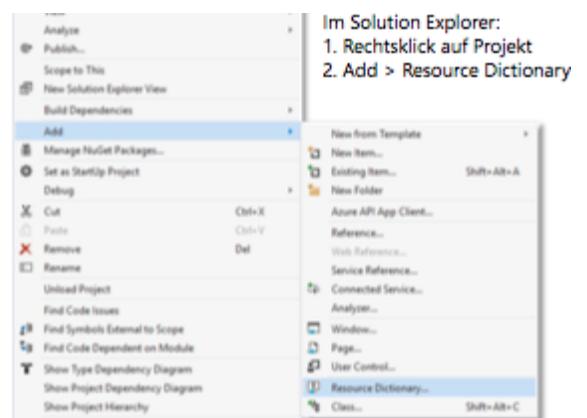
#### Eigene ResourceDictionaries

Ein separates .xaml-File, welches in anderen ResourceDictionaries einbindbar ist. XAML-Root-Node ist ResourceDictionary.

```
<ResourceDictionary
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:Microsoft.FamilyShow">

  <!-- place your resources here -->

</ResourceDictionary>
```



#### Eigene ResourceDictionaries einbinden

Dazu ein MergedDictionary erstellen und darin alle ResourceDictionaries auflisten.

```
<Application.Resources>
  <ResourceDictionary>
    <!-- you can mix resources and merged dictionaries -->
    <SolidColorBrush x:Key ="MyButtonBackground" Color="#EEEEEE" />

    <ResourceDictionary.MergedDictionaries>
      <!-- just list your external resource dictionaries, here -->
      <ResourceDictionary Source="Colors.xaml"/>
      <ResourceDictionary Source="Brushes.xaml"/>
    </ResourceDictionary.MergedDictionaries>

  </ResourceDictionary>
</Application.Resources>
```

## Kaskadierung möglich

Es ist möglich andere Dictionaries in einem eigenen ResourceDictionary zu verwenden. Zum Beispiel für hierarchisch festgelegte Formatierungen/Farben/Styling.

```
<ResourceDictionary
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:Microsoft.FamilyShow">

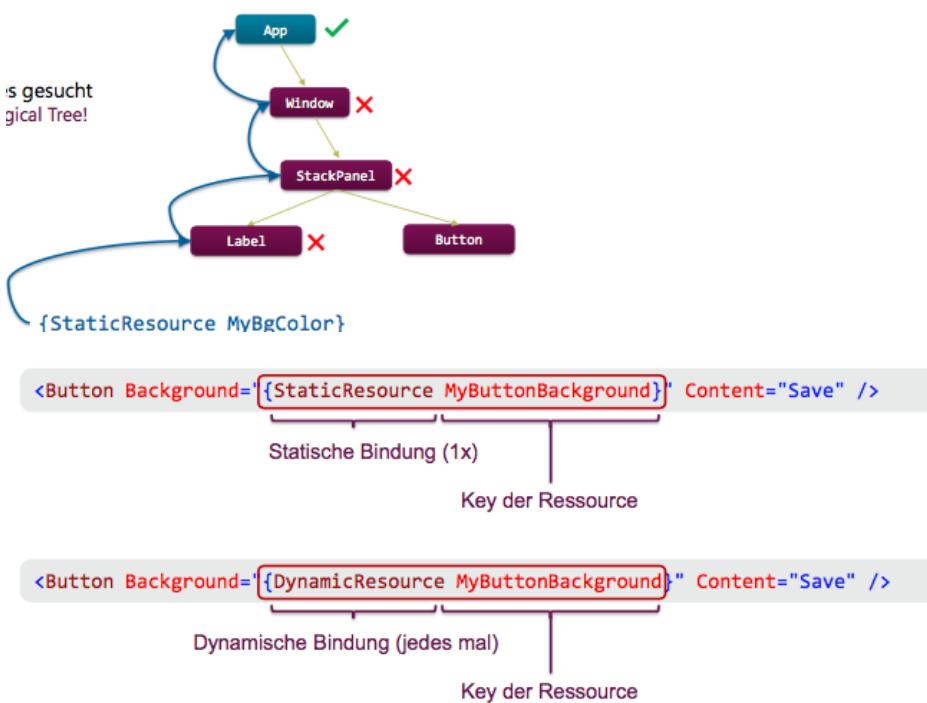
  <!-- include your base dictionaries, here -->
  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary Source="Colors.xaml"/> ←
  </ResourceDictionary.MergedDictionaries>

  <!-- now, access the externally defined resources -->
  <SolidColorBrush x:Key="ButtonBgBrush" Color="{StaticResource ThemeColor1}" />
</ResourceDictionary>
```

## Zugriff auf Resource

1. Key wird im Element und in allen Parent-Nodes gesucht → Logical Tree
2. Key wird in Application.Resources gesucht
3. Key wird in System-Ressourcen gesucht.

Dabei ist die Reihenfolge im XAML-Code entscheidend.



## Zugriff auf System-Ressourcen

x:Static ist eine weitere Markup Extension → Dazu später mehr.

```
<Button Background="{x:Static SystemColors.ControlBrush}" Content="Save" />
```



## Zugriff auf Ressourcen mit C#

Über die Methode `FrameworkElement.FindResource(...)`.

Beispiele:

```
var okText = (string)FindResource("OkText");
var bgBrush = FindResource("DarkBrush") as Brush;
```

### StaticResource vs. DynamicResource

`...="{StaticResource [name]}`

Anstelle von [name] steht der Key der Resource. Statische Bindung. Compile Time Check → Der Fehler wird daher früh gefunden. Die Auswertung findet bei der Objekt-Konstruktion statt (1-Mal).

`...="{DynamicResource [name]}`

Anstelle von [name] steht der Key der Resource. Dynamische Bindung. Runtime Check → Lässt dynamisch erzeugte und geladene Ressourcen zu. Die Auswertung findet bei jedem Zugriff statt (N-Mal).

### Markup Extension

- Ausdruck mit Binding-Syntax... (→ «Attribut Syntax»)

```
<Border BorderBrush="{DynamicResource MyBorderBrush}" />
```

- ... ist die Kurzform für: (→ «Property Element Syntax»)

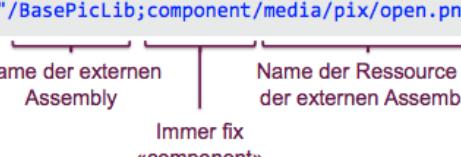
```
<Border ... >
<Border.BorderBrush>
<DynamicResource ResourceKey="MyBorderBrush" />
</Border.BorderBrush>
...
</Border>
```

### Externe Ressourcen

#### Externe Assembly

Ressourcen in einer anderen Assembly

```
<Image Source="/BasePicLib;component/media/pix/open.png" />
```



Zum Vergleich ein Zugriff auf ein Bild in der gleichen Assembly

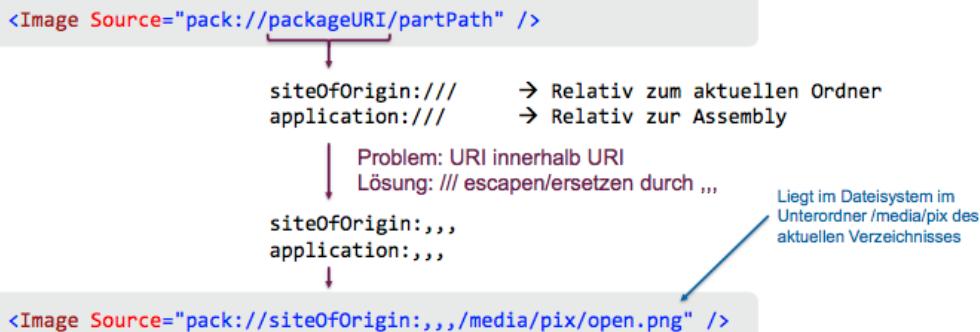
```
<Image Source="FileOpen.png" /> ← Relativ (gleicher Ordner wie .xaml-Datei)
<Image Source="/MeinOrdner/FileOpen.png" /> ← Absolut (innerhalb Assembly)
```

Ressourcen im Filesystem. Sollte aber nicht verwendet werden, da zu starke Bindung.

```
<Image Source="C:\Program Files\X-App\media\pix\open.png" />
```

## PackageURI

Ressourcen anhand der PackageURI ➔ Standardisiert in der XML Paper Specification (XPS).



## Styles

### Situation

#### Ohne Styles

Wie in HTML ohne CSS. Styles müssen bei jedem Element festgelegt werden.

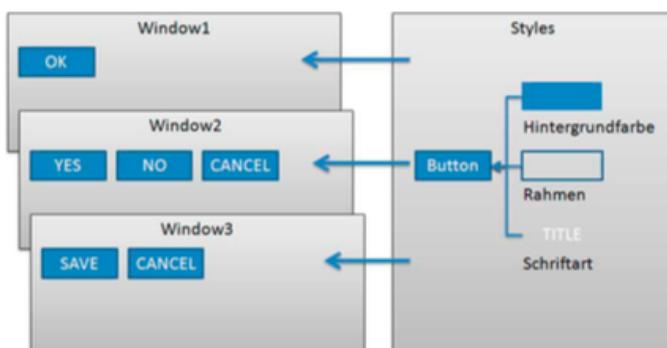
```

<StackPanel Orientation="Horizontal" HorizontalAlignment="Right" Margin="0 10 0 0">
  <Button Background="White" Foreground="Black" BorderBrush="Black"
    BorderThickness="1" FontFamily="Arial" FontSize="13" Padding="10 2 10 2"
    Margin="2">Speichern</Button>
  <Button Background="White" Foreground="Black" BorderBrush="Black"
    BorderThickness="1" FontFamily="Arial" FontSize="13" Padding="10 2 10 2"
    Margin="2">Abbrechen</Button>
  <Button Background="White" Foreground="Black" BorderBrush="Black"
    BorderThickness="1" FontFamily="Arial" FontSize="13" Padding="10 2 10 2"
    Margin="2">Hilfe</Button>
</StackPanel>
  
```

Schlecht! ➔ DONT DO THAT!

#### Mit Styles

Aussehen ausschliesslich über Styles und Templates definieren. (Style-Element nur 1x im Speicher).



- Vgl. eigene CSS Klasse bei HTML

Cancel    Ok    Save

- Nutzung:

```
<StackPanel Orientation="Horizontal">
  <Button Style="{StaticResource MyButtonStyle}" Content="Cancel" />
  <Button Style="{StaticResource MyButtonStyle}" Content="Ok" />
  <Button Style="{StaticResource MyButtonStyle}" Content="Save" />
</StackPanel>
```

```
<Style x:Key="MyButtonStyle">
  <Setter Property="Button.Foreground" Value="#2672EC" />
  <Setter Property="Button.Background" Value="Transparent" />
  <Setter Property="Button.BorderBrush" Value="#2672EC" />
  <Setter Property="Button.BorderThickness" Value="2" />
  <Setter Property="Button.FontFamily" Value="Segoe UI" />
  <Setter Property="Button.FontSize" Value="13" />
  <Setter Property="Button.Padding" Value="10 2 10 2" />
  <Setter Property="Button.Margin" Value="2" />
</Style>
```

Name der betreffenden  
Dependency Property

Wert der betreffenden  
Dependency Property

## Styles definieren

Es sind auch komplexere Werte möglich.

```
<Setter Property="Button.Background">
  <Setter.Value>
    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
      <GradientStop Offset="0" Color="#dddddd" />
      <GradientStop Offset="0.5" Color="#F0F0F0" />
      <GradientStop Offset="1" Color="#dddddd" />
    </LinearGradientBrush>
  </Setter.Value>
</Setter>
```

Cancel ← Farbverlauf im Hintergrund

## Typspezifische Styles

1

```
<Style x:Key="MyButtonStyle3" TargetType="Button">
  <Setter Property="Background" Value="Transparent" />
  <Setter Property="Foreground" Value="Black" />
  <Setter Property="BorderBrush" Value="Silver" />
  <Setter Property="BorderThickness" Value="2" />
  <Setter Property="FontFamily" Value="Segoe UI" />
  <Setter Property="FontSize" Value="13" />
  <Setter Property="Padding" Value="10 2 10 2" />
  <Setter Property="Margin" Value="2" />
</Style>
```

Verkürzter Name der  
Dependency Property

Klassenname kann  
wegelassen werden,  
da Typ bekannt

Typ / Control-Klasse'

1

✓ «Implizite Styles»

Wird der Key weggelassen, so gilt  
der Style standardmäßig für alle  
Controls des angegebenen Typs

⚠ Typspezifische Styles verhindern  
Style Sharing mit Controls  
anderer Typen

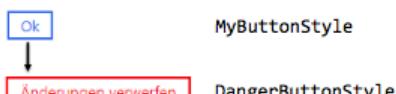
<sup>1</sup> Kurzform für:  
... TargetType="{x:Type Button}"

## Styles kombinieren

Styles sind mit einer Inline Eigenschaft kombinierbar. Eignet sich zum Beispiel zum Hervorheben von Control mit Sonderfunktion.

Styles können aber auch abgeleitet werden (BasedOn). Wie bei einer gut durchdachten Klassenhierarchie reduziert ein geschicktes Wiederverwenden von Styles den Umgang der Ressourcen drastisch.

```
<Style x:Key="DangerButtonStyle"
  TargetType="Button"
  BasedOn="{StaticResource MyButtonStyle}">
  <Setter Property="Background" Value="Red" />
</Style>
```

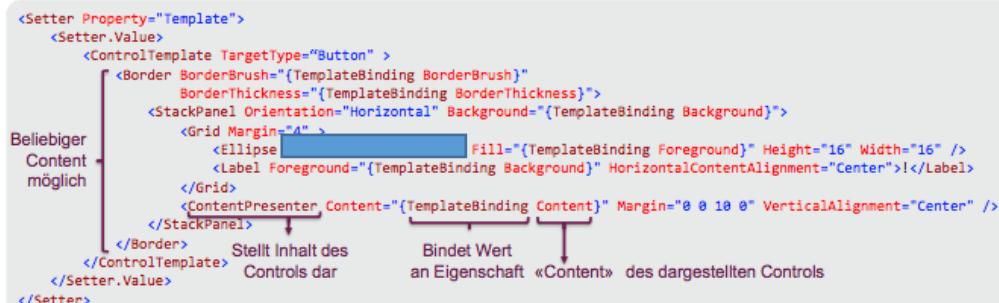


Ist eine XAML-Struktur, die die Inhaltsdarstellung eines Controls festlegt. Es ermöglicht stärkere Wiederverwendung der Basis-Control-Logik bei gleichzeitig trotzdem sehr flexibler Darstellung (Beispiel: ToggleButton, CheckBox und RadioButton).

Jedes Control hat ein Default-Control Template.

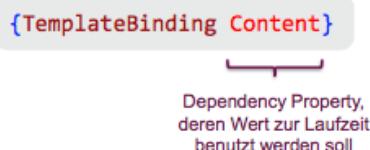
Die Control Templates sind einzeln als Ressource oder auch innerhalb des Styles definierbar.

## Festlegung via Style



## Template Binding

Markup Extension, die eine verkürzte Form des Data Bindings abbiete. Nur innerhalb des Control Templates anwendbar. Kann Wert einer Dependency Property im Control (oder Style) abrufen.



## Trigger

Style anhand unterschiedlicher UI-Zustände anpassen.



1 Trigger-Element

2 Dependency Property

3 Wert / Zustand

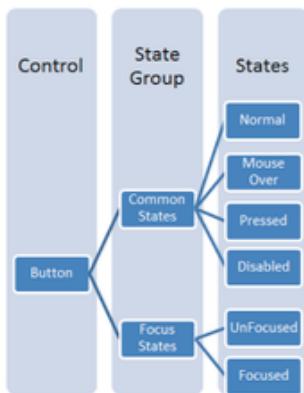


Default Button Style verhindert,  
dass dies für Buttons wie erwartet  
funktioniert (Visual States!)

## Visual State Manager

Ermöglicht innerhalb eines ControlTemplate-Elements unterschiedliche Darstellungen anhand eines Zustands.

## Beispiel des Buttons:



## Trigger vs. Visual State Manager

```

<Setter Property="Template">
  <Setter.Value>
    <ControlTemplate TargetType="Button">
      <Border BorderBrush="{TemplateBinding BorderBrush}"
        BorderThickness="{TemplateBinding BorderThickness}">
        <Label Background="{TemplateBinding Background}"
          Foreground="{TemplateBinding Foreground}"
          VerticalContentAlignment="Center"
          HorizontalContentAlignment="Center"
          Content="{TemplateBinding Content}" />
      </Border>
    </ControlTemplate>
  </Setter.Value>
</Setter>
  
```

### Problem

Visual State Manager übersteuert die Angaben gemäss Style.Trigger (z.B. bei „Button“)

### Abhilfe

Eigenes Control Template verwenden

### Beispiel

Border + Label als Control Template für Button verwenden.

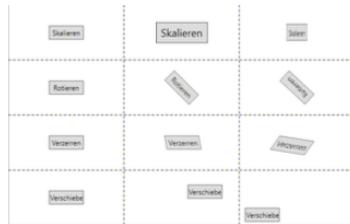
## Transformationen

- «LayoutTransform»
- Werden VOR der Layout-Phase angewendet

- «RenderTransform»
- Werden NACH der Layout-Phase angewendet

### Typen:

- <ScaleTransform ScaleX="1.5" ScaleY="1.5" />
- <RotateTransform Angle="45" CenterX="30" CenterY="12" />
- <SkewTransform AngleX="-35" AngleY="9" />
- <TranslateTransform X="40" Y="-10" />



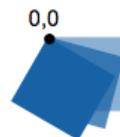
! Angelpunkt der Transformation beeinflusst die optische Darstellung sehr wesentlich!

## In den Styles

Ist Normal via Style Setter zu bewerkstelligen

```

<Setter Property="RenderTransform">
  <Setter.Value>
    <RotateTransform Angle="3"></RotateTransform>
  </Setter.Value>
</Setter>
  
```



Und um den Ursprungspunkt/Angelpunkt der Transformation anzugeben. Die Angaben bei RenderTransformOrigin sind relativ (0...1), innerhalb des Transform absolut (Pixels). Der Standardwert ist 0,0.

```
<Setter Property="RenderTransformOrigin" Value="0.5,0.5"/>
```



WPF hat kein explizites „Skin“ oder „Theme“ Konzept. Anhand separater Resource Dictionaries ist dies aber einfach simulierbar.

## Resources & Styles

### Kleinprojekte

Styling direkt in Window.Resources (oder teilweise direkt inline bei den Controls)

### Kleine bis mittlere Projekte

Eigene Resource Dictionaries für verschiedene Typen (z.B. nach Grundfarben, Pinsel, ControlStyles etc.)

### Grössere Projekte

Eigene Assemblies mit Ressource Dictionaries und Image-Resources pro Design/Theme

## GUI Design Principles (Desktop Apps)

### Windows 7

#### ■ «How to design a great User Experience» (Microsoft, Windows 7 Guidelines, 2009)

- |  |   |
|--|---|
| 1. Nail the basics<br>2. Design experiences, not features<br>3. Be great at something<br>4. Don't be all things to all people<br>5. Make the hard decisions<br>6. Make the experience like a friendly conversation<br>7. Do the right thing by default<br>8. Make it just work<br>9. Ask questions carefully<br>10. Make it a pleasure to use<br>11. Make it a pleasure to see<br>12. Make it responsive<br>13. Keep it simple | 14. Avoid bad experiences<br>15. Design for common problems<br>16. Don't be annoying<br>17. Reduce effort, knowledge and thought <ul style="list-style-type: none"> <li>- Explicit beats implicit</li> <li>- Automatic beats manual</li> <li>- Concise beats verbose</li> <li>- Constrained beats unconstrained</li> <li>- Enabled beats disabled</li> <li>- Remembered beats forgotten</li> <li>- Feedback beats being clueless</li> </ul> 18. Follow the guidelines<br>19. Test your UI |
|--|---|


 WIMP

### Windows 8 / 10

#### ■ «Microsoft Design Principles for Windows Store apps» (2014)

1	2	3	4	5
<b>«Pride in craftsmanship»</b> <small>"Engineer the experience to be complete, thorough, and polished at every stage. Devote time and energy to small things that are seen often by many of your users."</small>	<b>«Be fast and fluid»</b> <small>"Let people interact directly with content. Respond to actions quickly with matching energy. Bring life to the experience by creating a sense of continuity and telling a story through meaningful use of motion."</small>	<b>«Authentically digital»</b> <small>"Exemplify the capabilities of hardware and software. Take full advantage of the digital medium. Remove physical boundaries to create experiences that are more efficient and effortless than reality."</small>	<b>«Do more with less»</b> <small>"You can do more with less by reducing your design to its essence. Create a clean and purposeful experience by leaving only the most relevant elements on screen so people can be immersed in the content."</small>	<b>«Win as one»</b> <small>"Work with other apps, devices, and the system to complete scenarios for people. For example, let people get content from one app and share it with another. Take advantage of what people already know, like standard touch gestures and charms, to provide a sense of familiarity, control, and confidence."</small>

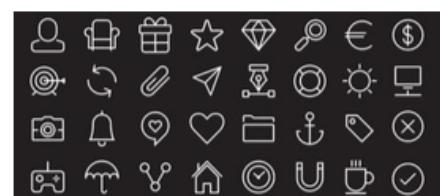
## Flat Design

### Trend 1: «Flat Design»

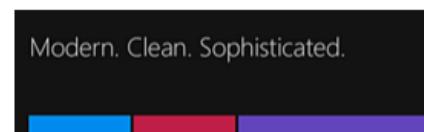
- Von Microsoft seit 2010 mit Mia.-schwerer Marketing-Kampagne in den Markt gedrückt
- Mittlerweile von iOS + Android (+ Web) adaptiert



## Minimalism



## Typography



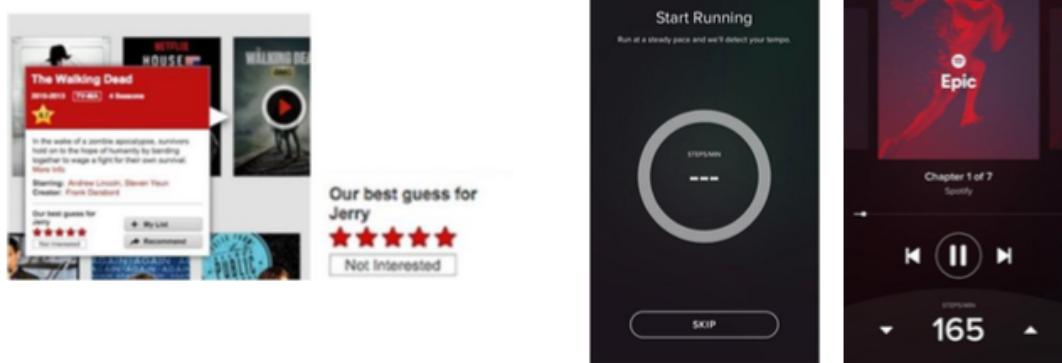
→ Font «Segoe UI»

<http://tomgabor.com/i-shot-the-serif/>

## Icons beat Text => a picture is worth a thousand words

Jetzt	22 Uhr	23 Uhr	00 Uhr	01 Uhr	02 Uhr	03 Uhr	04 Uhr
1°	2°	2°	2°	2°	2°	2°	2°
Sonntag				3	-2		
Montag				2	-3		
Dienstag				0	-2		
Mittwoch				4	3		
Donnerstag				4	1		
Freitag				4	1		
Samstag				4	2		



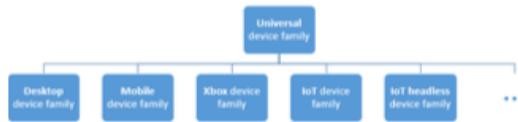


## Dynamic Layout (= Responsive Design in Web/App Development)

### Universal Windows Platform (UWP)



<https://dev.windows.com/en-us/design>



**⚠️ UWP = Fokus auf Single App Betrieb  
WPF = Fokus auf Multitask-Betrieb**

## Dynamic Layout in WPF Strategie 1 – Ändern der Position





### Strategie 3 – Neuanordnen



### Strategie 4 – Einblenden



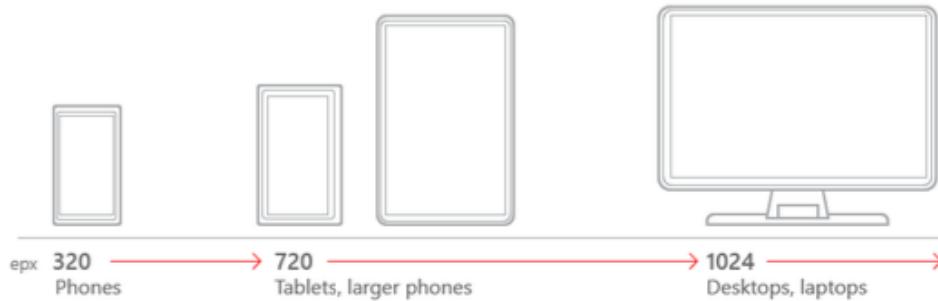
### Strategie 5 – Ersetzen (Wechsel der UI-Elemente)



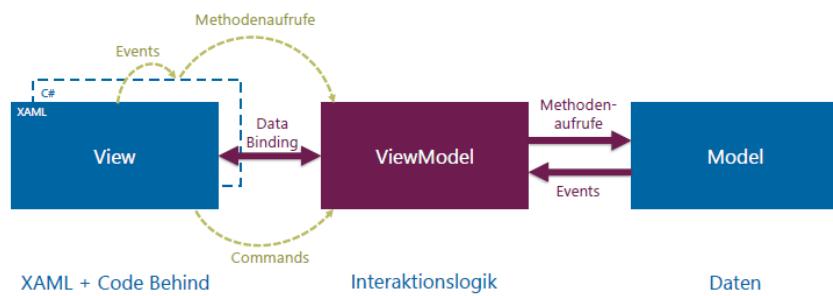


### Umsetzbarkeit in WPF

- Styles, Skins, Themes
- Trigger
- Eigene Layouts (= XAML-Files) pro Device-Klasse (vgl. zu Android)



Klare Trennung von der Präsentation und der Logik.



## Markup Extensions

{x:Type [Datentyp]}	Liefert angegebenen [Datentyp]/Klasse
{x:Static [Pfad]}	Bindet an Konstante, statische Property oder Field, sowie an Enums
{x:Null}	Null-Wert
{StaticResource [Name]}	Statische Bindung an Ressource
{DynamicResource [Name]}	Dynamische Bindung an Ressource
{Binding ....}	Data Binding Ausdruck
{RelativeSource...}	Setzt die Data Binding Source auf eine relativen Bezug im Logical Tree
{TemplateBinding...}	Bindet Wert an Eigenschaft des mittels Template dargestellten Controls
{x:Reference...}	Abkürzungen für {Binding ElementName=...}

## Überblick Data Binding

Nützliche Properties (Appendix mit Detailbeschrieb vorhanden)

- BindingBase
  - o StringFormat
- Binding
  - o Converter(IValueConverter)
  - o ElementName
  - o Mode
  - o Path (Standardproperty)
- MultiBinding
  - o Bindings
  - o Converter (IMultiValueConverter)

## Binding

Source vs. RelativeSource vs. ElementName

Die Datenquelle für Data Binding ist nicht standardmäßig gesetzt. Dies muss also explizit geschehen.

## Möglichkeiten

- Klasse FrameworkElement

## Mobile and GUI Engineering

- o «DataContext»
- Klasse Binding
  - o «Source»
  - o «RelativeSource »
  - o « ElementName »

### Source

Um DataContext für einzelne Controls zu übersteuern, kann die Datenquelle beim Binding mittels Angabe einer « Source» explizit angegeben werden.

### Beispiel – Beschriftung eines Labels aus Resource Dictionary beziehen

```
<Label Content="{Binding Source={StaticResource NameLabelCaption}}" />
```

Kann natürlich mit Angabe von Path sowie Properties kombiniert werden. Auf Ressourcen ({StaticResource...}/{DynamicResource...}) oder statische Objekte ({x:Static....} bindbar.

### Relative Source

Ermöglicht die Angabe einer relativen Datenquelle im Visual Tree. Dafür ist eine eigene MarkupExtension vorhanden {RelativeResource....}.

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>- <b>Mode</b></li> <li>o FindAncestor</li> <li>o PreviousData</li> <li>o Self</li> <li>o TemplatedParent</li> </ul> | <ul style="list-style-type: none"> <li>Gibt Suchmodus an</li> <li>Sucht übergeordnetes Element des Typs (AncestorType)</li> <li>Bindet auf das vorhergehende Element</li> <li>Bindet auf das Element selbst</li> <li>Bindet auf Element, für welches Control Template gilt (nur innerhalb des Templates sinnvoll)</li> </ul> |
| <ul style="list-style-type: none"> <li>- <b>AncestorLevel</b></li> <li>- <b>AncestorType</b></li> </ul>  | <ul style="list-style-type: none"> <li>Gibt Vorgängerposition an, 1 = erster gefundener Vorgänger</li> <li>Typ des zu suchenden Vorgänger-Elements</li> </ul>  |

```
<Label Content="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType=Window}, Path=Title}" />
```

### ElementName

Nutzt direkt anderes benanntes XAML-Element als Datenquelle. Namen müssen im gleichen Namensraum vorliegen.

### Beispiel – Binden an Property eines anderen Elements

```
<TextBlock Name="MyText" Margin="10" ... />
<TextBlock Name="OtherText"
  Margin="{Binding ElementName=MyText, Path=Margin}" ... />
```

### DataContext

Property, welches in FrameworkElement definiert ist. Daher besitzen alle FrameworkElement abgeleiteten WPF-Klassen dieses Property (Window, TextBox, Label).

Setzt grundlegenden Kontext für Datenbindungsausdrücke. Beliebiges Objekt! ➔ zum Beispiel beliebiges ViewModel oder auch Objekte der Klasse selbst (this).

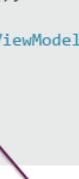
## Mobile and GUI Engineering

Datenbindungsausdrücke können anschliessend relativ zu diesem Kontext geschrieben werden. Beim Auswerten eines Datenbindungs ausdrucks wird im jeweiligen Control und in dessen Parent-Controls nach einem gesetzten Kontext gesucht (Kontext nicht null).

### Beispiel – Kontext für ganzes Fenster direkt im Code-Behind

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        var vm = new PersonViewModel();
        DataContext = vm;
    }
}
```



] Instanz erzeugen  
 ] Context des ganzen Fensters auf View Model setzen  
 ] Binding in XAML

<TextBlock Text="{Binding FirstName}" />

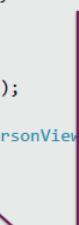
### Beispiel – Für ganzes Fenster direkt im Code-Behind, aber mit Fenster selbst als ViewModel

Für ganzes Fenster direkt im Code-Behind, aber mit Fenster selbst als ViewModel

```
public partial class MainWindow : Window
{
    public PersonViewModel MyViewModel { get; set; }

    public MainWindow()
    {
        InitializeComponent();

        MyViewModel = new PersonViewModel();
        DataContext = this;
    }
}
```



] Als Property zur Verfügung stellen  
 ] Instanz erzeugen  
 ] Context des ganzen Fensters auf das Fenster selbst setzen  
 ] Binding in XAML

<TextBlock Text="{Binding MyViewModel.FirstName}" />

### Beispiel – Für ganzes Fenster direkt im Code-Behind, aber mit Fenster selbst als ViewModel (Liste)

Für ganzes Fenster direkt im Code-Behind, aber mit Fenster selbst als ViewModel (Liste)

```
public partial class MainWindow : Window
{
    public ObservableCollection<PersonViewModel> PersonList { get; set; }

    public MainWindow()
    {
        InitializeComponent();

        var list = ...;
        PersonList = new ObservableCollection<PersonViewModel>(list);

        DataContext = this;
    }
}
```



] Property  
 ] Instanz erzeugen  
 ] Context setzen  
 ] Binding in XAML

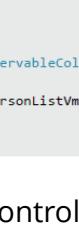
<ListBox ItemsSource="{Binding PersonList}" >...</ListBox>

### Beispiel – Für spezielle Control z.B. ListBox

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        var personList = ...;
        var personListVm = new ObservableCollection<PersonViewModel>(personList);

        myListBox.DataContext = personListVm;
    }
}
```



] Instanz erzeugen  
 ] Context der Liste setzen

Setzen auf gezieltes Control ist unüblich. => Bei jedem UI-Wechsel sind Anpassungen im Code Behind nötig. Stattdessen mit separaten ViewModels arbeiten nur nur DataContext für Fenster setzen.

## StringFormat

Umwandlung eines Wertes in einen String (zur Anzeige). Entspricht dem Format-String im Aufruf an die String.Format(...) Methode. Beispiel:

```
<TextBox Text="{Binding ScaleX, ElementName=WpfLabelScale, StringFormat={}{}{0:0.0}}" />
          ^_____
          string.Format("{0:0.0}", ScaleX)
```

## Bei MultiBindings

```
string.Format("{0} - Now only {1:C}", Description, Price)
```

---

```
<TextBlock>
  <TextBlock.Text>
    <MultiBinding StringFormat="{}{0} -- Now only {1:C}!">
      <Binding Path="Description"/>
      <Binding Path="Price"/>
    </MultiBinding>
  </TextBlock.Text>
</TextBlock>
```

 Normaler Format-String, bei welchem gebundene Properties als Argumente benutzt werden

Normaler Format-String, bei welchem gebundene Properties als Argumente benutzt werden.

## Path

Path ist die Standard-Property eines Binding-Ausdrucks. Folgende Ausdrücke sind daher identisch {Binding FirstName} und {Binding Path=FirstName}.

Für die Angabe der zu bindenden Property kann auch die Objektsyntax verwendet werden.

## Beispiele

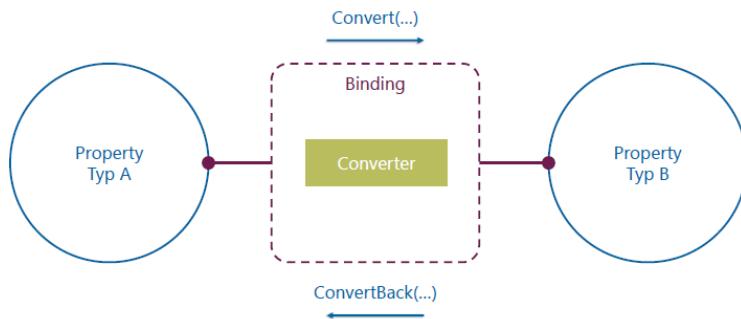
- {Binding} → Bindet an die Datenquelle selbst (nicht an deren Property)
- {Binding Address.Street} → Bindet an Property Street der Property Adresse der Datenquelle
- {Binding Groups[0].Name} → Bindet an Name des ersten Objekts in Groups-Auflistung der Datenquelle

## Mode

Mode ist die Richtung der Datenbindung. Der Default Mode ist je nach gebundenem Property unterschiedlich. Folgende Modis werden unterschieden:

- OneTime – Einmalig (bei erstem Zugriff)
- OneWay – Lesend → TextBlock.Text
- TwoWay – Lesend und schreibend → TextBox.Text
- OneWayToSource - Schreibend

Um sicher zu gehen, Mode jeweils explizit im Binding angeben



### Beispiel BooleanToVisibilityConverter

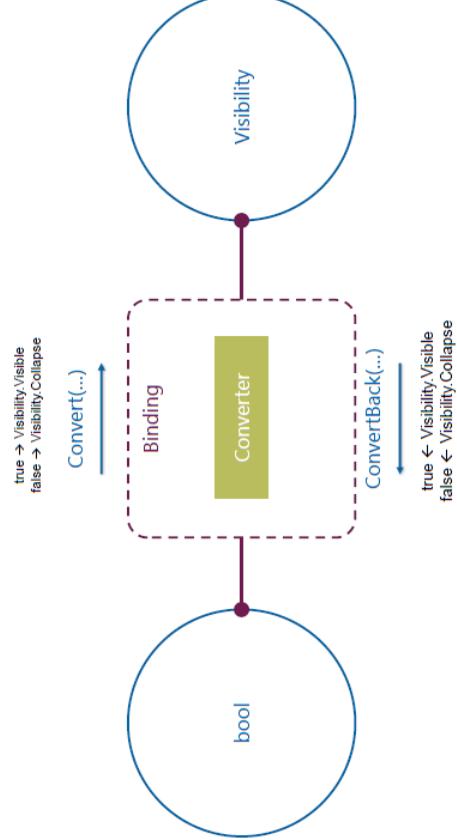
```

public sealed class BooleanToVisibilityConverter : IValueConverter
{
    /// <summary>
    /// Convert bool or Nullable<bool> to Visibility
    /// </summary>
    /// <param name="value">bool or Nullable</param>
    /// <param name="targetType">Visibility</param>
    /// <param name="parameter">null (not used)</param>
    /// <param name="culture">null (not used)</param>
    /// <returns>Visible or Collapsed</returns>
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return value is bool && (bool)value == true ? Visibility.Visible : Visibility.Collapsed;
    }

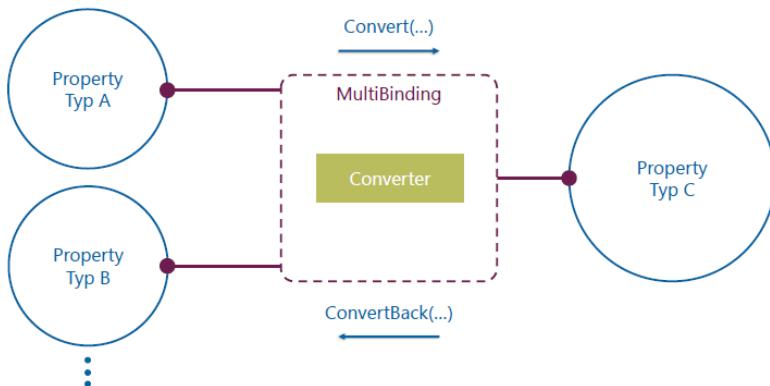
    ...

    /// <summary>
    /// Convert Visibility to boolean
    /// </summary>
    /// <param name="value">visibility value</param>
    /// <param name="targetType">bool</param>
    /// <param name="parameter">null (not used)</param>
    /// <param name="culture">null (not used)</param>
    /// <returns>true (Visible) or false</returns>
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return value as Visibility? == Visibility.Visible;
    }
}

```



### IMultiValueConverter



## Mobile and GUI Engineering

### Beispiel RgbToColorConverter

```

public class RgbToColorConverter : IMultiValueConverter
{
    public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)
    {
        if (values == null)
            return DependencyProperty.UnsetValue;

        // give some feedback about the usage
        if (values.Length != 3)
            throw new NotSupportedException($"3 values needed (R, G, B) but only {values.Length} given");

        // automatically throw another exception, if the passed values are not byte values
        var r = (byte)System.Convert.ToInt32(values[0]);
        var g = (byte)System.Convert.ToInt32(values[1]);
        var b = (byte)System.Convert.ToInt32(values[2]);
        return Color.FromRgb(r, g, b);
    }

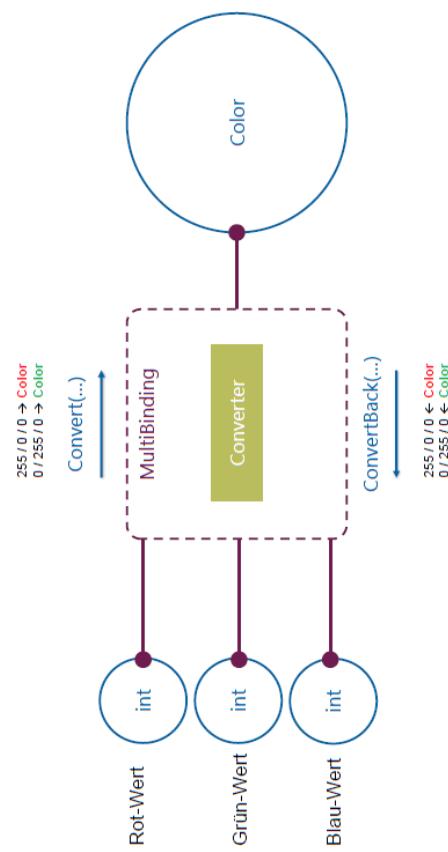
    ...
}

public object[] ConvertBack(object value, Type[] targetTypes, object parameter, CultureInfo culture)
{
    if (value == DependencyProperty.UnsetValue)
        return null;

    if (value is Color)
    {
        var color = (Color) value;
        var colors = new object[] {color.R, color.G, color.B};
        return colors;
    }

    return null;
}

```



## Nutzungshinweise IValueConverter & IMultiValueConverter

### Value Converter anwenden

Um einen Value Converter anzuwenden, wird eine Instanz benötigt.

#### Variante 1

Instanz als Ressource in XAML anlegen

#### Variante 2

Instanz als statische Property in C# bereitstellen

#### Variante 1

Instanz als Ressource anlegen

```

<Application.Resources>
    <local:BooleanToVisibilityConverter x:Key="MyVisibilityConverter" />
</Application.Resources>

```

Dann die Nutzung wie gehabt (z.B. für Binding der Visibility)

```
<Label Visibility="{Binding IsAvailable, Converter={StaticResource MyVisibilityConverter}}" />
```

Property des gesetzten  
Objekts gem. DataContext

Resource Key  
der Converter-Instanz

**Weiteres Beispiel****Value Converter anwenden**

z.B. in App.xaml

- Variante 1:  
Instanz als Ressource anlegen

```
<Application.Resources>
  <local:RgbToColorConverter x:Key="MyRgbToColorConverter" />
</Application.Resources>
```



- Dann: Nutzung wie gehabt, z.B.

```
<SolidColorBrush>
  <SolidColorBrush.Color>
    <MultiBinding Converter="{StaticResource MyRgbToColorConverter}"
      <Binding ElementName="ColorR" Path="Value"></Binding>
      <Binding ElementName="ColorG" Path="Value"></Binding>
      <Binding ElementName="ColorB" Path="Value"></Binding>
    </MultiBinding>
  </SolidColorBrush.Color>
</SolidColorBrush>
```

Resource Key  
der Converter-Instanz

} Binding auf Value Property  
der drei Slider-Controls

**Variante 2**

Instanz als statische Property in beliebiger Klasse (C#), zum Beispiel gleich im Value Converter selbst.

```
public class RgbToColorConverter : IMultiValueConverter
{
  // static field
  public static RgbToColorConverter Default = new RgbToColorConverter();

  // Alternative: lazily instantiated Singleton property
  private static RgbToColorConverter _instance;
  public static RgbToColorConverter Instance => _instance ?? (_instance = new RgbToColorConverter());
}
```

Dann Nutzung via {x:Static....} anstelle von {StaticResource}.

```
... Converter={x:Static local:RgbToColorConverter.Instance} ...
```

**Eigene Value Converters Implementieren?****Vorteile**

- XAML-Markup wird kürzer
- Entkoppelung von Wert und dessen Darstellungseigenschaften

**Aber... (Nachteile)**

- Setup Aufwand nötig
- Implementationsaufwand nötig
- Konversion evtl. auf Ebene ViewModel benötigt (damit zum Beispiel in Unit Tests möglich sind)

**Deshalb**

Gut abwägen, ob sich der Aufwand lohnt.

**Binding auf eigene Objekte**

INotifyPropertyChanged ist das wichtigste Interface beim Data Binding. Schreibt nur ein Event vor, das implementiert werden muss. Event Handler übermittelt Name der geänderten Property.

## Implementieren V1

```

public class Person : INotifyPropertyChanged
{
    private string _firstName;
    public string FirstName
    {
        get { return _firstName; }
        set
        {
            if (value != _firstName)
            {
                _firstName = value;
                OnPropertyChanged(nameof(FirstName));
            }
        }
    }
    // repeat pattern for other properties...
}

1 public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string name)
{
    var handler = PropertyChanged;
    if (handler != null)
        handler(this, new PropertyChangedEventArgs(name));
}

```

3 } Bei Änderung Methode aufrufen, die Event auslöst

1 } Event definieren

2 } Event auslösen (falls Abonnenten) → Thread-Safety!

## Implementieren V2

```

public class PersonV2 : INotifyPropertyChanged
{
    private string _firstName;
    public string FirstName
    {
        get { return _firstName; }
        set { SetProperty(ref _firstName, value, nameof(FirstName)); }
    }
    // repeat pattern for other properties...
}

1 public event PropertyChangedEventHandler PropertyChanged;
private void SetProperty<T>(ref T field, T value, string name)
{
    if (Equals(field, value))
        return false;

    field = value;
    var handler = PropertyChanged;
    if (handler != null)
        handler(this, new PropertyChangedEventArgs(name));
}

```

3 } Methode aufrufen, die auf Änderungen prüft

1 } Event definieren

2 } Auf Änderung prüfen und – falls ja – Event auslösen

## Implementieren V3

Via Basisklasse mit Infrastrukturmethoden ➔ Ableiten. Mit Zwang zum Ableiten sind keine POCOs mehr möglich.

```

1 public abstract class BindableBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected bool SetProperty<T>(ref T field, T value, string name = null)
    {
        if (Equals(field, value))
            return false;

        field = value;
        OnPropertyChanged(name);
        return true;
    }

    protected void OnPropertyChanged(string name = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
}

```

3 } Basisklasse mit Infrastruktur bereitstellen

1 } Notifikationen auch direkt zugreifbar machen

---

```

public class PersonV3 : BindableBase
{
    private string _firstName;
    public string FirstName
    {
        get { return _firstName; }
        set { SetProperty(ref _firstName, value, nameof(FirstName)); }
    }
    // repeat pattern for other properties...
}

```

3 } Von Basisklasse ableiten

1 } Methode aufrufen, die auf Änderungen prüft

## Problem

Änderung sollte immer dann kommuniziert werden, wenn Quellwerte ändern.

## Lösung

Für jeden Wechsel eine eigene Notifikation übermitteln

### Angepasste Code der Basisklasse

```
protected bool SetProperty<T>(ref T storage, T value, string name, [params string[]] otherNames)
{
    if (Equals(storage, value))
    {
        return false;
    }

    storage = value;
    OnPropertyChanged(name);
    foreach (var n in otherNames)
    {
        OnPropertyChanged(n);
    }
    return true;
}
```

// Nutzung innerhalb eines Property-Setters (Beispiel):  
SetProperty(ref \_firstName, value, prop, otherProp1, otherProp2, ...);

## Beispiel

```
private string _firstName;
public string FirstName
{
    get { return _firstName; }
    set { SetProperty(ref _firstName, value, nameof(FirstName), nameof(FullName)); } ↗ FirstName UND FullName ändern!
}

private string _lastName;
public string LastName
{
    get { return _lastName; }
    set { SetProperty(ref _lastName, value, nameof(LastName), nameof(FullName)); } ↗ LastName UND FullName ändern!
}

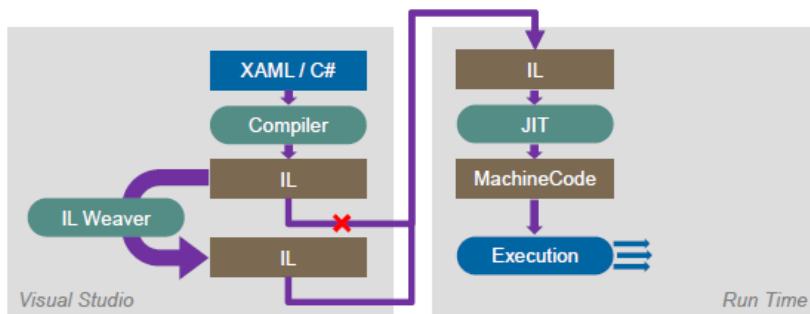
public string FullName => _firstName + " " + _lastName; ↗ Berechnete Property FullName
```

## Automatisch implementieren?

Erfordert eine Abhängigkeitsanalyse. Ist nicht trivial. Dies zeigt das Beispiel des FullNames. Bei der Änderung von FirstName muss auch die Änderung an FullName mitgeteilt werden. Bei der Änderung von LastName muss ebenfalls die Änderung an FullName mitgeteilt werden.

### INotifyPropertyChanged V4 (PropertyChanged.Fody)

Als weitere Möglichkeit kann ein Framework verwendet werden. Zum Beispiel Fody, welches auch via NuGet verfügbar ist. Es arbeitet mit IL-Weaving. Das Add-In «PropertyChanged.Fody» webt INotifyPropertyChanged ein.



## Vorgehen

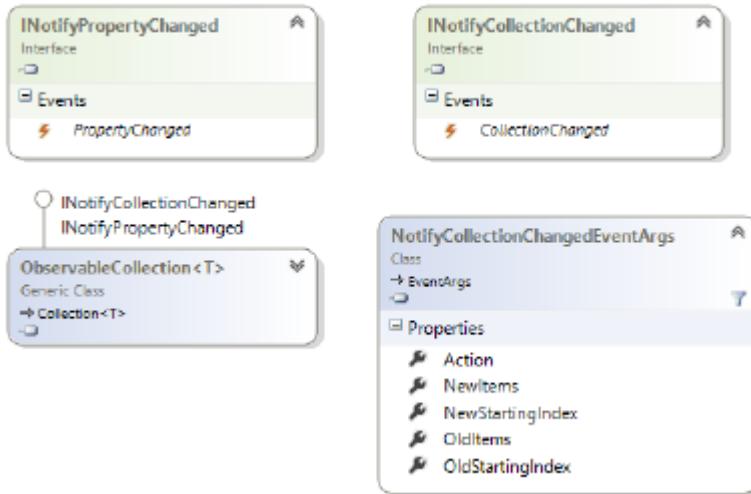
- POCO schreiben
- Mit Attribut «ImplementPropertyChanged» markieren, that's it ☺
- Der Rest wird mittels IL Weaving «eingewoben»

- Nachteil: Das Interface INPC auf Quellcode-Ebene nicht sichtbar, daher auch das Intellisense dazu fehlt!

```
[ImplementPropertyChanged]
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string FullName => FirstName + " " + LastName;
}
```

» Mit Attribut markieren

## ObservableCollection<T>



Implementiert **INotifyCollectionChanged** und **INotifyPropertyChanged**. Es meldet Änderungen an der Collection automatisch (Hinzufügen, Ersetzen, Verschieben und Entfernen). Event Handling ermöglicht Zugriff auf alte und neue Listeninhalte, sowie Listenposition.

## ObjectDataProvider

Ermöglicht die Verwendung einer Factory-Methode in XAML (**«ObjectType» + «MethodName»**). An den Konstruktor der Klasse können Parameter übergeben werden (**«ConstructorParameters»**). An die Methode können ebenfalls Parameter gebunden werden (**«MethodParameters»**).

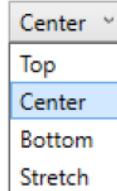
### Anwendungsbeispiel – Binding von Enums

```
<Window.Resources>
  <ObjectDataProvider x:Key="Alignments"
    MethodName="GetNames"
    ObjectType="{x:Type sys:Enum}">
    <ObjectDataProvider.MethodParameters>
      <x:Type TypeName="VerticalAlignment" />
    </ObjectDataProvider.MethodParameters>
  </ObjectDataProvider>
</Window.Resources>
```

Verwendungsbeispiel:

```
<ComboBox ItemsSource="{Binding Source={StaticResource Alignments}}" />
```

Resultat:

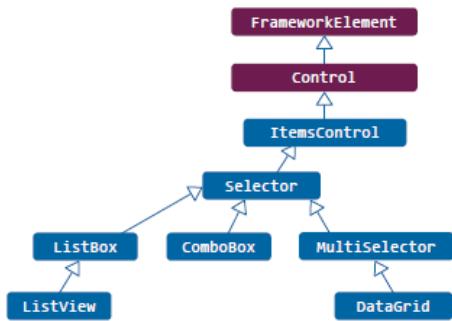


## Debugging Data Binding

Data Binding ist eines der mächtigsten Konzepte in WPF. Es wird zur Laufzeit ausgewertet. Dabei wird es keine Exceptions, sondern macht in der Regel einfach nicht, wenn das Binding fehlschlägt. Manchmal ist es schwierig Fehler zu finden.

## Hauptfehlerquellen

- Ausdruck ist ungültig
- Ausdruck ist gültig, aber es erscheint nicht das erwünschte Resultat



## ItemsControl

Ist die Basisklasse für ListBox, ComboBox und Datagrid. Vergleichbar mit dem «ControlTemplate».

ItemsControl (Ohne Selektierungsmöglichkeit)

- |                        |   |
|------------------------|---|
| - Items                | Gibt hart codierte Liste von Items an         |
| - ItemsSource          | Datenquelle / Data Binding Ausdruck für Items |
| - ItemTemplate         | Template für die Darstellung eines Items      |
| - ItemTemplateSelector | Selector für das Template eines Items         |

## Variante 1

Angabe des Template direkt im Control.

```

<ListBox ItemsSource="{Binding Source={StaticResource myList}}"
        HorizontalContentAlignment="Stretch">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel>
        <TextBlock Text="{Binding TaskName}" />
        <TextBlock Text="{Binding Description}" />
        <TextBlock Text="{Binding Priority}" />
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
  
```

} Wichtig, damit Items die ganze Breite ausfüllen

} Gibt Aussehen eines Items vor

## Variante 2 (Besser)

Ablage als (wiederverwendbare) Ressource

```

<DataTemplate x:Key="myTaskTemplate" DataType="local:Task">
  <DockPanel HorizontalAlignment="Stretch">
    <TextBlock Text="{Binding TaskName}" />
    <TextBlock Text="{Binding Description}" />
    <TextBlock Text="{Binding Priority}" />
  </DockPanel>
</DataTemplate>
  
```

} In einem Resource Dictionary

Die Verwendung geht dann via {StaticResource....}

```

<ListBox ItemsSource="{Binding Source={StaticResource myList}}"
        ItemTemplate="{StaticResource myTaskTemplate}"
        HorizontalContentAlignment="Stretch" />
  
```

Von Microsoft ursprünglich nicht in WPF integriert. «Die WPF ist für Anwendungen mit einer hohen Usability und User Experience ausgelegt. Ein Data Grid erfüllt keines dieser beiden Merkmale» - Microsoft. Später wurde es dann nachgeliefert, um die Lücke zu schliessen.

### Merkmale

- Kann Spalten autogenerieren (AutoGenerateColumns)
- Spalten-Layout kann anhand vordefinierter Spalten-Klassen oder eigener Spalten-Templates erfolgen
- Grouping wird unterstützt
- Erweitertes Markieren wird unterstützt (mehrere Spalten/Zeilen/Zellen)
- Es kann ein RowDetailTemplate angegeben werden → stellt für ausgewählte Zeile zusätzliche Detail View dar
- Sortierung erfordert zusätzliches Event-Handling

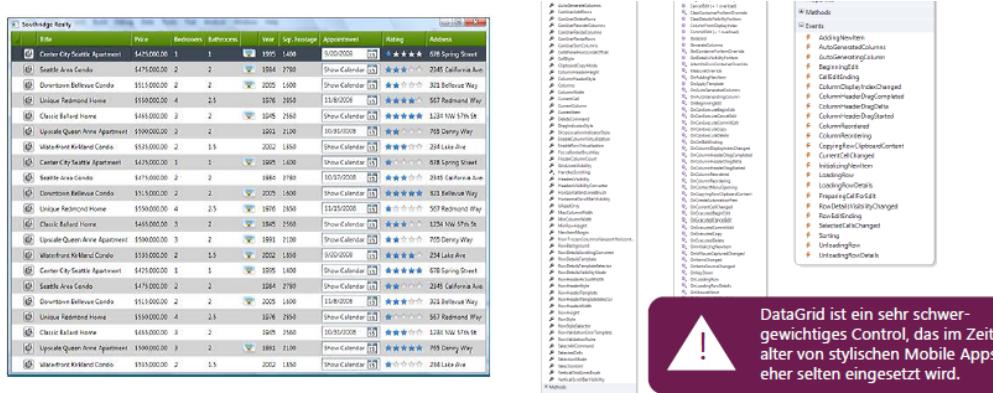
Beispiel – mit autogenerierten Spalten

```
<DataGrid ItemsSource="{Binding Customers}" />
```

Beispiel – mit eigenen Spalten

```
<DataGrid ItemsSource="{Binding Customers}" AutoGenerateColumns="False" >
    <DataGrid.Columns>
        <DataGridTextColumn Header="First Name" Binding="{Binding FirstName}" />
        </DataGrid.Columns>
        <DataGridTemplateColumn Header="Image" Width="SizeToCells" IsReadOnly="True">
            <DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <Image Source="{Binding Image}" />
                </DataTemplate>
            </DataGridTemplateColumn.CellTemplate>
        </DataGridTemplateColumn>
    </DataGrid.Columns>
</DataGrid>
```

1. Spalte  
2. Spalte



Address	Rating	Price	Bedrooms	Bathrooms	Year	Sq.Footage	Apartments
Center City Studio Apartment	4.5	\$425,000.00	1	1	1995	1200	0
Gentle Area Condo	4.5	\$175,000.00	2	2	1984	2700	0
Overtown Bellview Condo	5.0	\$115,000.00	2	2	2005	1600	0
Unique Redmond Home	5.0	\$150,000.00	4	2.5	1976	2650	11/05/2008
Classic Ballard Home	4.5	\$145,000.00	3	2	1985	2500	0
Upcale Queen Anne Apartment	5.0	\$900,000.00	2	2	1991	2100	10/15/2008
Waterfront Kirkland Condo	5.0	\$135,000.00	2	1.5	2002	1850	0
Center City Studio Apartment	4.5	\$175,000.00	1	1	1995	1400	0
Gentle Area Condo	4.5	\$175,000.00	2	2	1984	2100	0
Overtown Bellview Condo	5.0	\$115,000.00	2	2	2005	1600	0
Unique Redmond Home	5.0	\$150,000.00	4	2.5	1976	2850	11/15/2008
Classic Ballard Home	4.5	\$145,000.00	3	2	1985	2500	0
Upcale Queen Anne Apartment	5.0	\$900,000.00	3	2	1991	2100	10/15/2008
Waterfront Kirkland Condo	5.0	\$135,000.00	2	1.5	2002	1850	0
Center City Studio Apartment	4.5	\$175,000.00	1	1	1995	1400	0
Gentle Area Condo	4.5	\$175,000.00	2	2	1984	2700	0
Overtown Bellview Condo	5.0	\$115,000.00	2	2	2005	1400	11/05/2008
Unique Redmond Home	5.0	\$150,000.00	4	2.5	1976	2900	0
Classic Ballard Home	4.5	\$145,000.00	3	2	1985	2600	11/15/2008
Upcale Queen Anne Apartment	5.0	\$900,000.00	3	2	1991	2100	0
Waterfront Kirkland Condo	5.0	\$135,000.00	2	1.5	2002	1850	0

! DataGrid ist ein sehr schwerwiegiges Control, das im Zeitalter von stylischen Mobile Apps eher selten eingesetzt wird.

**CollectionViewSource**

Ermöglicht das Gruppieren, Sortieren und Filtern

**Properties (nur ein Auszug)**

- **SortDescriptions** Sortiermerkmale
- **Source** Quelldatenobjekt (eine ObservableCollection<T>)
- **Filter** Event, das beim Filtern ausgelöst wird. => Filterung muss mit eigenem Code erledigt werden

```
<CollectionViewSource x:Key="ListingDataView"
    Source="{Binding Source={x:Static Application.Current}, Path=AuctionItems}" />
```

**Sonstiges**

Weitere UI Features im Kontext von Data Binding (nicht besprochen).

**ListView**

Control, welches eine spaltenweise Darstellung der Listenelemente ermöglicht.

**StyleSelector**

Delegate, mit welchem ein Style dynamisch anhand von Merkmalen der Daten ausgewählt werden kann. Ermöglicht Spezialdarstellung für Objekte mit speziellen Merkmalen.

**ItemTemplateSelector**

Eigene Klasse, die ein Template anhand einer Logik dynamisch auswählt. Ermöglicht komplett anderes Layout für Objekte mit speziellen Merkmalen.

# Property Konzept C#

## OOP: Properties

- Property (engl.) == «Eigenschaft», «Besitz»
- Property in OOP = Beschreibende Eigenschaft einer Klasse von Objekten
- Setzen von Werten in der Regel mit Getter und Setter-Methoden realisiert
- Field = Feld, Datenfeld (=Klassenvariable)
- Backing field = Klassenvariable, in welcher der Wert einer Property (zwischen-)gespeichert wird.

## Vorteile

- Kontrolle darüber, welche Werte gesetzt werden können
- Bessere Kapselung

## Java Properties

Property Konzept nicht nativ in der Sprache verfügbar. Realisierung mittels Getter- und Setter-Methoden.

## Beispiel und Nutzung

```
public class Person {
    // backing field for Name property
    // (private, non-accessible from outside)
    private String name;

    // getter for Name property (public)
    public String getName() {
        return name;
    }

    // setter for Name property (public)
    public void setName(String newName) {
        name = newName;
    }
}
```

Selbstgewählter Name für den Methodenparameter des Setters

```
public class PersonExampleUsage {
    public void runDemo() {
        Person p = new Person();
        p.setName("Max");

        System.out.println("The person's name is:");
        System.out.println(p.getName());
    }
}
```

## C# Properties

Property Konzept NATIV in der Sprache verfügbar. Spezielle Syntax für Setter- und Getter-Methoden verfügbar.

## Beispiel und Nutzung

```
public class Person {
    // backing field for Name property
    // (private, non-accessible from outside)
    private string _name;

    // property definition (public)
    public string Name {
        get { return _name; }
        set { _name = value; }
    }
}
```

Vordefiniertes Schlüsselwort, welches die Rolle des Methodenparameters Für die Setter-Methode übernimmt

```
public class PersonExampleUsage {
    public void runDemo() {
        Person p = new Person();
        p.Name = "Max";           Zuweisung an Property
                                (=Setter wird aufgerufen)

        System.Console.WriteLine("The person's name is:");
        System.Console.WriteLine(p.Name);
    }
}
```

Lesen der Property  
(=Getter wird aufgerufen)  
→ OHNE Klammern

## Review C# Events

Publish / Subscribe-Mechanismus, Event in Klasse definieren, Event innerhalb der Klasse auslösen ("Publish"), Für Benachrichtigungen via Event registrieren («Subscribe»).

Beispiel

### EventArgs

Von EventArgs angeleitete Klasse für Übermittlung der Event-Details. Konvention: Klassenname endet immer auf «EventArgs».

```
public class PropertyChangedEventArgs : EventArgs
{
    public stringPropertyName { get; private set; }

    public PropertyChangedEventArgs(string propertyName)
    {
        PropertyName = propertyName;
    }
}
```

### Publish

Innerhalb der Klasse «Clock» senden wir allen Subscribers eine Nachricht, wenn zum Beispiel die Zeit ändert.

```
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
```



### Subscribe

UI meldet sich für das PropertyChanged-Event automatisch an (für alle Klassen, die von INPC ableiten). Wir können aber auch selbst subscriben. (z.B. innerhalb von OnStartup von App.xaml.cs)

```
Clock.PropertyChanged += OnPropertyChanged;
```

- ... brauchen dann einfach eine Methode mit kompatibler Signatur, um auf das Event zu reagieren:

```
private void OnPropertyChanged(object sender, PropertyChangedEventArgs args)
{ ... }
```

### Beispiel

- Beispiel: Ausgabe des geänderten Wertes auf Konsole:

```
Clock.PropertyChanged += OnPropertyChanged;
```

Name meines Event-Handlers

Signatur (=Parameterliste) muss Delegate gem. Event-Definition entsprechen

- Event Handler:

```
private void OnPropertyChanged(object sender, PropertyChangedEventArgs args)
{
    // extract the property name out of the event args parameter
    var propertyName = args.PropertyName;
    // get the new value of the mentioned property via reflection
    var newValue = sender.GetType().GetProperty(args.PropertyName).GetValue(sender);
    // write to console
    Console.WriteLine($"The property {propertyName} has changed to {newValue}");
}
```

Oft kein Unsubscribe nötig, deshalb meist Subscribe per Lambda. Anstelle von «sender» und «args» werden oft der Kürze wegen einfach «s» oder «o» und «a» genutzt.

```
Typ «object» Typ «EventArgs» oder ein davon abgeleiteter Typ, passend zum Event
Clock.PropertyChanged += ( o, a ) => {
};
```

Kürzestform, falls nur einziger Methodenaufruf im Lambda-Event-Handler (keine {}-Klammerung nötig).

```
Clock.PropertyChanged += (o, a) => MyMethodCall(...);
```

## Beispiel

```
Clock.PropertyChanged += (o, a) => System.Console.WriteLine("The property has changed");
```

## Subscribe via Lambda-Syntax (kürzer)

Lambda-Syntax → anonyme Event-Handler Methoden direkt dem Event zuweisen

**Beispiel** Ausgabe des geänderten Wertes auf Konsole

```
Typ «object» Typ «PropertyChangedEventArgs»
Clock.PropertyChanged += (sender, args) =>
{
    // extract the property name out of the event args parameter
    var propertyName = args.PropertyName;
    // get the new value of the mentioned property via reflection
    var newValue = sender.GetType().GetProperty(propertyName).GetValue(sender);
    // write to console
    Console.WriteLine($"The property {propertyName} has changed to {newValue}");
};
```

■ Nachteil Lambda-Syntax:

■ → kein Unsubscribe möglich

mit sep. Event-Handler-Methode ginge Unsubscribe per «-=»-Operator:  
Clock.PropertyChanged -= OnPropertyChanged

## Vollständiges Beispiel «Clock» (basierend auf Lösungsvorschlag Ü4)

```
public partial class App : Application
{
    public DispatcherTimer Timer { get; set; }
    public Clock Clock { get; set; }

    protected override void OnStartup(StartupEventArgs e)
    {
        Clock = new Clock();

        Timer = new DispatcherTimer();
        // Jede Sekunde das Tick-Event auslösen:
        Timer.Interval = TimeSpan.FromSeconds(1);
        Timer.Tick += (sender, args) =>
        {
            // Do something...
            Clock.Time = DateTime.Now;
        };
        Timer.Start();

        MainWindow = new MainWindow();
        MainWindow.DataContext = Clock;
        MainWindow.Show();
    }

    Clock.PropertyChanged += OnPropertyChanged;
}
```

### Ausgabe:

```
The property Time has changed to 04.12.2016 14:31:47
The property TimeString has changed to 04.12.2016 14:31:47
The property Time has changed to 04.12.2016 14:31:48
The property TimeString has changed to 04.12.2016 14:31:48
The property Time has changed to 04.12.2016 14:31:49
The property TimeString has changed to 04.12.2016 14:31:49
The property Time has changed to 04.12.2016 14:31:50
The property TimeString has changed to 04.12.2016 14:31:50
-
```

```
private void OnPropertyChanged(object sender, PropertyChangedEventArgs args)
{
    // extract the property name out of the event args parameter
    var propertyName = args.PropertyName;
    // get the new value of the mentioned property via reflection
    var newValue = sender.GetType().GetProperty(propertyName).GetValue(sender);
    // write to console
    Console.WriteLine($"The property {propertyName} has changed to {newValue}");
}
```

## WPF

UI Library stellt sehr, sehr viele Events zur Verfügung. Ansatzpunkte für eigene Aktionen zum richtigen Moment.

## Beispiele

- Wenn App startet (Application.Startup)

- Wenn Fenster geschlossen wird (Windows.Closing und Window.Closed)
- Wenn Text in TextBox markiert (TextBox.SelectionChanged) oder geändert (TextBox.TextChanged) wird.
- Wenn Wert eines Slider-Controls geändert wird (Slider.ValueChanged)

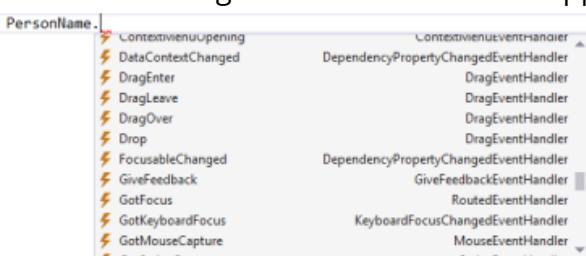
## C# Events in WPF

Welche Events verfügbar sind, können über MSDN/Google herausgefunden werden.

Oder auch via **Object-Browser** (→ Menu View) → WPF-Controls sind im Namespace «System.Window».

Achtung oft sind die Events in Basisklassen implementiert. Beispiel: TextBox => Events «SelectionChanged» und «TextChanged» in Basisklasse «TextBoxBase» verfügbar.

Eine weitere Möglichkeit ist direkt beim Tippen (Intellisense) nachzuschlagen.



## C# Events vs. Virtual Event Handlers

Vereinfachte Event-Nutzung in .NET-Frameworks wie WPF. Sie stellen neben den Events oft auch vordefinierte, virtuelle Event Handler Methoden zur Verfügung. (→ virtual = in abgeleiteter Klasse überschreibbar, Schlüsselwort «override» in C#).

## Konventionen

- Name des vordefinierten Event Handlers zu einem Event «On[NameDesEvents]»
- Methodenparameter: in der Regel auf den EventArgs-Parameter reduziert und daher ohne den «sender»-Parameter
- Zugriffskategorie: meist «protected» daher nur innerhalb gleicher Klassenhierarchie nutzbar

## Beispiele

Klasse	Event	Vordefinierter Event Handler
Application	Startup	OnStartup
Window	Closing	OnClosing
	Closed	OnClosed
TextBox	SelectionChanged	OnSelectionChanged
Slider	ValueChanged	OnValueChanged

## Beispiel Application.Startup

Variante Event → z.B. im Konstruktor subscriben

```
public partial class App : Application
{
    public App()
    {
        this.Startup += (o, args) =>
        {
            ...
        };
    }
}
```

Variante vordefinierter Handler → Methode «OnStartup» aus Basisklasse «Application» überschreiben

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        ...
    }
}
```

// In Basisklasse wird der vordefinierte Handler  
// bereits für das Event registriert, daher in  
// eigenem Code direkt via Handler nutzbar:  
this.Startup += (o, args) => OnStartup(args);

→ Framework-Code WPF

# Der WPF App Lifecycle

## App Startup

### Zur Erinnerung Java

```
class HelloWorldApp {
    public static void main(string[] args) {
        System.out.println("Hello WPF");
    }
}
```

Dazu muss eine statische main()-Methode festgelegt werden. Zum Starten den Klassennamen angeben. → Java VM ruft die main()-Methode von HelloWorldApp auf.

### C#(.NET) generell

Statische Main()-Methode festlegen → Grossbuchstabe «M» (= einzige public static Main() pro Assembly oder Startup Object in Build-Settings).

```
class HelloWorldApp {
    public static void Main(string[] args) {
        Console.WriteLine("Hello WPF");
    }
}
```

Zum Starten den Programmnamen (gem. Projekteinstellungen) ohne «.exe» angeben / Doppelklicken.

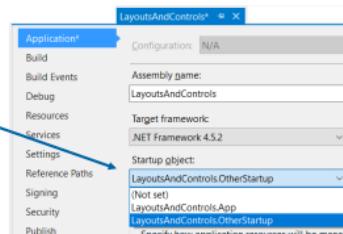
>HelloWorldApp

 HelloWorldApp.exe

1 Main Methode in Assembly (.dll oder .exe) → wird automatisch durch den Compiler als Startmethode festgelegt (Einstellung «Startup Object»: «No set»).

Mehrere Main()-Methoden in Assembly?

→ Hauptklasse («Startup Object») in Build-Settings festlegen

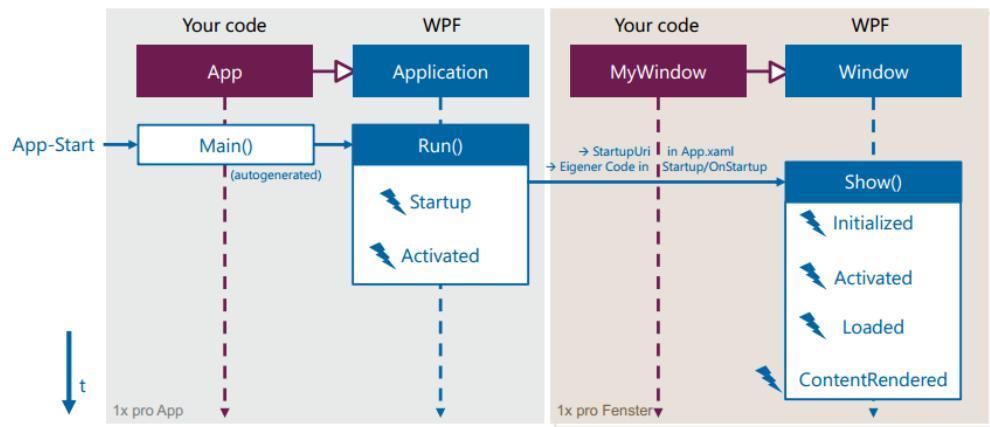


## WPF App

Autogenerierte (partial) Klasse, wird von Visual Studio im Hintergrund erzeugt.

```
public partial class App : System.Windows.Application
{
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
    public void InitializeComponent()
    {
        this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);
    }
    [System.STAThreadAttribute()]
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
    public static void Main()
    {
        LayoutsAndControls.App app = new HelloWpf.App();
        app.InitializeComponent();
        app.Run();
    }
}
```

Der «normale» Einsprungspunkt



## App Startup Beispiele

### Variante «StartupUri»

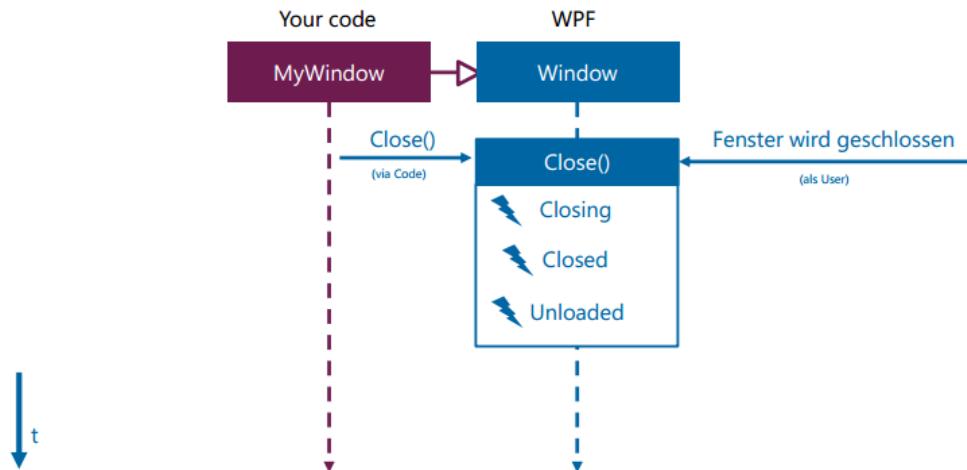
Kein manuelles Erzeugen des Fensters im Code Behind nötig.

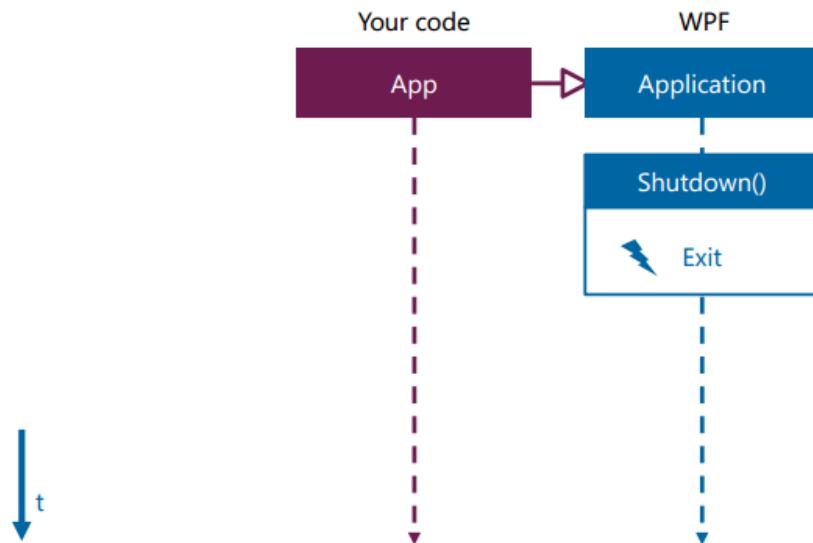
```
<Application x:Class="HelloWpf.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:HelloWpf"
    StartupUri="MainWindow.xaml">
<Application.Resources>
</Application.Resources>
</Application>
```

### Variante «OnStartup» im Code Behind

```
protected override void OnStartup(StartupEventArgs e)
{
    MainWindow = new MainWindow();
    MainWindow.Show();
}
```

## Window Close

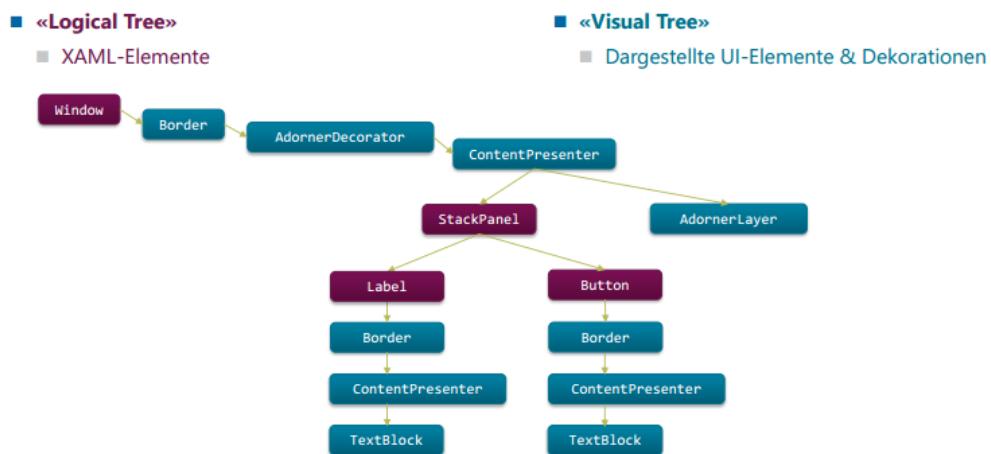




## Details der Benutzerinteraktion

### Review

#### Logical und Visual Tree



#### “Routed Events”

##### XAML UI Event

Interessante UI Ereignisse, auf die reagiert werden soll. Zum Beispiel

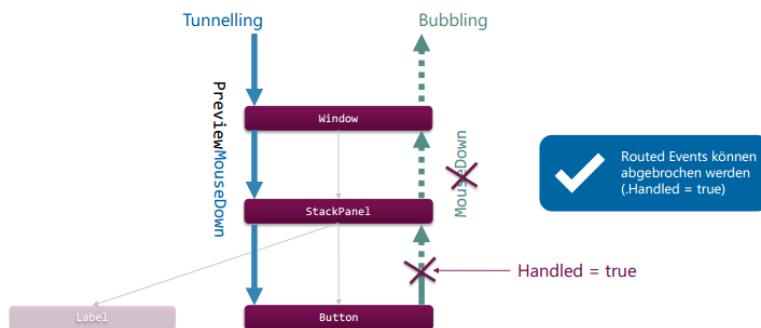
- Mouse-Ereignis, zum Beispiel Linke Maustaste gedrückt/losgelassen, Maus bewegt etc.
- Keyboard-Ereignis, zum Beispiel Taste B gedrückt oder CRTL losgelassen etc.
- Touch-Ereignis zum Beispiel Berühren oder Loslassen

Sie können abgebrochen werden und nutzen das C# Event Konzept mit Publish und Subscribe.

Werden abwärts (Preview/Tunnelling) und dann aufwärts (Bubbling) durch den Visual Tree gesendet.  
Meiste Routed Events in der Basisklasse «UIElement» verfügbar.

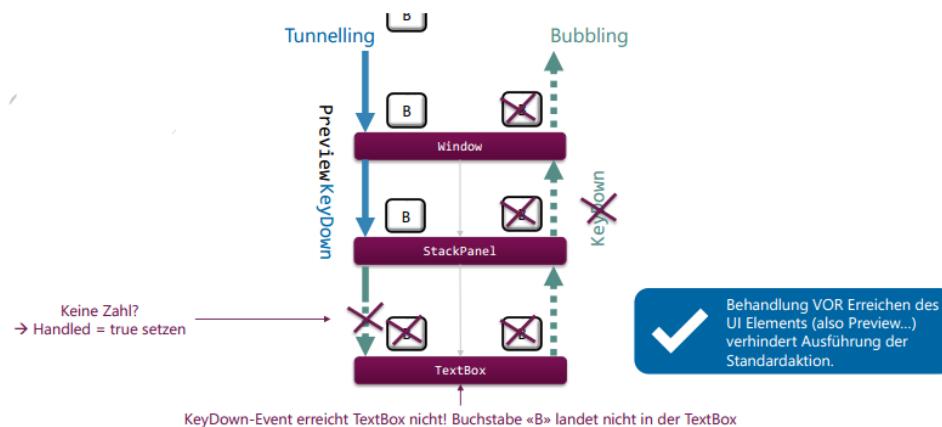
## Beispiel Button

Klick auf «Button», Event «MouseDown»



## Beispiel (TextBox)

z.B. nur Zahlen zulassen, Drücken der Taste «B», Event «KeyDown»



## Routed Events behandeln

### Möglichkeit A

Beim UI Element selbst.

- Beispiel: Texteingabe in TextBox

```
<TextBox Name="Amount">
    <!-- PreviewKeyDown="Amount_OnPreviewKeyDown" -->
    <!-- KeyDown="Amount_OnKeyDown" />
```

- Im Code-Behind:

```
private void Amount_OnPreviewKeyDown(object sender, KeyEventArgs e)
{
    ...
}
private void Amount_OnKeyDown(object sender, KeyEventArgs e)
{
    ...
}
```

### Möglichkeit B

Bei einem Parent Element, aber mit dem konkreten Event.

- Beispiel: Button Klick (resp. PreviewMouseDown)

```
<StackPanel PreviewMouseDown="StackPanel_OnPreviewMouseDown">
    ...
    <Button Name="SaveButton">
        <!-- PreviewMouseDown="SaveButton_OnPreviewMouseDown" -->
        <!-- MouseDown="SaveButton_OnMouseDown" >Save</Button>
    ...
</StackPanel>
```

## Möglichkeit B – Alternative

Bei einem Parent Element – falls Event dort nicht direkt verfügbar → «AttachedEvent». Click wird als kombiniertes Event von MouseDown und MouseUp ausgelöst. Intellisense für Attached Events ist nicht verfügbar.

- Beispiel: Button Klick

```
<StackPanel>
  <Button Click="StackPanel_OnClick">
    ...
    <Button Name="SaveButton" Click="SaveButton_OnClick">Save</Button>
    ...
  </StackPanel>
```

## Interessante Routed Events

### Ein genauer Blick – EventArgs

#### Maus

MouseDown, MouseUp, MouseMove

#### Keyboard

KeyDown, KeyUp, TextInput

#### Touch

TouchDown, TouchUp, TouchMove

### Viele weitere Events

Sind zu finden in den Appendix-Folien zur Vorlesung

### RoutedEventArgs

Von EventArgs abgeleitet.

### Eigenschaften

- Handled Flag, das aussagt, ob das Event behandelt wurde (Boolean)
- OriginalSource Quell-Element (→ Hit-Testing), welches das Event ausgelöst hat
- RoutedEvent Das Routed Event, welches mit diesem Objekt verknüpft ist
- Source Element, welches das Event rapportiert hat

Das Quell-Element kann durch WPF angepasst werden (Source vs. OriginalSource). Die OriginalSource ist das U.U. Kind-Element des Source-Element, welches per Hit-Testing als 1.-Adressat des Events bestimmt wird. Die Source ist das Element im LogicalTree, welches das Event dann rapportiert.

### EventArgs der verschiedenen Event-Gruppen

Von RoutedEventArgs abgeleitet.

### Maus (Auswahl)

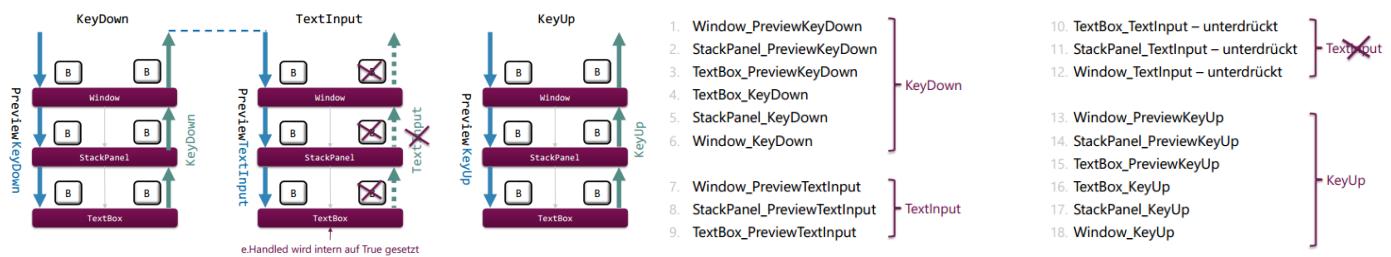
- LeftButton Zustand der linken Maustaste (Pressed/Released)
- MiddleButton Zustand der mittleren Maustaste (Pressed/Released)
- RightButton Zustand der rechten Maustaste (Pressed/Released)
- Delta Mausradbewegung (immer 120/-120)

### Keyboard (Auswahl)

- Key Taste (Enum)
- IsDown Gibt an, ob die Taste gedrückt ist
- IsUp Gibt an, ob die Taste nicht gedrückt ist
- IsRepeat Gibt an, ob es ein wiederholtes Event ist
- SystemKey Gibt bei gedrückter ALT-Taste die zusätzlich gedrückte Taste an (in Key steht dann Key.System)

## Ein genauer Blick – Keyboard-Events

Die Taste «B» wird auf dem Keyboard gedrückt und wieder losgelassen. Diese Abbildungen stellen das Standardverhalten dar. Die Abbildung tut dies in Listenform.



## Sonstige Events

### ■ Control (und abgeleitete)

- MouseDoubleClick Doppelklick
- PreviewMouseDoubleClick Preview-Event zu Doppelklick

### ■ TextBox

- SelectionChanged Die Markierung hat geändert
- TextChanged Der Inhalt hat geändert

### ■ ListBox, ComboBox

- SelectionChanged Selektiertes Element hat geändert

## Achtung! Routed Events!

Gleiches Routed Event oft in verschiedenen Controls im Logical Tree abrufbar. EventArgs geben Aufschluss über den Auslöser des Routed Events.

## Beispiel Problemstellung

Wir haben einen TabControl, welcher vom Selector ableitet. Zusätzlich haben wir eine verschachtelte ListBox (oder anders, von Selector abgeleitetes Control). Unser Plan ist es ein Lazy (Re-) Loading der Listeneinträge zu implementieren, sobald Tab 0 «Courses» aktiviert wird.

```
<Grid>
<TabControl Name="MainTabControl"
    SelectionChanged="MainTabControl_SelectionChanged">
    <TabItem Header="Courses">
        <TabItem.Content>
            <ListBox x:Name="ListBox1" />
        </TabItem.Content>
    </TabItem>
    <TabItem Header="Classrooms" >
    </TabItem>
</TabControl>
</Grid>
```



Quelle: <http://blogs.u2u.be/diederik/post/2009/09/09/In-WPF-SelectionChanged-does-not-mean-that-the-selection-changed.aspx>

## Erste Idee

Wir machen einen SelectionChanged-EventHandler, welcher für den Tab abgefangen wird. Darin dann die Liste neu laden und ersten Eintrag auswählen. Dies klappt leider nicht.

```
private void MainTabControl_SelectionChanged(object sender,
                                            SelectionChangedEventArgs e)
{
    if (this.MainTabControl.SelectedIndex == 0)
    {
        // (Re-)Populate ListBox
        this.ListBox1.Items.Clear();
        this.ListBox1.Items.Add("...");
```

■ Frage: Klappt das?

## Zweite Idee

Ein hilfloser Rettungsversuch. Wir wählen den ersten Listeneintrag nicht aus. Es klappt aber immer noch nicht.

```
private void MainTabControl_SelectionChanged(object sender,
                                            SelectionChangedEventArgs e)
{
    if (this.MainTabControl.SelectedIndex == 0)
    {
        // (Re-)Populate ListBox
        this.ListBox1.Items.Clear();
        this.ListBox1.Items.Add("...");
```

## Korrekte Lösung

Quelle des Events prüfen

```
private void MainTabControl_SelectionChanged(object sender,
                                            SelectionChangedEventArgs e)
{
    if (e.OriginalSource == this.MainTabControl)
    {
        if (this.MainTabControl.SelectedIndex == 0)
        {
            // (Re-)Populate ListBox
            this.ListBox1.Items.Clear();
            this.ListBox1.Items.Add("...");
```

## Event Handling vs. Data Binding

Gleichartiges Event kann für verschiedenen Controls ausgelöst werden. Die Quelle des Events ist entscheidend. Via Data Binding können solche unerwarteten Situationen nicht im gleichen Massen entstehen. Falls es möglich ist, soll Data Binding dem EventHandling in jedem Fall vorgezogen werden.

## Multitouch

Die Multitouch-API erfordert natürlich einen Touchscreen. Es nutzt darunterliegendes Multitouch-API von Windows. Abstrahiert komplexe Touch-Interpretationen und Berechnungen. Opt-In: Via IsManipulationEnabled=>True< für jedes Element einschalten!

## Achtung

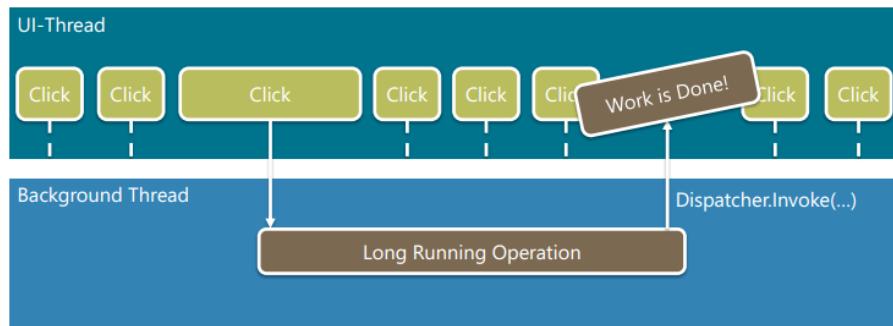
Die Controls müssen grösser dargestellt werden, damit diese bequem angewählt werden können. Touch kennt zudem kein Mouseover/Hover.

Ist in WPF mit 3 Phasen implementiert.

1. Maustaste wird gedrückt
  - a. Entwickler muss nun selbst beurteilen, ob und wann der Drag & Drop-Vorgang gestartet werden soll
  - b. In der Regel erst nach dem Überschreiten einer Mindestdistanz (Schwellwert) bei gedrückter Maustaste
2. Während des Ziehens (Drag)
  - a. Steuerelemente unterhalb des Mauszeigers müssen melden, ob Sie als Ziel in Frage kommen.
3. Fallenlassen (Drop)
  - a. Steuerelemente unterhalb des Mauszeigers müssen eine Aktion mit dem bewegten Objekt durchführen.

## Hintergrund-Operationen

### Review



### Im Hintergrund Daten transferieren

Ist gilt für lange dauernde Operationen. Beispiele dafür sind Berechnungen, Download, Aufbereitung und DB-Zugriff.

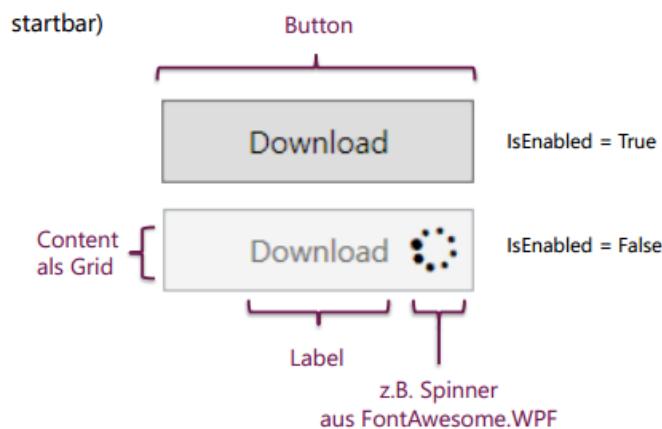
### Vorgehen

1. (Optional) Visuelles Feedback geben => Spinner, Overlay, Popup, ....
2. Starten eines Background-Threads als Reaktion auf ein Event, Kontrolle an UI Thread zurückgeben
3. Bei Thread-Ende aus dem Background-Thread den UI-Thread benachrichtigen => Dispatcher.Invoke(...).

Button deaktivieren (damit Aktion nicht nochmal startbar) oder einen Spinner anzeigen.

### Beispiel

- Button intern mit Grid darstellen (Control Template)
- Darin Label anzeigen
- Darin rechts aussen einen Spinner anzeigen
  - o Visibility an IsEnabled des Buttons binden
  - o Eigener Converter nötig (da vordefinierter Converter gerade anders herum arbeitet)



### Dann

In der Aktion die IsEnabled-Property des Buttons auf False setzen. Das Data Binding würden den Spinner dann automatisch einblenden.

### Schritt 2 – Starten eines Background-Threads

Task Parallel Library (TPL) verwenden! Der einfachste Weg ist die statische Methode Task.Run(...)

```
Lambdas! ☺
Task.Run(() =>
{
    // Ich laufe in einem Background-Thread, nice! :-)
    ...
    // Operationen hier laufen somit parallel zum UI Thread, ACHTUNG!!!
});
// UI-Thread läuft hier gleich weiter
```



### Schritt 3 – UI-Thread benachrichtigen

Falls der Dispatcher nicht aus Code-Behind aufgerufen (Dispatcher ist ein Property der UI-Elemente wie Window.). Stattdessen System.Windows.Threading.Dispatcher.CurrentDispatcher verwenden ! (=static global Zugangspunkt).

```
Task.Run(() =>
{
    // Ich laufe in einem Background-Thread, nice! :-)
    ...
    Lambdas! ☺
    // den Dispatcher nutzen, um die folgende Aktion im UI-Thread durchzuführen:
    Dispatcher.Invoke(() =>
    {
        // Ich laufe im UI-Thread, woohoo! :-)
        // Somit hier etwas im UI Thread erledigen
        ...
    });
});
```



**Internationalization (i18n)**

Wieso mehrere Sprachen in einer App. Grundsätzlich gibt es in der App-Welt alles nur in Englisch. In der Schweiz offiziell 4-sprachig. Englisch ist ebenfalls von den meisten verstanden. Daher sollte man in einer App mindestens Englisch und/oder Deutsch anbieten.

**Möglichkeiten zur Internationalisierung**

- Standard-.NET-Mechanismen → Embedded Resources
- WPF-spezifischer Mechanismus
- Eigene Implementierung → oft in unternehmensinternen Frameworks im Einsatz
- Mischlösungen



«Internationalisierung» / «Globalisierung»:  
Eine App für die Nutzung in verschiedenen Sprachen vorbereiten.



«Lokalisierung»:  
Die Ressourcen einer internationalisierten App in eine bestimmte Sprache übersetzen.

**Standard .NET-Mechanismen (Embedded Resources)****Nicht-lokalisiert**

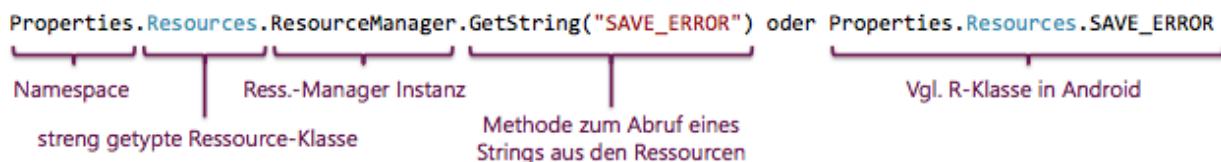
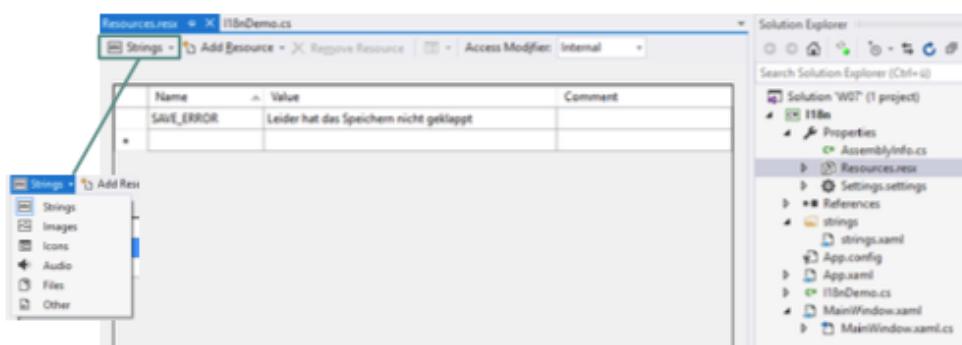
```
public void NonLocalizedDemo()
{
    MessageBox.Show("Leider hat das Speichern nicht geklappt");
}
```

**Lokalisiert (via Resource Manager)**

```
public void LocalizedDemo()
{
    ox.Show(Properties.Resources.ResourceManager.GetString("SAVE_ERROR"));
}
```

**Lokalisiert (streng getypt)**

```
public void StronglyTypedLocalizedDemo()
{
    MessageBox.Show(Properties.Resources.SAVE_ERROR);
}
```



## Standard .NET-Mechanismen (Embedded Resources)

Resources.resx → Standardsprache (siehe Projekteinstellungen)

Neue Sprache unterstützen:  
Kopie von Resources.resx erstellen und Sprachkürzel ergänzen.  
Beispiel: Resources.en-US.resx

Properties.Resources.ResourceManager.GetString("SAVE\_ERROR") oder Properties.Resources.SAVE\_ERROR

Vgl. R-Klasse in Android

Namespace      Ress.-Manager Instanz      Methode zum Abruf eines Strings aus den Ressourcen

streng getypte Ressource-Klasse

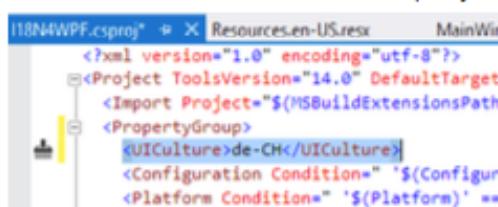
## Unterstützung zur Übersetzung

Dazu der Zeta Resource Editor (<http://www.zeta-resource-editor.com/index.html>). Er kann Visual Studio Solutions lesen. Zudem unterstützt er das Anzeigen der Übersetzungen aus den .resx-Files in Spalten nebeneinander.

## WPF-Spezifisch

### 1 UICulture setzen (Standardsprache) → Hinweis an Compiler, dass dies ein sprachabhängiges Projekt ist

- Rechtsklick auf Projekt > Unload Project
- Rechtsklick auf Projekt > Edit ...csproj
- <UICulture>-Knoten in erstem <PropertyGroup>-Knoten setzen (z.B. «de-CH»):



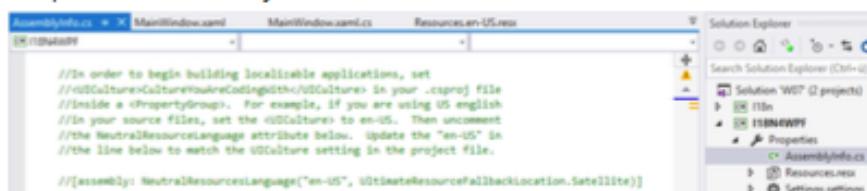
- Speichern und schliessen
- Rechtsklick auf Projekt > Reload Project

languagecode2[-country/regioncode2]

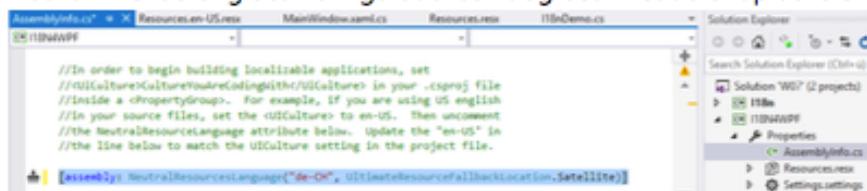
⚠ Wird der Country-/Region-Code weggelassen, gilt das .resx-File für alle Regionen der Sprache. Wird er gesetzt, gilt das File nur für die jeweilige Sprachregion.

### 2 Neutrale Sprache festlegen

- Properties > AssemblyInfo.cs öffnen



- Auskommentierung des Konfigurationseintrag betr. Neutraler Sprache entfernen und Sprache anpassen:



## Zugriff via Standard .NET Resource Manager

```
<Window x:Class="I18n.MainWindow" ...>
    xmlns:resx="clr-namespace:I18n.Properties"
    ...
    ...
    <TextBlock Text="{x:Static resx:Resources.LABEL_FIRST_NAME}" />
    ...
</Window>
```

Static Markup Extension

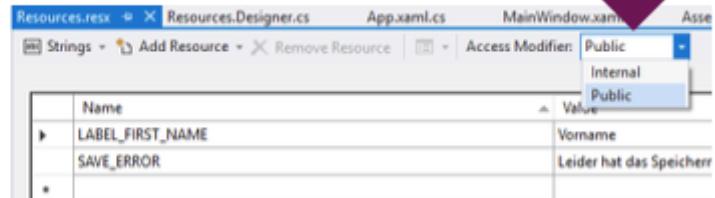
Zugriff auf streng getypte Embedded Resource



Access Modifier in .resx-Files  
unbedingt auf «Public» und  
nicht «Internal» stellen!

## Nachteil:

- Nur beim Initialisieren des Fensters ausgewertet
- Konsequenzen?

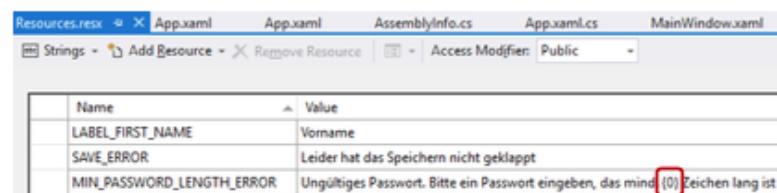


Name	Value
LABEL_FIRST_NAME	Vorname
SAVE_ERROR	Leider hat das Speichern nicht geklappt
MIN_PASSWORD_LENGTH_ERROR	Ungültiges Passwort. Bitte ein Passwort eingeben, das mind. {0} Zeichen lang ist

## Parameter in String?

## Beispiel

## Fehlermeldung mit einem Parameter



Name	Value
LABEL_FIRST_NAME	Vorname
SAVE_ERROR	Leider hat das Speichern nicht geklappt
MIN_PASSWORD_LENGTH_ERROR	Ungültiges Passwort. Bitte ein Passwort eingeben, das mind. {0} Zeichen lang ist

## Vorgehen

- String Ressource erstellen
- String.Format-Syntax für Parameter verwenden
  - o {0} ist der Platzhalter für den ersten Parameter, {1} für den zweiten, ...
- String einlesen und vor Ausgabe mittels String.Format{} und em passenden Parameter formatieren.
- Konsequenz
  - o Geht nicht mehr via {x:Static...} alleine, sondern nur indirekt über Binding/MultiBinding

## Die Sprache zur Laufzeit ändern

Unter anderem zum Fenster neu laden, damit neuste Sprachressourcen genutzt werden.

```
// sprachspezifische Culture erzeugen
var culture = new CultureInfo("de-CH");

// Standard-Culture für alle Threads setzen (inkl. Background Threads)
CultureInfo.DefaultThreadCurrentCulture = culture;
CultureInfo.DefaultThreadCurrentUICulture = culture;

// UI-/Culture im aktuellen Thread umstellen
Thread.CurrentThread.CurrentCulture = culture;
Thread.CurrentThread.CurrentUICulture = culture;
```

Translation-Workflow in WPF starten → LocBaml-Tool

Interpretiert BAML (kompilierte Version von XAML)  
 Nutzt x:Uid-Attribute (automatisch via Tool ergänzbar)  
 LocBaml erzeugt .csv-Files, die einfach übersetzt werden können  
 Mit LocBaml können die übersetzten .csv-Files in Satellite Assemblies übersetzt werden

- Aufwändig, dafür unabhängig von der Entwicklung
- Nicht weiter besprochen

➔ Die Nutzung des nativen WPF-Mechanismus zur Globalisierung / Lokalisierung wird nicht empfohlen.

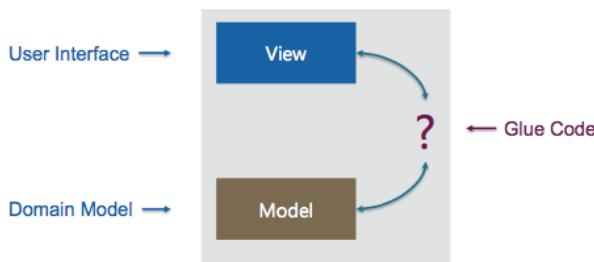
### Fazit

WPF hat den Prozess eher aufwändiger gemacht. Am einfachsten ist immer noch der Standard .NET-Mechanismus (Embedded Resources). In der Praxis sind sehr oft Eigenentwicklungen im Einsatz.

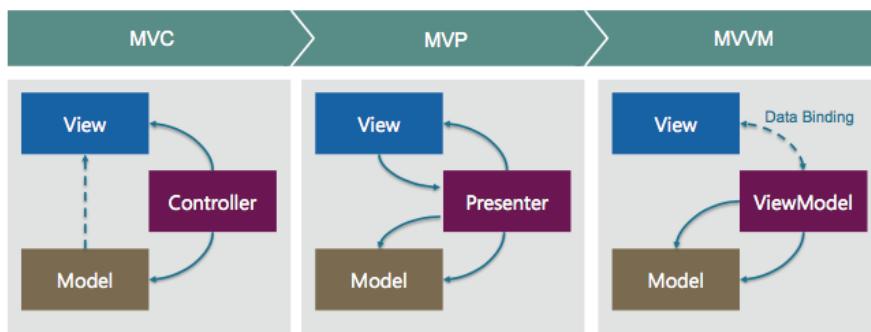
### MVVM

#### Von MVC zu MVVM

Eine abstrakte Sicht auf die User Interface Entwicklung. Die Daten (Modell) müssen irgendwie in einem User Interface (View) dargestellt werden.

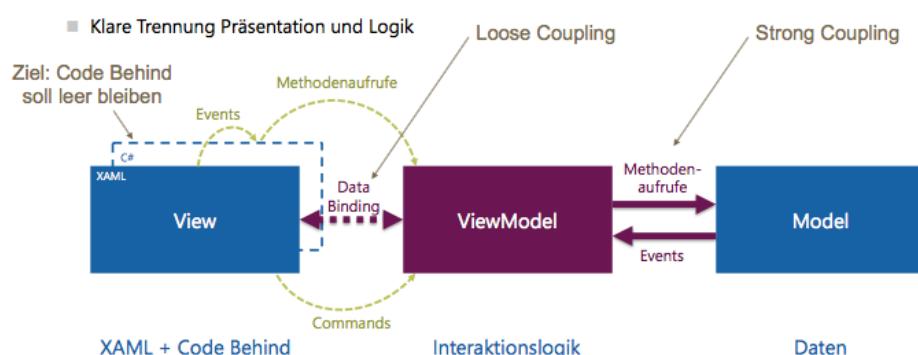


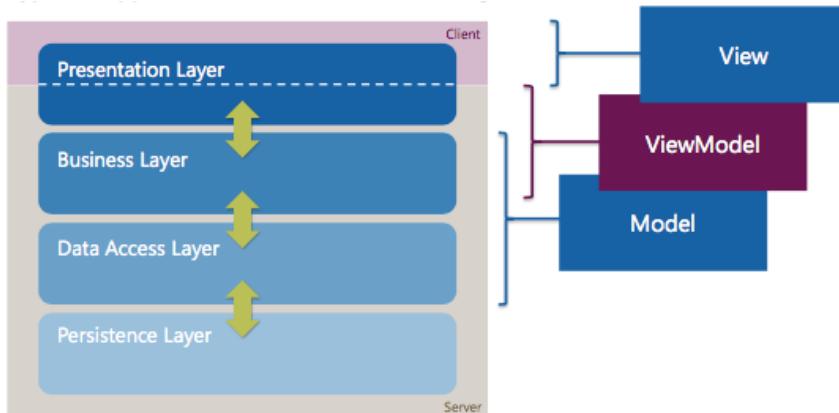
#### Verschiedene Ausprägungen



### MVVM im Detail

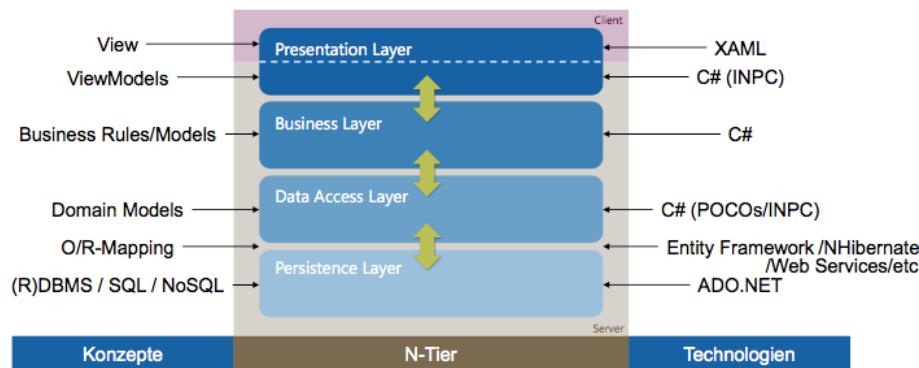
Klare Trennung von Präsentation und Logik.





## N-Tier Architektur (WPF)

Typische WPF-Applikations-Architektur.



## Eine typische WPF-App

### Das Model

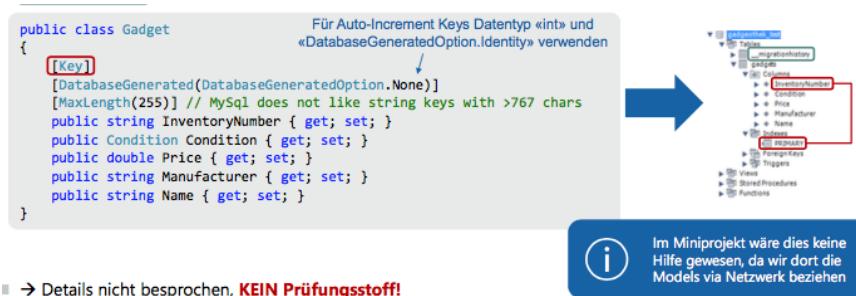
Die eigentlichen Daten, also die Domain Objekte. Kein Verhalten, nicht zuständig für die Darstellung und Formatierung der Daten und das Laden und Speichern. Ein Abbild der realen Welt oft ergänzt um Informationen zur Nutzung, Speicherung um zum Tracking.

### Hilfsmittel

- Entity Framework → Microsoft's empfohlene O/R-Mapping-Technologie für .NET-Anwendungen
- DB-Server & .NET-Zugriffskomponenten → z.B. MySql.Data & MySql.Data.Entity
- Oder falls via Web Service: JSON (vgl. Miniprojekt)

### Entity Framework für Persistenz

Code First-Ansatz benutzen → erzeugt die DB automatisch aus attribuierten POCOs. Auto-Migration des DB-Schemas bei geänderten Models.



→ Details nicht besprochen, **KEIN Prüfungsstoff!**

**Variante 1 – Model als Property**

Model als Property kapseln

```
public class GadgetVm:BindableBase // vgl. Slides zu INotifyPropertyChanged (Variante 3)
{
    private Gadget _gadget;
    public Gadget Gadget
    {
        get { return _gadget; }
        set { SetProperty(ref _gadget, value, nameof(Gadget)); }
    }

    ...
}
```

**Variante 2 – Properties des Models reimplementieren**

Für jede im UI benötigte Property eine gleichnamige Property implementieren und dann ein Object-Mapping Tool verwenden. Z.B. automapper via nuget.

```
public class GadgetVm:BindableBase // vgl. Slides zu INotifyPropertyChanged (Variante 3)
{
    private string _inventoryNumber;
    public string InventoryNumber
    {
        get { return _inventoryNumber; }
        set { SetProperty(ref _inventoryNumber, value, nameof(InventoryNumber)); }
    }

    ...
}

Mapper.Initialize(cfg => cfg.CreateMap<Gadget, GadgetVm>());
// Annahme: gadget sei eine Variable des Typs Gadget
var vm = Mapper.Map<GadgetVm>(gadget);
```

**„Commands“**

Problem: View soll nicht vom ViewModel wissen, aber Kommandos anstoßen können

Lösung: ICommand im ViewModel implementieren und via DataBinding auslösen

**Beispiel**

Ein Gadget View Model soll sich selbst speichern können (z.B. auf Server, in DB, etc.). Dies soll als Reaktion auf einen Button-Klick (z.B. „OK“ oder „Speichern“-Button in Gadget Detail View) erfolgen.

**Lösung**

Wir scheiben dazu ein eigenes Command („SaveCommand“).

- Command führt das Speichern aus
- Command Source ist der Button, auf den geklickt wird, um die Aktion auszulösen
- Command Target ist unser in der Detail View angezeigtes Gadget Objekt
- Das Command Binding definieren wir auf dem Button, so dass dieser das Kommando beim Klicken auslöst.

**MSDN zu Commands**

"The **command** is the action to be executed."

"The **command source** is the object which invokes the command."

"The **command target** is the object that the command is being executed on."

"The **command binding** is the object which maps the command logic to the command."

### Variante 1

Command direkt implementieren, Instanz davon als Property im ViewModel.

```
public class SomeCommand : ICommand
{
    public bool CanExecute(object parameter)
    {
        // ... irgendwie true oder false zurückgeben
    }

    public void Execute(object parameter)
    {
        // ... irgendwas ausführen
    }

    public event EventHandler CanExecuteChanged;
}
```

```
public class GadgetVm
{
    // streng getypt, oder auch nur ICommand:
    public SomeCommand MyCommand { get; set; }
    // public ICommand MyCommand { get; set; }

    public GadgetVm()
    {
        MyCommand = new SomeCommand();
    }
    ...
}
```

→ Nachteile dieser Variante?

### Variante 2

Command als innere Klasse im ViewModel implementieren. Vorteil gegenüber vorher: Zugriff auf private Member des ViewModels.

```
public class GadgetVm:BindableBase
{
    ...
    public class SomeCommand : ICommand
    {
        public bool CanExecute(object parameter)
        {
            // ... irgendwie true oder false zurückgeben
        }

        public void Execute(object parameter)
        {
            // ... irgendwas ausführen
            // ... jetzt aber mit Zugriff auf private Members
        }

        public event EventHandler CanExecuteChanged;
    ...
}
```

```
...
private GadgetVm ViewModel { get; set; }

// eine Instanz auf das ViewModel übergeben:
public SomeCommand(GadgetVm vm)
{
    ViewModel = vm;
}
} // Ende des Kommandos
} // Ende des ViewModels
```

→ Nachteile dieser Variante?

### Variante 3

Helper-Command für einfachere Wiederverwendung implementieren.

#### „ReplayCommand“

##### Teil 1

```
public class RelayCommand : ICommand
{
    private readonly Action _execute;
    private readonly Func<bool> _canExecute;

    public RelayCommand(Action execute, Func<bool> canExecute = null)
    {
        if (execute == null) throw new ArgumentNullException("execute");

        _execute = execute;
        _canExecute = canExecute;
    }

    public bool CanExecute(object parameter) => _canExecute?.Invoke() ?? true;
    public void Execute(object parameter) => _execute();
    ...
}
```

## Teil 1 – Variante mit generischem Parameter <T>

```
public class RelayCommand<T> : ICommand
{
    private readonly Action<T> _execute;
    private readonly Predicate<T> _canExecute;

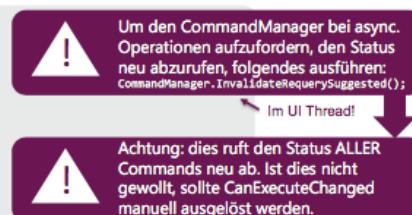
    public RelayCommand(Action<T> execute, Predicate<T> canExecute = null)
    {
        if (execute == null) throw new ArgumentNullException("execute");

        _execute = execute;
        _canExecute = canExecute;
    }

    public bool CanExecute(object parameter) => _canExecute?.Invoke((T)parameter) ?? true;
    public void Execute(object parameter) => _execute((T)parameter);
    ...
}
```

## Teil 2

```
...
// event an CommandManager delegieren (Benachrichtigung
// erfolgt so immer dann, wenn WPF denkt, dass sich etwas
// am Ausführungsstatus geändert hat, z.B. bei Key- oder
// Mouse Button-Klick)
public event EventHandler CanExecuteChanged
{
    add { CommandManager.RequerySuggested += value; }
    remove { CommandManager.RequerySuggested -= value; }
}
```



## Verwendung

```
public class GadgetVm
{
    public ICommand SaveCommand { get; set; }

    public GadgetVm()
    {
        SaveCommand = new RelayCommand(() => this.Save(), () => this.CanSave());
    }
    ...

    public CanSave => ...; // some condition
    public void Save() // some method
    {
        ...
    }
    ...
}
```

- Variante: Initialisierung direkt bei Definition (C# 6)

```
public ICommand SaveCommand { get; set; } = new RelayCommand(() => this.Save(), () => this.CanSave());

// Instanzierung im Konstruktor von GadgetVm nicht mehr nötig
```

- Variante: Lazy Initialization

```
// backing field needed:
private ICommand _saveCommand;
public ICommand SaveCommand => _saveCommand ?? (_saveCommand = new RelayCommand(() => this.Save(), () => this.CanSave()));

// Instanzierung im Konstruktor von GadgetVm nicht mehr nötig
```

Das DataBinding im XAML-Code ist wie gehabt. Voraussetzung ist, dass der DataContext für Binding korrekt auf ViewModel gesetzt ist.

```
<Button Content="Save"
        Command="{Binding SaveCommand}" />
```

Oder falls zusätzlich ein Parameter übermittelt werden sollen.

→ Commands werden beim Data Binding an die Standardaktion eines Controls gebunden. Bei Buttons ist dies das Click-Event.

```
<Button Content="Open"
        Command="{Binding OpenGadgetViewCommand}"
        CommandParameter="{Binding SelectedGadget}" />
```

## „Events“

**Problem:** View soll ViewModel nicht kennen, aber mittels Events Kommandos anstoßen können

**Lösung:** Event in Kommando konvertieren (EventToCommand) und diesen via Binding auslösen

Der SourceCode ist hier nicht angegeben, da er etwas umfangreich ist. Beispiel kann aber im Rahmen der MVVM Light Implementation angesehen werden. Die Nutzung geschieht via „Interaction“. Diese wird nicht weiter besprochen.

```
<ListBox ItemsSource="{Binding Gadgets}">
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="MouseDoubleClick">
      <mvvm:EventToCommand Command="{Binding OpenGadgetViewCommand}"
                           CommandParameter="{Binding SelectedGadget}"
                           PassEventArgsToCommand="False" />
    </i:EventTrigger>
  </i:Interaction.Triggers>
</ListBox>
```

Zusätzliche benötigte Namespaces (XAML-Code):  
 xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"  
 xmlns:mvvm="http://www.galasoft.ch/mvvmlight"

## ViewModel vs. View

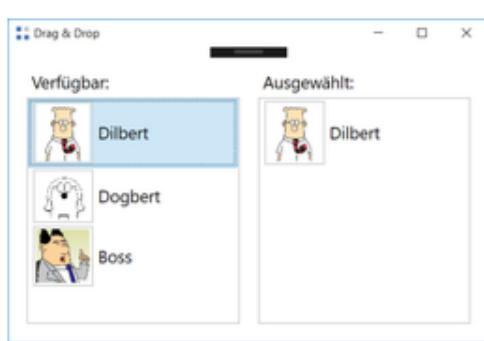
ViewModel kennt View nicht. Er hat keine Referenz auf die View. Falls Parameter benötigt werden sollen diese als ComandParameter übergeben werden und Klassenintern gecachet werden. Die View erstellt es nicht selbst.

Die View kennt keine Referenz auf das ViewModel (ausser den DataContext). Zudem auch keine Verbindung zu Datenelementen und Commands nur deklarativ via Data Binding.

Das ViewModel und View werden „von aussen“ erzeugt. Das ViewModel wird „von aussen“ als DataContext für die View gesetzt.

## Die View

= Das User Interface = XAML – Dateien = WPF-Controls



```
<Window x:Class="DragSample.MainWindow" Title="Drag & Drop" Height="350" Width="525">
  <Window.Resources>
    <DataTemplate x:Key="UserInfoTemplate" DataType="local:UserInfo">
      <Grid>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="64"/><ColumnDefinition/>
          <ColumnDefinition Width="28"/><ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Border BorderThickness="2" Padding="2" BorderBrush="LightGray">
          <Image Source="{Binding Image}" />
        </Border>
        <Label Content="{Binding NickName}" Grid.Column="1" VerticalContentAlignment="Center" />
      </Grid>
    </DataTemplate>
  </Window.Resources>
  <Grid Margin="20">
    <Grid.ColumnDefinitions>
      <ColumnDefinition></ColumnDefinition>
      <ColumnDefinition Width="28"/><ColumnDefinition/>
      <ColumnDefinition Width="28"/><ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/><RowDefinition/>
      <RowDefinition Height="Auto"/><RowDefinition/>
    </Grid.RowDefinitions>
    <Label Content="Verfügbar:"/>
    <Label Grid.Row="1" Grid.Column="2" Content="Ausgewählt:"/>
    <ListBox Name="AvailableListBox" Grid.Row="1" Grid.Column="0" ItemsSource="{Binding AvailableUsers}" ItemTemplate="{StaticResource UserInfoTemplate}" />
    <ListBox Name="SelectedListBox" Grid.Row="1" Grid.Column="1" ItemsSource="{Binding SelectedUsers}" ItemTemplate="{StaticResource UserInfoTemplate}" />
  </Grid>
</Window>
```

## Tipps

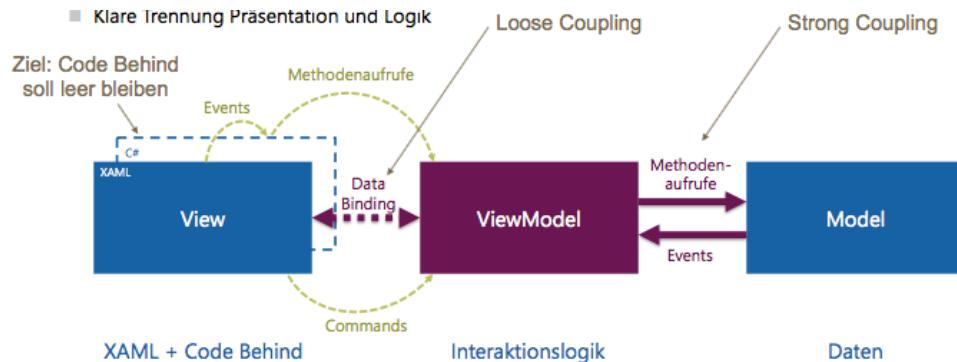
### Wiederverwendbarkeit

Design (z.B. Button): Styles mit Control Templates und Layouts als UserControls.

### Koppelung

Möglichst wenig (am besten: kein) Zugriff auf die UI Controls aus dem Code Behind. Doe Zustände via Data Binding zwischen speichern. Beispiel: SelectedItem einer ListBox oder eines DataGrids an eine Property des ViewModels binden. Steuerung der Anzeige via Flags des ViewModels binden.

MVVM Revised → Klare Trennung von Präsentation und Logik.



### Startup Code

**So weit so gut, aber...**

wer erstellt die ViewModels? Wer erzeugt die Views? Wer lädt die Models?

### Lösungsvorschläge

- Spezielles App-ViewModel verwenden (zentrale Controller-Komponente)
- App-ViewModel beim App-Start erzeugen
- Zusätzliche ViewModels aus App-ViewModel erzeugen(mittels Factory-Methoden)
- View direkt innerhalb der Views oder mit speziellen View-Factories (UIProjekt) erzeugen
- Models in ViewModel erzeugen/laden/speichern

### Initialisierungscode – Erzeugen des ViewModels

**Variante 1 – Internes Erzeugen (privates „Implementierungsdetail“ in der View)**  
**Variante 1: internes Erzeugen (privates «Implementierungsdetail» in der View)**

In früheren Beispielen hieß die View jeweils «MainWindow», der Name der View kann aber natürlich frei gewählt werden

```
// eigener init code...
MainWindow = new GadgeothekView();
MainWindow.Show();
«MainWindow» ist hier die
gleichnamige Property
der Application-Klasse
```

```
public GadgeothekView() ←
{
    InitializeComponent();
    DataContext = new GadgetVm();
}
```

**Variante 2 – Dependency Injection (via Property)**

```
// eigener init code...
var vm = new GadgetVm();
MainWindow = new GadgeothekView();
MainWindow.DataContext = vm;
MainWindow.Show();
```

```
public GadgeothekView()
{
    InitializeComponent();
}
```

**Variante 3 – Dependency Injection (via Konstruktor)**

```
// eigener init code...
var vm = new GadgetVm();
MainWindow = new GadgeothekView(vm);
MainWindow.Show();
```

```
public GadgeothekView(GadgetVm vm)
{
    InitializeComponent();
    DataContext = vm;
}
```

```
<Window ...>
  <Window.DataContext>
    <local:GadgetVm />←
  </Window.DataContext>
  ...
</Window>
```

Instanz des ViewModels in XAML

```
public GadgeothekView()
{
  InitializeComponent(); ← Keine Anpassung nötig,
}
```

Code Behind ist «leer»

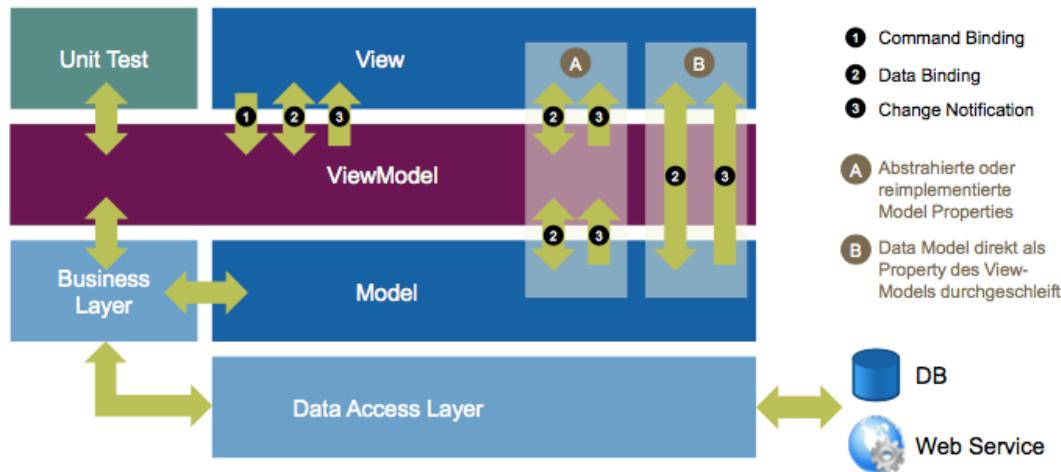
## Initialisierungscode – Erzeugen der Views

Direkt in App.xaml oder in anderen View ➔ sinnvollste Variante mit Wissensstand im Rahmen MGEG.

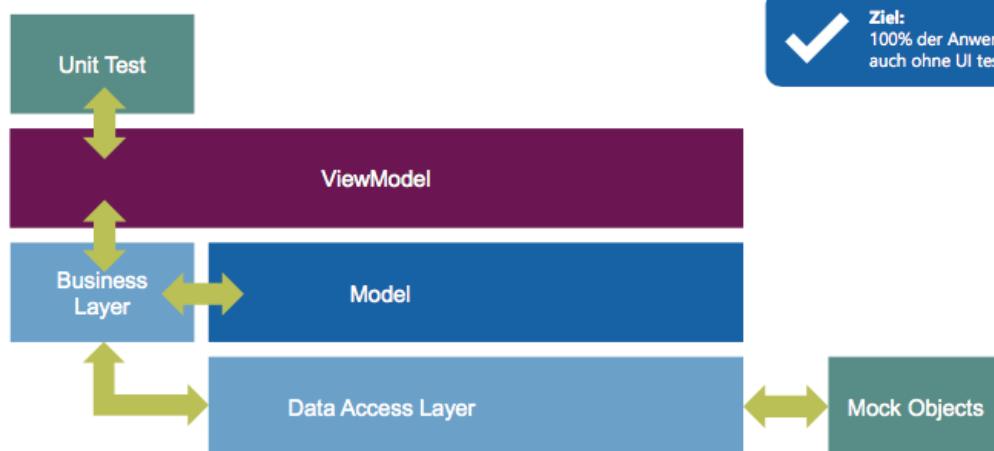
Weitere Varianten

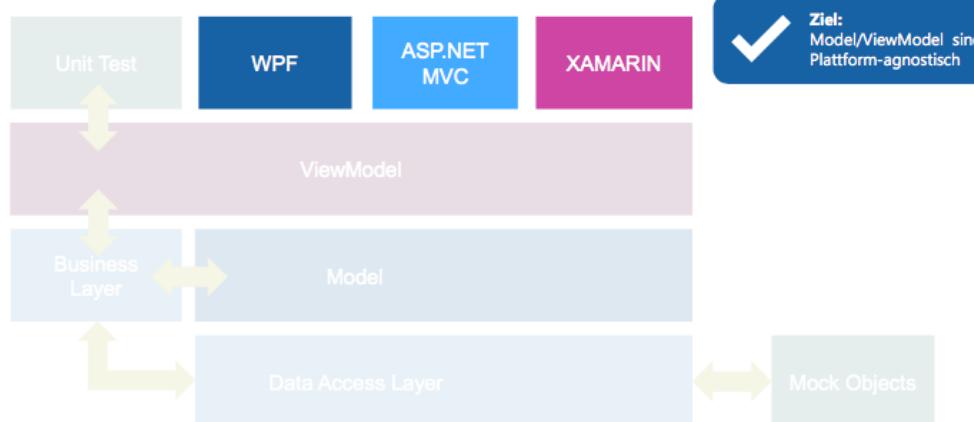
- Automatisch via DataTemplate (mit gesetztem DataType) als Ressource
  - o ContentPresenter mit Binding.
  - o Anhand des Typs des Binding wird das „richtige“ DataTemplate ausgewählt
- Via Custom Attributes

## Zusammenfassung WPF-Architektur



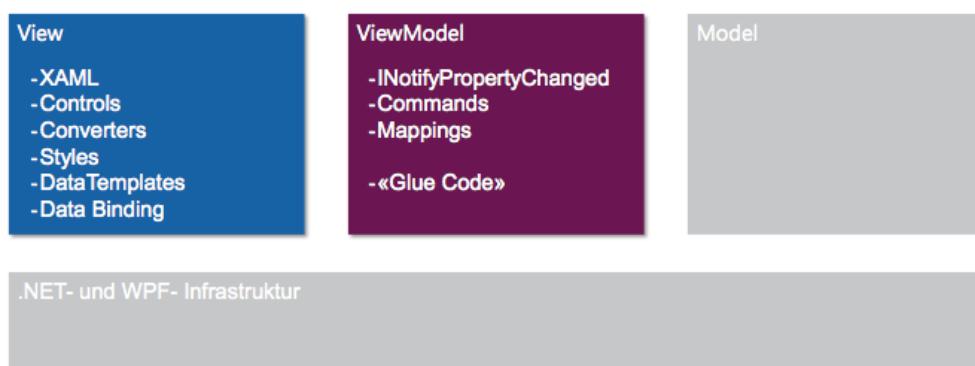
## MVVM Testing





## MVVM in WPF – Technologieübersicht

WPF-Bestandteile und MVVM.

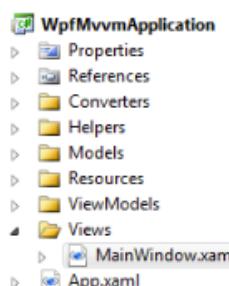


## Projekt-Layout

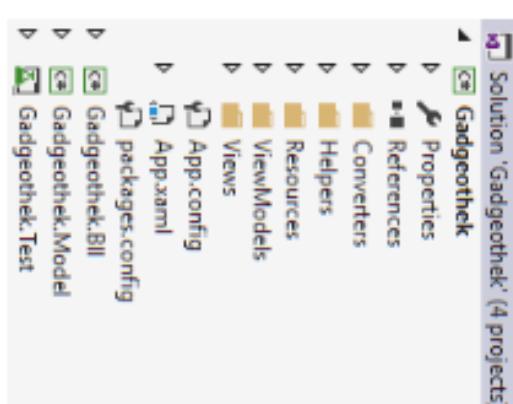
### Kleine(re) Projekte

All in One Ansatz. Also 1 Ordner pro Kategorie und 1 Ordner pro Rolle in MVVM.

Für die Tests ein separates Test-Projekt erstellen.



### Mittlere Projekte



Eigene Projekte für die verschiedenen Layer erstellen. 1 Projekt pro Schicht (Data/Model | Business Logik | UI). Innerhalb der Projekte 1 Ordner pro Rolle in MVVM / Kategorie.

Für die Tests ein separates Test-Projekt.

### Grosse Projekte

Projekte der verschiedenen Schichten weiter in (testbare) Teile aufsplitten.