Responsive Design, Weiterführende Interaktionsmuster

Inhaltsübersicht

1. Einführung in die Entwicklung mobiler Anwendungen
2. Erste grafische Oberflächen und Benutzerinteraktionen
3. Weiterführende Konzepte mobiler Plattformen
4. Standorbezogene Dienste, Sensoren und Kamera
5. Dauerhaftes Speichern von Daten (Persistenz)
6. Responsive Design, Weiterführende Interaktionsmuster
7. Asynchrone Verarbeitung

Responsive (Web) Design

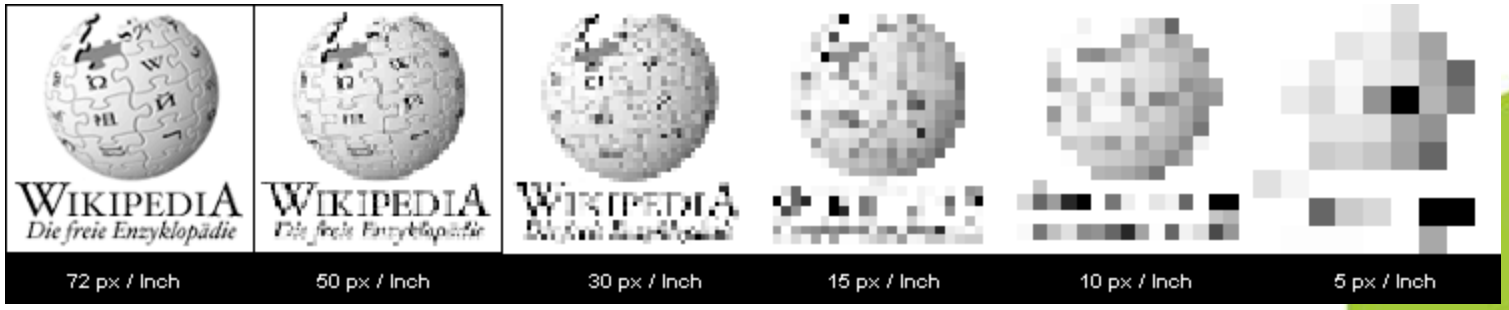
- Responsive Design ist ein Begriff aus der Webentwicklung
- Im WWW werden dazu MediaQueries und Grid Layouts bzw. auch JavaScript verwendet
- Das Ziel von RWD ist eine Webseite für alle typischen Ausgabemedien verfügbar machen

- Die Ansätze hinter RWD können auch auf mobile native Apps übertragen werden
- Gute Einführung: https://www.youtube.com/watch?v=zHirwKGEfoE

Android Geräte existieren in unterschiedlichsten Auflösungen

Auflösung (DPI)	Bezeichner in Android	Skalierung
160	MDPI (Baseline)	1
240	HDPI	1.5
320	XHDPI	2
480	XXHDPI	3

Zur einfacheren Handhabung gruppiert Android unterschiedliche Geräteauflösungen in Auflösungsklassen.



Density Independent Pixels (dp)

- Virtuelle Pixeleinheit, welcher 1 physischer Pixel auf einer MDPI (160 dpi) Auflösung entspricht
- dps werden proportional für unterschiedliche Auflösungen skaliert
- Mit dps können UI-Elemente die selbe physische Größe behalten auf unterschiedlichen Geräten mit unterschiedlicher Auflösung

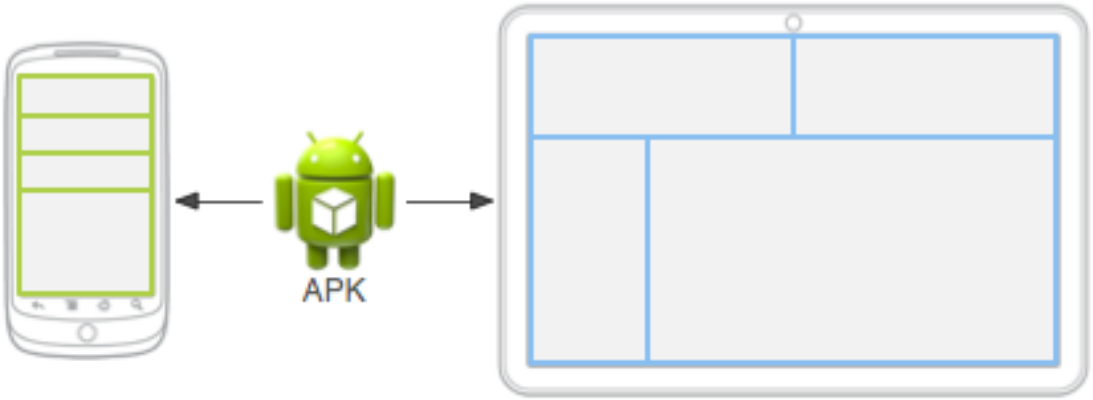
Qualifizierte Resourcen (Verzeichnisnamen)

- Aufgrund unterschiedlicher Gerätekonfigurationen können unterschiedliche Resourcen geladen werden
  - Sprachen: zB en, fr, de
  - Schreibrichtung: ldrtl, ldltr
  - kleinste Breite (sw<N>dp): zB sw320dp, sw600dp
  - Orientierung: port, land
  - Auflösung: mdpi, hdpi, xdpi, xxdpi, ...
- Es können mehrere Qualifier für Resourcen Verzeichnis angegeben werden
- Spezielle Ordnungsregeln müssen jedoch eingehalten werden

Laden unterschiedlicher Layouts


```
res/
   layout/
      activity_main.xml
   layout-sw320dp/
      activity_main.xml
   layout-sw600dp/
      activity_main.xml
```

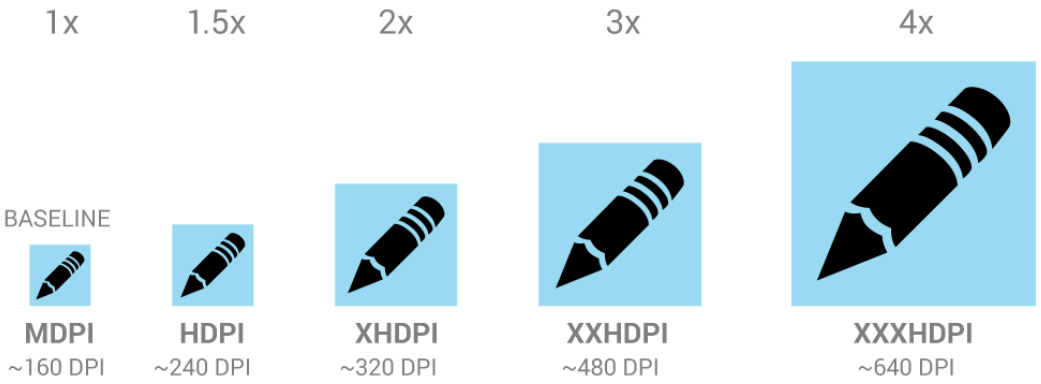
Layouts werden anhand der mindest Breite ausgewählt



Laden unterschiedlicher Bildgrößen


```
res/
   drawable-mdpi/
      icon.png
   drawable-hdpi /
      icon.png
   drawable -xdpi/
      icon.png
```

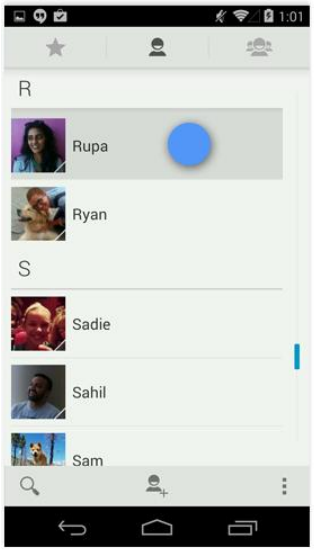
Bilder werden anhand der Auflösung ausgewählt



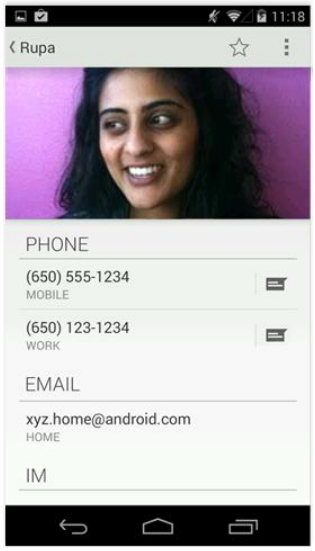
Generelle Best Practices

1. wrap_content, fill_parent und dp Angaben verwenden für Layouts
2. Keine hard-coded Pixel-Berechnungen
   - zb gibt die ***getWidth()*** Methode einer View die Pixel der aktuellen Auflösung wieder und kann nicht mit hard-coded Pixelwerten verwendet werden
3. AbsoluteLayouts sollen nicht mehr verwendet werden (Deprecated)
4. Größenabhängige Layouts und Auflösungsabhängige Bilder verwenden

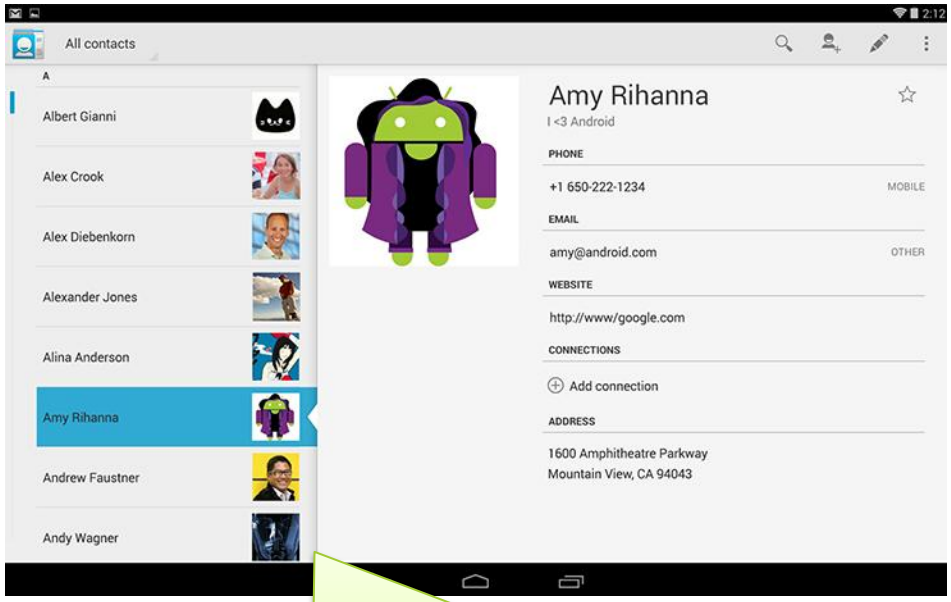
Herausforderung Wiederverwendbarkeit



List view



Detail view



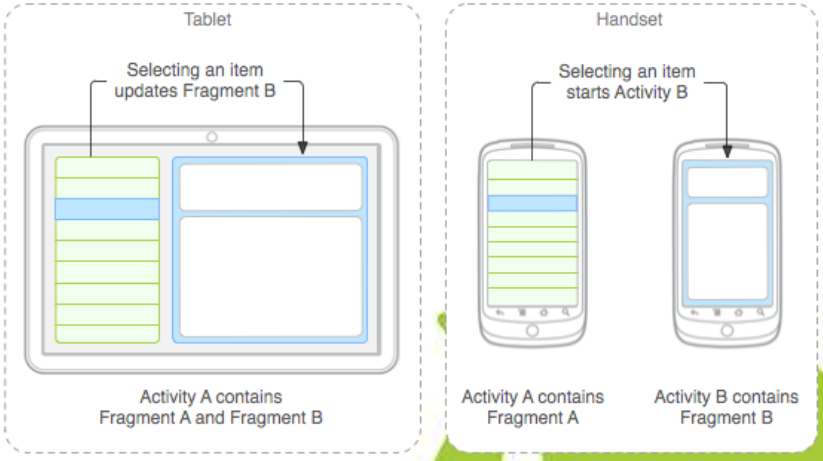
Realisierung mit zwei Activities am Smartphone

Realisierung mit einer Activity am Tablet

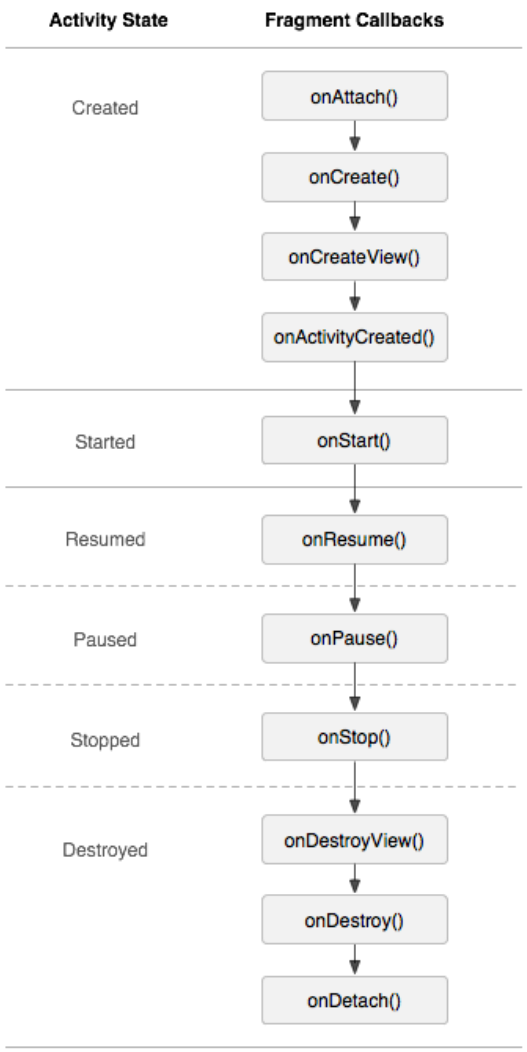
Activities sind zu grob-granulare Komponenten, um eine Wiederverwendbarkeit innerhalb verschiedener Geräteklassen zu gewährleisten. Deshalb wurden mit Android 3.0 Fragments eingeführt.

Fragments

- Fragments sind wiederverwendbare UI Komponenten, welche in das Layout von Activities eingebettet werden
- Fragments haben einen eigenen Lebenszyklus, welcher dem von Activities sehr ähnlich ist
- Fragments sind an den Lebenszyklus der Host Acivity gebunden
- Fragments können zum besseren Verständnis wie “Subactivities” gedacht werden



Fragment Lebenszyklus



- Fragments besitzen, ähnlich wie Activities, Callbacks für entsprechende Übergänge im Lebenszyklus
- Drei Zusände sind zu unterscheiden:
  - Resumed (Started): das Fragment ist sichtbar und aktiv
  - Paused: Eine andere Activity ist im Vordergrund, Teile der Host Activity sind noch sichtbar
  - Stopped: Fragment wurde beendet oder Host Activity wurde gestoppt

Hinzufügen eines Fragements (deklarativ)


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/andro
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Angabe eines Layout Platzhalters für das Einbinden eines Framgents

Eine Referenz auf die Klasse, welche das Fragment implementiert

Hinzufügen eines Fragments (programmatisch)


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

Innerhalb der Activity findet sich eine Referenz auf einen FragmentManager: ***getFragmentManager()***

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

Innerhalb des Layouts der Host Activity wird der Container für das Fragment angegeben. Dies kann zum Beispiel ein FrameLayout sein

Fragment Transaktionen

- Alle Operationen über den FragmentManager passieren innerhalb von Transaktionen
  - Die Transaktion wird eingeleitet durch ***beginTransation()***
  - Operationen wie ***add()*** , ***replace()*** oder ***remove()*** von Fragments können durchgeführt werden
  - Die Transaktion kann abgeschlossen werden mit ***commit()***
- Fragmentübergänge werden nicht wie Activities automatisch im Backstack gespeichert
- Vor dem Abschluss einer Transaktion kann ***addToBackStack()*** aufgerufen werden, um dies zu gewährleisten

Implementieren eines Fragments


```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

Innerhalb der ***onCreateView()*** Methode wird das Layout des Fragments erzeugt. Der Container wird von der Host Activity bereitgestellt.

- Ähnlich wie für Activities stehen für Fragments bereits eine Vielzahl von nützlichen Implementierungen zur Verfügung
  - zB ListFragment, DialogFragment, PreferenceFragment

Themes und Styles

- Ähnlich wie CSS für Webseiten bietet Android eine Möglichkeit Stylesheets zu erstellen, um Apps ein einheitliches Erscheinungsbild zu geben
- Zu unterscheiden ist dabei:
  - Themes können auf eine Activity oder auch auf die ganze Application angewendet werden
  - Themes können über das AndroidManifest.xml angewendet werden
  - Styles können für einzelne UI-Elemente angewendet werden
  - Themes können jedoch auch Styles für UI-Elemente vorgeben

Definition von Styles


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

- Styles werden im Ordner res/values/ abgelegt
- Styles werden innerhalb eines <resources> Wurzelknotens angegeben
- Jeder Style benötigt einen Namen
- Styles können vererbt werden
- <item> Elemente deklarieren Style Eigenschaften

Anwendung von Styles und Themes

- Anwendung über das style-Attribut innerhalb einer Layout xml-Datei

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

- Anwendung innerhalb des AndroidManifest.xml auf einzelne Activities oder die ganze Anwendung

```
<activity android:theme="@android:style/Theme.Dialog">
```

```
<application android:theme="@style/CustomTheme">
```

- Android liefert eine Vielzahl von Styles und Themes mit
- https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values/styles.xml
- https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values/themes.xml

Animationen

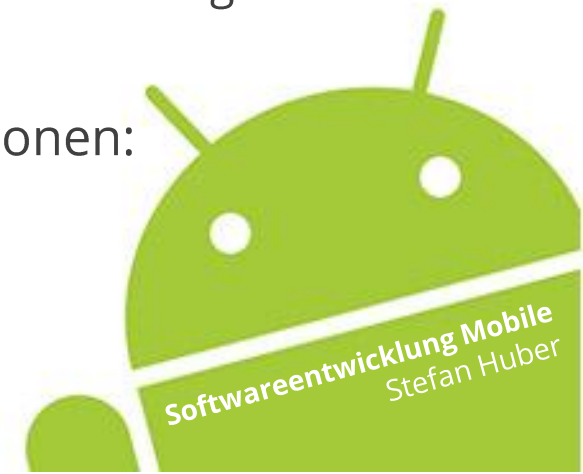
- Das Android Framework unterstützt von Haus aus Animationen
- Drei verschiedene Möglichkeiten stehen zur Verfügung:

1. **Animation von Views (Tweened View Animations)** ermöglichen die animierte Veränderung von Views in Bezug auf Position, Größe, Drehung, Durchsichtigkeit

2. **Frame-basierte Animation (Frame Animations)** realisieren animierte Effekte über die wechselnde Anzeige von Grafiken (drawables)

3. **Animation über Objekteigenschaften** (Interpolated Property Animations) animieren über die stufenweise Veränderung von Eigenschaften

- Einstiegspunkt der Android-Animations-Funktionen:
  - http://developer.android.com/guide/topics/graphics/overview.html



Animation: Tweened View Animations

- Views können als sog. Tweened View Animations animiert werden.
- Möglich sind Veränderung in
  - Position: <translate> -Tag
  - Größe: <scale> -Tag
  - Drehung: <rotation> -Tag und
  - Durchsichtigkeit: <alpha> -Tag
- Definition erfolgt in einem entsprechenden XML-File, das eine einzelne Animation oder eine Sequenz von Animationen definiert

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/anticipate_overshoot_interpolator">

  <rotate
    android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="3000"
    android:startOffset="0"/>
</set>
```

Definition einer zentrierten, rotierenden Animation um 360 Grad in 3 sek.

Animation: Tweened View Animations

- Im Code werden die Animationen mit Hilfe von AnimationUtils geladen und durch startAnimation auf dem entsprechenden View gestartet:

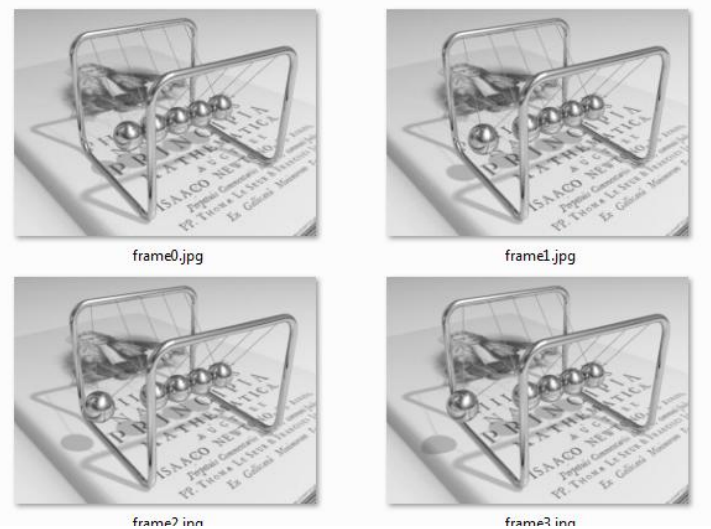
```
// Referenzen auf die Buttons und das Image besorgen
final Button buttonRotateCenter = (Button) findViewById(R.id.rotatecenter);
final Button buttonRotateCorner = (Button) findViewById(R.id.rotatecorner);
final ImageView floatingImage = (ImageView) findViewById(R.id.floatingimage);

// Animationen aus den entsprechenen XML-Files laden und einer Instanz
// der Klasse Animation zuordnen
final Animation animationRotateCenter = AnimationUtils.loadAnimation(this, R.anim.rotate_center);
final Animation animationRotateCorner = AnimationUtils.loadAnimation(this, R.anim.rotate_corner);
final Animation animationBlink = AnimationUtils.loadAnimation(this, R.anim.blink);

// Starten der Animationen durch Klicjen auf die entsprechende
// Buttons. startAnimation
buttonRotateCenter.setOnClickListener(new Button.OnClickListener() {
  @Override
  public void onClick(View arg0) {
    buttonRotateCenter.startAnimation(animationBlink);
    floatingImage.startAnimation(animationRotateCenter);

  }
});
```

Laden der Animationen

Start der Animationen

Animation: Frame Animations

- Framebasierter Animationen verwenden verschiedene Grafiken um einen animierten Effekt zu realisieren.
- Definition in einer xml Datei als BackgroundRessource, in der die einzelnen Dateien (jpg) und die Anzeigedauer festgelegt werden (in ms)




```xml
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
        android:oneshot="false">

        <item android:drawable="@drawable/frame0" android:duration="50" />
        <item android:drawable="@drawable/frame1" android:duration="50" />
        <item android:drawable="@drawable/frame2" android:duration="50" />
        <item android:drawable="@drawable/frame3" android:duration="50" />
        <item android:drawable="@drawable/frame4" android:duration="50" />
        <item android:drawable="@drawable/frame5" android:duration="50" />
        <item android:drawable="@drawable/frame6" android:duration="50" />
```

Animation: Frame Animations

- Initialisierung der Bildfolge als BackgoundResource

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  animation = (ImageView) findViewById(R.id.imageAnimation);

  // the frame-by-frame animation defined as a xml file within the drawable folder
  animation.setBackgroundResource(R.drawable.animation);
```

Referenz zum ImageView

Zuordnung der Bildfolge

- Starten/Stoppen der Animation

```
public void onWindowFocusChanged(boolean hasFocus) {
  super.onWindowFocusChanged(hasFocus);
  AnimationDrawable frameAnimation =
          (AnimationDrawable) animation.getBackground();

  if (hasFocus) {
    frameAnimation.start();
  } else {
    frameAnimation.stop();
  }
}
```

Zugriff auf den die Animationssequenz und Starten/Stoppen der Animation

Animation über Objekteigenschaften

- Über die ObjektAnimator-Klasse können Objekte animiert werden, indem ihre Eigenschaften schrittweise verändert werden
  - Für alle Eigenschaften möglich, die über Getter/Setter verfügen
  - Z.B. für die Hintergrundfarbe eines Buttons (setBackground)
  - Entweder direkte Implementierung im Programmtext oder Deklaration in einem XML-File
  - Animationen lassen sich parametrisieren (Dauer, Wiederholungsart, Anzahl der Wiederholungen)
  - Komplexere Animationen lassen sich über ein AnimationSet kombinieren
  - Auf Animationen kann über die entsprechende EventListener Einfluss genommen werden – AnimationListener (onAnimationStart, onAnimationEnd, onAnimationCancel, onAnimationRepeat)

Animation über Objekteigenschaften - Beispiel

- Direkte Implementierung einer Animation eines Buttons

```
Button b = (Button) findViewById(R.id.animationButton);
b.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {

    Button myButton = (Button) v;
    ObjectAnimator an = ObjectAnimator.ofInt(
            myButton, "height", 10, 400);
    an.setRepeatCount(4);
    an.setRepeatMode(ValueAnimator.REVERSE);
    an.setDuration(1500);
    an.setInterpolator(new BounceInterpolator());
    an.start();
  }
});
```

Erzeugen eines AnimatorObjekts um die Höhe des Buttons von 10 – 400 zu verändern

4 alternierende Wiederholungen

Dauer der Animation auf 1,5 sek. festgelegt

Animation gestartet

Animation über Objekteigenschaften - Beispiel

- Deklarative Animation über ein XML-File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"
                android:valueFrom="1"
                android:valueTo="0.2"
                android:duration="1000"
                android:propertyName="alpha"
                android.valueType="floatType"
                android.repeatCount="10"
                android.repeatMode="reverse"/>
```

Definition der Animationsparameter

- Einbindung in den Code:

```
b = (Button) findViewById(R.id.animationButton_2);
b.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {

    Animator anim = AnimatorInflater.loadAnimator(v.getContext(),
            R.anim.opacity);
    anim.setTarget(v);
    anim.start();
  }
});
```

Laden der definierten Animation und Zuordnung zu einem View

Animation über Objekteigenschaften – Beeinflussung der Animation

- Die Art und Weise der Annäherung an die Zielwerte kann über verschiedene Interpolatoren erreicht werden:

- `AccelerateDecelerateInterpolator`—The rate of change starts and ends slowly but accelerates through the middle.
- `AccelerateInterpolator`—The rate of change starts slowly but accelerates through the middle.
- `AnticipateInterpolator`—The change starts backward and then flings forward.
- `AnticipateOvershootInterpolator`—The change starts backward, flings forward, overshoots the target value, and finally goes back to the final value.
- `BounceInterpolator`—The change bounces at the end.
- `DecelerateInterpolator`—The rate of change starts out quickly and then decelerates.
- `LinearInterpolator`—The rate of change is constant.
- `OvershootInterpolator`—The change flings forward, overshoots the last value, and then comes back.

```
anim.setInterpolator(new AnticipateOvershootInterpolator());
```

You can also extend your own `TimeInterpolator` class to specify a custom interpolation algorithm.

„Take-Away“ für diese Einheit

- Erstellen von Responsive Android Apps
- Einsatz von Fragments
- Themes und Styles in Android
- Animationen in Android