Android: Buttons *(Projekt SelectionsAndroid)*

zu Abschnitt 4.3

Die Applikation demonstriert den Umgang mit einfachen Buttons.

java/.../ButtonsDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;

public class ButtonsDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.buttons);                                  Layout mit mehreren Buttons (siehe unten)

  ((Button) findViewById(R.id.buttonA))                              Referenzen auf die beiden ersten Buttons
                  .setOnClickListener(new MyButtonListener());       (siehe res/layout/buttons.xml) beschaffen
  ((Button) findViewById(R.id.buttonB))                              und den Buttons einen Listener (siehe unten) zuordnen.
                  .setOnClickListener(new MyButtonListener());

 }
```

Möglichkeit 1 (hier benutzt für die ersten beiden Buttons): Die Reaktion auf einen Button-Click durch die Methode `onClick()` eines Button-Listeners festlegen (hier: `MyButtonListener`) und ein entsprechendes Listener-Objekt durch `setOnClickListener()` an den Button binden (siehe oben).

```
 class MyButtonListener implements View.OnClickListener {

  public void onClick(View v) {
   String ausgabe = ((Button) v).getText().toString();
   Toast.makeText(getApplicationContext(),ausgabe, Toast.LENGTH_LONG).show();
  }

 }
```

Möglichkeit 2 (hier benutzt für die übrigen Buttons): Die Reaktion auf einen Button-Click durch eine Methode innerhalb der Activity definieren (hier: `clickReaktion()`) und diese Methode durch das XML-Attribut `android:onClick` an den Button binden (siehe res/layout/buttons.xml).

```
 public void clickReaktion(View v) {
  String ausgabe = "";
```

```
  if (v.getId()==R.id.buttonC || v.getId()==R.id.buttonD)
   ausgabe = ((Button) v).getText().toString();
  if (v.getId()==R.id.buttonDelete)
   ausgabe = "DELETE";
  if (v.getId()==R.id.buttonDone)
   ausgabe = "DONE";
  Toast.makeText(getApplicationContext(),ausgabe, Toast.LENGTH_LONG).show();
 }
```

res/layout/buttons.xml: Layout mit mehreren Buttons

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

 <Button
   android:id="@+id/buttonA"
   android:text="Alpha"                                             Beschriftung des Buttons
   android:layout_width="match_parent"                              Button soll so breit wie das Display sein
   android:layout_height="wrap_content" />                          Button soll nur so hoch wie nötig sein

 <Button
   android:id="@+id/buttonB"
   android:text="Beta"
   android:layout_width="match_parent"
   android:layout_height="wrap_content" />

 <Button
   android:id="@+id/buttonC"
   android:text="Gamma"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:onClick="clickReaktion" />                               Reaktion bei Button-Click: clickReaktion()

 <Button
   android:id="@+id/buttonD"
   android:text="Delta"
   android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:onClick="clickReaktion" />
```

```
  <LinearLayout                                                    Horizontales Layout für zwei Buttons nebeneinander
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button                                                        Button mit Icon und Text
     android:id="@+id/buttonDelete"
     android:drawableLeft="@drawable/ic_delete_black_24dp"         Icon des Buttons
     android:text="Delete"                                         Text des Buttons
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:layout_weight="1"                                     Darstellung genau so breit wie der andere Button
     android:onClick="clickReaktion" />                            Reaktion bei Button-Click: clickReaktion()

    <Button                                                        Button mit Icon und Text
     android:id="@+id/buttonDone"
     android:drawableLeft="@drawable/ic_done_black_24dp"           Icon des Buttons
     android:text="Done"                                           Text des Buttons
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:layout_weight="1"                                     Darstellung genau so breit wie der andere Button
     android:onClick="clickReaktion" />                            Reaktion bei Button-Click: clickReaktion()

  </LinearLayout>

</LinearLayout>
```

Android: Checkboxen und Radiobuttons *(Projekt SelectionsAndroid)*

zu Abschnitt 4.3

Die Applikation demonstriert den Umgang mit Checkboxen und Radiobuttons.

java/.../CheckboxRadioToggleDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.*;
import android.os.Bundle;
import android.widget.*;

public class CheckboxRadioToggleDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.checkboxradiotoggle);                    Drei Checkboxen; Gruppe mit drei Radiobuttons (s.u.)

  CheckBox checkbox1 = (CheckBox) findViewById(R.id.checkbox1);    Ermittlung der Checkbox ...
  checkbox1.setOnCheckedChangeListener(                            ... und Zuordnung eines Listeners
                      new MyCheckboxOnCheckedChangeListener());    (Definition des Listeners siehe unten)
```

... für `checkbox2`, `checkbox3` analog ...

```
  RadioGroup group = (RadioGroup) findViewById(R.id.radiogroup);   Ermittlung der RadioGroup ...
  group.setOnCheckedChangeListener(                                ... und Zuordnung eines Listeners
                      new MyRadioGroupOnCheckedChangeListener());  (Definition des Listeners siehe unten)

 }
```

Listener für die Checkboxen: Ändert sich der Auswahlstatus einer Checkbox, so erscheint ein Toast, der ihren neuen Status textuell anzeigt.

```
 class MyCheckboxOnCheckedChangeListener implements CompoundButton.OnCheckedChangeListener {

  public void onCheckedChanged(CompoundButton button, boolean isChecked) {

   String ausgabe = button.getText()+": ";
   if (isChecked) ausgabe += "markiert";
    else ausgabe += "nicht markiert";
   Toast.makeText(CheckboxRadioDemo.this,ausgabe,Toast.LENGTH_LONG).show();

  }
 }
```

Listener für die Gruppe der Radiobuttons: Wird ein Radiobutton ausgewählt, so erscheint ein Toast, der den aktuell gewählten Radiobutton nennt.

```
class MyRadioGroupOnCheckedChangeListener implements RadioGroup.OnCheckedChangeListener {
 public void onCheckedChanged(RadioGroup group, int checkedId) {
  int buttonId = group.getCheckedRadioButtonId();
  RadioButton button = (RadioButton) findViewById(buttonId);
  String ausgabe = "Gewählt: "+ button.getText();
  Toast.makeText(CheckboxRadioDemo.this,ausgabe, Toast.LENGTH_LONG).show();
 }  }  }
```

res/layout/checkboxradiotoggle.xml: Layout mit drei Checkboxen und einer RadioGroup mit drei Radiobuttons untereinander

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent" android:layout_height="match_parent" >
 <CheckBox
  android:id="@+id/checkbox1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/checkbox1" />                    Beschriftung der Checkbox
 ... checkbox2, checkbox3 analog ...
 <RadioGroup android:id="@+id/radiogroup"               RadioGroup, die die RadioButtons umschließt
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical">
  <RadioButton
   android:id="@+id/radio1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="@string/radio1" />                     Beschriftung des Radiobuttons
 ... radio2, radio3 analog ...
 </RadioGroup>
</LinearLayout>
```

Android: Togglebuttons *(Projekt SelectionsAndroid)* zu Abschnitt 4.3

Die Applikation demonstriert den Umgang mit Togglebuttons (= Ein-Aus-Buttons). Switches (= Ein-Aus-Schieberegler) werden genauso programmiert.

java/.../CheckboxRadioToggleDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.*;
import android.os.Bundle;
import android.widget.*;

public class CheckboxRadioToggleDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.checkboxradiotoggle);                    Zwei Togglebuttons

  ToggleButton toggle1 = (ToggleButton) findViewById(R.id.toggle1);Ermittlung des Togglebuttons ...
  toggle1.setOnCheckedChangeListener(                              ... und Zuordnung eines Listeners
                    new MyToggleButtonOnCheckedChangeListener());  (Definition des Listeners siehe unten)
```

... für `toggle2` analog ...

```
 }
```

Listener für die Togglebuttons: Ändert sich der Status eines Buttons, so erscheint ein Toast, der den neuen Status anzeigt.

```
 class MyToggleButtonOnCheckedChangeListener implements CompoundButton.OnCheckedChangeListener {

  public void onCheckedChanged(CompoundButton button, boolean isChecked) {

   String ausgabe = button.getText().toString();

   if (isChecked)
     ausgabe += " ist nun an";
    else
     ausgabe += " ist nun aus";

   Toast.makeText(CheckboxRadioDemo.this,ausgabe,Toast.LENGTH_LONG).show();

  }
 }
}
```

res/layout/checkboxradiotoggle.xml: Layout mit zwei Togglebuttons untereinander

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

 <ToggleButton
  android:id="@+id/toggle1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textOn="@string/toggle1on"                    Beschriftung des Buttons im Zustand “On”
  android:textOff="@string/toggle1off" />               Beschriftung des Buttons im Zustand “Off”

 ... toggle2 analog ...

</LinearLayout>
```

Android: Seekbar *(Projekt SelectionsAndroid)*

zu Abschnitt 4.3

Die Applikation demonstriert den Umgang mit Seekbars (= Schiebereglern).

java/.../SeekbarDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.*;
import android.os.Bundle;
import android.widget.*;

public class SeekbarDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.seekbar);
```

Anzeige von Seekbar und Ausgabefeld

```
  SeekBar mySeekBar = (SeekBar) findViewById(R.id.mySeekBar);
  mySeekBar.setOnSeekBarChangeListener(new MySeekBarListener());
```

Beschaffung eines Referenz auf die Seekbar
Zuordnung eines Listeners (siehe unten)

```
 }
```

Listener für die Seekbar: Der jeweils gewählte Wert wird im Textausgabefeld angezeigt.

```
 private class MySeekBarListener implements SeekBar.OnSeekBarChangeListener {

   EditText ausgabefeld = (EditText) findViewById(R.id.ausgabefeld);
```

`onProgressChanged()` wird aufgerufen, wenn der Benutzer den Regler bewegt. `progress` gibt die aktuelle Position des Reglers an (zwischen dem Minimal- und dem Maximalwert, die in der XML-Layout-Datei vorgegeben werden können – Defaultwerte sind 0 und 100).

```
  public void onProgressChanged(SeekBar seekBar, int progress, boolean fromTouch) {

   ausgabefeld.setText(progress+"");

  }
```

`onStartTrackingTouch()` wird aufgerufen, wenn die Reglerbewegung beginnt.

```
  public void onStartTrackingTouch(SeekBar seekBar) { }
```

`onStopTrackingTouch()` wird aufgerufen, wenn die Reglerbewegung endet.

```
  public void onStopTrackingTouch(SeekBar seekBar) { }
}
```

res/layout/seekbar.xml: Layout mit Seekbar und Textfeld

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

 <SeekBar
  android:id="@+id/mySeekBar"
  android:layout_width="match_parent"
  android:layout_marginRight="30dp"
  android:layout_height="wrap_content"
  android:max="200" />

 <EditText
  android:id="@+id/ausgabefeld"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" />

</LinearLayout>
```

Abstand vom rechten Rand (gibt Raum, so dass man den Regler bis zum Anschlag ziehen kann)

maximal wählbarer Wert

zur Ausgabe des aktuell gewählten Werts

Android: Menus und Action Bar *(Projekt MenusAndroid)*

zu Abschnitt 4.3

Die Applikation demonstriert die Benutzung von Optionsmenus bzw. Action Bars sowie von Kontextmenus.

java/.../ActivityAlpha.java: Erste Activity

Die Activity zeigt einen Text und eine Grafik an. Über ein Optionsmenu (= Menu der Menutaste des Geräts; im Emulator auch zugänglich mit F2) kann man zu einer anderen Activity weiterschalten. Statt eines Optionsmenus kann eine Action Bar benutzt werden (ab Android 3.0 / API-Level 11).

```
package de.thkoeln.cvogt.android.menus;

import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.content.Intent;

public class ActivityAlpha extends Activity {

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.layout_alpha);
 }
```

Zeigt eine Grafik an (siehe unten: layout_alpha.xml)

`onCreateOptionsMenu()` wird automatisch ausgeführt, wenn der Menu-Button zum ersten Mal geklickt wurde (bei Verwendung eines Menu Buttons) bzw. die Activity gestartet wird (bei Verwendung einer Action Bar). Hier wird die `inflate()`-Methode eines `MenuInflaters` aufgerufen, die das Menu bzw. die Action Bar aus einer XML-Beschreibung (siehe unten: menu_alpha.xml) erzeugt. Der erste `inflate()`-Parameter gibt die Menu-Elemente an, die dargestellt werden sollen (also die XML-Beschreibung). Der zweite Parameter nennt das Menu, in das diese Elemente gebracht werden sollen (hier: Standardmenu der Activity; wurde beim automatischen Aufruf von der Activity als Parameter an `onCreateOptionsMenu(Menu menu)` übergeben).

```
 public boolean onCreateOptionsMenu(Menu menu) {

  super.onCreateOptionsMenu(menu);
  MenuInflater mi = new MenuInflater(this);
  mi.inflate(R.menu.menu_alpha,menu);
  return true;

 }
```

Zusätzlich könnte man die Methode `onPrepareOptionsMenu()` ausprogrammieren, die bei jeder Anzeige des Menus (also auch bei den folgenden Betätigungen des Menu-Buttons) ausgeführt wird (bzw., bei neueren Android-Versionen, durch Aufruf der Methode `invalidateOptionsMenu()`). Hierdurch könnte man die Menu-Einträge jeweils aktualisieren.

`onOptionsItemSelected()` wird ausgeführt, wenn ein Item des Menus angeklickt wurde. Dabei wird dieses Item als Parameter übergeben.

```
 public boolean onOptionsItemSelected(MenuItem item) {
  if (item.getItemId()==R.id.menuItemAlpha)
    startActivity(new Intent(this,ActivityBeta.class));
  if (item.getItemId()==R.id.menuItemGamma)
    startActivity(new Intent(this,ActivityGamma.class));
  return true;
 }
}
```

Wenn Item „beta“ angeklickt wurde, dann starte `ActivityBeta`.

Wenn Item „gamma“ angeklickt wurde, dann starte `ActivityGamma`.

java/.../ActivityBeta.java: Zweite Activity – analog

java/.../ActivityGamma.java: Dritte Activity

In Erweiterung der beiden anderen Activities hat diese Activity ein Kontextmenu, das sich bei Anklicken der Grafik, die auf der Oberfläche angezeigt ist, öffnet. Über dieses Kontextmenu kann man zu einer der beiden anderen Activities umschalten.

```
package de.thkoeln.cvogt.android.menus;

import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.view.ContextMenu.*;
import android.content.Intent;

public class ActivityGamma extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.layout_gamma);
```

Der folgende `registerForContextMenu()`-Aufruf registriert die angezeigte Grafik „gamma“ zur Anzeige eines Kontextmenus – legt also fest, dass beim Klicken auf die Grafik ein Kontextmenu erscheinen soll. Fehlt dieser Aufruf, so hat ein Anklicken keinen Effekt. Das konkrete Kontextmenu, das dann erscheinen soll, wird durch `onCreateContextMenu()` (siehe unten) definiert. Um den Namen „gamma“ bekanntzumachen, muss in layout_gamma.xml in die `<ImageView>`-Komponente per Hand "`android:id="@+id/gamma`" eingesetzt werden.

```
  registerForContextMenu(findViewById(R.id.gamma));
 }
```

```
public boolean onCreateOptionsMenu(Menu menu) { wie oben }
public boolean onOptionsItemSelected(MenuItem item) { wie oben }
```

`onCreateContextMenu()` wird bei einem Klick auf die Grafik, die durch `registerForContextMenu()` dafür vorbereitet wurde, ausgeführt. Es wird die `inflate()`-Methode eines `MenuInflaters` aufgerufen, die das Menu aus einer XML-Beschreibung (gemäß der Datei menu_gamma.xml, die analog zu menu_alpha.xml definiert ist) erzeugt. Der erste `inflate()`-Parameter gibt die Menu-Elemente an, die dargestellt werden sollen (d.h. das XML-Menu-File, das sie spezifiziert). Der zweite Parameter nennt das Menu, in das diese Elemente gebracht werden sollen (hier: Standardmenu der Activity; wurde von der Activity als Parameter an `onCreateContextMenu()` übergeben).

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo info) {
 super.onCreateContextMenu(menu,v,info);
 MenuInflater mi = new MenuInflater(getApplication());
 mi.inflate(R.menu.menu_gamma, menu);
}
```

`onContextItemSelected()` wird ausgeführt, wenn ein Item des Kontextmenus angeklickt wurde. Dabei wird dieses Item als Parameter übergeben.

```
public boolean onContextItemSelected(MenuItem item) {
 if (item.getItemId()==menuItemAlpha)
   startActivity(new Intent (this, ActivityAlpha.class));
 if (item.getItemId()==menuItemBeta)
   startActivity(new Intent (this, ActivityBeta.class));
 return true;
}
}
```

Wenn Item „alpha“ angeklickt wurde, dann starte `ActivityAlpha`.
Wenn Item „beta“ angeklickt wurde, dann starte `ActivityBeta`.

Ein Popupmenu wird nahe bei der Grafik durch folgende Programmschritte angezeigt (einzubetten z.B. in den Listener eines Buttons; ein Benutzer-Click auf den Button lässt somit das Menu erscheinen):

```
PopupMenu menu = new PopupMenu(this,findViewById(R.id.gamma));
menu.getMenuInflater().inflate(R.menu.menu_gamma,menu.getMenu());
menu.setOnMenuItemClickListener(new PopupMenuItemListener());
menu.show();
```

Der Listener des Popupmenus entspricht den Methoden `onOptionsItemSelected()` für das Options- und `onContextItemSelected()` für das Kontextmenu:

```
private class PopupMenuItemListener implements PopupMenu.OnMenuItemClickListener {
 public boolean onMenuItemClick(MenuItem item) {
  if (item.getItemId()==R.id.menuItemAlpha)
   startActivity(new Intent(ActivityGamma.this, ActivityAlpha.class));
  if (item.getItemId()==R.id.menuItemBeta)
   startActivity(new Intent(ActivityGamma.this, ActivityBeta.class));
  return true;
 }
}
```

res/layout/layout_alpha.xml: Layout mit einem Text und einer Grafik darunter

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"                                  Senkrechte Ausrichtung
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
 <TextView android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:layout_marginTop="20dp"                        Abstand nach oben
           android:layout_marginBottom="20dp"                     Abstand nach unten
           android:textSize="16pt"
           android:gravity="center"
           android:text="@string/activity_alpha" />               Text „Activity Alpha“
 <ImageView android:src="@drawable/alpha"                         Grafik „Alpha“
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
           android:layout_gravity="center" />
</LinearLayout>
```

res/layout/layout_beta.xml, **layout_gamma.xml**: analog

res/menu/menu_alpha.xml: **Definition des Menus** für die erste Activity

```
<?xml version="1.0" encoding="UTF-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:id="@+id/menuItemBeta" android:title="beta" />      „Beta“-Item
       Wenn das Item in der Action Bar erscheinen soll, zusätzlich: android:showAsAction="always|withText"
 <item android:id="@+id/menuItemGamma" android:title="gamma" />    „Gamma“-Item
</menu>
```

Im Fall von Action Bars kann man statt des Texts ein Icon anzeigen lassen. Das geschieht über das `icon`-Attribut im `item`-Element, z.B. `android:icon="@drawable/ic_action_discard"`. Icons findet man unter https://material.io/tools/icons/.

res/menu/menu_beta.xml, **menu_gamma.xml**: analog

res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string name="app_name">MenusAndroid</string>
 ...
 <string name="activity_alpha">Activity Alpha</string>
 <string name="activity_beta">Activity Beta</string>
 <string name="activity_gamma">Activity Gamma</string>
</resources>
```

res/values/styles.xml:

Will man eine Action Bar anzeigen, so ist es wichtig, ein Theme für die Applikation zu wählen, das Action Bars unterstützt – wie z.B. `Theme.Holo.Light`:

```
<resources>

 <style name="AppBaseTheme" parent="android:Theme.Holo.Light">
 </style>

 <style name="AppTheme" parent="AppBaseTheme">
 </style>

</resources>
```

Soll keine Action Bar angezeigt werden, so wird das ebenfalls durch ein entsprechendes Theme spezifiziert – z.B. `Theme.Holo.NoActionBar`.

Android: ListActivity *(Projekt SelectionsAndroid)*

zu Abschnitt 4.3

Die Applikation demonstriert ein Auswahlangebot durch eine `ListActivity`, also ohne explizite Layout-Definition. (Einen alternativen Ansatz findet man im Beispielprojekt *SelectionsAndroid*: Dort wird ein Layout mit einem `ListView`-Element definiert und explizit durch `setContentView()` angezeigt.)

java/.../ListActivityDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.ListActivity; import android.os.Bundle;
import android.view.View; import android.widget.*;

public class ListActivityDemo extends ListActivity {
 String[] choices = { "Alpha", "Beta", "Gamma", "Delta" };
 ListAdapter demoListAdapter;
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setTitle("ListActivity");
  // setContentView(R.layout.main);

  demoListAdapter = new ArrayAdapter<String>(
           this,android.R.layout.simple_list_item_1,choices);
  setListAdapter(demoListAdapter);

 }
 protected void onListItemClick(ListView liste,
                  View datenElement, int position, long id) {
  super.onListItemClick(liste, datenElement, position, id);
  CharSequence tastenText =
                   ((TextView)datenElement).getText();
  final Toast hinweis = Toast.makeText(this,
                    "Taste: "+tastenText, Toast.LENGTH_LONG);
  hinweis.show();
 }  }
```

ListActivity, keine „normale“ Activity!

Anzuzeigende Listenelemente

Adapter zur Aufnahme der Listenelemente (siehe unten)

Dieser Aufruf muss fehlen: Der ContentView wird bei einer ListActivity automatisch gesetzt!
Der Adapter nimmt die Listenelemente auf und legt ein Layout für sie fest (hier: Standard-Layout).

Verknüpft den Adapter (und damit auch der Listenelemente) mit der ListActivity

Wird bei Auswahl eines Listenelements aufgerufen

Ermittelt die Beschriftung des gewählten Listenelements

Zeigt die Beschriftung in einem Toast (= temporärem Popup-Fenster) an

Android: Spinner *(Projekt SelectionsAndroid)*

zu Abschnitt 4.3

Die Applikation demonstriert einen `AdapterView` am Beispiel eines Spinners, also einer Auswahlliste.

java/.../SpinnerDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.*;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
import java.util.*;

public class SpinnerDemo extends Activity {
 String items[] = { "Alpha", "Beta", "Gamma" };
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.spinner);

  Spinner spinner = (Spinner) findViewById(R.id.spinner);
  List<String> spinnerList = new ArrayList<String>();
  for (int i=0;i<items.length;i++)
   spinnerList.add(items[i]);
  ArrayAdapter<String> spinnerAdapter =
   new ArrayAdapter<String>(
    this,android.R.layout.simple_spinner_item,spinnerList);

  spinnerAdapter.setDropDownViewResource(
          android.R.layout.simple_spinner_dropdown_item);
  spinner.setAdapter(spinnerAdapter);
  spinner.setOnItemSelectedListener(
                      new MyOnItemSelectedListener());
 }
```

Items, die im Spinner angezeigt werden sollen

Zeigt den Spinner an
(Definition in res/layout/spinner.xml – siehe unten)

Ermittelt Referenz auf den Spinner

Erstellt eine `ArrayList` mit den anzuzeigenden Items

Der Adapter nimmt die Items aus der ArrayList auf
und legt ein Layout für den geschlossenen Spinner fest
(hier: Standard-Layout).
Der Adapter wird unten mit der Auswahlliste verknüpft.

Legt ein Layout für die Items bei geöffneten Spinner
fest (hier: Standard-Layout).

Verknüpft den Spinner mit dem Adapter von oben

Ordnet dem Spinner einen Listener zu
(Definition siehe unten)

```
 class MyOnItemSelectedListener                                    Listener, der aktiv wird,
              implements AdapterView.OnItemSelectedListener {      wenn ein Item des Spinners ausgewählt wurde
  public void onNothingSelected(AdapterView<?> parent) { }

  public void onItemSelected(                                      Wird aufgerufen, wenn ein Item ausgewählt wurde
     AdapterView<?> parent, View view, int position, long id) {
   String ausgabe = "onItemSelected: position="
                                        +position+" id="+id;
   Toast.makeText(SpinnerDemo.this,ausgabe,                        Zeigt Position und Id des gewählten Items an
                                Toast.LENGTH_LONG).show();
  }
 }
}
```

res/layout/spinner.xml: Layout mit einem Spinner

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
 <Spinner                                                         Auswahlliste
   android:id="@+id/spinner"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="@string/spinner"
 />
</LinearLayout>
```

Android: Gallery *(Projekt SelectionsAndroid)*

zu Abschnitt 4.3

Die Applikation demonstriert, wie ein Folge von Bildern durch eine horizontale `Gallery` angezeigt wird. Die Klasse `Gallery` ist zwar seit API-Level 16 / Android 4.1 "deprecated", kann aber weiterhin benutzt werden und und ermöglicht eine recht bequeme Programmierung. Alternativ kann man die Klassen `HorizontalScrollView` oder `ViewPager` verwenden – siehe z.B. http://stackoverflow.com/questions/15833889/options-for-replacing-the-deprecated-gallery. Eine gute Alternative ist auch die Klasse `StackView`, die die Items schräg hintereinander anordnet. Die Programmierung ist hier sehr ähnlich zu der einer Gallery (siehe Code im Beispielprojekt SelectionsAndroid).

java/.../GalleryDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.Activity;
import android.os.Bundle;
import android.content.*;
import android.view.*;
import android.widget.*
import android.graphics.BitmapFactory;

public class GalleryDemo extends Activity {

 static Gallery gallery;                                  Galerie, die weiter unten mit Bildern gefüllt wird
 static Button leftButton, rightButton;                   Buttons zum Blättern in der Galerie

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.gallery);

  gallery = (Gallery) findViewById(R.id.bildauswahl);     Ermittelt die anzuzeigende Galerie (siehe gallery.xml)
  gallery.setAdapter(new MyImageAdapter(this));           Ordnet ihr den Adapter mit den Bildern zu (siehe unten)

  leftButton = (Button) findViewById(R.id.leftButton);    Buttons zum Blättern in der Galerie
  leftButton.setOnClickListener(new MyButtonListener());

  rightButton = ... analog ...  }

 class MyButtonListener implements View.OnClickListener {   Wird bei Click auf einen der beiden Buttons aktiv

  public void onClick(View v) {
   if (v==leftButton && gallery.getSelectedItemPosition()>0)   Blättert ein Bild nach links
    gallery.setSelection(gallery.getSelectedItemPosition()-1);
   if (v==rightButton
```

```
     && gallery.getSelectedItemPosition()<gallery.getCount()-1)      Blättert ein Bild nach rechts
    gallery.setSelection(gallery.getSelectedItemPosition()+1);
  }
 }
```

Die Klasse `MyImageAdapter` stellt, als Unterklasse von `BaseAdapter`, die anzuzeigenden Bilder für die `Gallery` bereit. Sie ist eine Vereinfachung eines Beispiels, das sich früher unter https://developer.android.com/resources/tutorials/views/hello-gallery.html befand.

```
class MyImageAdapter extends BaseAdapter {
 private Integer[] myImageIds = {                                    IDs der Bilder (mit entsprechenden
  R.drawable.granada01,                                                          PNG-Dateien in /res/drawable-hdpi))
  R.drawable.granada02, R.drawable.granada03, R.drawable.granada04, R.drawable.granada05 };
 public int getCount() {                                             Anzahl der anzuzeigenden Bilder
  return myImageIds.length; }
 public Object getItem(int position) {                               Implementiert abstrakte Methode von BaseAdapter
  return BitmapFactory.decodeResource(getResources(),myImageIds[position]); }
 public long getItemId(int position) {                               Implementiert abstrakte Methode von BaseAdapter
  return position; }
 public View getView(int position,                                   Liefert ein Bild zur Anzeige
                     View convertView, ViewGroup parent) {
  ImageView imageView = new ImageView(GalleryDemo.this);
  imageView.setImageResource(myImageIds[position]);                  Ermittelt das Bild an der aktuellen Galerie-Position
  imageView.setLayoutParams(new Gallery.LayoutParams(300,200));      Legt Größe der Bildanzeige fest
  imageView.setScaleType(ImageView.ScaleType.FIT_XY);                Bild soll diese Anzeige ausfüllen
  return imageView;
 }
}
}
```

res/layout/gallery.xml: Bildergalerie und darunter zwei nebeneinanderliegende Buttons zum Blättern in der Galerie

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

```
  android:orientation="vertical" >
 <Gallery
   android:id="@+id/bildauswahl"
   android:layout_width="match_parent"
   android:layout_height="180dp"
   android:spacing="30dp"
   android:animationDuration="400" />
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal" >
  <Button
    android:id="@+id/leftButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="14pt"
    android:layout_weight="1"
    android:text="@string/left" />
  <Button
    android:id="@+id/rightButton"
    ... analog ... />
 </LinearLayout>
</LinearLayout>
```

Android: GridView *(Projekt SelectionsAndroid)* zu Abschnitt 4.3

Die Applikation demonstriert die zweidimensionale Anordnung von Items (hier: Bildern) in einem GridView.
(Grundlage ist der Beispielcode aus https://developer.android.com/guide/topics/ui/layout/gridview.html.)

java/.../GridViewDemo.java:

```
package de.thkoeln.cvogt.android.selections;

import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
import android.widget.AdapterView.*;
import android.graphics.BitmapFactory;

public class GridViewDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.gridview);                          GridView-Layout (siehe res/layout/gridview.xml)

  GridView gridview = (GridView) findViewById(R.id.gridview);  Ermittlung des GridViews
  gridview.setAdapter(new MyImageAdapter());                  Adapter mit Bildern, die im GridView angezeigt werden
```

Setzen eines Listeners, der aktiv wird, wenn eines der Items im GridView angeklickt wird. Der Listener zeigt die Positionsnummer des gewählten Bilds durch einen Toast an.

```
  gridview.setOnItemClickListener(new OnItemClickListener() {
   public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
    Toast.makeText(GridViewDemo.this, "" + position, Toast.LENGTH_SHORT).show();
   }
  });  }
```

Die Klasse `MyImageAdapter` ist im Wesentlichen dieselbe wie im Gallery-Beispiel . Lediglich die Bilddarstellungen sind kleiner, um mehr als ein Bild auf der Oberfläche anzeigen zu können.

```
 class MyImageAdapter extends BaseAdapter {

  private Integer[] myImageIds = {                            IDs der Bilder (mit entsprechenden
   R.drawable.granada01,                                                    PNG-Dateien in /res/drawable-hdpi)
   R.drawable.granada02, R.drawable.granada03, R.drawable.granada04, R.drawable.granada05 };
```

```
  public int getCount() {                                          Anzahl der anzuzeigenden Bilder
   return myImageIds.length; }
  public Object getItem(int position) {                            Implementiert abstrakte Methode von BaseAdapter
   return BitmapFactory.decodeResource(getResources(),myImageIds[position]); }
  public long getItemId(int position) {                            Implementiert abstrakte Methode von BaseAdapter
   return position; }
  public View getView(int position,                                Liefert ein Bild zur Anzeige
                      View convertView, ViewGroup parent) {
   ImageView imageView = new ImageView(GalleryDemo.this);
   imageView.setImageResource(myImageIds[position]);               Ermittelt das Bild an der aktuellen Galerie-Position
   imageView.setLayoutParams(new Gallery.LayoutParams(75,50));     Legt Größe der Bildanzeige fest
   imageView.setScaleType(ImageView.ScaleType.FIT_XY);             Bild soll diese Anzeige ausfüllen
   return imageView;
  }
 }
}
```

res/layout/gridview.xml: Layout mit einem GridView

```
<?xml version="1.0" encoding="UTF-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/gridview"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:columnWidth="90dp"                                       Feste Spaltenbreite: 90 Pixel
 android:numColumns="auto_fit"                                    Anzahl zu zeigender Spalten: Ergibt sich aus
                                                                        Displaygröße und columnWidth
 android:verticalSpacing="10dp"                                   Abstand zwischen Spalten: 10 Pixel
 android:horizontalSpacing="10dp"                                 Abstand zwischen Zeilen: 10 Pixel
 android:stretchMode="columnWidth"                                Spalten sollen gleichmäßig gestreckt werden,
                                                                        um leeren Platz zu füllen
 android:gravity="center"                                         Zentrale Plazierung des GridViews
/>
```

Android: Toasts *(Projekt NotifDialogsAndroid)*

zu Abschnitt 4.4

Die Applikation demonstriert die Anzeige von Toasts (also temporären Popup-Fenstern) mit Text und mit Bild.

java/.../ToastDemo.java:

```
package de.thkoeln.cvogt.android.notifdialogs;

import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;

public class ToastDemo extends Activity {
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
```

Einfacher Toast mit Text:

```
  Toast.makeText(this,"Toast mit Text",Toast.LENGTH_LONG).show();
```

`this` referenziert die Activity

Toast mit selbstdefiniertem Layout:

```
  Toast t = new Toast(this);
  LayoutInflater inflater = getLayoutInflater();
  View layout = inflater.inflate(R.layout.toastlayout,
                (ViewGroup) findViewById(R.id.ToastLayout));
  t.setView(layout);
  t.setDuration(Toast.LENGTH_LONG);
  t.show();
 }
}
```

Legt das Layout des Toasts fest
(siehe nächste Seite: res/layout/toastlayout.xml)

Legt die Dauer der Anzeige fest

Zeigt den Toast an

res/layout/toastlayout.xml: Layout mit einem Text und einem Bild untereinander

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/ToastLayout"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

 <TextView
   android:layout_width="match_parent"
   android:layout_height="60dp"
   android:textSize="10pt"
   android:gravity="bottom"
   android:text="@string/toastueberschrift" />

 <ImageView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/sunset" />

</LinearLayout>
```

Bilddatei sunset.jpg in /res/drawable-hdpi

Android: Status Bar Notifications *(Projekt NotifDialogsAndroid)*

zu Abschnitt 4.4

Die Applikation demonstriert die Anzeige einer Meldung („Notification“) in der Statusbar.

java/.../StatusBarDemo.java:

```
package de.thkoeln.cvogt.android.notifdialogs;

import android.app.*;
import android.content.*;
import android.os.Bundle;

public class StatusBarDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
```

Attributwerte der Notification definieren:

```
  int notifIcon = android.R.drawable.ic_dialog_info;
```
Icon für die Status Bar.
Detailinformationen zu Icons findet man unter https://developer.android.com/guide/practices/ui_guidelines/icon_design_status_bar.

```
  CharSequence notifTitel
                    = "--- Benachrichtigung ---";
```
Titel für den Expanded View
= für das Notifications Window

```
  CharSequence notifText
              = "Dies ist eine Benachrichtigung für den Benutzer";
```
Text für den Expanded View
= für das Notifications Window

```
  long notifTime = System.currentTimeMillis();
```
Zeitstempel der Notification

Pending Intent erzeugen: Soll abgeschickt werden, wenn der Benutzer die Notification ausgewählt hat, und damit eine Activity zur Reaktion auf die Benutzerauswahl starten (siehe unten: ReactionToNotification.java).

```
  Intent notifIntent = new Intent(this,ReactionToNotification.class);
  PendingIntent pendingNotifIntent = PendingIntent.getActivity(this,0,notifIntent,0);
```

Notification erzeugen: Die `set`-Methoden eines Builders spezifizieren die Eigenschaften der Notification.

```
  Notification notif = new Notification.Builder(this)
                             .setContentTitle(notifTitel)
                             .setContentText(notifText)
                             .setSmallIcon(notifIcon)
                             .setContentIntent(pendingNotifIntent)
                             .setWhen(notifTime)
                               .build();
```

Referenz auf den Notification Manager beschaffen und die Notification dorthin senden, damit sie angezeigt wird. Der Notification Manager ist ein Systemdienst, der alle Notifications verwaltet.

```
  NotificationManager notifMgr = (NotificationManager)
                  getSystemService(Context.NOTIFICATION_SERVICE);
  final int NOTIF_ID = 1;                                          Eindeutige Id (innerhalb dieser App) für die Notification
  notifMgr.notify(NOTIF_ID,notif);                                 Notification anzeigen
 }
}
```

java/.../ReactionToNotification.java: Activity, die nach Anklicken der Notification ausgeführt wird (siehe oben)

```
package de.thkoeln.cvogt.android.notifdialogs;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;

public class ReactionToNotification extends Activity {
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.reactionactivity);                       Beliebiges Layout (z.B. Anzeige eines TextViews)
 }
}
```

Android: Alert Dialog *(Projekt NotifDialogsAndroid)*

zu Abschnitt 4.4

Die Applikation demonstriert Alertdialoge, also temporär erscheinende Fenster mit Auswahlbuttons und -listen. (Die hier besprochenen Methoden `onCreateDialog()` usw. sind zwar seit API-Level 13 / Android 3.0 "deprecated"; empfohlen wird, stattdessen Fragment-bezogene Methoden zu benutzen. Allerdings können auch die ursprünglichen Methoden weiter verwendet werden, und die Arbeit mit Fragments ist deutlich komplexer.)

java/.../AlertDemo.java:

```
package de.thkoeln.cvogt.android.notifdialogs;

import android.app.*;
import android.content.*;
import android.os.Bundle;
import android.widget.Toast;

public class AlertDemo extends Activity {

 static final int JA_NEIN_VIELLEICHT_DIALOG_ID = 1;
 static final int AUSWAHL_DIALOG_ID = 2;
```

Codes zur Übergabe an `showDialog()`, die verschiedene Dialoge identifizieren (siehe unten)

```
 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);

  showDialog(JA_NEIN_VIELLEICHT_DIALOG_ID,null);
  showDialog(AUSWAHL_DIALOG_ID,null); }
```

Zeigt die Dialoge an (Details zu ihrem Aussehen siehe `onCreateDialog()` im Anschluss)

`onCreateDialog()` wird ausgeführt, wenn im Programm die Methode `showDialog()` aufgerufen wird und zuvor noch kein Dialog mit der übergebenen Id erzeugt und zwischengespeichert wurde. Der `showDialog()`-Parameter wird an `onCreateDialog()` übergeben und kann (in einer switch-case-Anweisung) zur Unterscheidung verschiedener Dialoge genutzt werden. `onCreateDialog()` erzeugt ein `Dialog`-Objekt und gibt es zurück, worauf es automatisch angezeigt wird. Im zweiten Parameter kann ein Bundle mit weiteren Daten übergeben werden.

```
 protected Dialog onCreateDialog(int id, Bundle args) {

  AlertDialog.Builder dialogBuilder;
```

DialogBuilder zum Aufbau des Dialogs

```
  AlertDialog dialog;
```

Variable für den aufzubauenden Dialog

```
  switch(id) {
```

In switch-case: Definition zweier verschiedener Dialoge

1.) Dialog mit drei Buttons:

```
   case JA_NEIN_VIELLEICHT_DIALOG_ID:
```

(Konstante wurde oben definiert)

```
    dialogBuilder = new AlertDialog.Builder(this);
```

(`this` referenziert die Activity)

```
dialogBuilder.setMessage("Entscheide Dich!");                    Textausgabe im Dialog
```

Definition der Buttons mit ihren Beschriftungen und Listenern. Es können maximal drei Buttons vereinbart werden (Positive/Negative/Neutral):

```
dialogBuilder.setPositiveButton("Ja", new DialogInterface.OnClickListener() {
 public void onClick(DialogInterface dialog, int id) {
  Toast.makeText(getApplicationContext()," Ja  :-)  ",Toast.LENGTH_LONG).show();
 }
});

dialogBuilder.setNegativeButton("Nein",... analog ...);

dialogBuilder.setNeutralButton("Vielleicht",... analog ...);

dialog = dialogBuilder.create();                                 Erzeugt das Dialog-Objekt
return dialog;                                                   Gibt das Dialog-Objekt zurück
```

2.) Dialog mit Auswahlliste:

```
case AUSWAHL_DIALOG_ID:                                          (Konstante wurde oben definiert)

dialogBuilder = new AlertDialog.Builder(this);                   (this referenziert die Activity)

dialogBuilder.setTitle("Wohin geht die Reise?");                 Überschrift der Auswahlliste

final CharSequence[] items = {"Holland", "Spanien", "Kanada"};   Elemente der Auswahlliste
```

Zuordnung der Elemente der Auswahlliste sowie eines Listeners, der bei Auswahl eines Elements aktiv wird (Hier durch `setItems()`; alternativ: `setSingleChoiceItems()` für Radio Buttons, `setMultiChoiceItems()` für Checkboxen):

```
dialogBuilder.setItems(items, new DialogInterface.OnClickListener() {
 public void onClick(DialogInterface dialog, int item) {
  Toast.makeText(getApplicationContext(),                        Toast zeigt ausgewähltes Item an
                 items[item],Toast.LENGTH_LONG).show();
 }
});

dialog = dialogBuilder.create();                                 Erzeugt das Dialog-Objekt
return dialog;                                                   Gibt das Dialog-Objekt zurück

default: return null;
 }
} }
```

Android: Progress Dialog *(Projekt NotifDialogsAndroid)*

zu Abschnitt 4.4

Die Applikation zeigt eine Progress Bar, also eine Ablaufanzeige.

java/.../ProgressDemo.java:

```
package de.thkoeln.cvogt.android.notifdialogs;

import android.app.*;
import android.os.Bundle;

public class ProgressDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);

  ProgressDialog dialog =
      ProgressDialog.show(this, "", "Bitte warten ...", true);

  LanglaufenderThread thread = new LanglaufenderThread(dialog);
  thread.start();

 }

}
```

Zeigt den Progress Dialog an

Startet einen langlaufenden Threads,
der den Dialog später wieder löscht (siehe unten)

Thread, der für eine gewisse Zeit läuft und dann den ProgressDialog wieder löscht:

```
class LanglaufenderThread extends Thread {

 private ProgressDialog dialog;

 LanglaufenderThread(ProgressDialog dialog) {
  this.dialog = dialog;
 }

 public void run() {
  try { sleep(3000); } catch (Exception e) {}
  dialog.dismiss();
 }

}
```

Tut drei Sekunden lang etwas
Löscht dann den Dialog

Android: Date and Time Picker Dialog *(Projekt NotifDialogsAndroid)*

zu Abschnitt 4.4

Die Applikation demonstriert Date und Time Picker zur Eingabe von Datums- und Uhrzeitangaben.

java/.../DateTimePickerDemo.java:

```
package de.thkoeln.cvogt.android.notifdialogs;

import android.app.*;
import android.os.Bundle;
import android.widget.*;

public class DateTimePickerDemo extends Activity {

 static final int DATEPICKER_DIALOG_ID = 3;
 static final int TIMEPICKER_DIALOG_ID = 4;
```

Codes zur Übergabe an `showDialog()` die verschiedene Dialoge identifizieren (siehe unten)

```
 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);

  showDialog(DATEPICKER_DIALOG_ID,null);
  showDialog(TIMEPICKER_DIALOG_ID,null);
```

Zeigt die Dialoge an (Details zu ihrem Aussehen siehe `onCreateDialog()` im Anschluss)

```
 }
```

Definition der anzuzeigenden Dialoge (siehe hierzu vorheriges Beispiel AlertDemo.java):

```
 protected Dialog onCreateDialog(int id, Bundle args) {

  switch(id) {

   case DATEPICKER_DIALOG_ID:
```

`DatePickerDialog` mit Voreinstellung 1.2.2019 (0 = Januar, 1 = Februar, ...) und Listener, der bei Änderung der Einstellung aktiv wird (siehe unten). `this` referenziert dabei die Activity.

```
    return new DatePickerDialog(this,new MyDatePickerListener(),2019,1,1);

   case TIMEPICKER_DIALOG_ID:
```

`TimePickerDialog` mit Voreinstellung 12:00h und Listener, der bei Änderung der Einstellung aktiv wird (siehe unten). `this` referenziert dabei die Activity.

```
    return new TimePickerDialog(this,new MyTimePickerListener(),12,0,true);

   default: return null;
```

```
  }
 }
```

Listener, der bei Änderung der Einstellung des DatePickerDialogs aktiv wird: Anzeige des eingegebenen Datums (da die interne Monatszählung bei 0 beginnt, daher `monthOfYear+1`).

```
 class MyDatePickerListener implements DatePickerDialog.OnDateSetListener {
  public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
   Toast.makeText(DateTimePickerDemo.this,
          "Neues Datum: "+dayOfMonth+"."+(monthOfYear+1)+"."+year, Toast.LENGTH_LONG).show();
  }
 }
```

Listener, der bei Änderung der Einstellung des TimePickerDialogs aktiv wird: Anzeige der eingegebenen Uhrzeit

```
 class MyTimePickerListener implements TimePickerDialog.OnTimeSetListener {
  public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
   String std = hourOfDay+"";
   if (std.length()==1) std = "0"+std;                    Bei einstelligen Werten: führende 0 hinzufügen
   String min = minute+"";
   if (min.length()==1) min = "0"+min;
   Toast.makeText(DateTimePickerDemo.this, "Neue Zeit: "+std+":"+min+"h", Toast.LENGTH_LONG).show();
  }
 }
}
```

Android: PopupWindow *(Projekt NotifDialogsAndroid)*

zu Abschnitt 4.4

Die Applikation demonstriert die Texteingabe über ein PopupWindow.

java/.../PopupWindowDemo.java:

```
package de.thkoeln.cvogt.android.notifdialogs;

import android.app.*;
import android.os.Bundle;
import android.view.*;
import android.widget.*;

public class PopupWindowDemo extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.popupdemoactivity);

  Button popupbutton = (Button) findViewById(R.id.popupbutton);      Button zur Anzeige des PopupWindows
  popupbutton.setOnClickListener(new MyOnClickListener());           Setzt den Listener dieses Buttons (siehe unten)

 }
```

Listener für den Button in der Activity-Oberfläche: Lässt das PopupWindow erscheinen.

```
 class MyOnClickListener implements View.OnClickListener {

  public void onClick(View v) {

   final MyPopupWindow pw = new MyPopupWindow(PopupWindowDemo.this);Erzeugt das PopupWindow (siehe Klassendef. unten)

   pw.setOnDismissListener(new PopupWindow.OnDismissListener() {    Ordnet ihm einen OnDismissListener zu,
    public void onDismiss() {                                         der bei Schließen des PopupWindows aktiv wird
     String eingegebenerName=pw.eingabefeld.getText().toString();    Bei Schließen des PopupWindows: Toast-Anzeige des
     Toast.makeText(PopupWindowDemo.this,                            Texts, der im PopupWindow eingegeben wurde (s. u.)
               "Hallo "+eingegebenerName,Toast.LENGTH_LONG).show();
    }
   });

   pw.showAtLocation(pw.layout,Gravity.CENTER, 0, 0);                Zeigt das PopupWindow an
   pw.update(0,0,300,250);   }  } }
```

Definition des PopupWindows: Es enthält insbesondere ein `EditText`-Feld, über das eine Texteingabe erfolgen kann.

```
class MyPopupWindow extends PopupWindow {
 EditText eingabefeld;                                          Textfeld zur Eingabe eines Strings
 Button okButton;                                               Bestätigungsbutton
 LinearLayout layout;                                           Layout des PopupWindows
 MyPopupWindow(Activity aktuelleActivity) {                     Konstruktor
  super(aktuelleActivity);
  LayoutInflater inflater = (LayoutInflater) aktuelleActivity.  Beschafft das Layout aus der zugeordneten XML-Datei
             getSystemService(Context.LAYOUT_INFLATER_SERVICE);
  layout = (LinearLayout) inflater.inflate(R.layout.popupwindow,null,false);
  eingabefeld = (EditText) layout.findViewById(R.id.eingabefeld); Beschafft Referenzen auf das Eingabefeld ...
  okButton = (Button) layout.findViewById(R.id.okButton);       ... und den Bestätigungsbutton
  okButton.setOnClickListener(new ButtonListener());            Registriert einen Buttonlistener (siehe unten)
  setContentView(layout);                                       Zeigt das Layout im PopupWindow an
  setFocusable(true);                                           Das PopupWindow kann den Fokus erhalten
 }
```

Listener für den Button im PopupWindow: Schließt das Fenster durch `dismiss()`, worauf der OnDismissListener (siehe oben) aktiv wird und den eingegebenen Text durch einen Toast anzeigt.

```
 class ButtonListener implements View.OnClickListener {
  public void onClick(View v) {
   dismiss();                                                   Schließt das Popup Window
  }
 }
}
```

res/layout/popupwindow.xml: Definition des Layouts des PopupWindows.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical">
```

```
 <TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Wie heißt Du?" />
 <EditText
  android:id="@+id/eingabefeld"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" />
 <Button
  android:id="@+id/okButton"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="OK" />
</LinearLayout>
```

Android: Einfache Berührungen *(Projekt TouchGesturesAndroid)* zu Abschnitt 4.5

Das Programm demonstriert die Erkennung von einfachen Berührungen und die Reaktion darauf.

java/.../Singletouch.java:

Durch `SingleTouchExampleView` wird eine Unterklasse von `View` definiert. Ein View dieser Klasse ist ein einfaches Rechteck mit einer gegebenen Farbe, das das gesamte Display ausfüllt. Berührt der Benutzer diesen View auf dem Display, so wird dessen Methode `onTouchEvent()` ausgeführt. Im Beispiel hier wird die Bezeichnung des Views sowie die Position der Berührung durch einen Toast angezeigt.

```
class SingleTouchExampleView extends View {

 private int color;                                          Farbe des Rechtecks
```

Konstruktor:

```
 public SingleTouchExampleView(Context context, AttributeSet attrs, int defStyle, int color) {
  super(context, attrs, defStyle);
  this.color = color;
 }
```

`onDraw()` wird vom Laufzeitsystem aufgerufen, um den View auf dem Display zu zeichnen:

```
 public void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  canvas.drawColor(color);
 }
```

`onTouchEvent()` wird vom Laufzeitsystem aufgerufen, wenn es auf dem View einen Touch Event (ein Berührungsereignis) bemerkt. Die hier definierte Methode zeigt einen Toast mit der View-Bezeichnung und der Position des Ereignisses, wenn ein Pointer (Finger) das Display neu berührt oder verlässt:

```
 public boolean onTouchEvent(MotionEvent ev) {

  final int action = ev.getAction();                         Ermittlung der Art des Ereignisses

  switch (action & MotionEvent.ACTION_MASK) {
                                        ACTION_MASK blendet die ID des Pointers aus, der das Ereignis verursacht hat

   case MotionEvent.ACTION_DOWN: {                           ein Pointer berührt erstmals das Display
    String farbtext = ... Name der Farbe des Views ("roter", "grüner", "blauer") ...
    String anzeigetext = new String(farbtext+" View: Down an Position ("
                                                    + ev.getX() + "," + ev.getY() + ")");
    Toast.makeText(getContext(), anzeigetext, Toast.LENGTH_LONG).show();
    break; }
```

```
   case MotionEvent.ACTION_MOVE: {                                   ein Pointer, der das Display bereits berührt, wird bewegt
    break;                                                           keine Aktion
   }
   case MotionEvent.ACTION_UP: {                                     ein Pointer verlässt das Display
    String anzeigetext = new String("Up an Position (" + ev.getX() + "," + ev.getY() + ")");
    Toast.makeText(getContext(), anzeigetext, Toast.LENGTH_LONG).show();
    break;
   }
   case MotionEvent.ACTION_CANCEL: {                                 Geste wird abgebrochen
    break;                                                           keine Aktion
   }
  }
  return true;
 }
}
```

`Activity`: Zeigt drei Views der oben definierten Klasse auf dem Display untereinander an (rot, grün und blau):

```
public class SingleTouch extends Activity {
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  LinearLayout ll = new LinearLayout(this);
  ll.setOrientation(LinearLayout.VERTICAL);
  SingleTouchExampleView view = new SingleTouchExampleView(this,null,0,0xffff0000);
  ll.addView(view);
  view = new SingleTouchExampleView(this,null,0,0xff00ff00);
  ll.addView(view);
  view = new SingleTouchExampleView(this,null,0,0xff0000ff);
  ll.addView(view);
  setContentView(ll);
 }
}
```

Android: Berührungen mit mehreren Pointern *(Projekt TouchGesturesAndroid)*

zu Abschnitt 4.5

Das Programmfragment demonstriert die Verarbeitung von Berührungen mit mehreren Pointern.

java/.../Multitouch.java:

Die Klasse `MultitouchExampleView` definiert Views, die jeweils ein Polygon anzeigen. Das Polygon verbindet die Pointer (Finger), die sich aktuell auf dem Display befinden, und ändert seine Form mit den Bewegungen dieser Pointer.

```
class MultitouchExampleView extends View {
 private float xPos[] = null, yPos[] = null;          X- und Y-Koordinaten der Pointer auf dem Display
 public void onDraw(Canvas canvas) {                  onDraw() zeichnet das Polygon
  super.onDraw(canvas);
  canvas.drawColor(0XFFFFFFFF);                       Weiß als Hintergrundfarbe des Canvas
  if (xPos==null || yPos==null) return;
  Paint paint = new Paint();
  paint.setColor(0xFF000000);                         Schwarz als Füllfarbe des Polygons
  if (xPos.length>1) {
   Path path = new Path();                            Geschlossenen Pfad definieren,
   path.moveTo(xPos[0],yPos[0]);                      der alle Pointer miteinander verbindet
   for (int i=1;i<xPos.length;i++)
    path.lineTo(xPos[i],yPos[i]);
   path.lineTo(xPos[0],yPos[0]);
   paint.setStyle(Style.FILL);
   canvas.drawPath(path, paint);                      Polygon gemäß Pfad zeichnen
  }
 }
```

`onTouchEvent()` bewirkt, dass das Polygon jedesmal neu gezeichnet wird, wenn ein Pointer auf dem Display bewegt wird, neu hinzukommt oder das Display verlässt.

```
  public boolean onTouchEvent(MotionEvent ev) {
    final int pointerCount = ev.getPointerCount();   aktuelle Anzahl der Pointer
    xPos = new float[pointerCount];                  X- und Y-Koordinaten der Pointer (siehe oben)
    yPos = new float[pointerCount];
```

```
    for (int i=0; i<pointerCount; i++) {
       xPos[i] = ev.getX(i); yPos[i] = ev.getY(i); }
    invalidate();                                          onDraw() ausführen lassen
    return true;
  }
}
```

Android: Einfache Gesten *(Projekt TouchGesturesAndroid)* zu Abschnitt 4.5

Die Applikation demonstriert die Erkennung von einfachen Gesten: Long Press, Double Tap, Wischen. Es stützt sich auf das Beispiel, das früher unter http://android-developers.blogspot.com/2010/06/making-sense-of-multitouch.html verfügbar war.

java/.../GesturesOhneScale.java:

```
package de.thkoeln.cvogt.android.touchgestures;

import android.app.*;
import android.content.*;
import android.os.*;
import android.util.*;
import android.view.*;
import android.graphics.*;
import android.graphics.drawable.*;
```

Die Klasse `GesturesExampleView` definiert einen View, der ein Icon auf das Display zeichnet. Dieses Icon kann man dann durch Wischen über das Display ziehen, und man kann die Anzeige durch weitere Gesten verändern.

```
class GesturesExampleView extends View {
```

```
 private Bitmap mBitmap;
```
Bitmap für das Icon, das über das Display gezogen wird

```
 private float mPosX, mPosY;
```
Aktuelle X- und Y-Position des Icons (für `onDraw()`)

```
 private static final int INVALID_POINTER_ID = -1;
 private int mActivePointerId = INVALID_POINTER_ID;
```
Id des Pointers, der momentan zum Ziehen des Icons benutzt wird (= „aktiver Pointer“)

```
 private float mLastTouchX;
 private float mLastTouchY;
```
Letzte X-Position des aktiven Pointers
Letzte Y-Position des aktiven Pointers

```
 private Context context;
```
Die aktuelle Activity

```
 private int frameType = 0;
```
0 = keinen Rahmen um das Icon zeichnen,
1/2 = Rahmen einfacher bzw. doppelter Größe

```
 private int bgColor = 0xFF000000;
```
Hintergrundfarbe

```
 private String text = "";
```
Auszugebender Zusatztext

```
 private GestureDetector exampleGestureDetector;
```
Zur Erkennung von „Single Touch“-Gesten (siehe unten)

Konstruktor:

```
public GesturesExampleView(Context context, AttributeSet attrs, int defStyle, float xPos, float yPos) {
 super(context, attrs, defStyle);
 this.context = context;
 mBitmap =                                                          Initialisiert die Bitmap für das dazustellende Icon
   BitmapFactory.decodeResource(context.getResources(), R.drawable.icon);
 mPosX = xPos;                                                      Initialisiert die X-Position gemäß Parameter
 mPosY = yPos;                                                      Initialisiert die Y-Position gemäß Parameter
```

Der folgende `GestureDetector` erkennt automatisch komplexere „Single Touch"-Gesten, also Gesten mit einem Pointer (z.B. Finger). Er wird gemeinsam mit einem `SimpleOnGestureListener` erzeugt (siehe Konstruktorparameter und Definition der Klasse `ExampleGestureListener` unten). Wenn ein Detector eine Geste erkannt hat, aktiviert er den Listener. Die `onTouchEvent()`-Methode des GestureDetectors muss explizit in `onTouchEvent()` des Views aufgerufen werden; der zugehörige Listener wird dagegen automatisch aktiv. Die Klasse `GestureDetector` ist vorgegeben; die Klasse des Listeners muss dagegen selbst programmiert werden.

```
 exampleGestureDetector =
     new GestureDetector(getContext(),new ExampleGestureListener());
 exampleGestureDetector.setIsLongpressEnabled(true);                Es soll auch auf lange Druckereignisse reagiert werden
 exampleGestureDetector.setOnDoubleTapListener(
                       new ExampleGestureListener());  }            Registriert einen Listener, der auf Double Taps reagiert
}
```

`onDraw()` zeichnet das Icon an der Position (`mPosX`,`mPosY`). Diese Position wird durch den „Pointer", der das Display berührt, bestimmt (siehe Definition von `onEvent()` unten):

```
public void onDraw(Canvas canvas) {
 super.onDraw(canvas);
```

Zeichnet die Hintergrundfarbe (je nach Wert der Variablen `bgColor`):

```
 canvas.drawColor(bgColor);
```

Zeichnet einen rechteckigen grünen Rahmen um das Icon (je nach Wert der Variablen `frameType`):

```
 Paint p = new Paint();
 p.setColor(0xFF00FF00);
 if (frameType==1)
   canvas.drawRect(mPosX-5, mPosY-5, mPosX+bitmap.getWidth()+4, mPosY+bitmap.getHeight()+4, p);
```

```
 if (frameType==2)
    canvas.drawRect(mPosX-10, mPosY-10, mPosX+bitmap.getWidth()+9, mPosY+bitmap.getHeight()+9, p);
 }
```

Gibt einen Text aus (sofern vorhanden):

```
 if (text.length()>0) {
  p.setTextSize(18);
  canvas.drawText(text,30,30,p);
 }
```

Zeichnet das Icon:

```
 canvas.drawBitmap(bitmap,mPosX,mPosY,null);
}
```

Ende von `onDraw()`

`onTouchEvent()` des Views wird vom Laufzeitsystem automatisch aufgerufen, wenn auf dem View ein Berührungsereignis bemerkt wird. Insbesondere wird hier die Icon-Position (`mPosX`,`mPosY`) anhand des „Pointers“, der das Display berührt, neu gesetzt:

```
public boolean onTouchEvent(MotionEvent ev) {
```

Ruft `onTouchEvent()` des `GestureDetectors` explizit auf, um ihn über das eingetretene Ereignis zu informieren:

```
 exampleGestureDetector.onTouchEvent(ev);
```

Ab hier: Reaktionen auf einfache Berührungsereignisse

```
 final int action = ev.getAction();
```

`ev.getAction()` liefert die Art des Ereignisses

```
 switch (action & MotionEvent.ACTION_MASK) {
```

Reaktionen je nach Art des Ereignisses. `ACTION_MASK` blendet die ID des Pointers aus, der das Ereignis verursacht hat; übrig bleibt das Ereignis selbst:

```
 case MotionEvent.ACTION_DOWN:
  mLastTouchX = ev.getX();
  mLastTouchY = ev.getY();
  mActivePointerId = ev.getPointerId(0);
  break;
```

Ein Pointer berührt als erster das Display
Speichert die Position dieses Pointers

Speichert die ID dieses Pointers
als die ID des "aktiven" Pointers

```
 case MotionEvent.ACTION_MOVE:
  final int pointerIndex =
         ev.findPointerIndex(mActivePointerId);
  final float x = ev.getX(pointerIndex);
  final float y = ev.getY(pointerIndex);
```

Ein Pointer, der das Display bereits berührt, wird bewegt
Findet den Index des aktiven Pointers ...
... und seine Position

```
   final float dx = x - mLastTouchX;
   final float dy = y - mLastTouchY;
```
Ermittelt den Bewegungsvektor von der vorherigen zur aktuellen Pointerposition

```
   mPosX += dx;        mPosY += dy;
   mLastTouchX = x;    mLastTouchY = y;
```
Aktualisiert die Position des Icons
Speichert die neue Position des aktiven Pointers

```
   invalidate();
```
Veranlasst ein Neuzeichnen des Canvas: `onDraw()`

```
   break;
```

```
  case MotionEvent.ACTION_UP:
   mActivePointerId = INVALID_POINTER_ID;
   break;
```
Der letzte Pointer verlässt das Display, so dass jetzt kein Pointer aktiv ist

```
  case MotionEvent.ACTION_CANCEL:
   mActivePointerId = INVALID_POINTER_ID;
   break; }
```
Die Aktion wurde abgebrochen

```
  case MotionEvent.ACTION_POINTER_UP:
```
Ein Pointer verlässt das Display

```
   final int pointerIndex =
      (action & MotionEvent.ACTION_POINTER_ID_MASK)
                  >> MotionEvent.ACTION_POINTER_ID_SHIFT;
   final int pointerId = ev.getPointerId(pointerIndex);
```
Ermittelt den Index ...
... und die Id dieses Pointers

Wenn der aktive Pointer das Display verlassen hat, wird der Pointer, der jetzt in der Nummerierung der erste ist, als aktiver Pointer vermerkt:

```
   if (pointerId == mActivePointerId) {
    final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
    mLastTouchX = ev.getX(newPointerIndex);
    mLastTouchY = ev.getY(newPointerIndex);
    mActivePointerId = ev.getPointerId(newPointerIndex);
   }
   break;
  }
  return true;
 }
```
Speichert die Position des neuen aktiven Pointers

Definition des Listeners, der dem `GestureDetector` zugeordnet ist (siehe Konstruktoraufruf für den Detector oben): Wenn der Detector eine komplexere Geste erkannt hat, ruft er automatisch eine entsprechende Methode des hier definierten `ExampleGestureListeners` auf.

```
 private class ExampleGestureListener extends GestureDetector.SimpleOnGestureListener {
```

`onLongPress()` wird bei einem langem Druckereignis ausgeführt:

```
  public void onLongPress(MotionEvent e) {
   text="";                                                    Löscht den anzuzeigenden Text
   if (frameType>0) frameType=0; else frameType=1;             Schaltet den Rahmen um das Icon ein bzw. aus
   bgColor = 0xFF000000;                                       Setzt den Hintergrund auf Schwarz zurück
   invalidate();  }                                            Lässt den Canvas neu zeichnen
```

`onDoubleTap()` wird bei einem „Double Tap"-Ereignis (zwei schnell aufeinanderfolgenden Berührungen) ausgeführt:

```
  public boolean onDoubleTap(MotionEvent e) {
   if (frameType==1) frameType=2;                              Vergrößert den Rahmen, wenn er klein war
   invalidate();                                               Lässt den Canvas neu zeichnen
   return false;  }
```

`onFling()` wird bei einem „Fling"-Ereignis („Wischen" = schnelles Ziehen des Pointers / Fingers) ausgeführt:

```
  public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
   text += " FLING";                                           Ändert den anzuzeigenden Text
   bgColor += 00x00220000;                                     Intensiviert den roten Hintergrund
   invalidate();                                               Lässt den Canvas neu zeichnen
   return false;  }

 }
}
```

Activity zur Anzeige des oben definierten Views:

```
public class GesturesOhneScale extends Activity {

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(new GesturesExampleView(this,null,0,10,10));
 }
}
```

Android: Skalierungsgesten *(Projekt TouchGesturesAndroid)* zu Abschnitt 4.5

Das Programmfragment demonstriert die Erkennung von Skalierungsgesten an der Benutzeroberfläche eines Android-Programms, also des Auseinanderziehens und Zusammenschiebens von zwei Pointern. Es wurde größtenteils übernommen von http://android-developers.blogspot.com/2010/06/making-sense-of-multitouch.html und ist eine Erweiterung des Beispiels "Android: Einfache Gesten".

java/.../ScaleGestures.java:

Die Klasse `ScaleGestureExampleView` definiert einen View, der ein Icon `mIcon` auf das Display zeichnet. Dieses Icon kann dann durch Auseinanderziehen von zwei Fingern vergrößert und durch Zusammenschieben der Finger verkleinert werden:

```
class ScaleGestureExampleView extends View {
...
 private ScaleGestureDetector mScaleDetector;        Erkennt Skalierungsgesten (siehe unten)
 private float mScaleFactor = 1.f;                   Aktueller Skalierungsfaktor für die Icon-Darstellung
...
   }
```

Konstruktor:

```
 public ScaleGestureExampleView(Context context, AttributeSet attrs, int defStyle) {
  ...
```

Der `ScaleGestureDetector` , der hier erzeugt wird, erkennt automatisch Skalierungsgesten mit zwei Pointern (insbesondere Finger). Wenn er eine Geste erkannt hat, aktiviert er automatisch den `ExampleScaleListener` (siehe ganz unten), der seinem Konstruktor als Parameter übergeben wurde. Die Klasse `ScaleGestureDetector` muss nicht selbst programmiert werden, sondern ist vorgegeben:

```
  mScaleDetector = new ScaleGestureDetector(context, new ExampleScaleListener());
 }
```

`onTouchEvent()` des Views wird automatisch aufgerufen, wenn auf dem View ein Berührungsereignis bemerkt wird.

```
 public boolean onTouchEvent(MotionEvent ev) {
```

Expliziter Aufruf von `onTouchEvent()` des `ScaleGestureDetectors`, um ihn über das eingetretene Ereignis zu informieren:

```
  mScaleDetector.onTouchEvent(ev);
  ... Rest wie im Beispiel "Android: Einfache Gesten" ...
 }
```

`onDraw()` zeichnet das Icon an der Position (`mPosX`,`mPosY`) und berücksichtigt dabei den Skalierungsfaktor `mScaleFactor`:

```
public void onDraw(Canvas canvas) {
 super.onDraw(canvas);
 canvas.save();
 canvas.translate(mPosX, mPosY);                    Positioniert das Icon
 canvas.scale(mScaleFactor, mScaleFactor);          Skaliert das Icon
 mIcon.draw(canvas);
 canvas.restore();
}
```

Wenn der `ScaleGestureDetector` eine Skalierungsgeste erkannt hat, ruft er automatisch die Methode `onScale()` des hier definierten `ExampleScaleListeners` auf. Der Listener wurde dazu bei der Erzeugung des `ScaleGestureDetector` dort registriert (siehe Konstruktor). Seine Klasse muss explizit ausprogrammiert werden:

```
private class ExampleScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener {
 public boolean onScale(ScaleGestureDetector detector) {
  mScaleFactor *= detector.getScaleFactor();        Ändert den Skalierungsfaktor entsprechend der Größe der Geste
  mScaleFactor =                                    Beschränkt die Größe des Skalierungsfaktors
   Math.max(0.1f, Math.min(mScaleFactor, 5.0f));
  invalidate();
  return true;
 }
}
}
```

Android: Canvas *(Projekt GrafAnimMMAndroid)* zu Abschnitt 4.6

Die Applikation demonstriert das Zeichnen auf einem Canvas.

java/.../GrafikMitCanvas.java:

```
package de.thkoeln.cvogt.android.grafanimmm;

import android.app.Activity;
import android.os.Bundle;
import android.content.*;
import android.view.*;
import android.widget.*;
import android.graphics.*;
```

Activity, die auf ihrer Oberfläche einen Canvas (also eine Zeichenfläche) anzeigt:

```
public class GrafikMitCanvas extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  ScrollView scv = new ScrollView(this);

  CanvasDemoView cv = new CanvasDemoView(this);
  cv.setMinimumHeight(800);
  scv.addView(cv);
  setContentView(scv);
 }
}
```

Erzeugt einen `ScrollView`, mit dem „gescrollt“ werden kann
Erzeugt den Canvas (Definition siehe unten)
Legt die Höhe des Canvas fest
Fügt den Canvas in den ScrollView ein
Zeigt ScrollView mit eingebettetem Canvas an

View-Klasse zur Anzeige des Canvas. In seiner `onDraw()`-Methode ist festgelegt, was zu zeichnen ist:

```
class CanvasDemoView extends View {

 Context context;
```

Konstruktor:

```
 public CanvasDemoView(Context context) {
  super(context);
  this.context = context;
 }
```

`onDraw()` legt fest, was zu zeichnen ist. Die Methode wird automatisch durch das Laufzeitsystem aufgerufen, wenn die Anzeige erneuert werden muss oder soll.

```
protected void onDraw(Canvas canvas) {
  Paint paintProperties = new Paint();
```

Wird im Folgenden für Stil- und Farbfestlegungen benutzt

Zeichnen einer bunten Figur aus geometrischen Mustern:

```
  paintProperties.setStyle(Paint.Style.FILL);
  paintProperties.setColor(0xff00ff00);
  canvas.drawRect(70,50,250,300,paintProperties);

  paintProperties.setColor(0xffff0000);
  RectF rahmen = new RectF(10,10,310,90);
  canvas.drawOval(rahmen,paintProperties);

  paintProperties.setColor(0xff0000ff);
  rahmen = new RectF(115,135,145,165);
  canvas.drawOval(rahmen,paintProperties);

  rahmen = new RectF(175,135,205,165);
  canvas.drawOval(rahmen,paintProperties);

  rahmen = new RectF(115,210,205,230);
  canvas.drawArc(rahmen,0,180,true,paintProperties);
```

Zeichnen von Linien in verschiedenen Stärken und Farben:

```
  paintProperties.setStrokeWidth(2);
  paintProperties.setColor(0xFF000000);
  canvas.drawLine(0,330,50,380,paintProperties);

  paintProperties.setStrokeWidth(4);
  paintProperties.setColor(0xFFFF0000);
  canvas.drawLine(50,330,100,380,paintProperties);

  paintProperties.setStrokeWidth(6);

  paintProperties.setColor(0xFF00FF00);
  canvas.drawLine(100,330,150,380,paintProperties);

  paintProperties.setStrokeWidth(8);
  paintProperties.setColor(0xFF0000FF);
```

```
canvas.drawLine(150,330,200,380,paintProperties);
```

Zeichnen von Rechtecken, Kreisen und Kreisbögen:

```
paintProperties.setStrokeWidth(2);
paintProperties.setColor(0xFF000000);
canvas.drawRect(0,400,90,450,paintProperties);

paintProperties.setStyle(Paint.Style.STROKE);
canvas.drawRect(100,400,190,450,paintProperties);

canvas.drawCircle(225,425,25,paintProperties);
canvas.drawArc(new RectF(210,350,310,450),0,90,false,paintProperties);
```

Zeichnen von Linienzügen und Punkten:

```
paintProperties.setStyle(Paint.Style.FILL);
float[] linienpunkte = { 0, 500, 70, 470, 70, 470, 140, 500, 140, 500, 210, 470 };
paintProperties.setColor(0xFF0000FF);
canvas.drawLines(linienpunkte, paintProperties);

float pathPunkteX[] = { 210, 280, 280, 210 };
float pathPunkteY[] = { 470, 500, 470, 500 };
Path path = new Path();
path.moveTo(pathPunkteX[0], pathPunkteY[0]);
for (int i=1; i<pathPunkteX.length; i++)
 path.lineTo(pathPunkteX[i],pathPunkteY[i]);
path.lineTo(pathPunkteX[0], pathPunkteY[0]);
canvas.drawPath(path, paintProperties);

float[] einzelpunkte = { 280, 470, 315, 485, 350, 500 };
paintProperties.setStrokeWidth(8);
canvas.drawPoints(einzelpunkte, paintProperties);
```

Ausgabe von Text:

```
paintProperties.setStrokeWidth(2);
paintProperties.setColor(0xFFFFFFFF);
paintProperties.setStyle(Paint.Style.FILL);
paintProperties.setTextSize(14);
paintProperties.setTextAlign(Paint.Align.LEFT);
canvas.drawText("Hallo",0,520,paintProperties);
```

```
  paintProperties.setTextAlign(Paint.Align.CENTER);
  canvas.drawText("Hallo",canvas.getWidth()/2,520,paintProperties);
  paintProperties.setTextAlign(Paint.Align.RIGHT);
  canvas.drawText("Hallo",canvas.getWidth(),520,paintProperties);
```

Zeichnen der Bitmap eines Icons:

```
  canvas.drawBitmap(BitmapFactory.decodeResource(context.getResources(),
                                                          R.drawable.ic_launcher),0,590,null);
 }
}
```

Android: Drawables *(Projekt GrafAnimMMAndroid)* zu Abschnitt 4.6

Die Applikation demonstriert das Zeichnen mit Hilfe von Drawables.

java/.../GrafikMitDrawable.java:

```
package de.thkoeln.cvogt.android.grafanimmm;

import android.app.Activity;
import android.os.Bundle;
import android.content.*;
import android.view.*;
import android.graphics.*;
import android.graphics.drawable.*;
import android.graphics.drawable.shapes.*;
```

Activity, die auf ihrer Oberfläche einen View mit Drawables anzeigt:

```
public class GrafikMitDrawable extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new MyDrawableView(this));
    }
}
```

View mit Drawables (Definition siehe unten)

View mit Drawables: Durch Aufruf seiner Methode `onDraw()` (geschieht automatisch durch das Laufzeitsystem) werden die Drawables angezeigt.

```
class MyDrawableView extends View {
 private ShapeDrawable rechteck, oval, kreis1, kreis2, bogen;
```

Anzuzeigende Drawables

Konstruktor des Views:

```
 public MyDrawableView(Context context) {
  super(context);
```

Ab hier: Erzeugung der anzuzeigenden Drawables; dabei Rückgriff auf vordefinierte Shapes.

```
  rechteck = new ShapeDrawable(new RectShape());
  rechteck.getPaint().setColor(0xff00ff00);
  rechteck.setBounds(70,50,250,300);
```

```
  oval = new ShapeDrawable(new OvalShape());
  oval.getPaint().setColor(0xffff0000);
  oval.setBounds(10,10,310,90);

  kreis1 = new ShapeDrawable(new OvalShape());
  kreis1.getPaint().setColor(0xff0000ff);
  kreis1.setBounds(115,135,145,165);

  kreis2 = new ShapeDrawable(new OvalShape());
  kreis2.getPaint().setColor(0xff0000ff);
  kreis2.setBounds(175,135,205,165);

  bogen = new ShapeDrawable(new ArcShape(0,180));
  bogen.getPaint().setColor(0xff0000ff);
  bogen.setBounds(115,210,205,230);
 }
```

`onDraw()` wird automatisch durch das Laufzeitsystem aufgerufen, wenn die Anzeige erneuert werden muss oder soll.

```
 protected void onDraw(Canvas canvas) {
  rechteck.draw(canvas);
  oval.draw(canvas);
  kreis1.draw(canvas);
  kreis2.draw(canvas);
  bogen.draw(canvas);
 }
}
```

Zeigt die Drawables
durch Aufruf ihrer `draw()`-Methoden an

Android: Frame Animation *(Projekt GrafAnimMMAndroid)*

zu Abschnitt 4.6

Die Applikation demonstriert eine Animation, bei der eine Folge von Bilddateien angezeigt wird.

java/.../FrameAnimation.java:

```
package de.thkoeln.cvogt.android.grafanimmm;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.widget.ImageView;

public class FrameAnimation extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
```

Anzeige eines ImageViews (siehe layout/frameanimationlayout.xml). Das Background-Attribut dieses ImageViews referenziert die Datei drawable/frameanimationsequence.xml. Dort ist eine "Animation List" definiert, die aus einer Folge von Bilddateien besteht.

```
  setContentView(R.layout.frameanimationlayout);
```

Start der Animation = Anzeige der Folge der PNG-Dateien aus der Animation List.

```
  ImageView smileyImage = (ImageView) findViewById(R.id.smileyanimationview);
  ((AnimationDrawable)(smileyImage.getBackground())).start();

 }

}
```

res/layout/frameanimationlayout.xml: Definiert das Layout, innerhalb dessen die Animation ablaufen soll.

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout 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"
 tools:context=".FrameAnimation">

 <ImageView                                                    zu animierender View
  android:id="@+id/smileyanimationview"
  android:layout_width="300dp"
```

```
 android:layout_height="300dp"
 android:layout_centerInParent="true"
 android:background="@drawable/frameanimationsequence" />
```

Verweis auf die Datei, die die Animation spezifiziert

```
</RelativeLayout>
```

res/drawable/frameanimationsequence.xml: Spezifiziert die nAnimation als Folge von Bilder, die angezeigt werden sollen.

```
<?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/smiley1" android:duration="300" />
 <item android:drawable="@drawable/smiley2" android:duration="300" />
 <item android:drawable="@drawable/smiley3" android:duration="300" />
 <item android:drawable="@drawable/smiley4" android:duration="300" />
 <item android:drawable="@drawable/smiley5" android:duration="300" />
 <item android:drawable="@drawable/smiley4" android:duration="300" />
 <item android:drawable="@drawable/smiley3" android:duration="300" />
 <item android:drawable="@drawable/smiley2" android:duration="300" />
</animation-list>
```

Android: View Animation *(Projekt GrafAnimMMAndroid)* zu Abschnitt 4.6

Die Applikation demonstriert eine Animation, bei der die Eigenschaften eines Views geändert werden.

java/.../ViewAnimation.java:

```
package de.thkoeln.cvogt.android.grafanimmm;

import android.app.Activity;
import android.os.Bundle;
import android.view.animation.*;
import android.widget.*;

public class ViewAnimation extends Activity {

 public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);

  ImageView icon = new ImageView(this);                    Generiert ein Icon aus einer PNG-Datei
  icon.setImageResource(R.drawable.smiley1);                                    in res/drawable-hdpi
  setContentView(icon);                                    Zeigt das Icon an.
```

Startet die Animation des Icons, die durch res/anim/viewanim (siehe unten) definiert wird:

```
  icon.startAnimation(AnimationUtils.loadAnimation(this,R.anim.viewanim));

 }

}
```

res/anim/viewanim.xml: Definiert die Animation

```
<?xml version="1.0" encoding="utf-8"?>

<set android:shareInterpolator="false"
  xmlns:android="http://schemas.android.com/apk/res/android">

 <rotate                                                   Rotation des Icons
   android:repeatCount="100"                               Angabe, wie oft die Animation wiederholt werden soll
   android:fromDegrees="0"                                 Grad-Bereich der Rotation
   android:toDegrees="-359"
   android:pivotX="50%"                                    Zentrum der Rotation
   android:pivotY="50%"                                          (hier: Mitte des Objekts)
   android:duration="4000" />                              Dauer der Animation (in Millisekunden)
```

```
  <scale                                                                    Skalierung des Icons
   android:interpolator="@android:anim/accelerate_decelerate_interpolator"
   android:repeatCount="100"                                                Angabe, wie oft die Animation wiederholt werden soll
   android:repeatMode="reverse"                                             Wiederholung jeweils in umgekehrter Abfolge
   android:fromXScale="0.5"                                                 Skalierungsfaktor zu Beginn ...
   android:toXScale="0.2"                                                   ... und zum Ende der Animation (X-Richtung)
   android:fromYScale="0.5"                                                 Skalierungsfaktor zu Beginn ...
   android:toYScale="0.2"                                                   ... und zum Ende der Animation (Y-Richtung)
   android:pivotX="50%"                                                     Zentrum der Skalierung
   android:pivotY="50%"                                                           (hier: Mitte des Objekts)
   android:duration="4000" />                                               Dauer der Animation (in Millisekunden)
</set>
```

Android: Animation auf SurfaceView *(Projekt GrafAnimMMAndroid)* zu Abschnitt 4.6

Die Applikation demonstriert eine Animation mit Hilfe eines Threads, der ein Icon über das Display bewegt und dafür wiederholt neu zeichnet. Grundlage ist ein Beispielprogramm aus http://android-developers.blogspot.com/2010/06/making-sense-of-multitouch.html.

java/.../SurfaceViewAnimation.java:

```
package de.thkoeln.cvogt.android.grafanimmm;

import android.annotation.SuppressLint;
import android.app.*;
import android.content.*;
import android.os.*;
import android.util.*;
import android.view.*;
import android.graphics.*;
```

View mit einer Animation, in der sich ein Icon über das Display bewegt:

```
@SuppressLint("WrongCall")
```
Damit der Aufruf `onDraw(c)` von Lint akzeptiert wird.

```
class AnimationExampleView extends SurfaceView implements SurfaceHolder.Callback {
 private Context context;
```
Activity, die dem View zugeordnet ist

```
 private Bitmap bitmap;
```
Zu bewegendes Icon

```
 private float xPos, yPos;
```
Aktuelle X- und Y-Position des Icons

```
 private float xDirect, yDirect;
```
Aktueller Bewegungsvektor des Icons

```
 private AnimationThread animThread = null;
```
Thread, der die Animation steuert

```
 private GestureDetector exampleGestureDetector;
```
GestureDetector, über den das Icon mit dem Finger neu „angestoßen“ werden kann

Konstruktor des Views

```
 public AnimationExampleView(Context context, AttributeSet attrs, int defStyle,
                                              float xPos, float yPos, float xDirect, float yDirect) {
  super(context, attrs, defStyle);
```

```
  this.context = context;
  this.xPos = xPos;
  this.yPos = yPos;
  this.xDirect = xDirect;
```
Initialisiert die Attribute

```
 this.yDirect = yDirect;
 bitmap = BitmapFactory.decodeResource(                          Erzeugt eine Bitmap
                  context.getResources(), R.drawable.ic_launcher); für das darzustellende Icon
 exampleGestureDetector = new GestureDetector(                   Erzeugt den GestureDetector
                      getContext(), new ExampleGestureListener());
```

Registrierung für Callback-Methoden des Holders, der diesem SurfaceView zugeordnet ist.
Bewirkt, dass die Methode `surfaceCreated()` (siehe unten) aufgerufen wird, sobald der SurfaceView bereit ist:

```
 getHolder().addCallback(this);
}
```

`onDraw()` zeichnet den SurfaceView neu und damit das Icon an seiner aktuellen Position:

```
public void onDraw(Canvas canvas) {
 super.onDraw(canvas);
 canvas.drawColor(0xFF000000);                                   Löscht das vorherige Icon: alle Bildpunkte schwärzen
 canvas.drawBitmap(bitmap, xPos, yPos, null);                    Zeichnet das Icon an seiner aktuellen Position
}
```

Die folgenden drei Methoden sind erforderlich, um das Interface `SurfaceHolder.Callback` zu implementieren. Wichtig ist davon hier `surfaceCreated()`: Die Methode wird automatisch aufgerufen, sobald der SurfaceView bereit für das Zeichnen ist, und startet dann den Animations-Thread.

```
public void surfaceCreated(SurfaceHolder holder) {
 if (animThread!=null) return;                                   Nichts tun, wenn der Animations-Thread schon existiert.
 animThread = new AnimationThread(holder);
 animThread.start();
}
public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) { }
public void surfaceDestroyed(SurfaceHolder holder) {            Animations-Thread anhalten, wenn der SurfaceView
 animThread.stop = true; }                                      nicht mehr angezeigt wird
```

Thread, der die Oberfläche wiederholt zeichnet („rendert“) und damit die Animation ausführt, also das Icon bewegt:

```
private class AnimationThread extends Thread {
 boolean stop = false;                                           Gibt an, ob der Thread weiterlaufen soll
 SurfaceHolder surfaceHolder;                                    Zum Zugriff auf den SurfaceView
```

Konstruktor des Animations-Threads:

```
public AnimationThread(SurfaceHolder surfaceHolder) {
 this.surfaceHolder = surfaceHolder;
}
```

Ausführungsschritte des Threads:

```
public void run() {
 while (!stop) {                                          Schleife zum Bewegen des Icons
  xPos += xDirect;                                        Verändert die x- und y-Position
  yPos += yDirect;                                         anhand des aktuellen Richtungsvektors
  if (xPos<0) { xDirect = -xDirect; xPos = 1; }           Wenn Rand des Displays erreicht:
  if (xPos>getWidth()-bitmap.getWidth()) {                 Ändert den Richtungsvektor
   xDirect = -xDirect;                                     nach dem Prinzip Einfallswinkel = Ausfallswinkel
   xPos = getWidth()-bitmap.getWidth()-1; }
  if (yPos<0) {  yDirect = -yDirect; yPos = 1; }
  if (yPos>getHeight()-bitmap.getHeight()) {
   yDirect = -yDirect;
   yPos = getHeight()-bitmap.getWidth()-1; }
  Canvas c = null;                                        Canvas, auf den gezeichnet werden soll
  try {
   c = surfaceHolder.lockCanvas(null);                    Belegt den Canvas
   synchronized (surfaceHolder) { onDraw(c); }            Zeichnet auf dem Canvas das Icon an der neuen Position
  } catch (Exception e) {}                                Insbes. Null Pointer Exception, wenn Canvas nicht ex.
  finally {
   if (c != null) surfaceHolder.unlockCanvasAndPost(c);   Gibt den Canvas wieder frei
  }
 }
}
```

`onTouchEvent()` wird automatisch aufgerufen, wenn auf dem View ein Berührungsereignis bemerkt wird: Der aktuelle Animations-Thread wird angehalten, und der GestureDetector wird benachrichtigt. Der GestureDetector startet dann ggf. die Animation mit einem neuen Richtungsvektor neu.

```
public boolean onTouchEvent(MotionEvent ev) {
 animThread.stop = true;
```

```
  exampleGestureDetector.onTouchEvent(ev);
  return true;
 }
```

Listener für den GestureDetector: Startet die Animation aufgrund der beobachteten Fling-Geste mit einem neuen Richtungsvektor neu. Der Benutzer kann somit das Icon in einer neuen Richtung "anstoßen".

```
 public class ExampleGestureListener extends GestureDetector.SimpleOnGestureListener {
  public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
   float xDirect = e2.getX()-e1.getX();
   float yDirect = e2.getY()-e1.getY();
   ((Activity)context).setContentView(new AnimationExampleView(
                                              context,null,0,xPos,yPos,xDirect,yDirect));
   return false;
  }
 }
}
```

Activity, die den View mit dem Icon anzeigt. Der Thread wird automatisch gestartet, wenn der View sichtbar ist (siehe oben: `surfaceCreated()`).

```
public class SurfaceViewAnimation extends Activity {
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(new AnimationExampleView(this,null,0,10,10,10,10));
 }  }
```

Android: Multimedia *(Projekt GrafAnimMMAndroid)* zu Abschnitt 4.6

Die Applikation demonstriert das Abspielen eines Audio- und eines Videoclips. (Ein Emulator kann das Video möglicherweise nicht abspielen.)

java/.../Multimedia.java:

```
package de.thkoeln.cvogt.android.grafanimmm;

import android.app.Activity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
import android.media.*;
import android.net.*;

public class Multimedia extends Activity {

 MediaPlayer audioPlayer;                                  Player zur Audiowiedergabe
 VideoView videoView;                                      View zur Wiedergabe eines Videos
 MediaController mediaController;                          Media Controller für das Video

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.multimedia); }                   Layout der Oberfläche: siehe unten

 public void onPause() {                                   Stoppt die Audio- und Video-Wiedergabe,
  super.onPause();                                                wenn die Activity pausiert.
  if (audioPlayer!=null) audioPlayer.stop();
  if (videoView!=null) videoView.stopPlayback(); }

 public void startAudio(View v) {                          Listener-Methode: Startet die Audio-Wiedergabe
  audioPlayer = MediaPlayer.create(this,R.raw.audio);       der Audio-Datei audio im Verzeichnis R.raw
  audioPlayer.start(); }

 public void stopAudio(View v) {                           Stoppt die Audio-Wiedergabe
  if (audioPlayer!=null)
   audioPlayer.stop(); }

 public void startVideo(View v) {                          Listener-Methode zur Video-Wiedergabe:
  videoView = (VideoView) findViewById(R.id.video);
  mediaController = new MediaController(this);             MediaController zur Steuerung der Wiedergabe
  mediaController.setAnchorView(videoView);                Soll das Video im VideoView anzeigen
```

```
  videoView.setMediaController(mediaController);
  EditText urlInput = (EditText) findViewById(R.id.urlInput);      EditText zur Eingabe der Video-URL
  String url = urlInput.getText().toString();                      Liest die Video-URL ein
  videoView.setVideoURI(Uri.parse(url));
  // videoView.setVideoURI(Uri.parse("android.resource://"         Alternativ: Abspielen einer Video-Datei
  //                         + getPackageName() +"/"+R.raw.video)); aus den Ressourcen
  videoView.start();                                               Startet die Wiedergabe
  videoView.requestFocus(); }                                      Verschiebt den Fokus zum VideoView

 public void showController(View v) {                             Listener-Methode: Zeigt den Video-Controller an
  if (mediaController!=null)
   mediaController.show(); }

 public void stopVideo(View v) {                                  Listener-Methode: Stoppt die Video-Wiedergabe
  if (videoView!=null)
   videoView.stopPlayback(); }

}
```

res/layout/multimedia.xml: Layout der Multimediapräsentation, u.a. mit einem `VideoView` zur Video-Wiedergabe und mehreren Steuerungs-Buttons

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

 <VideoView
  android:id="@+id/video"
  android:layout_width="320dp"
  android:layout_height="240dp" />

 <TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginTop="14dp"
  android:layout_marginBottom="10dp"
  android:text="Audio played from fixed res file" />

 <LinearLayout
  android:layout_width="match_parent"
```

```
 android:layout_height="wrap_content"
 android:orientation="horizontal" >

 <Button
  android:id="@+id/startAudio"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_weight="1"
  android:text="Start Audio"
  android:onClick="startAudio" />

 <Button
  android:id="@+id/stopAudio"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_weight="1"
  android:text="Stop Audio"
  android:onClick="stopAudio" />

</LinearLayout>

<TextView
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_marginTop="14dp"
 android:layout_marginBottom="10dp"
 android:text="Video played from URL:" />

<EditText
 android:id="@+id/urlInput"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:inputType="textUri"

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

```
  <Button
   android:id="@+id/startVideo"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_weight="1"
   android:text="Start Video"
   android:onClick="startVideo" />
  <Button
   android:id="@+id/showController"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_weight="1"
   android:text="Controller"
   android:onClick="showController" />
  <Button
   android:id="@+id/stopVideo"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_weight="1"
   android:text="Stop Video"
   android:onClick="stopVideo" />
 </LinearLayout>
</LinearLayout>
```

Android: Fragments *(Projekt FragmentsAndroid)* zu Abschnitt 4.8

Die Applikation demonstriert die Verwendung von Fragments auf Geräten mit unterschiedlichen Displaygrößen (Smartphones, Tablets).

Kern des Programms sind die beiden Fragments `Fragment1` und `Fragment2`. In `Fragment1` kann der Benutzer einen Text (hier: einen Namen) eingeben, in `Fragment2` wird der Name dann angezeigt. Je nach verfügbarem Platz werden entweder beide Fragments gleichzeitig nebeneinander (auf einem Tablet) oder abwechselnd einzeln bildschirmfüllend (auf einem Smartphone) angezeigt. Auf einem Tablet wird daher nur die eine Activity `MainActivity` genutzt, die beide Fragments anzeigt. Auf einem Smartphone sind zwei Activities nötig – `MainActivity` zur Anzeige von `Fragment1` und `SecondActivity` zur Anzeige von `Fragment2`.

java/.../MainActivity.java:

```
package de.thkoeln.cvogt.android.fragments;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {
```

Die `onCreate()`-Methode zeigt das Layout an, das in `activity_main.xml` definiert ist:
Auf einem Smartphone mit kleinem Display wird `res/layout/activity_main.xml` verwendet (automatische Auswahl durch das Laufzeitsystem). Die Datei definiert ein Layout mit nur einem Fragment. In diesem Fragment kann ein Name eingegeben und diese Eingabe durch Click auf einen Button beendet werden (siehe `Fragment1.java` und `fragment1.xml`).
Auf einem Tablet mit großem Display wird `res/layout-sw600dp/activity_main.xml` verwendet (automatische Auswahl durch das Laufzeitsystem). Die Datei definiert ein Layout mit zwei Fragments. Das zweite Fragment zeigt den Namen an, der in `Fragment1` eingegeben wurde (siehe zusätzlich `Fragment2.java` und `fragment2.xml`).

```
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
 }
```

`showInput()` wird aufgerufen, wenn in `Fragment1` der Button zum Abschluss der Eingabe geklickt wurde.

```
 public void showInput(View v) {
```

```
  EditText input = (EditText) findViewById(R.id.input);
  String name = input.getText().toString();
```

Beschafft den eingegebenen Namen

Beschafft eine Referenz auf das evtl. angezeigte `Fragment2` (`null`, wenn nicht angezeigt - das ist auf einem Smartphone der Fall)

```
  Fragment2 frag2 = (Fragment2) getFragmentManager().findFragmentById(R.id.fragment2);
  if (frag2 == null) {
```

Wenn nicht angezeigt: `SecondActivity` starten, die `Fragment2` mit dem eingegebenen Namen anzeigt. Dabei Übergabe des Namens im Intent.

```
   Intent intent = new Intent(this, SecondActivity.class);
   intent.putExtra("name", name);
   startActivity(intent);
  } else {
```

Wenn angezeigt (als Teil dieser `MainActivity`): Den eingegebenen Namen in `Fragment2` ausgeben.

```
   frag2.updateContent(name); }
 }
}
```

res/layout/activity_main.xml: Layout der `MainActivity` für Smartphones – zeigt nur ein Fragment (`Fragment1`, Definition weiter unten) an.

```
<LinearLayout xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical" >
 <fragment
  android:id="@+id/fragment1"
  android:name="de.thkoeln.cvogt.android.fragments.Fragment1"
  android:layout_weight="1"
  android:layout_width="match_parent"
  android:layout_height="0dp" />
</LinearLayout>
```

res/layout-sw600dp/activity_main.xml: Layout der `MainActivity` für Tablets – zeigt zwei Fragments (`Fragment1`, `Fragment2`) an.

```
<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="vertical" >
```

```
 <fragment
  android:id="@+id/fragment1"
  android:name="de.thkoeln.cvogt.android.fragments.Fragment1"
  android:layout_weight="1"
  android:layout_width="match_parment"
  android:layout_height="0dp" />
 <fragment
  android:id="@+id/fragment2"
  android:name="de.thkoeln.cvogt.android.fragments.Fragment2"
  android:layout_weight="1"
  android:layout_width="match_parment"
  android:layout_height="0dp" />
</LinearLayout>
```

java/.../Fragment1.java:

```
package de.thkoeln.cvogt.android.fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

`Fragment1` enthält einen `EditText`, in den der Benutzer einen Namen eingeben kann, und einen `Button`, mit dem er die Eingabe abschließen kann. Ein `Fragment1` wird entweder allein (auf einem Smartphone) oder zusammen mit einem `Fragment2` (auf einem Tablet) angezeigt (siehe oben).

Wichtig: Die `Fragment`-Klasse muss `public` sein. `onCreate()` muss `super.onCreate()` aufrufen; hierzu ist mindestens der API-Level 8 erforderlich. `onPause()` muss `super.onPause()` aufrufen; hierzu ist ebenfalls mindestens der API-Level 8 erforderlich.

```
public class Fragment1 extends Fragment {

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState); }

 public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
  return inflater.inflate(R.layout.fragment1, container, false); }
```

Macht das Layout auf dem Display sichtbar.

```
 public void onPause() {
  super.onPause(); }   }
```

res/layout/fragment1.xml und **res/layout-sw600dp/fragment1.xml**: Layout von `Fragment1` – auf allen Geräten gleich

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_height="match_parent"
 android:layout_width="match_parent"
 android:orientation="vertical"
 android:name="layoutFragment1"
 android:background="#ffff6347" >                          Hintergrundfarbe rötlich
 <TextView                                                 Anzeige einer Überschrift
  android:id="@+id/title1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Fragment 1" />
 <EditText                                                 Zur Eingabe des Namens
  android:id="@+id/input"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:hint="Your name here, pls" />
 <Button                                                   Zum Abschluss der Eingabe
  android:id="@+id/nextButton"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Next"
  android:onClick="showInput" />                           Veranlasst Anzeige des Namens in Fragment2
</LinearLayout>
```

java/.../Fragment2.java:

```
package de.thkoeln.cvogt.android.fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

```
import android.widget.TextView;
```

`Fragment2` zeigt den Namen an, der in `Fragment1` eingegeben wurde. `Fragment2` wird entweder allein (in `SecondActivity` auf einem Smartphone) oder zusammen mit `Fragment1` (in `MainActivity` auf einem Tablet) angezeigt.

```
public class Fragment2 extends Fragment {

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState); }

 public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
  return inflater.inflate(R.layout.fragment2, container, false); }
```

Macht das Layout auf dem Display sichtbar.

```
 public void onPause() {
  super.onPause(); }

 public void updateContent(String name) {
  TextView output = (TextView) getActivity().findViewById(R.id.output);
  output.setText("Hello "+name+"!");
 }
}
```

Zeigt den Namen an

res/layout/fragment2.xml und **res/layout-sw600dp/fragment2.xml**: Layout von `Fragment2` – auf allen Geräten gleich

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_height="match_parent"
 android:layout_width="match_parent"
 android:orientation="vertical"
 android:name="layoutFragment2"
 android:background="#ffcaff70" >
```

Hintergrundfarbe gelbgrünlich

```
 <TextView
  android:id="@+id/title2"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Fragment 2" />
```

Anzeige einer Überschrift

```
 <TextView
  android:id="@+id/output"
  android:layout_width="match_parent"
```

Zur Anzeige des Namens aus `Fragment1`

```
  android:layout_height="wrap_content"
  android:text="" />

</LinearLayout>
```

src/de/thkoeln/cvogt/android/fragments/SecondActivity.java:

```
package de.thkoeln.cvogt.android.fragments;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
```

`SecondActivity` wird nur auf einem Smartphone genutzt, da hier jedes Fragment in seiner eigenen Activity angezeigt werden muss und dabei das Display ganz ausfüllt. `SecondActivity` wird aus `MainActivity` gestartet, wenn über das dortige `Fragment1` ein Name eingegeben wurde, und zeigt dann in `Fragment2` diesen Namen an.

```
public class SecondActivity extends Activity {

 protected void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_second);                          Zeigt Layout mit Fragment2 an

  Fragment2 frag2 = (Fragment2)                                      Beschafft Referenz auf das angezeigte Fragment2
                getFragmentManager().findFragmentById(R.id.fragment2);

  Intent intent = getIntent();                                       Beschafft den Namen,
  String name = intent.getStringExtra("name");                       der von MainActivity übertragen wurde

  frag2.updateContent(name);                                         Zeigt den Namen in Fragment2 an

 }

}
```

res/layout/activity_second.xml: Layout von `SecondActivity` – zeigt `Fragment2` an

```
<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="vertical" >

 <fragment
```

```
  android:id="@+id/fragment2"
  android:name="de.thkoeln.cvogt.android.fragments.Fragment2"
  android:layout_weight="1"
  android:layout_width="match_parent"
  android:layout_height="0dp" />
</LinearLayout>
```

Android: Navigation *(Projekt AppNavigationAndroid)* zu Abschnitt 4.9

Die Applikation demonstriert, wie Navigationspfade durch die Oberflächen einer App graphisch spezifiziert werden und wie diese Navigation durch einen Navigation Controller unterstützt wird. Für Details wird auf die Quellcodes des AndroidStudio-Projekts verwiesen, die hier aus Platzgründen nicht abgedruckt sind.

Die Wurzel der Applikation ist eine Main Activity, die oben eine Action Bar, in der Mitte ein Fragment (das "NavHostFragment") und unten eine Bottom Bar anzeigt (siehe `layout/activity_main.xml`). Bei der Navigation durch die App wird an der Stelle des `NavHostFragments` jeweils eines der vier Fragments `layout/fragment_xxx.xml angezeigt` (siehe `navigation/nav_graph.xml`).

Die Applikation demonstriert drei Arten der Navigation:

1.) Navigation über Buttons: Die möglichen Navigationspfade werden durch Pfeile in `navigation/nav_graph.xml` (siehe Design-Ansicht dieser Datei) spezifiert. Jeder Pfeil entspricht einem `action`-Element, das das Ziel der Navigation (ein anderes Fragment) sowie die Animationen bei der Navigation festlegt (siehe Text-Ansicht von `nav_graph.xml`). Diese `action`-Elemente werden mit den jeweiligen Buttons verknüpft, indem man entsprechende Listener setzt (siehe z.B. `MainFragment.java`).

2.) Navigation über die Action Bar: Die Action Bar wird, wie üblich, anhand eines Menus (hier: `menu/menu.xml`) gefüllt (siehe `onCreateOptionsMenu()`). Die Aktionen bei Anklicken eines Items werden, wie ebenfalls üblich, durch `onOptionsItemSelected()` spezifiert. Bei der Auswahl eines Items wird die `NavigationController`-Methode `navigate()` mit dem jeweiligen Ziel-Fragment als Parameter aufgerufen. Zusätzlich wird noch ein Home-Item unterstützt, das zum Wurzel-Fragment zurückführt.

3.) Navigation über die Bottom Bar (`BottomNavigationView`): Die Vorgehensweise ist analog zu der Vorgehensweise bei der Action Bar. Die auszuführenden Aktionen werden hier allerdings durch die Methode `onNavigationItemSelected()` eines `OnNavigationItemSelectedListeners` festgelegt.