Landesinstitut für Schulentwicklung

Thema:	**App-Entwicklung mit Android Studio**
Name der Autorin/ des Autors:	Jochen Pogrzeba,StR Max-Weber-Schule, Freiburg
Fach:	Informatik, Wirtschaftsinformatik
Klasse/Jahrgangsstufe:	JG1
Schulart:	Berufliches Gymnasium
Lehrplanbezug:	LPE 9: Objektorientierte Systementwicklung (Fortführung) LPE 9: Objektorientierte Systemanalyse und -entwicklung
Zeitumfang:	-
Betriebssystem/e:	Android
Apps:	-
Technische Settings:	Schüler-PC's mit Windows und Android Studio Ggf. Schüler-Tablets mit Android

Fachliche Ziele:

- Kenntnisse der Besonderheiten der App-Entwicklung.
- Umsetzung des ereignisorientierten Ansatzes der GUI-Entwicklung.
- Vertiefung der Kenntnisse in einer objektorientierten Programmiersprache.
- Vertiefung der Kenntnisse im objektorientierten Fachklassenkonzept.
- Vertiefung der Kenntnisse in einer strukturierenden Auszeichnungssprache.

Sonstige Ziele:

- Erweiterung der Sozialkompetenz durch zusammenführende Gruppen- oder Projektarbeit.
- Schulung der gestalterischen Kreativität.
- Förderung der Medienkompetenz.
- Evtl. Schulung der Präsentationskompetenz.
- Evtl. individuelle Förderung.

Diese Unterrichtseinheit ist nicht als direkt umsetzbares Material konzipiert. Vielmehr soll dieses Beispiel der Lehrerin/dem Lehrer die Möglichkeit geben, sich ggf. schnell in das Thema einzuarbeiten. Die direkte Umsetzung des Beispiels mit Schülern ist möglich, aber nicht Hauptziel. Diese Unterrichtseinheit soll den Lehrer motivieren, ein eigenes, evtl. profilbezogenes Beispiel, mit den Schülern umzusetzen.
Aus diesem Grunde wurde darauf verzichtet einen Verlaufsplan hinzuzufügen.

Landesinstitut für Schulentwicklung

App-Entwicklung mit Android Studio
Vorstellung tablet-typischer Besonderheiten

Autor: Jochen Pogrzeba, StR
Max-Weber-Schule, Freiburg

Inhaltsverzeichnis:

jochen.pogrzeba@max-weber-schule.de

1 Vorbemerkungen

In dieser Unterrichtseinheit werden mehrere kleine Android-Projekte behandelt, die tablet-typische Funktionen begandeln.

Als Programmieroberfläche wurde für dieses Unterrichtsbeispiel *Android Studio* benutzt, da diese Software von Google als offizielle IDE zur App-Entwicklung für Android unterstützt wird. Eclipse mit seinen Plugins wird von Google nicht mehr unterstützt.
Android Studio basiert auf der IDE *IntelliJ IDEA*, welche sich ähnlich wie Eclipse bedienen lässt, der Einarbeitungsaufwand für Lehrer und Schüler ist daher sehr überschaubar.

Hauptaugenmerk dieser Beschreibung liegt in der Vorstellung der grundsätzlichen Prinzipien der Android-Entwicklung. Ziel ist es, der Lehrerin/dem Lehrer eine Anleitung mitzugeben, mit der er/sie sich leicht und schnell in die Thematik einarbeiten kann. Eine direkte Umsetzung des vorgestellten Projektes im Unterricht ist zwar möglich, aber nicht primäres Ziel. Der Leser ist also eingeladen, eine Unterrichtseinheit mit einer eigenen Softwareidee durchführen.

In dieser Handreichung wird kein Projekt von Beginn an beschrieben, es wird nur die Vorgehensweise für tablet-typische Funktionen beschrieben. Es wird daher empfohlen, dass zuvor schon ein kleines grundlegendes Android-Projekt (z.B. BMI-Rechner) umgesetzt wurde.
Die vorgeschlagenen Programmierlösungen sind nicht zwingend die besten oder einzigen Lösungen, sie wurden nach Verständlichkeit und Umsetzbarkeit ausgewählt.

Bei der Lektüre dieser Unterrichtseinheit wird folgendes vorausgesetzt:

- Ein installiertes und eingerichtetes Android Studio ist verfügbar (siehe *Handreichung - Installation und Einrichtung Android Studio*)
- Die Grundstruktur eines Android-Projektes ist bekannt (siehe *Handreichung - Installation und Einrichtung Android Studio*)
- Es wurde schon ein kleines grundlegendes Android-Projekt (z.B. BMI-Rechner) umgesetzt.
- Kenntnisse in Java und idealerweise Swing.

Sinnvoll wären Kenntnisse in:

- Umgang mit einer IDE wie z.B. Eclipse.
- XML

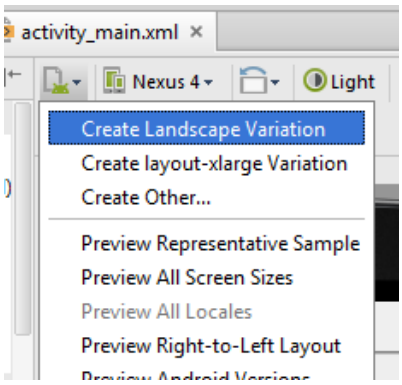
2 Kippen

Codebeispiel_Kippen.zip

Die Anpassung einer App auf die Ausrichtung des Gerätes über Kippen/Drehen ist denkbar einfach. Diese Technik benötigt keine gesonderte Java-Programmierung.

Man gestaltet wie gewohnt eine Oberfläche. Über Android Studio lässt sich dann eine „gekippte“ Variante erstellen:

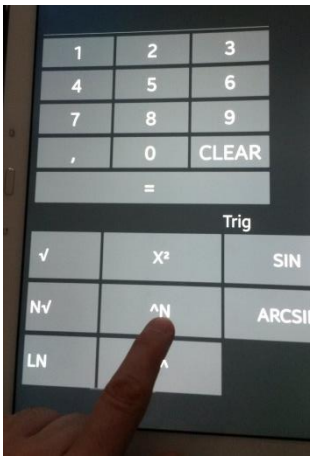
Im Designmodus lässt sich im Orientation-Menü  der Eintrag *Create Landscape Variation* auswählen. Damit wird automatisch eine *activity_main.xml (land)* erstellt, die später zur Laufzeit automatisch geladen wird, wenn das Tablet gedreht wird.



Das Design der Vertikalvariante wird übernommen, so dass man es nun zu einem Design umbauen kann, was besser zur Horizontalansicht passt. Daher empfiehlt es sich diesen Schritt am Ende durchzuführen, da es einfacher ist, die bestehenden GUI-Objekte anzupassen, als diese völlig neu zu erstellen.

3 Scrollen

Codebeispiel_Scrollen.zip

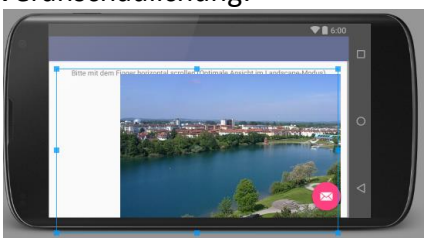


Ein wichtiges Feature einer App ist die Möglichkeit einen Bereich des Bildschirms mit dem Finger zu scrollen. Dies ist vergleichsweise einfach zu bewerkstelligen. Diese Technik benötigt keine gesonderte Java-Programmierung, sie wird in der *activity_main.xml* gelöst.

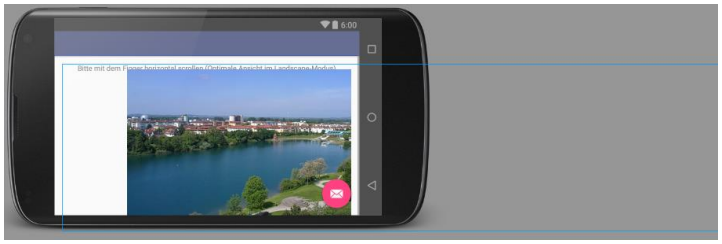
Hierzu empfiehlt es sich aber nicht mehr im Design-Modus, sondern direkt in der *activity_main.xml* zu arbeiten.

Der Activity wird ein *HorizontalScrollView*-Objekt hinzugefügt. In dieses Objekt wird wiederum ein neues *LinearLayout* eingefügt. Der Trick dabei ist, dass das *LinearLayout*-Objekt breiter sein muss als das *HorizontalScrollView*-Objekt. Damit kann man in dem Bereich des *HorizontalScrollView*-Objektes mit dem Finger scrollen. Innerhalb des LinearLayouts werden dann weitere GUI-Objekte wie Bilder eingefügt.

Veranschaulichung:



Der *HorizontalScrollView* wird in die GUI eingefügt. In diesem Bereich kann dann mit dem Finger gescrollt werden.



Das innere *LinearLayout* wird über den Paramater *layout_width* breiter eingestellt als die *HorizontalScrollView*.

Auf das innere *LinearLayout* werden nun die weiteren GUI-Objekte gesetzt. Diese liegen dann im scrollbaren Bereich.

```
<HorizontalScrollView
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ImageView
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:padding="10dp"
      android:id="@+id/imageView1"
      android:src="@drawable/seepark1"
    />

    ...weitere Bilder

  </LinearLayout>

</HorizontalScrollView>
```

Es ist dabei unbedingt darauf zu achten, dass die Baumstruktur der XML-Datei erhalten bleibt.
Kurze Schematische Darstellung des Baumes der *activity.main.xml* mit einem Scrollbereich:

```
<RelativeLayout ...>
      <EditText ... />
      <HorizontalScrollView ...>
            <LinearLayout ... >
                  <TextView ... />
            </LinearLayout>
      </HorizontalScrollView>
</RelativeLayout>
```

Das äußere *Layout* enthält alle GUI-Objekte.
zum Beispiel Textfelder oder Buttons
Der *HorizontalScrollView* wird eröffnet
Das innere *LinearLayout* wird eröffnet und enthält alle scrollenden GUI-Objekte.
Beispielsweise weitere Textfelder oder Buttons
Das innere *LinearLayout* wird geschlossen
Der *HorizontalScrollView* wird geschlossen

Um hier einen vertikalen Scrollbereich einzurichten, muss anstelle eines *HorizontalScrollView* ein *ScrollView* eingefügt werden.

4 Zoomen

Codebeispiel_Zoomen.zip

Um Objekte (z.B. Bilder) zoomen zu können, müssen diese (wie sonst auch) in einem *ImageView* eingefügt werden. Die Zoom-Funktion wird nun mittels Java umgesetzt:

```
public class MainActivity extends Activity
{
   Matrix matrixBeginn = new Matrix();
   Matrix matrixNeu = new Matrix();
   PointF mittelpunkt;
   float distanzBeginn = 1f;
   int mode = 0;
```

Ein Matrix-Objekt repräsentiert die Ausmaße eines Bildes. Hier benötigen wir zwei Matrizen: Eine vor, und eine nach dem Zoomen.

Ein PointF-Objekt repräsentiert eine Punktkoordinate aus zwei float-Werten. Diese speichert den Mittelpunkt des Bildes.

mode speichert, ob die Finger gerade das Bild berühren. (0-nein, 1-ja)

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

    bild.setOnTouchListener(new View.OnTouchListener() {

        private float getFingerSpreizWeite(MotionEvent event)
        {
            float x = event.getX(0) - event.getX(1);
            float y = event.getY(0) - event.getY(1);
            return (float) Math.sqrt(x * x + y * y);
        }

        private PointF getFingerSpreizMittelpunkt(MotionEvent event)
        {
            PointF point = new PointF();
            float x = event.getX(0) + event.getX(1);
            float y = event.getY(0) + event.getY(1);
            point.set(x / 2, y / 2);
            return point;
        }

        @Override
        public boolean onTouch(View view, MotionEvent event) {
            ImageView imageView = (ImageView) view;

            switch (event.getAction() & MotionEvent.ACTION_MASK)
            {
                case MotionEvent.ACTION_POINTER_DOWN:

                    distanzBeginn = getFingerSpreizWeite(event);
                    mittelpunkt = this.getFingerSpreizMittelpunkt(event);
                    matrixNeu.set(matrixBeginn);
                    mode = 1;
                    break;

                case MotionEvent.ACTION_POINTER_UP:
                    mode = 0;
```

Das Bild wird -wie üblich- registriert und ihm einen *OnTouchListener* hinzugefügt. Dieser horcht, ob das Bild berührt wird.

Diese Methode gibt uns die Länge der Diagonale zwischen den beiden Fingern zurück.

Diese Methode gibt uns den Mittelpunkt zwischen den beiden Fingern zurück (die Hälfte der Diagonale zwischen den beiden Fingern).

Nun werden diverse Fälle unterschieden, je nachdem ob ein Event ausgelöst wurde (getAction()) und welche Bewegung mit dem Bild ausgeführt wurde (MotionEvent).

Die Konstante ACTION_POINTER_DOWN bedeutet, dass mit zwei Finger auf das Bild getippt wird.

Die aktuellen Ausmaße des Bildes werden in der Matrix *matrixNeu* gespeichert. *mode* wird auf 1 (=ja) gesetzt.

Die Konstante ACTION_POINTER_UP bedeutet, dass ein oder zwei Finger vom Bild weggenommen werden.

mode wird auf 0 (=nein) gesetzt.

```
            break;

        case MotionEvent.ACTION_MOVE:

            if (mode == 1) {
                float distanzNeu = getFingerSpreizWeite(event);
                matrixBeginn.set(matrixNeu);
                float auflösung = distanzNeu / distanzBeginn;
                matrixBeginn.postScale(auflösung, auflösung, mittelpunkt.x, mittelpunkt.y);
            }

            break;
        }
        imageView.setImageMatrix(matrixBeginn);
        return true;
      }
    });
  }
}
```

Die Konstante ACTION_MOVE bedeutet, dass die Finger auf dem Bild verweilen.

Wenn *mode* 1 ist (=ja), dann wird die aktuelle Matrix *matrixNeu* der *matrixBeginn* zugewiesen.

Das gezoomte Bild ist die neue *matrixBeginn*. Die Auflösung ist das Streckenverhältnis der Fingerdistanzen vor und nach dem Zoomen.

Die neuen Bildausmaße werden dem ImageView zugewiesen, d.h. das Bild wurde gezoomt.

5 Objekte verschieben

Codebeispiel_Drag.zip

Dieses Beispiel ist sehr einfach gehalten. So ist es hier nur möglich, ein Objekt über die Oberfläche zu ziehen. Grund dafür ist, dass jedes zu bewegende Objekt auf einem individuellen Layer liegt, der in den Vordergrund geholt werden muss, um das entsprechende Objekt zu bewegen. Um die grundsätzliche Idee des Ziehens zu präsentieren, wurde hier davon abstrahiert.

Zunächst wird ein ImageView (oder ähnliches) auf dem Layout platziert. Die Verschiebe-Funktionalität wird in Java umgesetzt.
(zur Erklärung mancher Codeteile siehe Kapitel *Zoomen*)

```
public class MainActivity extends Activity
{
   ImageView bild;
   Matrix MatrixBeginn = new Matrix();
   Matrix MatrixNeu = new Matrix();
   PointF startpunkt = new PointF();
   int mode = 0;
```

Ein PointF-Objekt repräsentiert eine Punktkoordinate aus zwei float-Werten. Diese speichert den Startpunkt des Bildes.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    bild = (ImageView) findViewById(R.id.bild1);

    bild.setOnTouchListener(new View.OnTouchListener() {

        @Override
        public boolean onTouch(View view, MotionEvent event) {
            ImageView imageView = (ImageView) view;

            switch (event.getAction() & MotionEvent.ACTION_MASK) {
                case MotionEvent.ACTION_DOWN:
                    MatrixNeu.set(MatrixBeginn);
                    startpunkt.set(event.getX(), event.getY());
                    mode = 1;
                    break;

                case MotionEvent.ACTION_POINTER_UP:
                    mode = 0;
                    break;

                case MotionEvent.ACTION_MOVE
                    if (mode == 1)
                    {
                        MatrixBeginn.set(MatrixNeu);
                        MatrixBeginn.postTranslate(event.getX() - startpunkt.x, event.getY() - startpunkt.y);
                    }
                    break;
            }
            imageView.setImageMatrix(MatrixBeginn);

            return true;
        }
    });
  }
}
```

Die Konstante ACTION_POINTER_DOWN bedeutet, dass mit einem Finger auf dem Bild getippt wird.

Der Startpunkt wird auf die Mitte des Bildes gesetzt.

Die Ausmaße des Bildes werden neue gesetzt. postTranslate() setzt den neuen Punkt des Bildes.

6 Zugriff auf Internet-Ressourcen

Codebeispiel_Internet.zip

Dieses Beispiel zeigt, wie in einer App Ressourcen aus dem Internet bezogen werden können. Zum einen wird ein Text aus einer Textdatei gezeigt, die auf einem fremden Server liegt. Zum anderen wird ein beliebiges Bild einer externen Nachrichten-Seite gezeigt.

Diese Funktionalität wird im Java-Code umgesetzt.

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

    new TextAusURL().execute("http://www.jochen-pogrzeba.de/AndroidTest.txt");
    new PicAusURL().execute("http://cdn2.spiegel.de/images/image-922389-breitwandaufmacher-cwuo-922389.jpg");
}

private class TextAusURL extends AsyncTask<String, Void, String> {
    ProgressDialog dialog;
    String uri;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        // show progress dialog when downloading
        dialog = ProgressDialog.show(MainActivity.this, null, "Stelle Verbindung her");
    }

    @Override
    protected String doInBackground(String... params) {
        try {
            uri = params[0];
            URL url = new URL(uri);

            url.openStream();
            BufferedReader zeilenleser = new BufferedReader(new InputStreamReader(url.openStream()));

            StringBuilder result = new StringBuilder();
            String zeile;
            while ((zeile = zeilenleser.readLine()) != null) {
                result.append(zeile + "\n");
```

Zwei Objekte der u.g. Klassen werden erstellt, und deren Methode „.execute“ ausgeführt. Der Methoden werden zwei Internetressourcen (Text und Bild) übergegeben.

Diese Methode legt fest, was vor der Verbindungsaufnahme geschehen soll. Sie überschreibt die Methode der Superklasse.

Diese Methode öffnet die Textdatei als Internetressource und zeigt deren Inhalt.

Öffnet die Internetressource als Stream und schreibt den Inhalt in einen BufferedReader

Alle Zeilen des BufferedReader werden ausgelesen und in ein String „result“ geschrieben.

```
            }
            return result.toString();
        } catch (Exception e) {
            System.out.println("----------------------------------------------------Get Url Error in downloading: " + e.toString());
        }
        return null;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        TextView txtFeld1 = (TextView) findViewById(R.id.txtFeld1);

        if (result == null) {
            txtFeld1.setText("Error in downloading. Please try again.");
        } else {
            txtFeld1.setText("Nachricht von "+uri+": \n"+ result);
        }
        dialog.dismiss();
    }
}

private class PicAusURL extends AsyncTask<String, Void, String> {
    ProgressDialog dialog;
    String uri;
    Drawable image;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        dialog = ProgressDialog.show(MainActivity.this, null, "Stelle Verbindung her");
    }

    @Override
    protected String doInBackground(String... params) {

        try {
            uri = params[0];
            URL url = new URL(uri);
```

Diese Methode legt fest, was nach der Verbindungsaufnahme geschehen soll. Sie überschreibt die Methode der Superklasse.
Ihr wird das Resultat des Ressourcenzugriffs übergeben

Wenn das Resultat leer ist, ist wohl ein Fehler aufgetreten und es wird eine entsprechende Meldung gezeigt. Ansonsten wird das Resultat als Text gezeigt.

Diese Methode legt fest, was vor der Verbindungsaufnahme geschehen soll. Sie überschreibt die Methode der Superklasse.

```
        Object content;
        content = url.getContent();
        InputStream is = (InputStream) content;
        image = Drawable.createFromStream(is, "src");

        } catch (Exception e) {
            System.out.println("----------------------------------------------Get Url Error in downloading: " + e.toString());
    }
    return null;
  }

  @Override
  protected void onPostExecute(String result) {
    super.onPostExecute(result);

    ImageView bild1 = (ImageView) findViewById(R.id.bild1);
    bild1.setImageDrawable(image);

    dialog.dismiss();
  }
}
```

Die Internetressource (content) wird als InputStream gespeichert. Der wiederrum wird als drawable (=Bild) gespeichert.

Diese Methode legt fest, was nach der Verbindungsaufnahme geschehen soll. Sie überschreibt die Methode der Superklasse.
Ihr wird das Resultat des Ressourcenzugriffs (das Bild) übergeben

7 Zugriff auf eine Datenbank

Codebeispiel_Datenbank.zip

Dieses Beispiel zeigt, wie eine App Inhalte aus einer Datenbank zeihen kann.

Diese Funktionalität wird im Java-Code umgesetzt.

```
public class Database extends Activity
{
  SQLiteDatabase mydb;

  public void oeffnen()
  {
    mydb = openOrCreateDatabase("DBEinkaeufe", 0,null);
  }

  public void testDatenHinzufuegen()
  {
```

Die Datenbank wird geöffnet.

```
    mydb.execSQL("create table if not exists ekliste (item VARCHAR(45));");
    mydb.execSQL("insert into ekliste (items) values('Salami');");
    mydb.execSQL("insert into ekliste (items) values('Eier');");
    mydb.execSQL("insert into ekliste (items) values('Milch');");
    mydb.execSQL("insert into ekliste (items) values('Bananen');");
    mydb.execSQL("insert into ekliste (items) values('Erdnüsse');");
    mydb.execSQL("insert into ekliste (items) values('Küchenrolle');");
}

public Cursor lesen()
{
    Cursor ergebnis = mydb.rawQuery("select * from ekliste.item;",null);
    return ergebnis;
}

public void schliessen()
{
    mydb.close();
}
```

Mit der Methode *.execSQL* werden SQL-Befehle der DDL und DML an die DB übergeben.

Mit der Methode *.rawQuery* werden SQL-Befehle der DQL an die DB übergeben. Das Ergebnis ist ein *cursor*-Objekt (ähnlich Record- oder Resultset)