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
- Entwicklung einer Taschenrechner-App –
(Ein Beispiel für Fortgeschrittene)

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

Inhaltsverzeichnis:

jochen.pogrzeba@max-weber-schule.de

1 Vorbemerkungen

In dieser Unterrichtseinheit wird mit Hilfe der Entwicklungsumgebung *Android Studio* eine Taschenrechner-App entwickelt, die auf Android-Tablets lauffähig ist.

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.

Als Programmierbeispiel wurde ein Taschenrechner gewählt, da dieses Projekt schulartübergreifend einsetzbar ist und die prinzipielle Funktionsweise eines Taschenrechners den Schülern bekannt ist.
Er beherrscht typische Tablet-Funktionalität wie Kippen oder Scrollen. Dieses Projekt ist durchaus komplex, ein simpleres Projekt zum Einstieg wird in *UE BMI-App* beschrieben.

Hauptaugenmerk dieser Beschreibung liegt in der Vorstellung der grundsätzlichen Prinzipien der Android-Entwicklung. Hauptziel dieser Unterrichtseinheit ist es, der Lehrerin/dem Lehrer eine Anleitung mitzugeben, mit der er 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.
Ziel ist es auch nicht, die Entwicklung einer App zu beschreiben, da alles abdeckt, was eine App "so alles kann", die Umsetzungsmöglichkeit mit Schülern steht im Vordergrund.

Das Android Studio-Projekt finden Sie in den Archiven *Projekt Taschenrechner Tablet* oder *Projekt Taschenrechner Phone.* Beim ersten ist die Ausgabe für große Tablets optimiert, beim zweiten für kleinere Telefon-Displays.

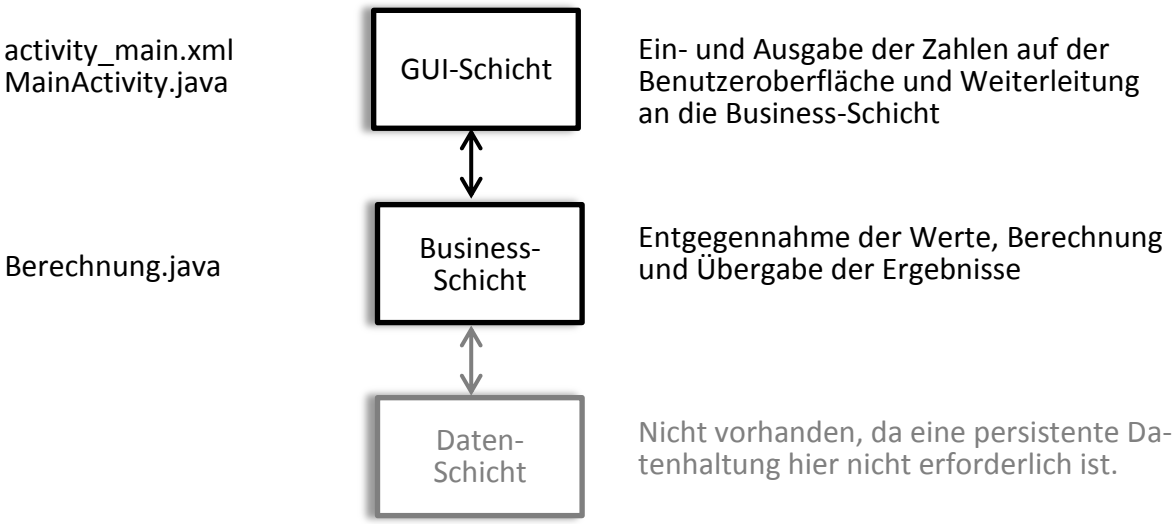
Bei der Lektüre dieser Unterrichtseinheit wird folgendes vorausgesetzt:

- Ein installiertes und eingerichtetes Android Studio ist verfügbar (siehe *Handreichung - Installation und Einrichtung Android Studio*)
- Das Android Studio-Projekt *Projekt Taschenrechner* liegt zum aktiven Mitarbeiten vor (siehe Archive *Projekt Taschenrechner Tablet* oder *Projekt Taschenrechner Phone*)
- Die Grundstruktur eines Android-Projektes ist bekannt (siehe *Handreichung - Installation und Einrichtung Android Studio*)
- Kenntnisse in Java und idealerweise Swing.

Sinnvoll wären Kenntnisse in:

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

2 Umsetzung des 2-Schichtenmodells



Die beiden zentralen Dateien der GUI-Schicht sind:

Activity_main.xml

In dieser XML-Datei werden die GUI-Elemente als XML-Tags eingetragen. Auch die Attribute und deren Werte finden sich hier wieder. Diese Datei ist somit der zentrale Ort, der das Layout der App festlegt. Diese Datei wird beim Start automatisch erzeugt. Die Einträge für die GUI-Elemente werden automatisch hinzugefügt, wenn man im Designmodus die GUI-Objekte auf die Activity zieht. Trotzdem kann es nötig sein, bei komplexeren Projekten auch direkt in dieser Datei zu arbeiten.

Im Projektbaum finden sich zwei *Activity_main.xml*-Dateien:
Mit Zusatz *(land)*: Hier wird das Design für die Horizontalansicht festgelegt
Ohne Zusatz: Hier wird das Design für die Vertikalansicht festgelegt.

Dies bedeutet, dass man für beide Bildschirmausrichtungen völlig unterschiedliche Designs festlegen kann. Unsere App kann also auf das Kippen des Tablets reagieren. Der Nachteil ist, dass man manche Aufgaben (z.B. Hinzufügen eines neuen Buttons) zweimal erledigen muss (siehe Kap. 3.5).

MainActivity.java

Diese Java-Klasse repräsentiert die MainActivity, die unsere App steuert. Sie ist somit die zentrale Datei, was die Steuerung der App über die GUI-Elemente angeht. Sie erbt Funktionalität von anderen Activity-Klassen, je nachdem was bei der Projekterstellung als voreingestellte Activity ausgewählt wurde.

Diese Klasse besitzt die Methode *onCreate()*, die beim Starten der App ausgeführt wird. In ihr wird zum einen ein Exemplar der Superklasse erzeugt, zum anderen die *Activity_main.xml* als Layout-Datei eingefügt. Diese wird im weiteren Verlauf durch die Klasse *R* repräsentiert (siehe Kap. 5).

Ähnlich einer GUI-Klasse bei der Swing-Programmierung werden hier auch die GUI-Objekte deklariert und deren Eventhandler hinzugefügt.

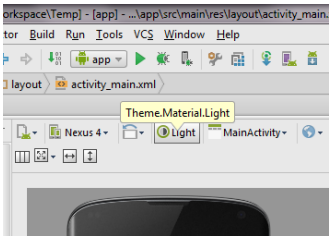
3 Erstellung der Benutzeroberfläche

3.1 Layout

Eine einfache Activity kann nahezu vollständig mittels Drag-and-Drop im Designmodus erstellt werden. Dabei wird dann im Hintergrund die *activity_main.xml* mit den entsprechenden Tags gefüllt.
Bei komplexen Projekten wie diesem wird man aber nicht darum herumkommen direkt in dieser Datei zu arbeiten, da bei GUIs mit vielen GUI-Objekten und Scroll-Bereichen die Handhabung im Design-Modus sehr unübersichtlich werden kann.

Bei einem neu erstellten Projekt sind bereits ein Layout und ein Textfeld mit dem Inhalt "Hello World" vorbereitet.

Hinweis: Sollte der Designmodus hier "Rendering Problems" anzeigen, so empfiehlt es sich das Grafikthema zu ändern. Ändern Sie in diesem Falle *ProjectTheme* auf ein anderes Thema, z.B. *MaterialLight*.



Bevor die Elemente auf der Activity platziert werden können, muss ein Layout platziert werden. Ein Layout kann man sich wie einen Teppichboden vorstellen, auf dem Möbelstücke nach bestimmten Mustern angeordnet werden. Das voreingestellte Layout ist das *RelativeLayout*, hier werden die GUI-Objekte relativ zu der Position der anderen Elemente platziert (ähnlich wie HTML ohne css).

Für dieses Projekt wurde ein *GridLayout* gewählt. Dieses zerlegt die Activity in ein Gitter (ähnlich einer Tabelle), in das dann in jeden Bereich ein GUI-Objekt gelegt werden kann.

In der *activity_main.xml* wird nun das GridLayout als XML-Tag angezeigt. Zwischen `<GridLayout… >` und `</GridLayout>` werden dann später die XML-Tags für die anderen GUI-Objekte stehen.

```
<GridLayout xmlns:android="http://schemas.android.co
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity"
    android:id="@+id/Taschenrechner"
    android:orientation="vertical"
    android:textAlignment="center"
    android:background="#00000000">
```

Im Designmodus kann man die Anzahl, sowie die aktuelle Höhe und Breite der Gitterzellen an den hellgrünen Hilfssymbolen oberhalb und links des Vorschaufensters erkennen.

Die Breite und Höhe der Gitterzellen werden dann nachfolgend durch die Breite und Höhe der in ihnen eingefügten GUI-Objekte bestimmt.

Ein GUI-Objekt lässt sich nun per Drag-und-Drop in das *GridLayout* platzieren.

3.2 Platzierung der GUI-Objekte

Das Ein- und Ausgabefeld (Lösungsfeld) des Taschenrechners wurde als *EditText*-Objekt realisiert. Um ein GUI-Objekt im Grid zu platzieren, sind folgende Einstellungen wichtig:

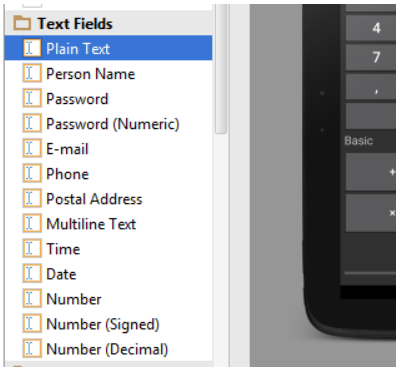
```
<EditText
    android:layout_width="660dp"
    android:layout_height="100dp"
    android:inputType="numberDecimal"
    android:ems="10"
    android:id="@+id/txtLsg"
    android:layout_row="0"
    android:layout_column="0"
    android:layout_columnSpan="3"
    android:textSize="30sp"
    android:gravity="end"
    android:editable="false"
    android:enabled="false"
    android:linksClickable="false"
    android:longClickable="false"
    android:clickable="false"
    android:textStyle="bold"
    android:textColor="@android:color/white"
    android:focusable="false" />
```

layout-width und *layout-height* legen die Ausmaße des Textfeldes in Bildpunkten fest. Dies bestimmt dann auch Breite und Höhe der Gitterzellen. Wird hier anstatt einer Zahlenangabe der Wert *wrap_content* eingetragen, ist das GUI-Objekt genauso hoch/breit wie der Text benötigt. Dieser Eintrag ist jedoch mit Vorsicht zu genießen, da hiermit schnell ein Layout verschoben werden kann.

layout_row und *layout_column* legen die Gitterzelle fest in der das GUI-Objekt abgelegt wird. Die Zählung beginnt von links oben mit dem Wert 0.

layout-columnSpan und *layout-rowSpan* werden benutzt um ein GUI-Element auf mehrere Zellen auszubreiten. In diesem Beispiel breitet sich das Lösungsfeld horizontal über drei Gitterzellen aus.

3.3 Weitere wichtige Parameter der GUI-Objekte



Für Textfelder sind mehrere *EditText*-Varianten auswählbar. Für das Lösungsfeld des Taschenrechners wurde *Number (Decimal)* gewählt.

Auch können nun wie oben beschrieben die Buttons platziert werden.

Im Properties-Bereich oder direkt in der *activity_main.xml* werden nun weitere Einstellungen für die GUI-Objekte getätigt.

- Wichtig ist vor allem der Name des Objektes. Vergeben Sie bei *id* (im Properties-Bereich) oder `android:id="@+id/txtNameDesObjektes"` (in der *activity_main.xml)* einen aussagekräftigen Namen.

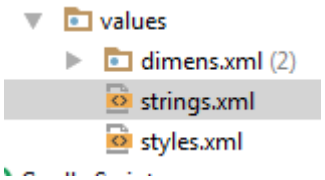
- Über `android:drawableLeft="@drawable/bildname"` im Button-Tag kann einem Button ein Icon hinzugefügt werden. Das ico- oder png-Bild muss sich dabei im Projektordner *drawable* befinden und ohne Endung angegeben werden. (*bildname* wird hier nur symbolisch verwendet und ist durch den tatsächlichen Dateinamen zu ersetzen).

- Bei dieser App wird die Eingabe über die hinzugefügten Buttons erledigt. Im Normalzustand wird das Android-Betriebssystem immer einen Eingabebereich zur Zahleneingabe auf den Bildschirm schieben. Dies ist hier unerwünscht und kann dadurch verhindert werden, dass man das Lösungsfeld nicht erreichbar (focusable, clickable, editable) macht.
  Dem Tag `<EditText` werden daher folgende Attribute hinzugefügt:

```
android:editable="false"
android:enabled="false"
android:linksClickable="false"
android:longClickable="false"
android:clickable="false"
android:focusable="false"
```

  Damit ist das Lösungsfeld für alle Veränderungen von außen abgeschottet und der Eingabebereich erscheint nicht mehr.

- Die Beschriftungen für die Buttons können bei Android Studio auf zwei Wegen hinzugefügt werden: Zum einen direkt in der *activity_main.xml* mit dem Parameter `android:text="e"` (Bsp.) oder, viel eleganter, als String-Ressource. Dabei wird der Beschriftungstext in der Datei *strings.xml* im Projektordner *values* eingetragen und dessen Bezug in der *activity_main.xml* verwendet.



  Konkret könnte dies so aussehen:

  Eintrag in *strings.xml*:

```
<resources>
    <string name="euler">e</string>
    …
</resources>
```

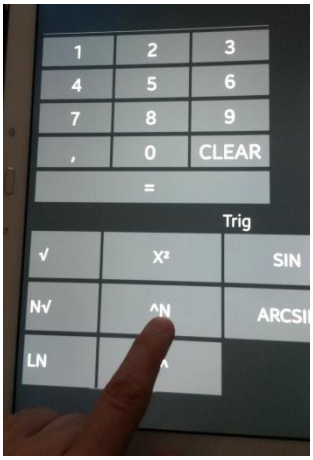
  Eintrag in *activity_main.xml*:

```
<Button
        android:layout_width="330dp"
        android:layout_height="140dp"
        android:text="@string/euler"
        android:id="@+id/btnEuler"
        android:layout_row="1"
        android:layout_column="9"
        android:textSize="40sp"/>
```

  Auf dem Button *btnEuler* ist dann der Text 'e' zu sehen.

  Der Vorteil der Methode kommt in unserem Beispiel nicht wirklich zum Tragen. Bei kommerziellen Projekten liegt er darin, dass der Übersetzer/in nur die Einträge in der Datei *string.xml* übersetzen muss, der restliche Quellcode muss nicht angefasst werden.

3.4 Erstellen eines Scrollbereichs



Ein wichtiges Feature einer App ist die Möglichkeit einen Bereich des Bildschirms mit dem Finger zu scrollen. Dies ist vergleichsweise einfach zu bewerkstelligen.

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, analog zur oben beschriebenen Vorgehensweise. In dieses Objekt wird wiederrum ein neues *GridLayout* eingefügt. Der Trick dabei ist, dass das *GridLayout*-Objekt breiter sein muss als das *HorizontalScrollView*-Objekt. Damit kann man in dem Bereich des *HorizontalScrollView*-Objektes mit dem Finger scrollen. Innerhalb des GridLayout werden dann weitere GUI-Objekte wie Buttons eingefügt.

Veranschaulichung:

```
<HorizontalScrollView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/horizontalScrollView"
    android:paddingTop="10dp"
    android:layout_row="6"
    android:layout_column="0"
    android:layout_columnSpan="3" >

    <GridLayout
        android:layout_height="match_parent"
        android:id="@+id/TaschenrechnerScroll"
        android:layout_width="660dp" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="60dp"
            android:textAppearance="?android:attr/t
            android:text="Basic"
            android:id="@+id/txtInfo1"
            android:layout_row="0"
            android:layout_column="0"
```

Der *HorizontalScrollView* wird in das äußere Grid eingefügt. In diesem Bereich kann dann mit dem Finger gescrollt werden.

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

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

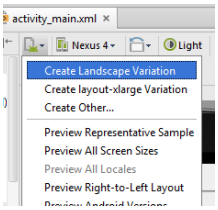
Die Zählung der Zellenposition im inneren Grid beginnen wieder bei 0 !

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:

```
<GridLayout ...>
      <EditText ... />
      <HorizontalScrollView ...>
            <GridLayout ... >
                  <TextView ... />
            </GridLayout>
      </HorizontalScrollView>
</GridLayout>
```

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

3.5 Erstellen einer horizontalen Variante



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.

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

4 Programmierung der Fachklasse

Die Taschenrechner-App wurde so konzipiert, dass sie ähnlich einem echten Taschenrechners nur über ein einziges Feld für Ein- und Ausgabe (Lösungsfeld) verfügt. Der Benutzer tippt dabei eine erste Zahl ein, die im Lösungsfeld gezeigt wird. Danach wählt er eine Rechenoperation aus, anschließend die zweite Zahl, die ebenfalls im Lösungsfeld gezeigt wird. Schließlich tippt er die Taste = um das Ergebnis im Lösungsfeld angezeigt zu bekommen (Zweistellige Verknüpfung).

Einige Berechnungsarten (√, sin, ln, 1/x usw.) benötigen aber nur eine Zahl. Hier wird der Vorgang verkürzt. Es wird nun keine = Taste mehr benötigt, um das Ergebnis zu berechnen, dies wird von der Operator-Taste durchgeführt. Der Benutzer tippt dabei eine erste Zahl ein, die im Lösungsfeld gezeigt wird. Danach wählt er eine Rechenoperation aus, um das Ergebnis im Lösungsfeld angezeigt zu bekommen (Einstellige Verknüpfung).

Beide Varianten werden nachfolgend schematisch dargestellt:

Zweistellige Verknüpfung:
Beispiel für einen Ablauf einer einfachen Berechnung mit zwei Zahlen (Bsp. 9 + 5 = 14):

Ein Berechnungsobjekt *br* wird erzeugt

Die Zahl **9** wird in das Lösungsfeld geschrieben

- Das Attribut *zahl1* des *br*-Objektes wird mit der Zahl **9** aus dem Lösungsfeld gefüllt.
- Das Attribut *operation* des *br*-Objektes wird mit dem String **"+"** gefüllt.
- Das Lösungsfeld wird leergeräumt.

Die Zahl **5** wird in das Lösungsfeld geschrieben

- Das Attribut *zahl2* des *br*-Objektes wird mit der Zahl **5** aus dem Lösungsfeld gefüllt.
- Die Methode *rechne()* des br-Objektes fragt das Attribut *operation* ab und führt die gewählte Operation durch.
- Die Antwort der Methode *rechne*() wird in das Ergebnisfeld geschrieben.

Dies heißt also, dass nicht die Buttons mit den Rechenoperationen die eigentlichen Berechnungen durchführen, sondern erst die = Taste. Die Tasten mit den Rechenoperationen geben lediglich weiter, welche Berechnungsart dann von der = Taste durchgeführt wird.

Einige Berechnungsarten (√, sin, ln, 1/x usw.) benötigen aber nur eine Zahl. Hier wird der Vorgang verkürzt. Es wird nun keine =-Taste mehr benötigt, um das Ergebnis zu berechnen, dies wird von der Operator-Taste durchgeführt:

Einstellige Verknüpfung:
Beispiel für einen Ablauf einer einfachen Berechnung mit einer Zahl (Bsp. √9 = 3):

Ein Berechnungsobjekt *br* wird erzeugt

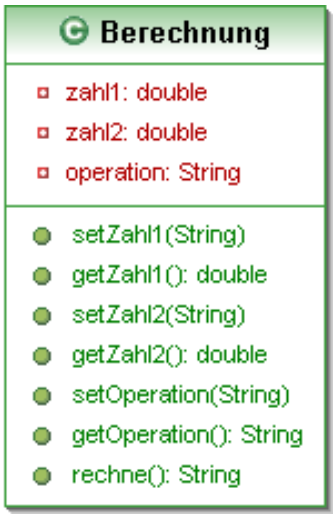
Die Zahl **9** wird in das Lösungsfeld geschrieben

- Das Attribut *zahl1* des br-Objektes wird mit der Zahl **9** aus dem Lösungsfeld gefüllt.
- Das Attribut *operation* des *br*-Objektes wird mit dem String **"w"** gefüllt.
- Die Methode *rechne()* des *br*-Objektes fragt das Attribut *operation* ab und führt die gewählte Operation durch.
- Die Antwort der Methode *rechne()* wird in das Ergebnisfeld geschrieben.

Gemäß dem 2-Schichten-Modell werden die Daten der beiden Zahlen und des Operatorsymbols, sowie die Methode zur Berechnung, in einer eigenen Klasse *Berechnung* gehalten. Diese Klasse wird dann von der *MainActivity.java* genutzt.

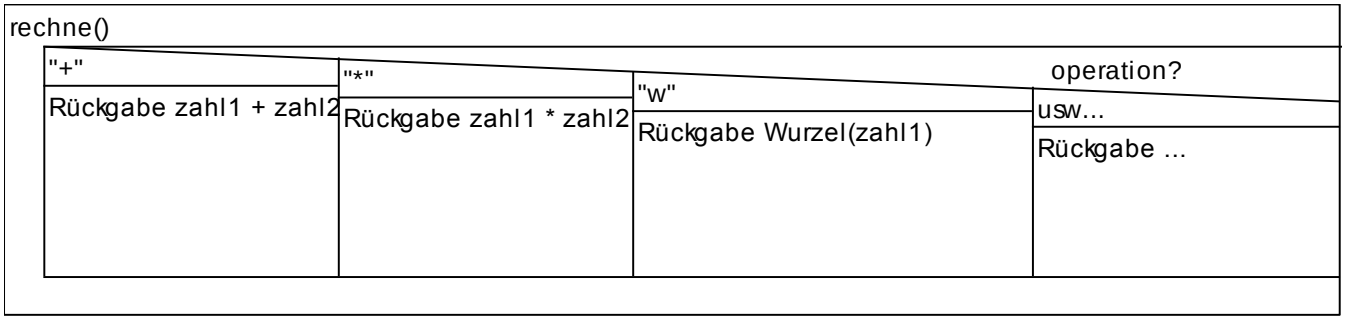
Die Klasse *Berechnung* enthält folgende Attribute:

zahl1	enthält die erste übergebene double-Zahl
zahl2	enthält die zweite übergebene double-Zahl. (Diese wird bei der einstelligen Verknüpfung nicht benötigt.)
operation	enthält einen String, der die übergebene Operation als Symbolzeichen enthält.



Neben den Getter- und Setter-Methoden der obigen Attribute enthält die Klasse *Berechnung* die Methode *rechne()*:
Diese fragt das Attribut *operation* ab, führt die entsprechende Rechenoperation durch und gibt das Ergebnis als String zurück. Dies deshalb, damit die aufrufende Methode das Ergebnis sofort in das Lösungsfeld schreiben kann und sich nicht noch mit der Umwandlung aufhalten muss.

Die Methode *rechne()* besteht aus einer Mehrfachabfrage.



Bei folgenden Rechenoperationen müssen noch folgende Einschränkungen beachtet werden:

- Bei der Division muss geprüft werden, ob *zahl2* ungleich 0 ist. Ansonsten gibt *rechne()* den String "Error" zurück.

- Beim Ziehen einer Quadratwurzel muss überprüft, ob *zahl1* größer oder gleich 0 ist. Ansonsten gibt *rechne()* den String "Error" zurück.
- Beim Berechnen des natürlichen Logarithmus (ln) muss überprüft, ob *zahl1* größer 0 ist. Ansonsten gibt *rechne()* den String "Error" zurück.
- Bei der Berechnung des reziproken Wertes (1/x) muss geprüft werden, ob *zahl1* ungleich 0 ist. Ansonsten gibt *rechne()* den String "Error" zurück.

Die Methode *rechne()* der Klasse *Berechnung*:

```
public String rechne()
{
	if (this.getOperation().equals("+"))
	{
		return String.valueOf(this.getZahl1()+this.getZahl2());
	}
	else if (this.getOperation().equals("-"))
	{
		return String.valueOf(this.getZahl1()-this.getZahl2());
	}
	else if (this.getOperation().equals("*"))
	{
		return String.valueOf(this.getZahl1()*this.getZahl2());
	}
	else if (this.getOperation().equals(":"))
	{
		if (this.getZahl2()!=0)
		{
			return String.valueOf(this.getZahl1()/this.getZahl2());
		}
		else
		{
			return "Error";
		}
	}
	else if (this.getOperation().equals("w"))
	{
		if (this.getZahl1()>=0)
		{
			return String.valueOf(Math.sqrt(this.getZahl1()));
		}
		else
		{
			return "Error";
		}
	}
```

```
else if (this.getOperation().equals("p"))
{
      return String.valueOf(Math.pow(this.getZahl1(),2));
}
else if (this.getOperation().equals("nw"))
{
      return String.valueOf(Math.pow(this.getZahl2(),1/this.getZahl1()));
}
else if (this.getOperation().equals("np"))
{
      return String.valueOf(Math.pow(this.getZahl1(),this.getZahl2()));
}
else if (this.getOperation().equals("ln"))
{
      if (this.getZahl1()>0)
      {
            return String.valueOf(Math.log(this.getZahl1()));
      }
      else
      {
            return "Error";
      }
}
else if (this.getOperation().equals("e"))
{
      return String.valueOf(Math.pow(Math.E,this.getZahl1()));
}
else if (this.getOperation().equals("sin"))
{
      return String.valueOf(Math.sin(this.getZahl1()));
}
else if (this.getOperation().equals("cos"))
{
      return String.valueOf(Math.cos(this.getZahl1()));
}
else if (this.getOperation().equals("asin"))
{
      return String.valueOf(Math.asin(this.getZahl1()));
}
else if (this.getOperation().equals("acos"))
{
      return String.valueOf(Math.acos(this.getZahl1()));
}
else if (this.getOperation().equals("%"))
{
```

```
            return String.valueOf(this.getZahl1()*this.getZahl2()/100);
      }
      else if (this.getOperation().equals("/"))
      {
            if (this.getZahl1()!=0)
            {
                  return String.valueOf(1 / this.getZahl1());
            }
            else
            {
                  return "Error";
            }
      }
      else
      {
            return "Error";
      }
}
```

5 Programmierung in der MainActivity.java

Diese Java-Klasse repräsentiert die MainActivity, die unsere App steuert. Sie ist somit die zentrale Datei, was die Steuerung der App über die GUI-Elemente angeht. Sie wird beim Erstellen des Projektes mit einigen vorbereiteten Methoden bereitgestellt.

5.1 Die MainActivity im Ausgangszustand

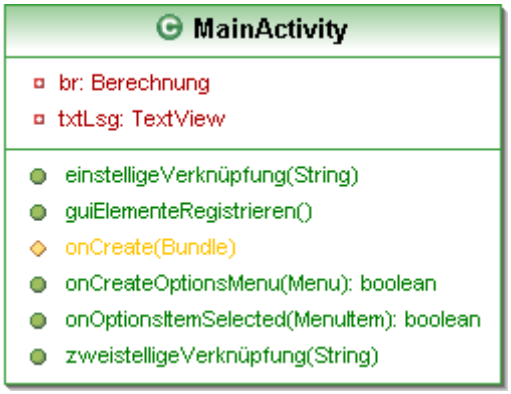
Die Klasse *MainActivity* erbt von der Klasse *ActionBarActivity*.
Die Methoden *onCreateOptionsMenu()* und *onOptionsItemSelected()* werden nur benötigt, wenn die App ein Menü bekommen soll und werden deshalb hier nicht weiter behandelt.

Die Methode *onCreate()* wird ausgeführt, wenn die App ein Exemplar der *MainActivity* erstellt, soll heißen, wenn die App gestartet wird. Sie führt folgendes durch:

- Erstellung eines Exemplars der Superklasse *ActionBarActivity.*
- Der Klasse *R* wird die Datei *activity_main.xml* als Attribut hinzugefügt. Die Klasse *R* repräsentiert somit die dort erstellte GUI. Die GUI-Objekte wie Buttons, Textfelder usw. sind somit Attribute dieser Klasse *R* und können nun in der *MainActivity* verwendet werden.
- Die GUI-Objekte wie Buttons, Textfelder usw. werden registriert und mit einem Eventhandler ausgestattet der Ereignisse wie Klicken o.ä. verarbeitet.

Der Methode *onCreate()* wird beim Start vom System ein Bundle-Objekt mitgegeben. In ihm können Zustände einer App gespeichert werden, was aber hier nicht benötigt wird.

5.2 Die MainActivity für die Taschenrechner-App



Wir haben der Klasse zwei neue Attribute und drei Methoden spendiert.
Die drei Methoden wurden nur aus Gründen der Übersichtlichkeit hinzugefügt. Man hätte diese Funktionalität auch in die *onCreate()*-Methode packen können, schwer wartbarer Code wäre aber das Ergebnis gewesen.

Die neuen Attribute:

- *br* ist ein Objekt der Fachklasse *Berechnung*, die oben vorgestellt wurde. Dieses Objekt repräsentiert eine Berechnung, die mit dem Taschenrechner durchgeführt wird.
- *txtLsg* ist das Textfeld-Objekt in dem die Ein- und Ausgabe erfolgt (Lösungsfeld). Es ist das einzige GUI-Objekt, welches als Attribut hinzugefügt wurde. Der Grund dafür ist, dass mehrere Methoden (und die Methoden der inneren Klassen der Eventhandler) dieses Objekt benötigen.

Die drei neuen Methoden werden in den nachfolgenden Kapiteln erläutert.

5.2.1 Methode *void guiElementeRegistrieren()*

Hier werden die GUI-Objekte mit Hilfe der Klasse *R* erstellt. Ähnlich wie bei JavaScript werden die in der Klasse *R* repräsentierten GUI-Elemente über eine Methode *findViewById* als GUI-Objekt gespeichert. Ein Cast übernimmt die Umwandlung der allgemeinen GUI-Elemente in ein spezielles GUI-Objekt.

Button btn1 = (Button) findViewById(R.id.btn1);

Die GUI-Objekte erhalten hier ihre Eventhandler mit folgender Funktionalität:

- Die Buttons mit Zahlen, Konstanten oder dem Komma nehmen den Inhalt des Lösungsfeldes und fügen den getippten Wert hinzu. Damit kann man mehrstellige Zahlen eingeben.
- Die Taste "Clear" räumt das Lösungsfeld leer und setzt die Attribute *zahl1* und *zahl2* des *br*-Objektes gleich 0.
- Die Buttons für die zweistelligen Verknüpfungen (+, -, * …) rufen die Methode *zweistelligeVerknüpfung()* auf und übergeben ihr den Operator als String-Zeichen.
- Der Button – (Minus) setzt zusätzlich noch ein Vorzeichen-Minus falls das Lösungsfeld leer ist.
- Die Buttons für die einstelligen Verknüpfungen (√, x^2 …) rufen die Methode *einstelligeVerknüpfung()* auf und übergeben ihr den Operator als String-Zeichen.
- Die = Taste setzt die aktuelle Zahl im Lösungsfeld als *zahl2* des *br*-Objektes und ruft dort die Funktion *rechne()* auf.

Die Methode *guiElementeRegistrieren()* wird in der *onCreate()*-Methode aufgerufen.

Quellcode:

```
public void guiElementeRegistrieren()
{
        txtLsg = (TextView) findViewById(R.id.txtLsg);
        br = new Berechnung();

        Button btn1 = (Button) findViewById(R.id.btn1);
        btn1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v)
            {
                txtLsg.setText(txtLsg.getText()+ "1");
            }
        });

        Button btn2 = (Button) findViewById(R.id.btn2);
        btn2.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v)
            {
                txtLsg.setText(txtLsg.getText()+ "2");
            }
        });

        Button btn3 = (Button) findViewById(R.id.btn3);
```

```
btn3.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "3");
    }
});

Button btn4 = (Button) findViewById(R.id.btn4);
btn4.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "4");
    }
});

Button btn5 = (Button) findViewById(R.id.btn5);
btn5.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "5");
    }
});

Button btn6 = (Button) findViewById(R.id.btn6);
btn6.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "6");
    }
});

Button btn7 = (Button) findViewById(R.id.btn7);
btn7.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "7");
    }
});

Button btn8 = (Button) findViewById(R.id.btn8);
btn8.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "8");
    }
```

```
});

Button btn9 = (Button) findViewById(R.id.btn9);
btn9.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "9");
    }
});

Button btn0 = (Button) findViewById(R.id.btn0);
btn0.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText(txtLsg.getText()+ "0");
    }
});

Button btnKomma = (Button) findViewById(R.id.btnKomma);
btnKomma.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
    //Ein Komma wird nur benötigt, wenn noch keines in der Anzeige vorhanden
    ist
        if (!txtLsg.getText().toString().contains("."))
        {
            txtLsg.setText(txtLsg.getText() + ".");
        }
    }
});
Button btnClear = (Button) findViewById(R.id.btnClear);
btnClear.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        txtLsg.setText("");
        br.setZahl1("0");
        br.setZahl2("0");
    }
});

Button btnGleich = (Button) findViewById(R.id.btnGleich);
btnGleich.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
          //'Gleich' soll nur ausgeführt werden, wenn
```

```
            //Das Lösungsfeld nicht leer ist,
            //es nicht nur ein Komma enthält
            //oder nicht mit "Er" (wie "Error") beginnt.
           if(!txtLsg.getText().toString().contentEquals("")&&!txtLsg.getText().
           toString().contentEquals(".")&&!txtLsg.getText().toString().
           startsWith("Er"))
           {
               br.setZahl2(txtLsg.getText().toString());
               txtLsg.setText(br.rechne());
           }
       }
});
Button btnPlus = (Button) findViewById(R.id.btnPlus);
btnPlus.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        zweistelligeVerknüpfung("+");
    }
});
Button btnMinus = (Button) findViewById(R.id.btnMinus);
btnMinus.setOnClickListener(
        new View.OnClickListener() {
            public void onClick(View v)
            {
                //Wenn das Lösungsfeld leer ist, ist das - ein Vorzeichenminus
                //ansonsten das Rechenminus
                if (txtLsg.getText().toString().equals(""))
                {
                    txtLsg.setText("-");
                }
                else
                {
                    zweistelligeVerknüpfung("-");
                }
            }
        });
Button btnMal = (Button) findViewById(R.id.btnMal);
btnMal.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        zweistelligeVerknüpfung("*");
    }
});
Button btnGeteilt = (Button) findViewById(R.id.btnGeteilt);
btnGeteilt.setOnClickListener(new View.OnClickListener() {
```

```
    public void onClick(View v)
    {
        zweistelligeVerknüpfung(":");
    }
});
Button btnWurzel = (Button) findViewById(R.id.btnWurzel);
btnWurzel.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        einstelligeVerknüpfung("w");
    }
});
Button btnPotenz = (Button) findViewById(R.id.btnPotenz);
btnPotenz.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        einstelligeVerknüpfung("p");

    }
});
Button btnNWurzel = (Button) findViewById(R.id.btnNWurzel);
btnNWurzel.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        zweistelligeVerknüpfung("nw");
    }
});
Button btnNPotenz = (Button) findViewById(R.id.btnNPotenz);
btnNPotenz.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
        zweistelligeVerknüpfung("np");
    }
});
Button btnLn = (Button) findViewById(R.id.btnLn);
btnLn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        einstelligeVerknüpfung("ln");
    }
});
Button btnE = (Button) findViewById(R.id.btnE);
btnE.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        einstelligeVerknüpfung("e");
```

```
    }
});
Button btnSinus = (Button) findViewById(R.id.btnSinus);
btnSinus.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        einstelligeVerknüpfung("sin");
    }
});
Button btnCosinus = (Button) findViewById(R.id.btnCosinus);
btnCosinus.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        einstelligeVerknüpfung("cos");
    }
});
Button btnArcsinus = (Button) findViewById(R.id.btnArcsinus);
btnArcsinus.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        einstelligeVerknüpfung("asin");
    }
});
Button btnArccos = (Button) findViewById(R.id.btnArccos);
btnArccos.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        einstelligeVerknüpfung("acos");
    }
});
Button btnProzent = (Button) findViewById(R.id.btnProzent);
btnProzent.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        zweistelligeVerknüpfung("%");
    }
});
Button btnRezi = (Button) findViewById(R.id.btnRezi);
btnRezi.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        einstelligeVerknüpfung("/");
    }
});
Button btnPi = (Button) findViewById(R.id.btnPi);
btnPi.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v)
    {
          //Ein Pi soll nur hinzugefügt werden, wenn das Lösungsfeld leer ist
        if (txtLsg.getText().toString().equals("")) {
            txtLsg.setText(txtLsg.getText() + String.valueOf(Math.PI));
```

```
                }
            }
        });
        Button btnEuler = (Button) findViewById(R.id.btnEuler);
        btnEuler.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v)
            {
                  //Die Zahl e soll nur hinzugefügt werden, wenn das Lösungsfeld leer
                  //ist
                if (txtLsg.getText().toString().equals("")) {
                    txtLsg.setText(txtLsg.getText() + String.valueOf(Math.E));
                }
        }
        });
}
```

5.2.2 Methode: *void zweistelligeVerknüpfung(String)*

Diese Methode

- setzt das Attribut *zahl1* des *br*-Objektes.
- setzt das Attribut *operation* des *br*-Objektes.

Quelltext:

```
public void zweistelligeVerknüpfung(String op)
{
      //Dies soll nur ausgeführt werden, wenn
      //Das Lösungsfeld nicht leer ist,
      //es nicht nur ein Komma enthält
      //oder nicht mit "Er" (wie "Error") beginnt.
      if(!txtLsg.getText().toString().contentEquals("")&&
      !txtLsg.getText().toString().contentEquals(".")&&
      !txtLsg.getText().toString().startsWith("Er"))
      {
            br.setZahl1(txtLsg.getText().toString());
            br.setOperation(op);
            txtLsg.setText("");
      }
}
```

5.2.3 Methode: *void einstelligeVerknüpfung(String)*

Diese Methode

- setzt das Attribut *zahl1* des *br*-Objektes.
- setzt das Attribut *operation* des *br*-Objektes.
- Berechnet das Ergebnis mit der Methode *rechne()* und schreibt dies in das Lösungsfeld.

Quelltext:

```
public void einstelligeVerknüpfung(String op)
{
      //Dies soll nur ausgeführt werden, wenn
      //Das Lösungsfeld nicht leer ist,
      //es nicht nur ein Komma enthält
      //oder nicht mit "Er" (wie "Error") beginnt.
      if(!txtLsg.getText().toString().contentEquals("")&&
      !txtLsg.getText().toString().contentEquals(".")&&
      !txtLsg.getText().toString().startsWith("Er"))
      {
            br.setZahl1(txtLsg.getText().toString());
            br.setOperation(op);
            txtLsg.setText(br.rechne());
      }
}
```

6 Weitere Vorgehensweise

Sie können nun die App in Ihrer Testumgebung testen. Siehe dazu *Handreichung - Installation und Einrichtung Android Studio*.