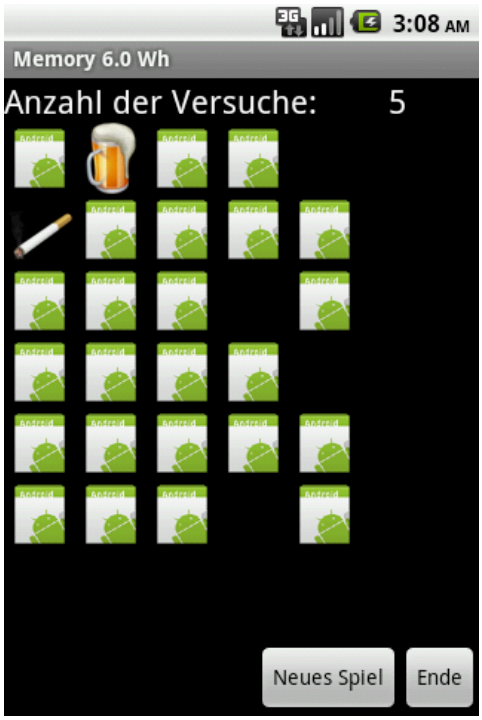
Objektorientiertes Modellieren und Implementieren mit Java für Android-Smartphones
- eine neue pädagogische Dimension für den Informatikunterricht

Unter den Informatik Aufgaben des hessischen Landesabiturs finden sich mehrere, die im Kontext von Smartphone-Apps[1] stehen. Solche Aufgaben lassen sich leicht als reale Apps implementieren. Am Beispiel des bekannten Memory-Spiels werden hier die Modellierung, die Wirkprinzipien und die Implementierung für Android-Smartphones erläutert.



Smartphones sind vollwertige Informatiksysteme, die den Alltag der Lernenden prägen. Im Unterricht entwickelte Apps können von Schülerinnen und Schülern jederzeit ortsunabhängig benutzt und mit einem gewissen Stolz auf die eigene Leistung auch anderen präsentiert werden. Das darin liegende Motivationspotenzial sollte unterrichtlich genutzt werden. Realitätsnahe und motivierende Anwendungsbeispiele gibt es viele, u. a. mit Animationen und Sounds, sowie programmierte grafische Darstellungen mit Auswertung von Sensordaten.

Das Memory-Spiel ist als Unterrichtsprojekt in der Qualifikationsphase gut geeignet um einen Einblick in professionelle Softwarearchitekturen und Entwicklungs-umgebungen zu vermitteln. Dies ist in der Oberstufe im wissenschafts-propädeutischen Sinne unerlässlich. In Hessen werden in der Oberstufe noch keine Blöcke gestapelt und dies wird auch noch lange Zeit so bleiben. Fundamentale Konzepte, wie die Speicherung der Struktur und Attribute von Benutzeroberflächen und von der Landessprache abhängige Textkonstanten in XML-Dateien, aber auch Konzepte der ereignisgesteuerten Programmierung und die Konzepte der objektorientierten Modellierung, u. a. Assoziation, Aggregation und Vererbung können an diesem Beispiel gut erarbeitet werden.

Dabei empfehlen sich eine projektartige und produktorientierte Vorgehensweise und eine sorgfältige methodische Planung. Manches muss man vorgeben, vieles kann aber auch von den Lernenden selbstständig erkundet und erarbeitet werden. Insbesondere sollten die dazu notwendigen instrumentellen Fertigkeiten, wie z. B. der Umgang mit der Entwicklungsumgebung und das Anlegen eines neuen Android-Projektes können im Sinne Inverted Classroom[2] am besten mit selbst erstellten Video-Tutorials vermittelt werden.

Die unterrichtlichen Voraussetzungen sind in Hessen am Ende der Einführungsphase in der Oberstufe i. d. R. gegeben, wenn man mit Java in das Programmieren einführt und HTML mit Stylesheets, zur Trennung von Layout und Inhalten, verwendet hat. Es lohnt sich daher, bereits in der Einführungsphase mit Eclipse zu arbeiten und einfache Benutzeroberflächen mit dem *WindowBuilder* zu implementieren[3].

[1] App, Kurzform für Applikation
[2] Vgl. http://wiki.zum.de/Flipped_Classroom
[3] Vgl. Wehrheim, Otto, LOG-IN, Jg. 2010, H. 163/164, S. 122 ff.

Eclipse unterstützt die Entwicklung von Android-Projekten in hervorragender Weise. Die meisten Fehler, auch Android-spezifische, werden von Eclipse schon beim Entwickeln erkannt. Zur Behebung der Fehler werden intelligente Korrekturvarianten vorgeschlagen.

Der nachstehend beschriebene Entwicklungsprozess ist exemplarisch für ähnliche Projekte. Die Vorgaben für eine Veröffentlichung im Android-Marketplace werden nur ansatzweise beachtet, näheres findet man im Internet.

Was muss man installieren?

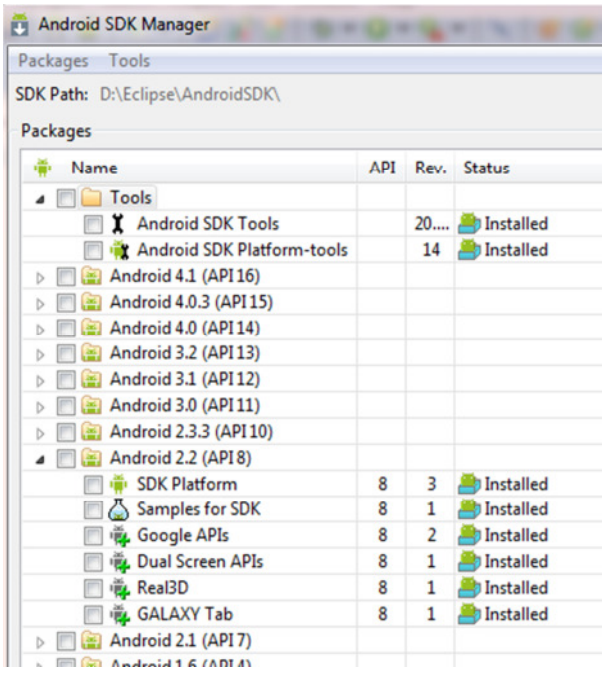
Die nachstehend beschriebene Installation ist problemlos, benötigt aber etwas Zeit.

1. JDK (Java Development Kit = JRE + Compiler) (32 bzw. 64 bit)
http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1637583.html

2. Die neueste Distribution Juno von Eclipse
Laden Sie dazu die 32 bzw. 64 bit Version *Eclipse IDE for Java Developers* herunter.
http://www.eclipse.org/downloads/
Entpacken Sie das *zip*-File in einen selbst gewählten Ordner. Es ist zweckmäßig, dann eine Verknüpfung auf die Datei *eclipse.exe*. anzulegen.
Legen Sie weiterhin ein Verzeichnis (Workspace) für Ihre *Eclipse-Android-Projekte* an.

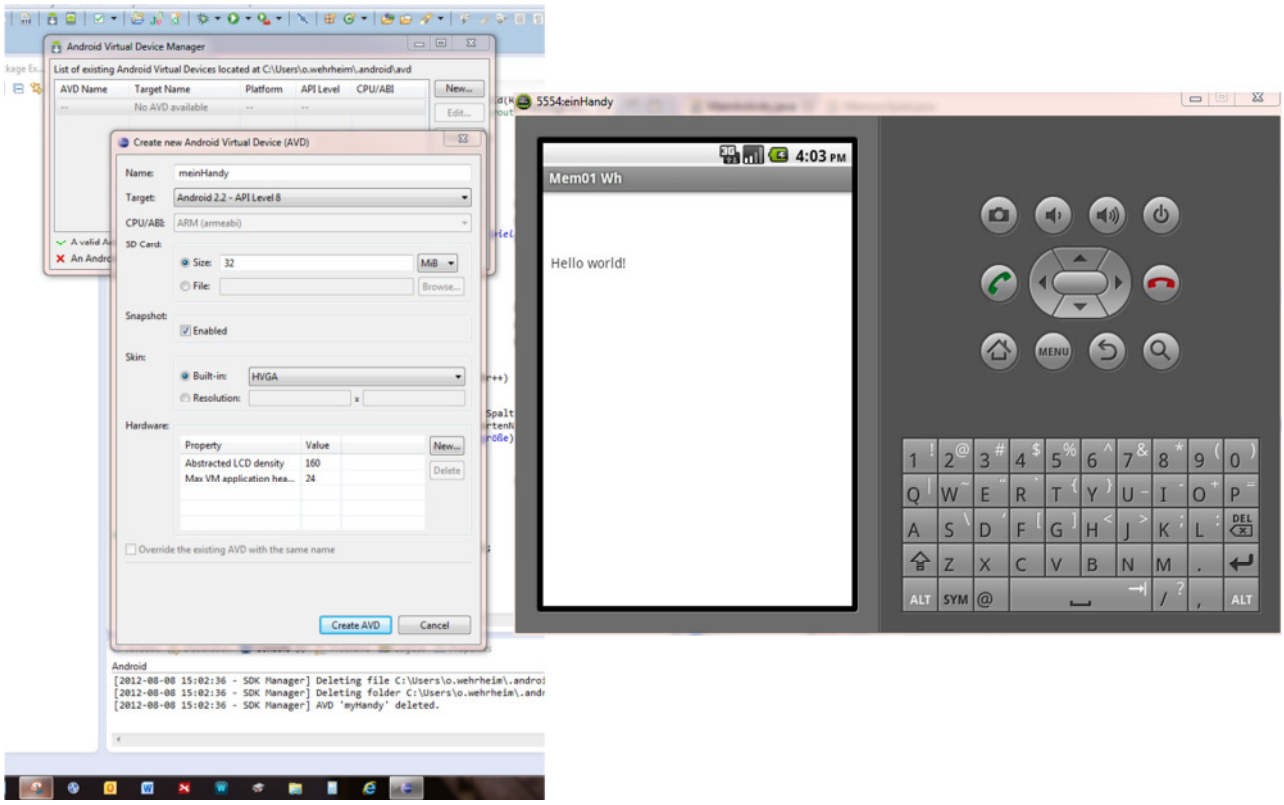
3. Das Android Development Tool (ADT)
Starten Sie dazu Eclipse und wählen Sie den angelegten Workspace aus. Wählen Sie in der Eclipse-Menüleiste *Help / Install New Software* und fügen Sie mit *ADD* eine neue Repository mit dem Namen Android und der Location https://dl-ssl.google.com/android/eclipse hinzu. Wählen Sie alles aus und akzeptieren Sie die Lizenzen.

4. Das Android Software Development Kit (Android SDK)
http://developer.android.com/sdk/index.html
Laden Sie das SDK herunter und installieren Sie es in ein selbstgewähltes Verzeichnis. Installieren Sie für den Anfang nur *API 8*. Eclipse muss nun den Installationsort des Android SDK kennenlernen.
Wählen Sie dazu in der Eclipse-Menüleiste *Window / Preferences / Android* und geben Sie das Installationsverzeichnis an, z. B. *D:\Eclipse\AndroidSDK\android-sdk*



Ein virtuelles Smartphone erzeugen

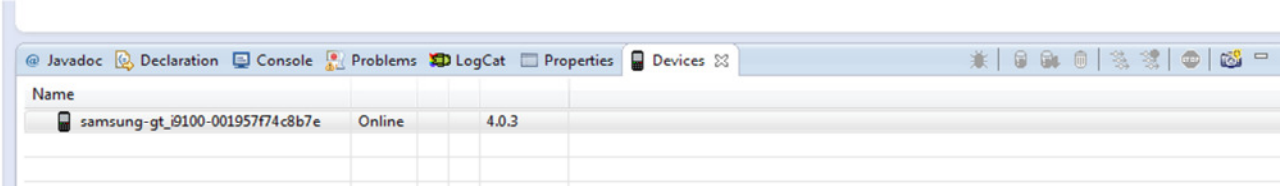
Starten Sie mit einem Klick auf das Icon in der Eclipse-Menüleiste den AVD-Manager (Android Virtual Device Manager). Für viele Handys gibt es vom Hersteller spezielle AVDs. Für die meisten genügt die abgebildete Konfiguration.



Starten Sie den Emulator und machen Sie sich damit vertraut. Natürlich kann man damit nicht telefonieren. Die verfügbaren Geräte kann man sich beispielweise unten im Consolen-Fenster mit *Window / Show View / Other /Devices* anzeigen lassen. Rechts gibt es dann einen Button für *Device Screen Capture*

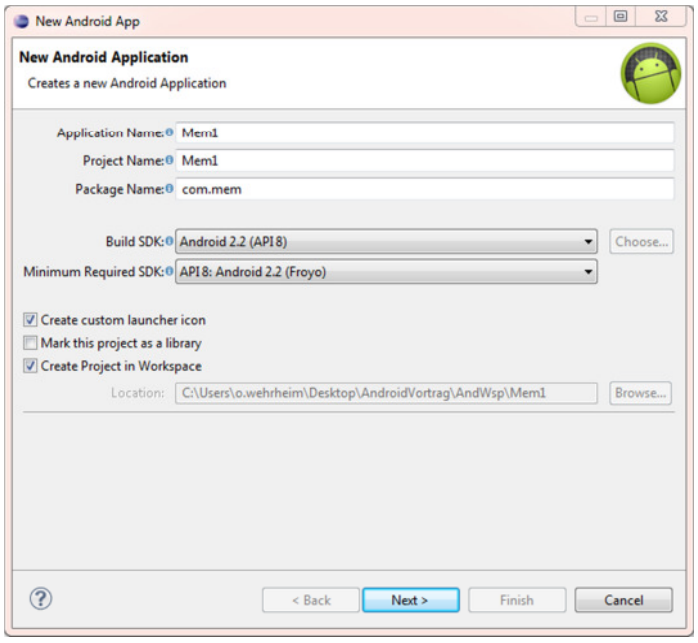
1.6 Das Smartphone mit Eclipse verbinden

Man kann jetzt auch das Smartphone z. B. über USB mit dem PC verbinden. Unter Devices wird das angeschlossene Handy angezeigt. Nach den Standardeinstellungen für die Run Configurations in Eclipse wird ein Programm bevorzugt auf dem Handy automatisch installiert und gestartet. Auf dem Handy sollten Sie unter *Einstellungen / System / Entwickler Optionen* das USB-Debugging aktivieren.

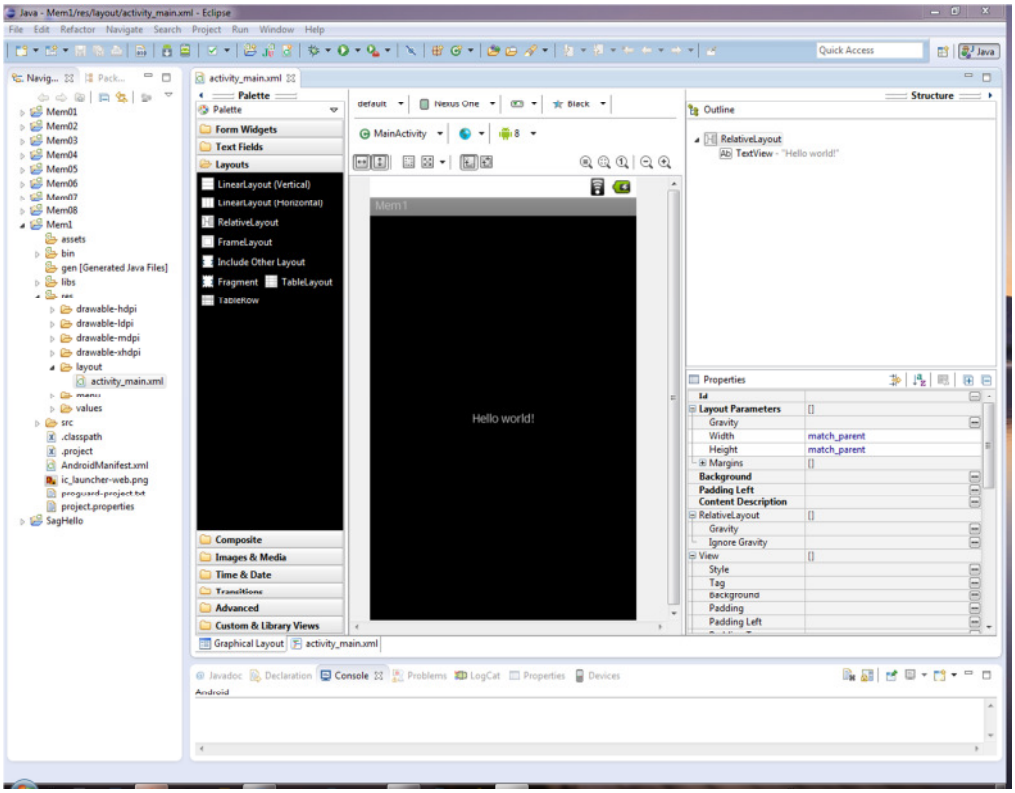


Ein erstes APP

Legen Sie mit *File / New / Android Application Project* ein neues Projekt an.
Wählen Sie ein *ClipArtImage* aus und fahren Sie fort mit *Create BlankActivity* und *Finish*.



Es öffnet sich die Ansicht des GUI-Builders für die Datei *activity_main.XML*.



Jetzt sollten Sie alle Fester schließen, etwas warten und mit einem *RechtsClick* auf das Projekt im Navigator mit *Run As / Android Applcation* die App im Emulator starten. Auf der Konsole werden die Installation des Apps auf dem Smartphone und der Start der App protokolliert. Dazu muss man rechts mit den Konsolen-Icon die Android-Konsole auswählen.

Die GUIs (Graphical User Interfaces) heißen unter Android Activitys. Das Layout einer Activity wird im Verzeichnis *layout* als XML-Datei gespeichert. Man kann die XML-Datei direkt edieren oder auch mit dem Graphical-Layout-Manager bearbeiten. Eine APP kann mehrere Activities haben. Wir verwenden in unserem Projekt jedoch nur eine.

Die Struktur eines Android-Projekts in Eclipse

Im ersten Augenblick erscheint die Verzeichnisstruktur sehr komplex. Aber wenn man diese internalisiert hat, ist alles transparent und verständlich, nichts bleibt verborgen.
Die wichtigste Datei ist die *AndroidManifest.xml*. Sie enthält die Metadaten des Projekts. In Eclipse gibt es dazu viele Eingabefenster und viele Definitionsmöglichkeiten. Für das *HelloWord*-App schaut man sich am besten die XML-Ansicht an. Die XML-Werte, welche mit @ beginnen, sind Referenzen auf Verzeichnisse in denen die tatsächlichen Inhalte stehen.
Im Verzeichnis *src* steht der Quellcode für die Java-Klassen. Die vorgenerierte Startklasse heißt *MainActivity.java.*

```
package com.mem;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Hier Programmcode einfügen
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        // zunächst unverändert lassen
        return true;
    }
}
```

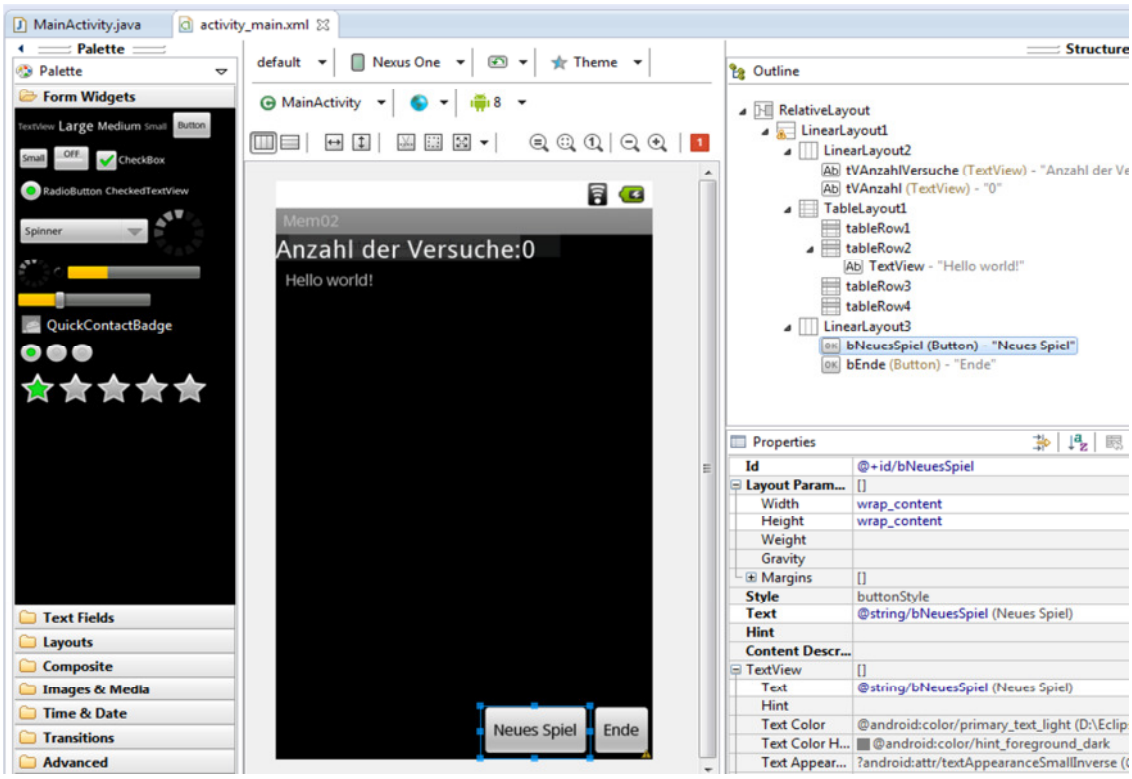
Die Methode *setContentView(R.layout.activity_main)* teilt dem Eclipse-Project-Builder mit, dass das Layout *activity_main* verwendet werden soll. Schauen Sie sich dazu die Klasse *R* im Verzeichnis *gen* an. Hier werden die Bezüge zu den Projekt-Ressourcen und den XML Dateien hergestellt. Unmittelbar nach Erstellung oder Änderung einer Ressourcendatei startet

das Eclipse-Plug-in den Ressourcencompiler AAPT[4] des Android-SDK. Die Dateien im Verzeichnis *gen* werden automatisch generiert und aktualisiert. An den Dateien, die zum Compilieren benötigt werden, sollten Sie nichts ändern.
Die Verzeichnisse *bin* und *lib* haben die gleiche Funktion wie in Java.
Im Verzeichnis *assets* kann man Dateien ablegen, die nicht durch Android-Standardmechanismen verwaltet werden. Bei den meisten Apps bleibt *assets* leer.

Das Wichtigste außer dem Programmcode steht im Verzeichnis *res*. Hier werden die Ressourcen abgelegt. Die Bilddateien stehen in verschiedenen Größen in den Verzeichnissen, die mit *drawable* beginnen. Für die Icons verwenden wir in unserem Project nur eine Größe von 48 x48 dpi. Zur Vereinfachung kann man das Verzeichnis *drawable-mdi* mit Eclipse in *drawable* umbenennen und die anderen *drawable*-Verzeichnisse löschen. Im Verzeichnis *layout* stehen die XML-Dateien für die Layouts der Activities und die die XML-Dateien für Menüs. Von großer Bedeutung ist das Verzeichnis *values*. In keiner Activity dürfen Texte in Form von String-Konstanten verwendet werden. Diese müssen in der XML-Datei *strings* hinterlegt werden. Das gilt auch für alle Beschriftungen von GUI-Objekten, also auch für Buttons. Damit erreicht man eine leichte Portierung in Sprachen anderer Länder. Für eine App kann man verschiedene Themes definieren, wie man das von vielen Content-Management-Systemen kennt. In der Datei styles.XML sollte man *Theme.Black* eintragen.

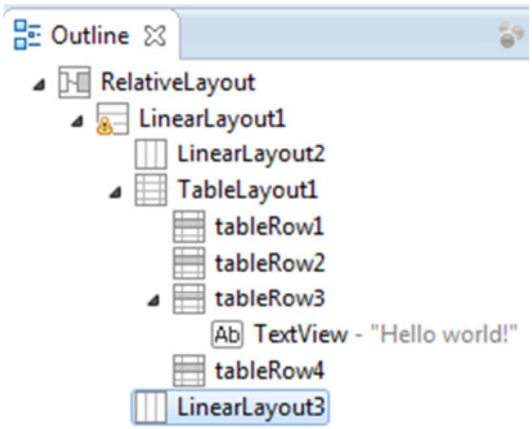
Das Layout für das Memory-Spiel

Kopieren Sie mit einem *RechtsClick* im Navigator das Projekt *Mem1* und fügen Sie mit Paste das Projekt mit dem Name Mem2 ein. Damit ist sichergestellt, dass der bisherige Zustand erhalten bleibt, falls etwas schief geht. Öffnen Sie die Datei strings.xml in der XML-Ansicht und ändern Sie die Werte von *Mem1* nach *Mem2* und Titel entsprechend.



[4] AAPT,

Öffnen Sie jetzt die Datei activity_main.xml in der *Graphical Layout Ansicht*. Stellen Sie sicher, dass die Layout-Struktur im Outline Fenster angezeigt wird (*Window / Show View / Outline*). Platzieren Sie als erstes ein LinearLayout innerhalb des ersten *RelativeLayout*. Mit einem RechtsClick und *AssignID* geben Sie dem Layout eine ID z. B.. *LinearLayout1*. In dem Property-Fenster muss dann ein Eintrag mit @ stehen, bspw. *@+id/LinearLayout1*. Arbeiten Sie bevorzugt mit dem Outline-Fenster, eine Stärke von Eclipse. Stellen Sie die abgebildete Layout-Struktur her. Setzten Sie im Property-Fenster für die Layouts folgende Attribute:




```
LinearLayout1
layout_width ="match_parent"
layout_height="match_parent"
orientation="vertical"

LinearLayout2
layout_width ="wrap_content"
layout_height="wrap_content"
orientation="horizontal"

TableLayout1
layout_width ="match_parent"
layout_height="wrap_content" >

LinearLayout3
layout_width="match_parent"
layout_height="match_parent"
android:gravity="bottom|right"
android:orientation="horizontal"
```

Fügen Sie jetzt innerhalb von *LinearLayout2* zwei *TextView*-Felder vom Typ *Large Text* mit der ID *tVAnzahlVersuche* und *tVAnzahl* ein. Die *TextView*-Objekte liegen im Ordner *Form Widgets*. Setzen Sie die ID wie oben beschrieben. Jetzt müssen die dazugehörenden Strings definiert werden. Mit einem Click auf den …-Icon in *Properties / Text* öffnet sich der String-Editor. Mit *New String* definieren Sie nun die Strings. Fügen Sie analog, entsprechend der obenstehenden Abbildung, zwei Buttons in das LinearLayout3. Achten Sie auf die Beschriftungen und Referenzen. Die Datei *strings* müsste wie folgt aussehen:

```
<resources>
    <string name="app_name">Mem2</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">Memory2</string>
    <string name="tVAnzahlVersuche">Anzahl der Versuche:</string>
    <string name="tVAnzahl">0</string>
    <string name="bNeuesSpiel">Neues Spiel</string>
    <string name="bEnde">Ende</string>
</resources>
```

Kontrollieren Sie auch die Datei *activity_main.XML*.

Die App soll jetzt auf Button-Ereignisse reagieren

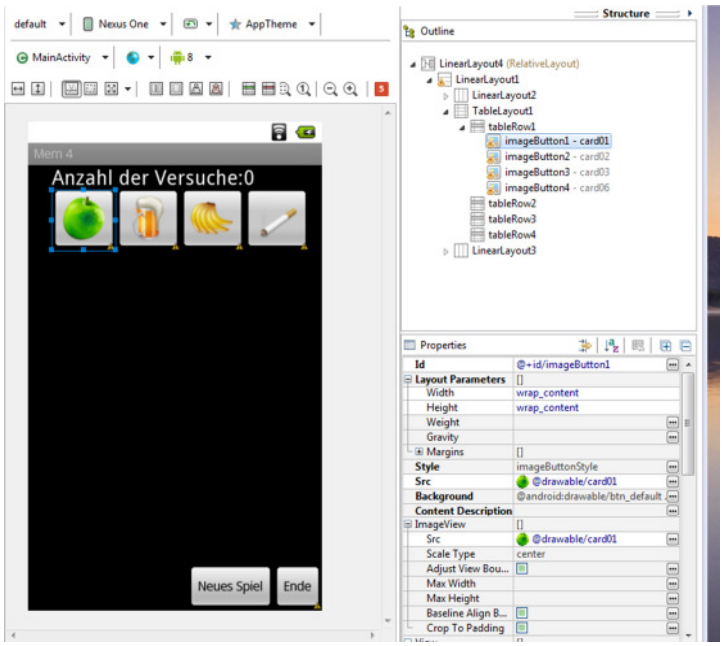
Mit dem Button „Neues Spiel" soll zunächst nur die Anzahl der Versuche hochgezählt werden um die dazu erforderlichen Programmiertechniken kennen zu lernen. Dazu wird die Methode *onCreate* in der Datei MainActivity.java entsprechend ergänzt. Das geschieht wie in Java. Man muss nur die entsprechende Objektvariable mit der Methode *findViewById* auf das in der R-Klasse hinterlegte Objekt referenzieren. Anschließend wird mit *setOnClickListener* die Referenz zu einem neuen Listener hergestellt. Auch für den Zugriff auf das Textfeld muss man mit *findViewById* eine entsprechende Objektvariable auf das Feld *tVAnzahl* setzen. Hier wird die Bedeutung der Klasse R deutlich.

```
public class MainActivity extends Activity {
    private int Anzahl = 0;

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

        Button EndeButton = (Button) findViewById(R.id.bEnde);
        EndeButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
            }
        });

        Button NeuesSpiel = (Button) findViewById(R.id.bNeuesSpiel);
        NeuesSpiel.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                TextView tvAnzahl = (TextView) findViewById(R.id.tVAnzahl);
                Anzahl++;
                tvAnzahl.setText(Integer.toString(Anzahl));
            }
        });

    }
```

Vorbereitung der Graphik-Dateien

Kopieren Sie *Mem3* nach *Mem4* und benennen Sie die Datei *drawable.mdpi* um, in *drawable*. Die anderen drawable-Verzeichnisse sollten Sie löschen. Unter http://www.iconeasy.com/ findet man gute, für ein Memory-Spiel geeignete Icons. Wir verwenden 48 dpi Icons im png-Format. Stellen Sie sich ein Icon-Set mit mindestens fünfzehn Icons zusammen. Damit man den Dateinamen automatisch generieren kann, erhalten die Dateien die Namen *icon01.png*, *icon02.png* usw. Kopieren Sie alle Icons in das *drawable*-Verzeichnis. Eclipse müsste nach kurzer Zeit die Klasse R im Verzeichnis gen aktualisiert haben.

Zum Testen können Sie jetzt einige *ImageButtons* in ein *TableRow* einfügen.



Eine neue Klasse *Karte*

Es wird schnell klar, dass man mit den Standardkonzepten nicht weiterkommt. Eine Karte muss u.a. zwei Bilder haben, eine Methode *umdrehen* und einen *OnClickListener*. Im Projekt *Mem5* legen wir im Verzeichnis *src* eine neue Java-Klasse *Karte* als Nachfolger von der Android-Klasse *Button* an und implementieren einen *OnClickListner*. Achten Sie dabei auf den korrekten Import *android.widget.Button*.

```
public class Karte extends Button implements OnClickListener{

private Drawable backImage;
private Drawable iconImage;
private Boolean  Bildseite;

    public Karte(Context view, String Bildname){
        super(view);
        int resourceId =
            getResources().getIdentifier("com.mem:drawable/"+Bildname,null,null);
        iconImage = getResources().getDrawable(resourceId);
        backImage = getResources().getDrawable(R.drawable.icon);
        this.setId(resourceId);
        this.setBackgroundDrawable(backImage);
        this.setOnClickListener((OnClickListener) this);
        Bildseite = false;
    }
    public boolean isAufgedeckt(){
        return Bildseite;}
```

```
    public void zeigeBildseite(){
        this.setBackgroundDrawable(iconImage);
        Bildseite = true;   }

    public void zeigeRückseite(){
    . . .   }

    public void umdrehen() {
    . . .}

    @Override
    public void onClick(View view) {
        umdrehen();
        MainActivity.inkrementAnzahl();
    }
}
```

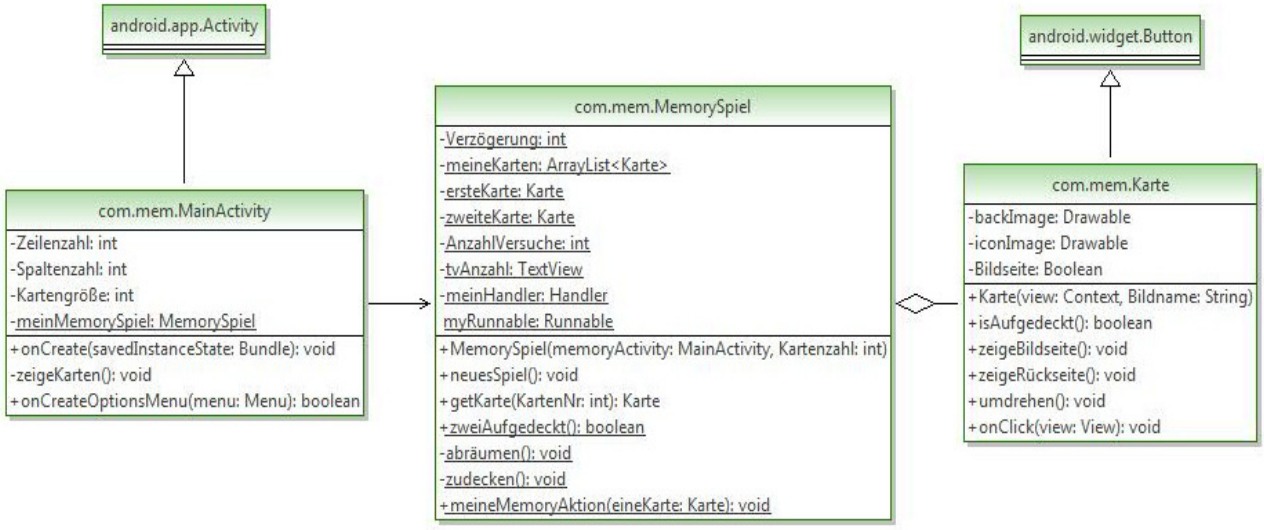
Auch das für das Spiel erforderliche *TableLayout* sollte flexibel zur Laufzeit aufgebaut werden. Sechs oder mehr *TableRows* mit dem Layout-Manger einzurichten, ist nicht sinnvoll. Es ist zweckmäßig, später über das Option-Menü die Anzahl der Icons zu variieren.

Löschen Sie im Outline-Fenster des Layout-Managers alle *Rows* im T*ableLayout1*.
Das nachstehende Codefragment zeigt, wie man das erforderliche *TableLayout* zur Laufzeit aufbauen und die Karten einfügen kann. Fügen Sie dazu in der
Klasse *MainActivity* folgende Anweisungen ein, die sich selbst erklären.

```
TableLayout meinTablelayout = (TableLayout) findViewById(R.id.TableLayout1);

        Karte eineKarte;
        TableRow Zeile;

        Zeile = new TableRow(this);
        eineKarte = new Karte(this, "card01");
        Zeile.addView(eineKarte, 48, 48);
        eineKarte = new Karte(this, "card02");
        Zeile.addView(eineKarte, 48, 48);
        meinTablelayout.addView(Zeile);

        Zeile = new TableRow(this);
        eineKarte = new Karte(this, "card06");
        Zeile.addView(eineKarte, 48, 48);
        eineKarte = new Karte(this, "card07");
        Zeile.addView(eineKarte, 48, 48);
        meinTablelayout.addView(Zeile);



public static void inkrementAnzahl() {
        Anzahl++;
        tvAnzahl.setText(Integer.toString(Anzahl));
    }
```

Modellierung und Implementierung des Memory-Spiels

Das eigentliche Memory-Spiel (*Mem6*) wird in einer eigenen Klasse *MemorySpiel* implementiert. Der Konstruktor der Klasse *MainActivity* zeigt lediglich mit der Methode *zeigeKarten()* die in einer *ArrayList* in der Klasse *MemorySpiel* gepeicherten Karten im TableLayout an. Man beachte, dass dies nur einmal beim Start der App geschieht. Das nachstehende UML-Diagram zeigt die Modellierung, danach folgt der vollständige Code der Klasse *MainActivity*.




```
public class MainActivity extends Activity {
    private final int Zeilenzahl = 6;
    private final int Spaltenzahl = 5; //Zeilenzahl * Spaltenzahl muss gerade sein
    private final int Kartengröße = 48;
    private static MemorySpiel meinMemorySpiel;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        meinMemorySpiel = new MemorySpiel(this, Zeilenzahl * Spaltenzahl);
        ZeigeKarten();

        Button EndeButton = (Button) findViewById(R.id.bEnde);
        EndeButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
                System.exit(0); // wegen Garbage Collection
            }
        });

        Button NeuesSpiel = (Button) findViewById(R.id.bNeuesSpiel);
        NeuesSpiel.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                meinMemorySpiel.neuesSpiel();
            }
        });

    }
```

```
    private void zeigeKarten() {
        TableLayout meinTableLayout=(TableLayout) findViewById(R.id.TableLayout1);
        int KartenNr = 0;
        for (int ZeilenNr = 0; ZeilenNr < Zeilenzahl; ZeilenNr++) {
            TableRow Zeile = new TableRow(this);
            for (int SpaltenNr = 0; SpaltenNr < Spaltenzahl; SpaltenNr++) {
                Karte eineKarte = meinMemorySpiel.getKarte(KartenNr);
                Zeile.addView(eineKarte, Kartengröße, Kartengröße);
                KartenNr++;
            }
            meinTableLayout.addView(Zeile);
        }
    }
    ...
}
```

Anstelle einer Array-Liste könnte man zum Speichern der Karten auch ein normales Array verwenden. Dies ist aber aufwendiger zu verwalten und außerdem müsste das Mischen der Karten in Java elementar programmiert werden. Wir benutzen hier dafür einfach die Anweisung *Collections.shuffle(meineKarten)* in der Methode *neuesSpiel()*. Nachstehend die Implementierung des Konstruktors und der elementaren Methoden der Klasse *MemorySpiel*. Wie man sieht werden die Kartenobjekte mit der Methode *abräumen()* nicht in der Array-Liste gelöscht, sondern nur auf *INVISIBLE* gesetzt. Damit sind sie nicht sichtbar und auch nicht anklickbar.

```
public class MemorySpiel {
    private final static int Verzögerung = 1500; // Millisekunden
    private static ArrayList<Karte> meineKarten = new ArrayList<Karte>();
    private static Karte ersteKarte;
    private static Karte zweiteKarte;
    private static int AnzahlVersuche;
    private static TextView tvAnzahl;

    public MemorySpiel(MainActivity memoryActivity, int Kartenzahl) {
        tvAnzahl = (TextView) memoryActivity.findViewById(R.id.tVAnzahl);
        String  Kartenname;
        for (int i = 1; i <= 2; i++) {
            Integer KartenNr = 1;
            for (int j = 1; j <= Kartenzahl / 2; j++) {
                if (KartenNr < 10)
                    Kartenname = "card0" + KartenNr.toString();
                else
                    Kartenname = "card" + KartenNr.toString();
                Karte k = new Karte(memoryActivity, Kartenname);
                meineKarten.add(k);
                KartenNr++;
            }
        }
        neuesSpiel();
    }
```

```
public void neuesSpiel(){
    ersteKarte = null;
    zweiteKarte = null;
    AnzahlVersuche = 0;
    tvAnzahl.setText(Integer.toString(AnzahlVersuche));
    Collections.shuffle(meineKarten); // in java.util
    for (int i = 0; i < meineKarten.size(); i++)  {
        meineKarten.get(i).zeigeRückseite();
        meineKarten.get(i).setVisibility(View.VISIBLE);
    }
}

public Karte getKarte(int KartenNr) {
    return meineKarten.get(KartenNr);
}

public static boolean zweiAufgedeckt() {
    return ersteKarte != null && zweiteKarte != null;
}

private static void abräumen() {
    ersteKarte.setVisibility(View.INVISIBLE);
    zweiteKarte.setVisibility(View.INVISIBLE);
}

private static void zudecken() {
    ersteKarte.zeigeRückseite();
    zweiteKarte.zeigeRückseite();
}
```

```
public static void prüfeZweiKarten(Karte eineKarte) {
    Handler handler = new Handler();
    if (ersteKarte == null)
        ersteKarte = eineKarte;
    else {
        zweiteKarte = eineKarte;
        handler.postDelayed(new Runnable() {
            public void run() {
                if (ersteKarte.getId() == zweiteKarte.getId())
                    abräumen();
                else
                    zudecken();
                ersteKarte = null;
                zweiteKarte = null;
                AnzahlVersuche++;
                tvAnzahl.setText(Integer.toString(AnzahlVersuche));
            }
        }, Verzögerung);
    }
}
```

Implementierung des Memory-Algorithmus

Wenn zwei Karten aufgedeckt sind, sollen diese nach einer gewissen Zeit (vgl. Attribut *Verzögerung*) entweder abgeräumt oder zugedeckt werden. Für die Zeitverzögerung benötigt man einen Handler mit der Methode *postDelayed* und ein *Runnable*-Objekt, dessen *run*-Methode wie folgt überschrieben wird. Mit *meinHandler.postDelayed(myRunnable, Verzögerung)* kann man dann ein Ereignis erzeugen, welches die Methode *run* genau nach der angegebenen Zeit ausführt. Wenn zwei Karten aufgedeckt sind, darf ein weiterer Click auf eine Karte nichts bewirken.

```
final static Runnable myRunnable = new Runnable() {

    public void run() {
        if (ersteKarte.getId() == zweiteKarte.getId())
            abräumen();
        else
            zudecken();
        ersteKarte = null;
        zweiteKarte = null;
        AnzahlVersuche++;
        tvAnzahl.setText(Integer.toString(AnzahlVersuche));
    }
};

public static void meineMemoryAktion(Karte eineKarte) {
    if (!zweiAufgedeckt()) {
        eineKarte.umdrehen();
        if (ersteKarte == null)
            ersteKarte = eineKarte;
        else {
            zweiteKarte = eineKarte;
            meinHandler.postDelayed(myRunnable, Verzögerung); }
    }
}
```

Falls dies für den Unterricht zu schwierig erscheint, kann man auf die Zeitverzögerung verzichten.
Die Karten werden erst abgeräumt bzw. umgedreht, wenn ein weiterer Click auf irgendeine Karte erfolgt.

```
public static void meineMemoryAktion(Karte eineKarte) {
    if (zweiAufgedeckt()) {
        if (ersteKarte.getId() == zweiteKarte.getId())
            abräumen();
        else
            zudecken();
        ersteKarte = null;
        zweiteKarte = null;
        AnzahlVersuche++;
        tvAnzahl.setText(Integer.toString(AnzahlVersuche));
    } else {
        eineKarte.umdrehen();
        if (ersteKarte == null)
            ersteKarte = eineKarte;
        else
            zweiteKarte = eineKarte;
    }
}
```

Anregungen zum Weiterarbeiten

Es gibt viele Erweiterungsmöglichkeiten, die u. a. im Buch von Uwe Post[5] beschrieben werden.
Im Folgenden ein paar Anregungen:

1. Entwicklung und Implementierung eines Algorithmus zum Mischen der Karten
2. Implementierung von Sound-Dateien
3. Implementierung eines Option-Menüs zur Einstellung der Verzögerungszeit, Anzahl der Karten und Kartengröße
4. Implementierung von Bestenlisten, eigene Highscores und Highscores anderer Spieler. Mit dem Google App Engine Projekt wird ein fertiger Highscore-Server zur Verfügung gestellt.

Studiendirektor Otto Wehrheim
Fachleiter für Informatik
Studienseminar Offenbach
E-Mail: otto.wehrheim@afl.hessen.de

[5] Post, Uwe: Android-Apps entwickeln

Literatur und Quellen

Arnold, J., & Plüss, A.: Spiele als Einstieg in das objektorientierte Programmieren.
Zeitschrift für Didaktik der Informatik , 2010, 1, S. 9.
http://www.zfdi.info/fileadmin/zfdi/Hefte/ZfDI_01_2010_Web.pdf

Felker, Donn: Android Apps für Dummies
Wiley VCH Verlag GmbH, 2011, 1. Aufl.

Hemig, Matthias: Einsatzszenarien von Mobiltelefonen im Informatikunterricht
Abschlussarbeit zur Erlangung des akademischen Grades eines Master of Education
Bergische Universität Wuppertal 2009.

Künneth, Thomas: Andoid 3, Apps entwickeln mit dem Android SDK
Galileo Press, 2011

Lau, Oliver: App Inventor, Android-Apps aus Puzzleteilen zusammenklicken
In c't, Jg 2010, H. 19, S. 136

Lindenau, Martin: Inwiefern kann die Programmierung von Android Smartphones in Java die Motivation der SchülerInnen in Bezug auf die objektorientierte Programmierung steigern?
Pädagogische Prüfungsarbeit im Rahmen der 2. Staatsprüfung am Studienseminar Offenbach, 2010

Linke, Andreas: Programmieren für Android 3 und 4
In: c't extra Andoid, 2012

Linke, Andreas: Appéritif, Einführung in die Entwicklung von Android-Apps, Teil 1
In: c't, Jg. 2010, H. 22, S. 188

Linke, Andreas: A la carte, Einführung in die Entwicklung von Android-Apps, Teil 2
In: c't, Jg. 2010, H. 24, S. 194

Linke, Andreas: Gut geschüttelt, Einführung in die Entwicklung von Android-Apps, Teil 3
In: c't, Jg. 2011, H. 1, S. 172

Post, Uwe: Android-Apps entwickeln, Ideal für Programmiereinsteiger geeignet
Galileo Computing, 1. Aufl., 2012

Ullenboom, Christian: Sonnenfinsternis in der Schule
In: LOG IN, Jg. 2007, H.145

Wehrheim, Otto: Eclipse – Ein universelles Entwicklungssystem für den Informatikunterricht
In: LOG-IN, Jg. 2010, H. 163/164, S. 122

Wehrheim, Otto: Objektorientiertes Modellieren mit "DELPHI" am Beispiel des Memory-Spiels
In: LOG-IN, Jg. 2003, H. 125, S. 30-35

Zechner, Mario: Spieleentwicklung
http://www.androidpit.de/de/android/wiki/view/Spieleentwicklung_101#toc56

Android Wiki:
http://www.androidpit.de/de/android/wiki/special/allPages

Verwendete Icons:
http://www.iconeasy.com

200 Videos für Android Application Development, englischsprachig
http://thenewboston.org/list.php?cat=6
http://www.youtube.com/playlist?list=PL34F010EEF9D45FB8

Sensor-Simulator: http://www.openintents.org/en/node/23/

Alle Internetquellen wurden zuletzt im August 2012 geprüft.