Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Seminararbeit

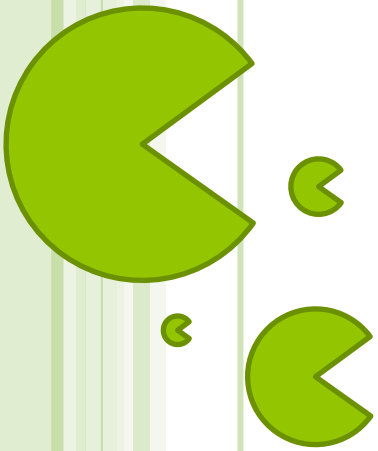
Tobias Fritz

Android Grafik API

Eine Einführung in die 2D und 3D Grafikprogrammierung mit Android.

Betreuer: **Prof. Dr. Henning**

Datum: **06.06.2011**



Inhaltsverzeichnis

Abbildungsverzeichnis

Vorwort

Ende 2010 war jedes dritte verkaufte Mobiltelefon ist ein Smartphone [1] und innerhalb der nächsten zwei Jahren, wird jeder vierte Deutsche eins besitzen [2]. Davon werden im Jahr 2011 fast vierzig Prozent mit dem Betriebssystem Android ausgeliefert. Täglich werden 350.000 neue Android Geräte in Market registriert, so dass es zusammen bis Ende des Jahres 180 Millionen Geräte sind [3]. Die Konkurrenten Symbian, iOS und Blackberry sind mit 20, 15 und 16 Prozent deutlich weniger verbreitet. Windows Phone 7 kommt selbst zusammen mit dem veralteten Windows Mobile gerade mal auf 5,5 Prozent [4].

Mit diesem rasanten Wachstum an leistungsfähigen mobilen Endgeräten, haben sich auch die Möglichkeiten für Entwickler verbessert. Sie können nun auch leistungsfordernde Programme mit anspruchsvollen Multimedia Inhalten für breite Massen entwickeln. Fortschritte in den Grafischen Möglichkeiten, werden wie im Desktop Bereich auch, durch die Spieleindustrie vorangetrieben. Fast zwanzig Prozent der Smartphone Käufer geben als Anschaffungsgrund für ein Handy den Entertainment Bereich an und in drei Jahren sollen 43 Prozent der Spiele auf dem Handy gespielt werden [5]. Darum ist davon auszugehen dass sich neben der Rechenleistung vor allem die Grafikleistung bei Smartphones weiter verbessern wird. Spiele Entwickler aber auch alle anderen, die Multimedia Inhalte für mobile Endgeräte entwickeln wollen, sollten sich daher unbedingt mit den Möglichkeiten der Android Grafik API vertraut machen.

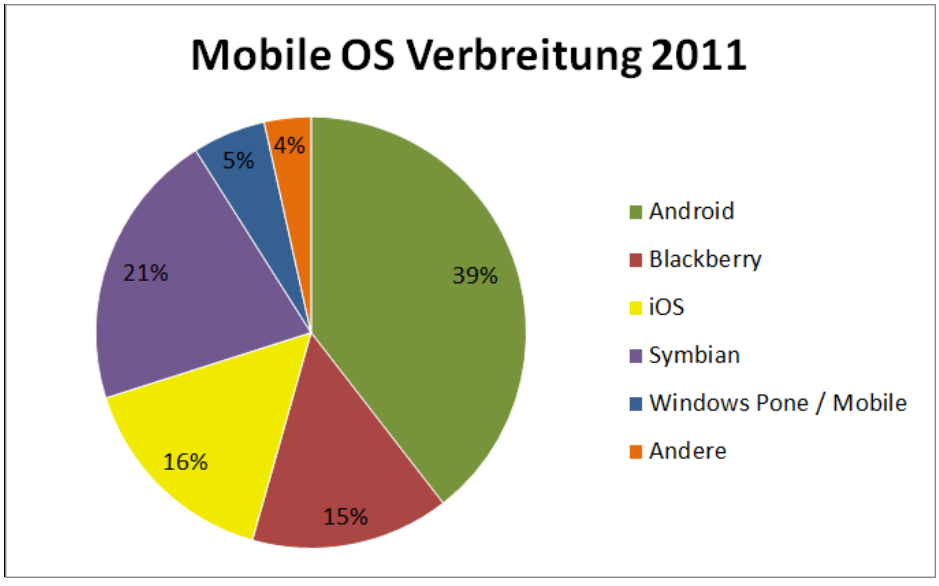


Abbildung 1 Mobile OS Verbreitung 2011 © 5. Mai 2011 Marktforschungsinstituts Gartner [4]

Android SDK einrichten

Um Anwendungen für Android zu programmieren, benötigen wir als erstes das Android SDK. Darin sind unter anderen der Compiler, Bibliotheken für die unterschiedlichen Android Versionen und der Emulator enthalten. Mit ihm lassen sich Programme testen, ohne sie auf ein physisches übertragen zu müssen. Im SDK nicht enthalten ist die Entwicklungsumgebung. Diese muss separat installiert und mit den Android Development Tools (*ADT*) ausgestattet werden.

Vorbereitung

Für die Android Entwicklung werden Windows, Linux und Mac unterstützt. Spezielle Systemvoraussetzungen sind nicht gegeben. Lediglich das Java SE Development Kit (JDK) müssen wir vor dem Einrichten des SDK installiert haben.

Das JDK kann unter folgender Quelle für alle genannten Plattformen heruntergeladen werden:

http://www.oracle.com/technetwork/java/javase/downloads/index.html [6]

Wichtig beim Download ist, dass man nicht das kleinere JRE herunterlädt. Dieses reicht nicht aus und muss auch nicht gesondert installiert werden, da es im JDK bereits enthalten ist.

Installation des Android SDK

Das Android SDK steht unter folgender Quelle zum Download bereit:

http://developer.android.com/sdk/index.html [7]

Der Download ist jedoch nur eine Minimalinstallation. Mit dem *SDK Manager* der sich im Hauptverzeichnis der Installation befindet, werden alle weiteren Bibliotheken und Tools nachinstalliert. Mit ihm werden später auch die *Virtual devices*[1] verwaltet, und das SDK durch Updates auf dem neuesten Stand gehalten.

Auswahl des Installationsverzeichnisses

Prinzipiell kann das Android SDK an in ein beliebiges Verzeichnis installiert werden. Der SDK Manager erkennt aus welchem Verzeichnis er gestartet wurde und installiert Updates in die entsprechenden Unterordner.

Achtung: Unter Windows Vista und Windows 7 empfiehlt es sich, das SDK nicht in das vorgeschlagene Strandartverzeichnis unter Programme zu installieren, da dieses durch die Windows Benutzerkontensteuerung geschützt wird. Das führt beispielweise beim Laden der im SDK vorhanden Beispielprojekte zu Problemen, da Eclipse ohne Administrator Rechte ausgeführt wird und ausschließlich nicht schreibend auf die *.project* Konfigurationsdatei zugreifen kann. Für diese Installation wurde folgendes SDK Verzeichnis gewählt: *C:\Android\android-sdk*. Folgend wird dieses Verzeichnis durch die Schreibweise *$SDK_ROOT* abgekürzt.

[1] Android System das ein Smartphone simuliert, indem es im Emulator virtualisiert wird.

Nach der Installation können die fehlenden Pakete mit Hilfe des *SDK Manager*s installiert werden. Dieser ist unter *$SDK_ROOT/SDK Manager.exe* zu finden.

Unterschiedliche Android Versionen

Jede Version die als Update auf ein Android Gerät ausgeliefert wird, ist mit einer Plattform Version und einem API Level gekennzeichnet. Alle Versionen bauen auf einander auf und sind zu älteren abwärtskompatibel. Neue Versionen beinhalten neben Sicherheit Updates und Detailverbesserungen oft auch neue Funktionen. So kam beispielweise in Version 2.2 die Anbindung an Microsoft Exchange Server und mit 2.3 die Schnittstelle zur Nutzung von *Near Field Communication*[2] dazu. Als Entwickler sollte man sich daher überlegen welche Funktionen man Benötigt und sich unter Berücksichtigung dessen Verbreitung für ein API Level entscheiden.

Verbreitung der API Versionen

Die Smartphone Hersteller verteilen Versionsupdates für Android in voneinander unterschiedlichen Update Zyklen. Oft werden ältere Modelle nur sehr langsam, oder auch überhaupt nicht mehr mit neuen Versionen versorgt. Daher sind Android Versionen auf unterschiedlichen Modellen verschiedener Hersteller fragmentiert. Wie stark eine SDK verbreitet ist, lässt sich gut anhand Googles Statistik der Market Zugriffe ablesen (Abbildung 2, Abbildung 3).

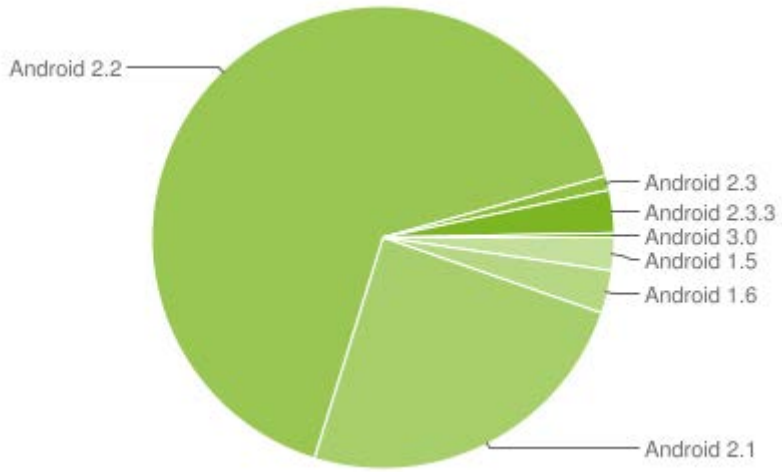


Abbildung 2 Android Versionsverbreitung, Auswertung von 14 Tagen im Mai 2011[3]

[2] Die Near Field Communication, NFC, ist ein *Übertragungsstandard* zum kontaktlosen *Austausch von Daten* über kurze Strecken.
[3] *http://developer.android.com/resources/dashboard/platform-versions.html* [6]

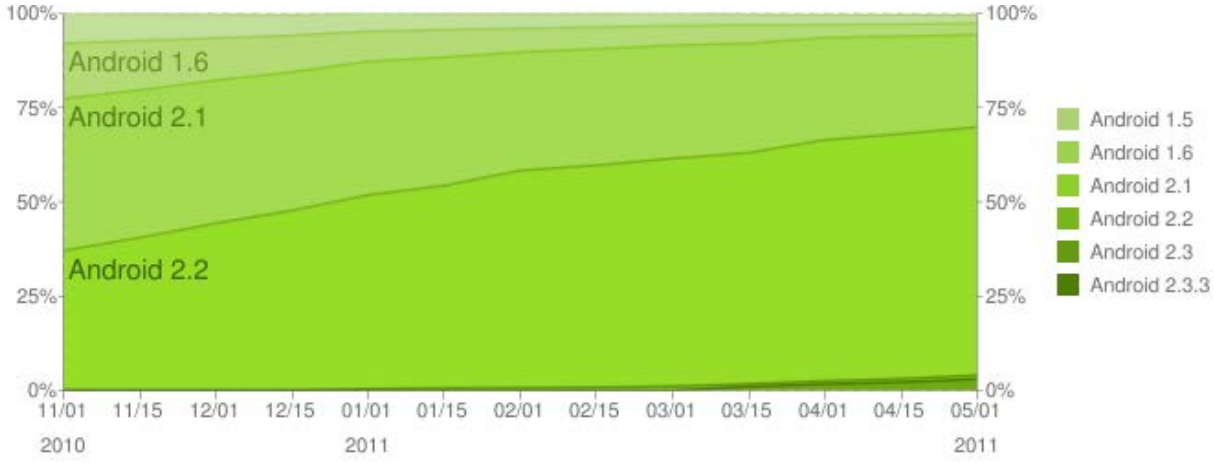


Abbildung 3 Android Versionsverbreitung, Auswertung der letzten 6 Monate ab Mai 2011[3]

Deutlich zu erkennen ist der Trend der Version 2.2 mit stetigem Wachstum auf fast 75% der Nutzer kommt. Wegen dieser hohen Verbreitung und den Neuerungen im Bereich Grafik wie beispielweise einer neuen Image Format Klasse und eine OpenGL ES 2.0 API[4], wurde für diese Ausarbeitung die Android Version 2.2 mit dem API Level 8 gewählt.

Nachladen der Plattform Pakete

Beim ersten Start des *SDK Manager*s poppt ein Fenster „*Choose Packages to Install*“ auf. Die darin vorgeschlagen Standartauswahl an zu installierenden Pakete kann getrost beibehalten werden. Daran sind dann auch gleich alle derzeit verfügbaren Android Versionen auf einmal dabei. Um die in dieser Ausarbeitung gezeigten Beispiele nachvollziehen zu können, reicht aber auch folgende Minimalkonfiguration[5]:

- Android SDK Platform-tools
- Android SDK Docs for Android API 8
- Android SDK Platform 2.2, API 8
- Android SDK Samples for Android API 8

Ein Klick auf *Install* lädt alle mit einem grünen Haken markierten Pakete herunter und installiert sie. Das Nachladen der Pakete ist damit abgeschlossen.

Plattform Version	API Level
Android 3.1	12
Android 3.0	11
Android 2.3.4	10
Android 2.3.3	
Android 2.3	9
Android 2.2	8
Android 2.1	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

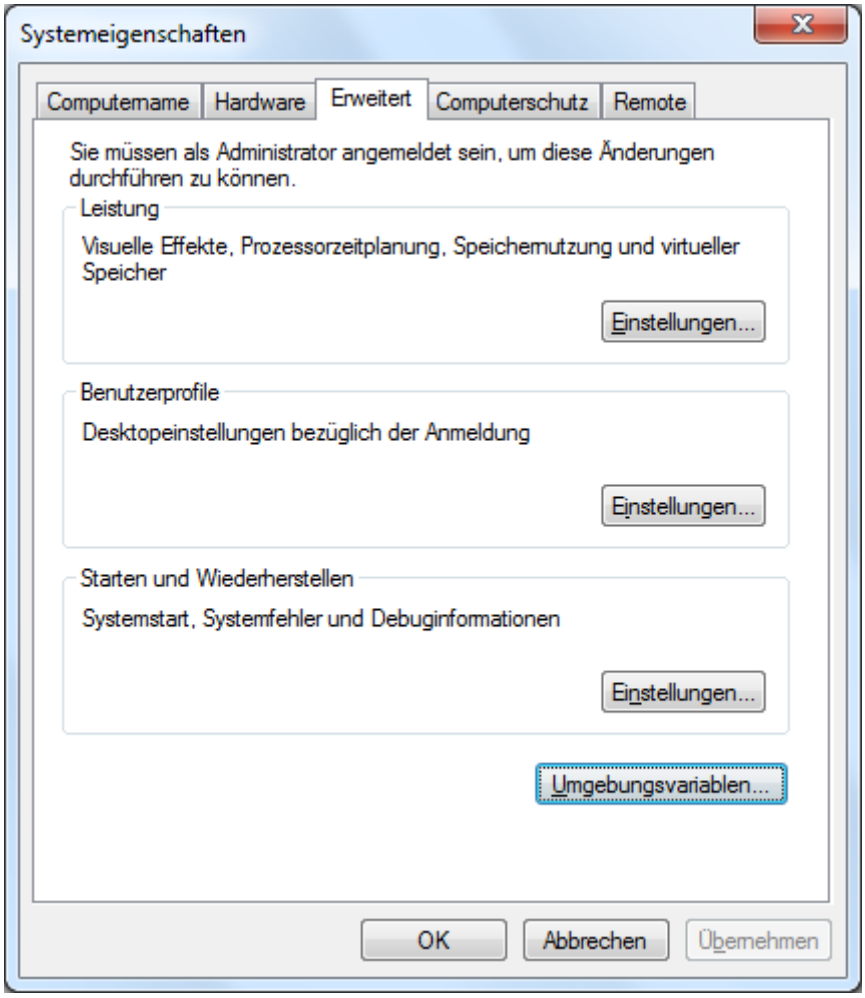
Tabelle 1 API Level der Plattformen[6]

Platform Tools

Das Android SDK ist nun vollständig installiert. Viele darin enthaltene Tools werden über die Konsole bedient. Um die Arbeit mit der Konsole zu erleichtern lohnt es sich den Pfad *$SDK_ROOT \platform-tools*
als Systemvariable zu setzen:

[4] http://developer.android.com/sdk/android-2.2.html [6]
[5] Warum die SDK Platform 2.2, mit dem API Level 8 ausgewählt wurde wird unter genauer xyz beleuchtet.
[6] *http://developer.android.com/guide/appendix/api-levels.html*

1. Navigiere dazu wie folgt zu den Systemeigenschaften:
   Systemsteuerung\System und Sicherheit\System\Erweiterte Systemeinstellungen
   (oder über Windows Suche „*Systemumgebungsvariablen bearbeiten*“)
2. Wähle anschließend im Fenster *Systemeigenschaften* den Reiter *Erweitert* aus und klicke auf *Umgebungsvariablen…* (Abbiludng 4 Systemeigenschaften).



Abbiludng 4 Systemeigenschaften

3. Wähle im Fenster *Umgebungsvariablen* aus der Gruppe *Systemvariablen* die Tabellenzeile mit der Variable *Path* aus und klicke *bearbeiten* (Abbildung 5 Umgebungsvariablen).

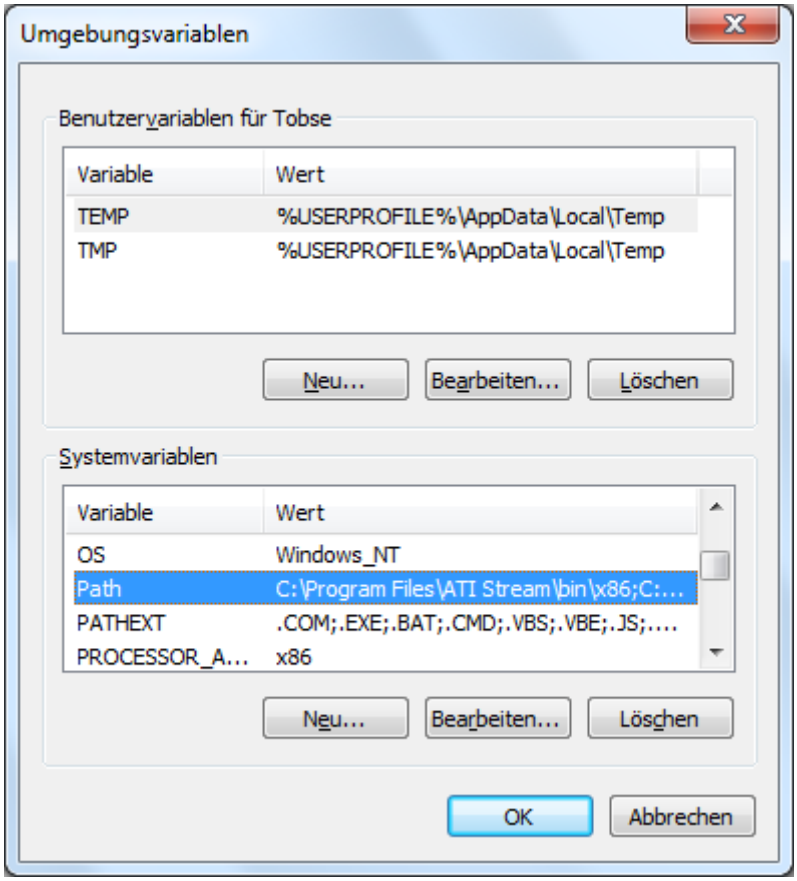
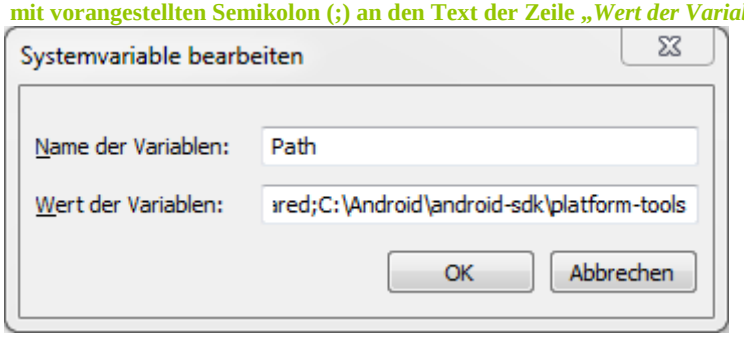


Abbildung 5 Umgebungsvariablen

Im Fenster Umgebungsvariablen kann nun der Pfad
$SDK_ROOT \platform-tools
mit vorangestellten Semikolon (;) an den Text der Zeile *„Wert der Variablen“* angehängt werden (



4. Abbildung 6 Systemvariable bearbeiten).

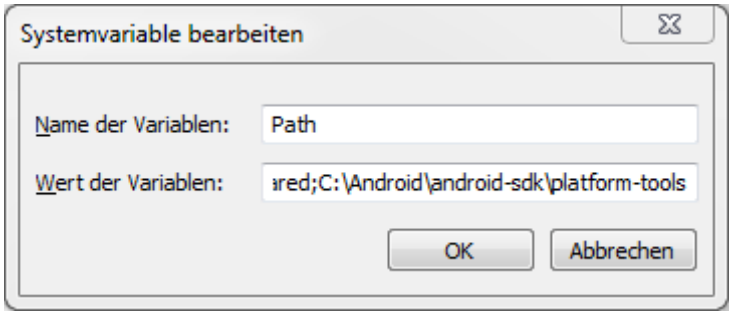


Abbildung 6 Systemvariable bearbeiten

Android Development Tools

Die ADT können in eine beliebige aktuelle[7] Eclipse Installation eingebunden werden. Bei einer Neuinstallation empfiehlt sich die Version *Eclipse IDE for Java Developers*. Diese kann unter folgendem Link heruntergeladen werden:

http://eclipse.org/downloads/

Installation des ADT

Eclipse bedarf keiner Installation, sondern ist schon nach dem entpacken in ein beliebiges Verzeichnis betriebsbereit. Die ADT werden über den Eclipse Plugin Mechanismus wie folg installiert[8]:

1. Starte Eclipse
2. Wähle unter ***Help / Install New Software....***
3. Klicke auf ***Add***, in der rechten oberen Ecke.
4. Gib im erscheinenden *Add Repository* Dialog, folgende Date nein:

   Name: ***ADT Plugin***

   Location: ***https://dl-ssl.google.com/android/eclipse/***

   Hinweis: Wenn beim beziehen des Plugins Probleme auftreten, probiere “http” anstelle von “https” in der Location URL.

   Bestätige mit **OK**.

5. Markiere die Checkbox vor ***Developer Tools*** und klicke auf ***Next***.
6. Im nächsten Dialog wird eine Liste mit Tools die herunter geladen werden angezeigt.

   Bestätige auch diese mit einem Klick auf ***Next***.
7. Lese und bestätige das Lizenz Abkommen mit einem Klick auf ***Finish***.
8. Starte nach der erfolgreichen Installation Eclipse neu.
9. Wähle in Eclipse unter *Window / Preferences / Android* das *$SDK_ROOT* aus.

[7] Die aktuelle ADT Version ist nur noch mit Eclipse Galileo oder höher (Ganymede) kompatibel.
[8] *http://developer.android.com/sdk/eclipse-adt.html#installing* [6]

Ein Android Projekt Einrichten

Das Anlegen eines Android Projekts funktioniert fast genauso so einfach wie bei einem normalen Java Projekt. Wähle in Eclipse *File / New / Other…* und wähle aus dem Verzeichnis *Android* das *Android Project* aus. Im folgenden *New Android Project* Fenster wird das Projekt konfiguriert. Folgendes ist einzustellen (Abbildung 7):

Project Name	Name des Projekts. Wird als Projektverzeichnis eingesetzt.
Build Target	Die Android Version 2.2
Application Name	Anwendungsname unter dem die Anwendung nach einer Installation auch auf dem Android Gerät zu finden ist.
Packe Pfad	Package Name.
Create Activity	Name der Activty die später gestartet werden kann.
Min SDK Version	Einstellige Zahl, Minimales API Level für dieses Projekt (8).

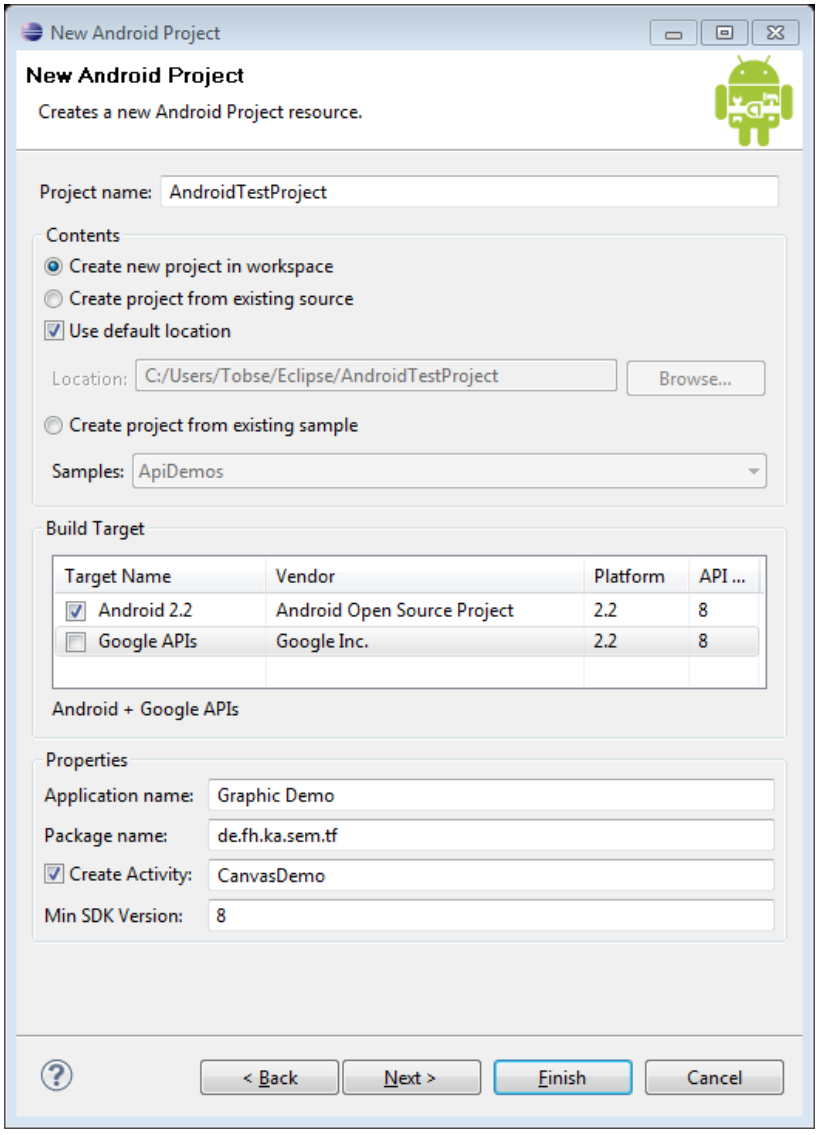


Abbildung 7 Wizard zum Anlegen eines neuen Android Projekts

Einrichten der JavaDoc

JavaDoc ist eine standardisierte Dokumentation im HTML Format, die zum schnellen nachschlagen in Eclipse eingebunden werden kann (Abbildung 8). Leider muss Eclipse dafür auf die Sprünge geholfen werden. Obwohl der Pfad zur Javadoc der *android.jar* automatisch korrekt eingestellt wird, muss er manuell neu ausgewählt werden, damit der Code Editor ihn ebenfalls berücksichtigt. Dies geschieht folgendermaßen:

1. Das Projekt im *Package Explorer* mit Linksklick auswählen.
2. Unter *Project / Properties / Java Build Path* den Reiter *Libraries* auswählen.
3. Im Baum erst dem Knoten *Android 2.2* und anschließend *android.jar* aufklappen.
4. Den Eintrag *Javadoc location*: doppelklicken und folgenden Pfad übernehmen: *file:/$SDK_ROOT/docs/reference/*

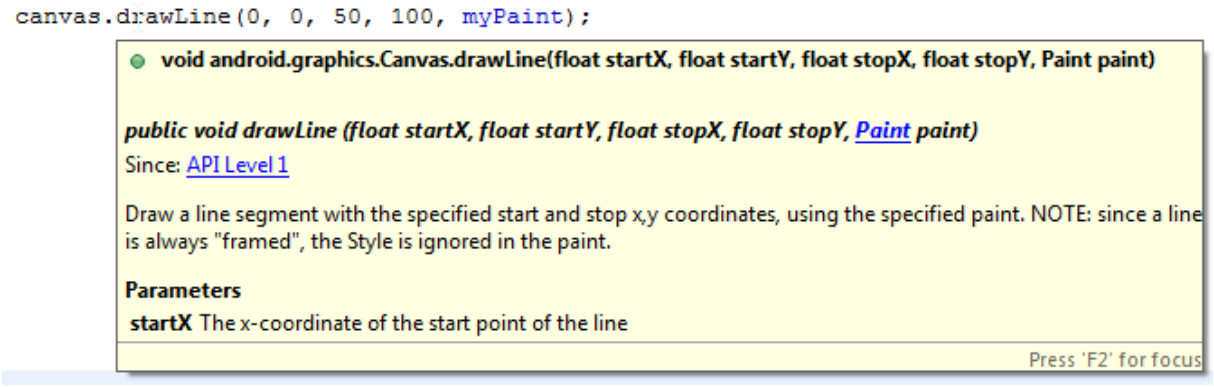


Abbildung 8 Javadoc Einblendung beim Überfahren einer Methode

Source Code beziehen und einbinden

Bei der Entwicklung ist es oft sehr nützlich neben der JavaDoc auch den Source Code einer API in Eclipse eingebunden zu haben. Der Source Code der in Android Projekten Eingebundenen *android.jar* wird leider nicht mit dem SDK Paketen mitgeliefert. Er muss separat über das aktuelle *git*[9] Repository herunter geladen werden. Der folgende Link stellt die neuste Code Version in einem tar.gz Archiv bereit:

http://git.source.android.com/?p=platform/frameworks/base.git;a=snapshot;h=HEAD;sf=tgz

Dieses rund 130MB große Archiv, kann mit einem Zip-Programm wie *7-Zip*[10] entpackt werden. Darin enthalten ist als Hauptverzeichnis der Ordner mit der Bezeichnung *base-HEAD* gefolgt von einer Versionscodierung. Dessen Inhalt müssen wir nun wieder mit Hilfe eines Zip Programms in ein neues *zip*-Archiv packen. Danach haben wir eine Zip Date in der unter anderem direkt, ohne Unterordner, die Datei Android.mk liegt. Diese Zip kann nun mit der *android.jar* verknüpft werden:

1. Das Projekt im *Package Explorer* mit Linksklick auswählen.
2. Unter *Project / Properties / Java Build Path* den Reiter *Libraries* auswählen.
3. Im Baum erst dem Knoten *Android 2.2* und anschließend *android.jar* aufklappen.

[9] Git ist ein freies, verteiltes open source Versionierungsystem *http://git-scm.com*.
[10] *http://www.7-zip.org*

4. Den Eintrag *Source attachment*: doppelklicken und nach einem Klick auf *External File* das bereits erstellte zip-Archiv auswählen.

Eine Virtuelles Android Gerät einrichten

Eine Android Anwendung wird entweder auf einem physischen, oder auf einem virtuellen Gerät installiert und ausgeführt. Eines von beiden muss daher bevor eine Android Anwendung getestet werden kann, eingerichtet werden. Bei physischen Geräten muss dafür ein Hersteller spezifischer Treiber installiert und das Smartphone in den debug Modus versetzt werden. Da sich dieses Vorgehen je nach Smartphone unterscheidet, wird folgend der allgemeine Weg mit einer Virtual Android Device (AVD) beschrieben. Diese ist auch das Mittel der Wahl, wenn man selbst keine Android Smartphone besitzt.

Eine AVD wird im *Android SDK and AVD Manger* eingerichtet. Im Menüpunkt *Virtual devices* ist dazu *New…* zu wählen. Anschließend kann für die AVD ein Name gewählt und als Target unsere Android Version 2.2 API Level 8 gewählt werden (Abbildung 9). Unter Size geben wir der AVD noch einen Applikationsspeicher von 200 MB an die Hand. Ein Klick auf Create AVD schließt den Wizard ab. Zum Testen kann die gerade erstelle AVD mit *Start…* ausgeführt werden (Abbildung 10).

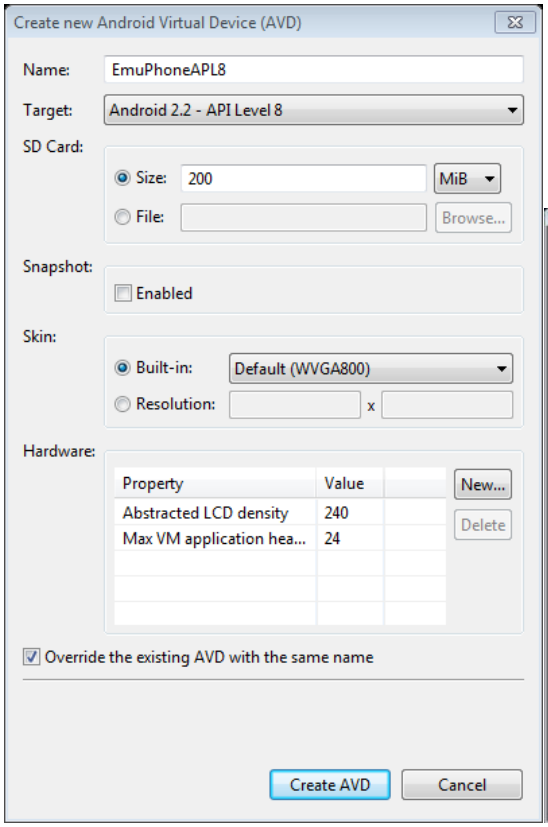


Abbildung 9 Erstellen einer AVD

Abbildung 10 Eine AVD im Emulator

Zeichenoperationen mit dem Canvas

Im Folgenden Kapitel werden die Zeichen Operationen von Text und Primitiven vorgestellt. Für das zweidimensionale Zeichnen bringt das SDK das Canvas mit -eine begrenzte Zeichenfläche. Das Canvas wird innerhalb einer *View* bereitgestellt, der Oberklasse aller GUI Elemente. Die *View* wiederum wird einer *Activity* zugeordnet. Eine Android Anwendung befindet sich stets in einer *Activity*, einer Seite auf dem Bildschirm. Dieser Aufbau ist nochmal in Abbildung 11 dargestellt.

Abbildung 11 Hierarchie einer Anwendung

Primitive Zeichnen

Für das unser erstes Programm nehmen wir das im Abschnitt *Ein Android Projekt Einrichten* erstelle Projekt. Dort legen wir neben der Klasse *CanvasDemo* eine *CanvasView* an, in der unsere Zeichenoperationen ausgeführt werden sollen. Diese lassen wir von *View* erben. Bevor wir zeichnen können, benötigen wir eine *Paint* Instanz, die die Style und Farbinformationen zum Zeichnen beinhaltet. Danach kann die Methode `onDraw(Canvas canvas)` überschrieben werden, um in ihrem übergebenen *Canvas* zu zeichnen:
`canvas.drawCircle(100.0f, 100.0f, 50.0f, myPaint);`
zeichnet einen Kreis dessen Mittelpunkt an der Position 100px , 100px liegt mit dem Radius 50px. Der Ursprung des Kartetischen Koordinatensystems liegt dabei in der linken oberen Ecke und wächst positiv nach unten (Abbildung 12).

Die Klasse CanvasDemo sieht für den ersten Hello World Test dann wie folg aus:

```
package com.example.android.apis;

import android.content.Context;
import android.graphics.*;
import android.view.View;

public class CanvasView extends View {

	private Paint myPaint;

	public CanvasView(Context context) {
		super(context);
		setPaintAttributes();
	}

	private void setPaintAttributes() {
		myPaint = new Paint();
```

```
            myPaint.setAntiAlias(true);
            myPaint.setStyle(Paint.Style.FILL);
            myPaint.setColor(Color.GREEN);
      }

      @Override
      protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.WHITE);
            canvas.drawCircle(100.0f, 200.0f, 50.0f, myPaint);
            canvas.drawText("Hello World", 10, 25, myPaint);
      }

}
```

Durch den Aufruf `drawColor(Color.WHITE)` wird der gesamte View Bereich mit weiß übermalt. Das `setStyle(Paint.Style.FILL)` legt fest, dass Formen ausgefüllt und nicht nur deren Rahmen gezeichnet werdem. Damit unsere Canvas View dargestellt wird, müssen wir sie noch mit der Standard Activity *CanvasDemo* verknüpfen. Dazu wird in der Klasse *CanvasDemo* in der automatisch generierten `onCreate` Methode der Aufruf

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

durch

```
setContentView(new CanvasView(this));
```

ersetzt. Zum Ausführen wird im *Package Explorer* das Projekt *AndroidTestProject* markiert und unter *Run / Run As / Android Application* gestartet. Die AVD wird dabei, falls noch nicht geschehen, automatisch gestartet. Da das Hochfahren der AVD lange dauet, empfiehlt es sich sie so lange wie möglich am Laufen zu lassen. Neustarts sind zum Ausführen neuer Programmversionen nicht nötig.

Weitere Zeichenoperationen

In der Javadoc des Canvas sind weitere Zeichenoperationen zu finden. Die wichtigsten sind folgend aufgelistet[11]:

- **`drawArc`**`(RectF oval, float startAngle, float angle, boolean useCenter, Paint paint)`
  - Zeichnet einen Bogen, der skaliert wird, bis er in das angegebene Oval passt.
- **`drawBitmap`**`(Bitmap bitmap, float left, float top, Paint paint)`
  - Zeichnet das angegebene Bitmap ab der oberen, linken Ecke von (x,y).
- **`drawCircle`**`(float cx, float cy, float radius, Paint paint)`
  - Zeichnet einen Kreis mit der Mitte in (cx, cy).
- **`drawColor`**`(int color)`
  - Füllt das gesamte Canvas mit der gegeben Farbe.
- **`drawLine`**`(float startX, float startY, float stopX, float stopY, Paint paint)`
  - Zeichnet eine Line zwischen den Start und Stopp Koordinaten.
- **`drawPicture`**`(Picture picture)`
  - Zeichnet das Bild ohne den Canvas State dabei zu verändern.
- **`drawPoint`**`(float x, float y, Paint paint)`
  - Zeichnet an den gegebenen Koordinatoren einen einzelnen Punkt.
- **`drawRect`**`(float left, float top, float right, float bottom, Paint paint)`
  - Zeichnet das angegebene Rechteck.
- **`drawRoundRect`**`(RectF rect, float rx, float ry, Paint paint)`

[11] *http://developer.android.com/reference/android/graphics/Canvas.html* -Frei übersetzt. [6]

- Zeichnet das angegebene Rechteck mit runden Ecken.

- **`drawText`**`(String text, float x, float y, Paint paint)`
  - Zeichnet den Text mit dem Ursprung in (x,y)

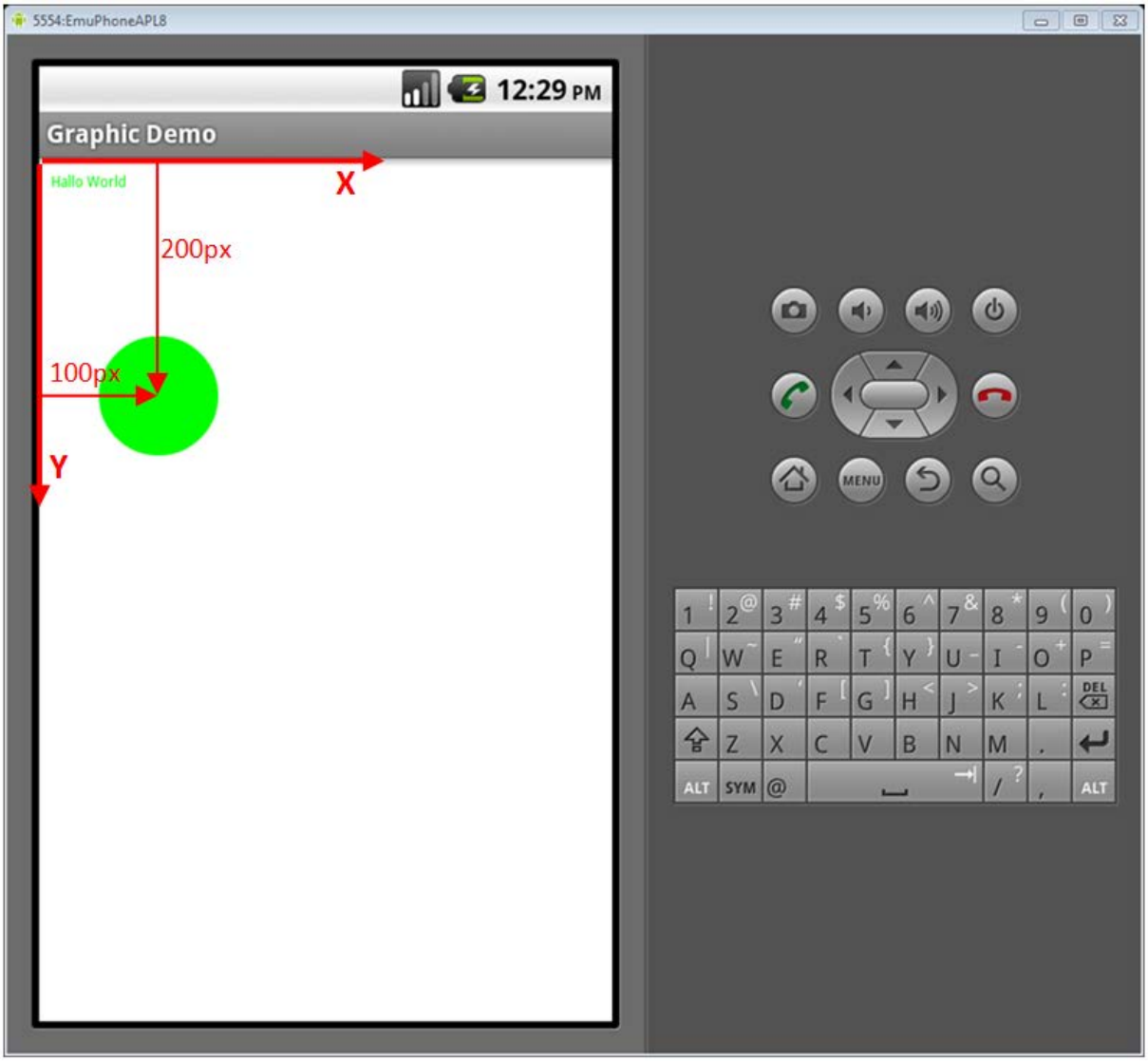


Abbildung 12 Erstes Zeichenoperationen im Canvas

Im Fehlerfall „Re-installation failed“

Im *Console* Fenster von Eclipse kann das starten der Anwendung nachvollzogen werden. Das Log zum Starten der *CanvasDemo* könnte in etwa so aussehen:

```
[..]Performing com.example.android.apis.GraphicDemo activity launch
[..]Automatic Target Mode: Preferred AVD 'EmuPhoneAPL8' is available on emulator 'emulator-55'
[..]WARNING: Application does not specify an API level requirement!
[..]Device API version is 8 (Android 2.2)
[..]Uploading AndroidProject.apk onto device 'emulator-55'
[..]Installing AndroidProject.apk...
[..]Success!
[..]Starting activity com.example.android.apis.GraphicDemo on device emulator-55
[..]ActivityManager: Starting: Intent { act=android.intent.action.MAIN
    cat=[android.intent.category.LAUNCHER] cmp=com.example.android.apis/.GraphicDemo }
```

Abbildung 13 Auszug aus der Console beim Starten einer Android Anwendung

Hier kommt man Startproblemen schnell auf die Schliche. Auffallend ist die Warnung wegen des fehlenden API Levels. Das Programm wird in diesem Fall aber trotzdem fehlerfrei ausgeführt. Zum beseitigen dieser Warnung muss in der AndroidManifest.xml im Projekt hinter das *application* Tag der Eintrag

```
<uses-sdk android:minSdkVersion="8" />
```

eingefügt werden.

Bei der Ausgabe

```
Re-installation failed due to different application signatures.
```

wird das Programm nicht mehr in dem Emulator hoch geladen. Was zu tun ist, wird dann zwei Zeilen weiter auch beschrieben:

```
Please execute 'adb uninstall com.example.android.apis' in a shell.
```

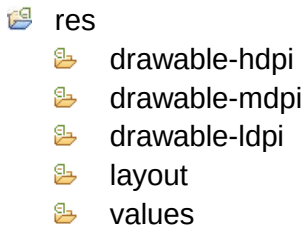
Das bedeutet, dass man selbst in die Windows Konsole wechseln und den genannten Befehl ausführen muss. An dieser Stelle wünscht man sich eine etwas selbstständigere Entwicklungsumgebung.

Grafiken zeichnen

Auch Bilder können geladen und im Canvas gezeichnet werden. Dabei werden unter anderem die Formate PNG, JPG und GIF unterstützt. Diese können an einer beliebigen Stelle im Programm mit Hilfe der *BitmapFactory* geladen werden. Beim laden stehen zwei verschiedene Methoden zu Auswahl die sich darin unterscheiden aus welcher Quelle die Bilder geladen werden.

Ressourcen Laden

Grafiken aber auch Layout XML-Dateien, oder Property-Dateien werden in Android als Ressourcen bezeichnet. Diese befinden sich in einer vorgegeben Ordnerstruktur:



Die drei Abkürzungen hinter *drawable* stehen für High, Medium und Low DPI[12]. Darin können Grafiken in unterschiedlichen Auflösungen gespeichert werden, die dann automatisch abhängig vom Endgerät ausgewählt werden.

Legt man in einem der *drawable* Ordner eine Grafik ab, wird von Eclipse automatisch im Source Verzeichnis *gen* im Package *com.example.android.apis* in der Klasse *R* eine Konstante mit dem Dateinamen dieser Grafik angelegt. Manuell darf man diese Datei nicht modifizieren. Sie dient dazu, Ressourcen im Code zu referenzieren. Der Zugriff geschiegt wie folgt:

```
Bitmap myBitmap = BitmapFactory.decodeResource(getResources(),
                  R.drawable.android_robot);
```

[12] dots per inch, englisch für „Punkte pro *Zoll*“

Dass vor diesem Aufruf kein Objekt steht liegt daran, dass dies eine Methode der Vaterklasse *View* ist. Über die Konstante *android_robot* der Klasse *R* wird dann die eigentliche Grafik referenziert. Interessant dabei ist, dass bei diesem Zugriff die Dateiendung keine Rolle mehr spielt. Dateien die sich ausschließlich in im Dateiformat unterscheiden sind darum als Ressourcen nicht zugelassen.

Alle Dateien die sich innerhalb eines Projekts und dadurch später auch in einer Android Paket Datei (akp) befinden, müssen über diesen Mechanismus geladen werden.

Dateien Laden

Für Dateien die auf einem Dateisystem, wie Beispiel auf einer SD-Karte liegen, geschieht der Zugriff nicht über den Ressource Mechanismus, sondern wie gewohnt über Pfadangaben. Dies ermöglicht eine andere Methode der *BitmapFactory*:

```
Bitmap myBitmap = BitmapFactory.decodeFile("/sdcard/android_robot.png");
```

Bitmaps darstellen

Das geladene Bitmap kann über die Zeichenmethoden des Canvas ausgegeben werden. Zum setzen von Größe und Position wird der Zeichen Methode eine Transformationsmatrix mitgegeben. Unsere bereits zum Zeichnen von Primitiven genutzte `onDraw` Methode sieht dann beim Zeichen eines Bildes folgendermaßen aus:

```
@Override
protected void onDraw(Canvas canvas) {
	canvas.drawColor(Color.WHITE);
	canvas.drawCircle(100.0f, 200.0f, 50.0f, myPaint);
	canvas.drawText("Hello World", 10, 25, myPaint);

	Bitmap myBitmap = BitmapFactory.decodeResource(getResources(),
				R.drawable.android_robot);
	if(myBitmap != null ){
		Matrix matrix = new Matrix();
		matrix.preTranslate(50, 50);
		matrix.preScale(0.5f, 0.5f);
		canvas.drawBitmap(myBitmap, matrix, myPaint);
	}else{
		Log.w("CanvasView", "Coulnd't load android_robot.png");
	}
}
```

Die Matrix verschiebt das Bild um jeweils 50px nach rechts unten und skaliert es auf die Hälfte. Die Methode `drawBitmap` wurde erst mit dem API Level 9 (Version 2.4) so überladen, dass das Bild an einer bestimmten Position auch ohne Matrix gezeichnet werden kann.

3D Grafik mit Open ES

Einen ganz anderen Weg zum Zeichnen von Grafiken in Android, kann mit Hilfe der *GLSurfaceView* gegangen werden. Mit ihr ist es möglich mit OpenGL ES Grafische Ausgaben zu programmieren. OpenGL ES ist der Standard für plattformübergreifende

beschleunigte 3D Grafiken in Embedded Systemen[13]. Es stellt eine Teilmenge der Funktionalität von OpenGL bereit. Ab der Android Version 2.2 wird auch OpenGL ES in der Version 2.0 unterstützt. Die Schnittstellen sind dabei aber weiterhin abwärts kompatibel zu älteren OpenGL ES Versionen.

Einen Bereich zum Ausführen von OpenGL Befehlen bekommt man folgendermaßen: Als ersten brauchen wir wie bei der *CanvasDemo auch*, eine Startklasse die von *Activity* erbt –In unserem Fall die Klasse *OpenGLDemo.* In dieser wird in der Methode `protected void onCreate(Bundle savedInstanceState)` mit dem Aufruf `setContentView(View view)`eine *GLSurfaceView* gesetzt. Das eigentliche Zeichnen bzw. Rendern, erledigt ein *Renderer*, der der *GLSurfaceView* zugeordnet wird. Der Aufbau ist mit Hilfe eines Klassendiagramms in Abbildung 14 noch einmal veranschaulicht.

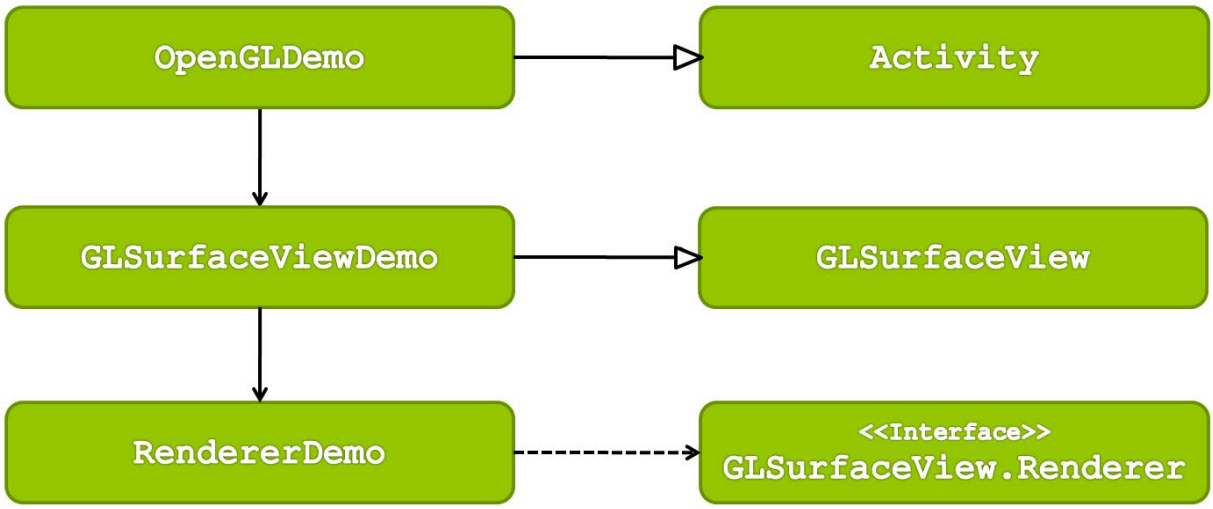


Abbildung 14 Klassendiagramm OpenGLDemo

Die drei beteiligten Klassen sind folgend dargestellt:

```
import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;

public class OpenGLDemo extends Activity {

	private GLSurfaceView glSurfaceView;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		glSurfaceView = new GLSurfaceViewDemo(this);
		setContentView(glSurfaceView);
	}
}
```

[13] *http://www.khronos.org/opengles/* [13]

```
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.view.MotionEvent;

class GLSurfaceViewDemo extends GLSurfaceView {

	private RendererDemo renderer;

	public TouchSurfaceView(Context context) {
		super(context);
		renderer = new RendererDemo();
		setRenderer(renderer);
		setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
	}
}


import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLSurfaceView;

class RendererDemo implements GLSurfaceView.Renderer {

	private Cube cube;

	public RendererDemo() {
		cube = new Cube();
	}

	@Override
	public void onSurfaceCreated(GL10 gl, EGLConfig config) {
		gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
				GL10.GL_DEPTH_BUFFER_BIT);
		gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
				GL10.GL_FASTEST);

		gl.glClearColor(1, 1, 1, 1);
		gl.glEnable(GL10.GL_CULL_FACE);
		gl.glShadeModel(GL10.GL_SMOOTH);
		gl.glEnable(GL10.GL_DEPTH_TEST);
	}

	@Override
	public void onDrawFrame(GL10 gl) {
		gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
				 GL10.GL_DEPTH_BUFFER_BIT);
		cube.draw(gl);
	}

}
```

In der Klasse RendererDemo wird mit Hilfe der OpenGL ES API das rendern vorbereitet. In der `onDrawFrame(GL10 gl)` Methode werden dann die Zeichenoperationen ausgeführt. In diesem Beispiel wurde eine eigene Cube Klasse zum Zeichnen eines Würfels erstellt. Die dafür nötigen OpenGL Zeichenoperationen würden jedoch den Rahmen dieser Ausarbeitung sprengen. Vielmehr ist der Aufbau, der für dieses Beispiel benötigten Klassen, wichtig. Mit dem entsprechenden Wissen über OpenGL kann an dieser Stelle direkt mit der Grafikprogrammierung begonnen werden. Zum hinein schnuppern eignen sich dafür auch

hervorragend die Demo Klassen im Verzeichnis *$SDK_ROOT \samples*. Darin sind unter anderem eine Gruppe mit kleinen Beispielprogrammen zum Thema OpenGL ES enthalten.

Plattformübergreifendes Programmieren

Zwei unterschiedliche Ansätze für die Grafik Programmierung unter Android wurden bereits vorgestellt: Zeichnen mit dem Canvas und mittels OpenGL ES in einer *GLSurfaceView*. Bei letzterem könnte man meinen dass man dank OpenGL schon plattformübergreifend programmiert. Dem ist jedoch nicht so, da man stets die speziellen OpenGL Klassen von Android verwendet. Damit entwickelte Anwendungen lassen sich daher nur im Emulator, oder auf dem Android Endgerät starten. Möchte man eine Anwendung schreiben, die auch noch unter Windows, Linux oder Mac nativ starten kann, muss man andere Wege gehen. Einer ist die Bibliothek libGDX, die im Folgenden vorgestellt wird.

Das libGDX Framework

Das Framework *libGDX* ist eine Plattform übergreifende OpenGL ES Bibliothek unter der Apache 2 Lizenz. Freie als auch kommerzielle Anwendungen dürfen mit libGDX Programmiert werden und der Quellcode ist frei verfügbar. Ins Leben gerufen wurde das noch recht junge Projekt Ende 2009 von Mario Zechner. Nun arbeiten um die zehn Hobby Entwickler daran und erfreuen sich einer stetig wachsenden Community.

Die Hauptseite des Projekts ist folgende:
http://libgdx.badlogicgames.com
Downloads stehen unter Google Code bereit:
http://code.google.com/p/libgdx/

Aufbau von libGDX

LibGDX abstrahiert den Zugriff auf folgende Komponenten: OpenGL ES, Sound, Dateizugriff und ein Input System von Tuchsensor, Tastatur und Beschleunigungssensor. Das alles kann dann plattformübergreifend angesteuert werden. Um das zu ermöglichen arbeiten die Bibliotheken *LWJGL*[14], das Lightweight Java Game Library und *JOGL*[15], die Java OpenGL Library dazwischen. Beide stellen die gleichen Funktionen bereit, so dass man sich bei einer *libGDX* Desktop Applikation entscheiden kann, ob das Programm mit *LWJGL*, oder mit *JOGL* gerendert werden soll. Der Aufbau ist in der angeführten Grafik nochmal dargestellt (Abbildung 15).

[14] *http://www.lwjgl.org*
[15] *http://jogamp.org/jogl/www/*

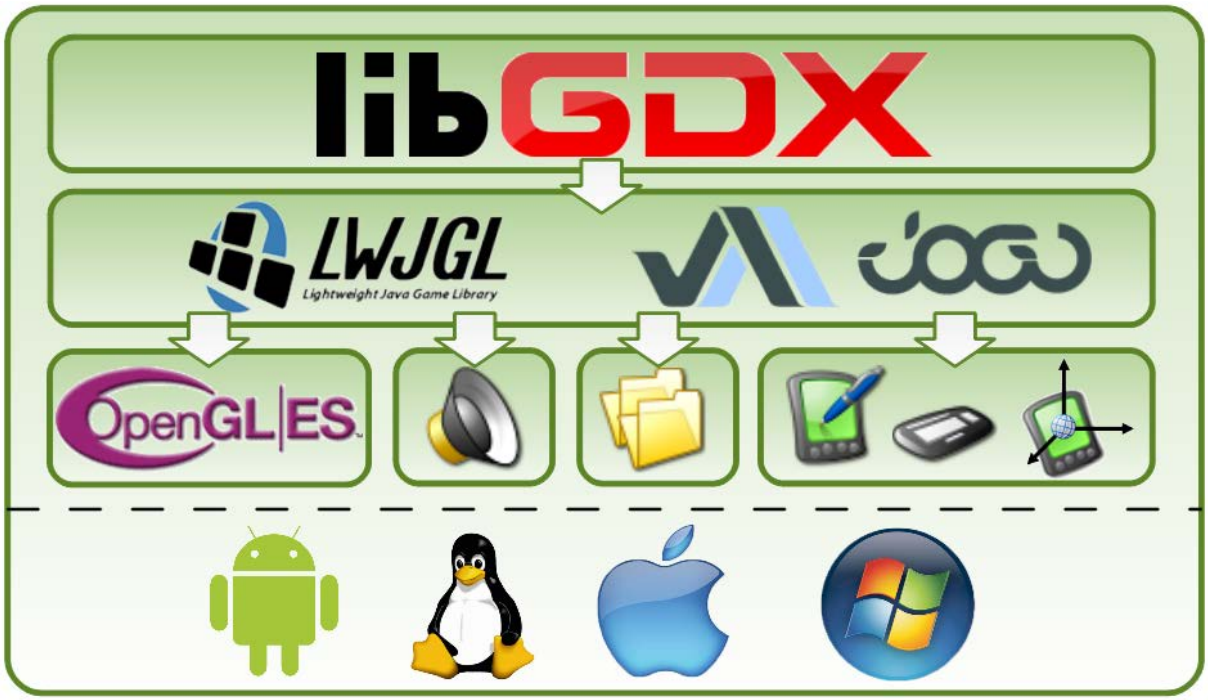


Abbildung 15 Aufbau des libGDX Frameworks

Lebenszyklus einer libGDX Applikation

Eine Anwendung die auf Android gestartet werden soll, muss sich einem speziellen Lebenszyklus unterwerfen. Das bedeutet, dass sich die Anwendung stets in einem definierten Zustand befindet und jederzeit damit rechnen muss in einen anderen überführt zu werden. Das ist wichtig, damit beispielsweise ein ankommender Anruf auf dem Smartphone nicht von einer laufenden Applikation im Vordergrund blockiert wird.

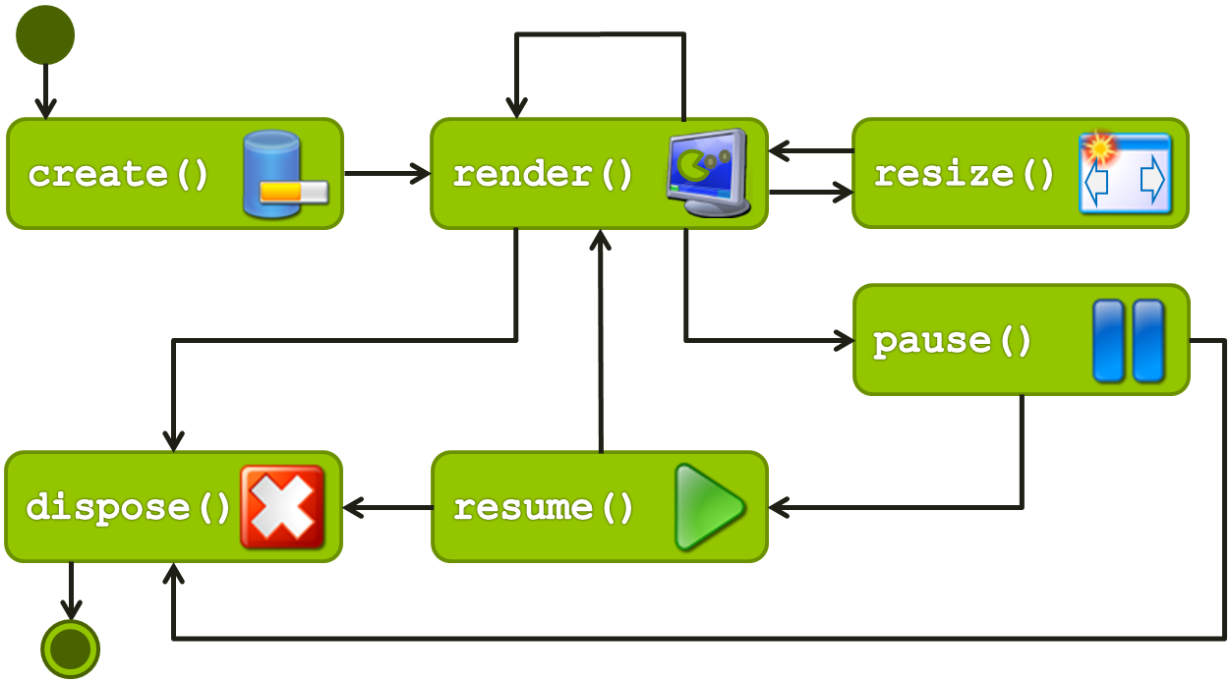


Abbildung 16 Lebenszyklus einer libGDX Applikation

Eine libGDX Applikation stellt den Ablauf über die Schiedsstelle *ApplicationListener* bereit. Als erstes wird die `create()` Methode aufgerufen in der alle Datenstrukturen zu initialisieren sind und Ressourcen wie Grafiken oder Sounds geladen werden müssen. Damit wird sichergestellt, das nach dem Starten der Anwendung keine Ladezeiten durch unkontrolliertes nachladen entstehen. Nach create kommt das Programm in die Renderschleife, in der die `render()` Methode zyklisch aufgerufen wird. Die Renderschleife kann durch den Aufruf von `pause()` pausiert und mit `resume()` wieder gestartet werden. Aus jedem dieser Zustände kann das Programm auch mittels `dispose()` beendet werden. In den Zustand `resize()` wird die App überführt, wenn sich die Bildschirmauflösung ändert. Dies ist bei einem mobilen Endgerät vor allem dann der Fall, wenn der Bildschirm gedreht wird.

Ein Java Projekt mit libGDX einrichten

Zum Programmieren mit libGDX wird zuerst ein Java Projekt eingerichtet. In diesem wird die Anwendung entwickelt und die Desktop Version gestartet. Für dessen Android Version wird ein separates Android Projekt konfiguriert, das vom ersten Projekt abhängt und nur noch eine *Activity* zum Starten enthält. Das Einrichten des Android Projekts wird im Kapitel *Ein Android Projekt mit libGDX einrichten* beschrieben.

Zum Entwickeln mit libGDX benötigt man dessen Bibliotheken. Diese gibt's als Zip Archiv gepackt unter folgendem Link zum Download:
http://code.google.com/p/libgdx/downloads/list
Die benötigte Datei ist die *libgdx-{versionsummer}.zip* mit der höchsten Versionsnummer, die auch als *Featured* gekennzeichnet ist.

Neues Projekt anlegen

Um ein neues Eclipse Projekt anzulegen, muss man unter *File / New / JavaProject* ein neues Java Projekt erstellen. In unserem Beispiel mit dem Namen: *LibGDXDemo*.

In dem noch leeren Projekt müssen folgende Verzeichnisse angelegt werden:

Bibliotheken kopieren

Folgend müssen einige Dateien in das Eclipse Projekt kopiert werden. Dies kann direkt per Drag&Drop in Eclipse geschehen (View *Package Explorer* oder *Navigation*), oder über den Windows Explorer. Bei der zweiten Variante muss nach den Kopierarbeiten das Eclipse Projekt noch aktualisiert werden: Rechtsklick auf das Projekt / Aktualisieren (F5).

In den Ordner *libs* sind nun die für die Entwicklung einer libGDX Desktop Applikation nötigen jar-Dateien zu kopieren. In den Ordner *libsrc* wird anschließend der Source Code von libGDX hinterlegt, was die Entwicklung in Eclipse durch JavaDoc Unterstützung erleichtert. Beides ist nach folgendem Dateibaum zu erledigen:

- LibGDXDemo
  - libs
    - gdx-backend-jogl-natives.jar
    - gdx-backend-jogl.jar
    - gdx-backend-lwjgl-natives.jar
    - gdx-backend-lwjgl.jar
    - gdx-natives.jar
    - gdx-twl.jar
    - gdx.jar
  - libsrc
    - gdx-sources.jar
  - res

Bibliotheken verlinken

Damit die ins Projekt kopierten Bibliotheken auch verwendet werden können, müssen sie noch in den *Java Build path* eingetragen werden. Dies geschieht wie folgt:

Navigiere nach: *Project / Properties / Java Build Path / Libraries*.
Mit *Add JARs...* können nun alle .jar- Dateien aus dem Ordner *lib* hinzugefügt werden -Hier lassen sich alle auf einmal auswählen.

Für die JavaDoc Unterstützung ist der *gdx.jar* noch ein *Source attachmet* anzuhängen. Dazu muss im Reiter Libraries (*Project / Properties / Java Build Path / Libraries*) der Knoten *gdx.jar* aufgeklappt werden. Anschließend ist nach einem Doppelklick auf *Source attachmet* und einem weiteren auf den Button *Workspace*, die gdx-sources.jar aus Verzeichnis *libsrc* auszuwählen.

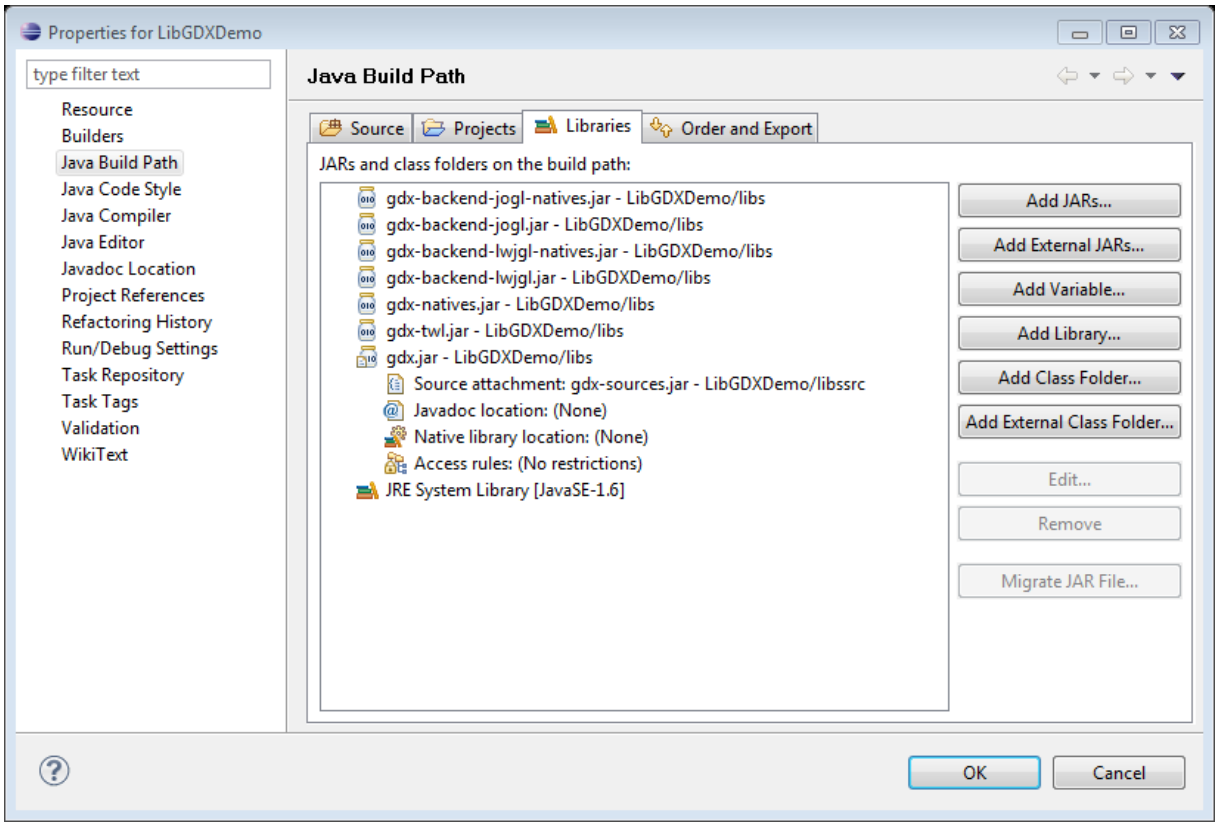
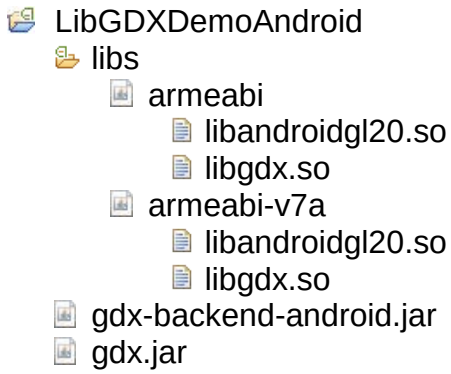


Abbildung 17 Mit libGDX fertig konfiguriertes Java Projekt

Ein Android Projekt mit libGDX einrichten

Nachdem das libGDX Java Projekt angelegt ist, kann damit begonnen werden dessen Android Version einzurichten. Dazu wird ein neues Android Projekt mit dem Namen *libGDXDemoAndroid* erstellt. Der Wizard dazu wird wie folgt aufgerufen: Klicke In Eclipse unter *File / New / Other…* an und wähle aus dem Verzeichnis *Android* das *Android Project* aus. Der folgende Dialog kann bis auf den Projektnamen genauso ausgefüllt werden wie im Kapitel *Ein Android Projekt Einrichten*.

Ist das Projekt angelegt, muss abermals ein Ordner *libs* angelegt werden. In diesen sind die Ordner *armeabi* und *armeabi-v7a* aus der libGDX zip-Datei zu kopieren. Anschließend kommen noch die zwei Bibliotheken *gdx.jar* und *gdx-backend-android.jar* in das *libs* Verzeichnis. Danach sollte der Dateibaum folgendermaßen aussehen:



Wie im Kapitel *Bibliotheken verlinken* genauer beschrieben müssen wieder alle Bibliotheken aus dem Ordner *libs* in den Build Path aufgenommen werden (*Project / Properties / Java Build Path / Libraries* ➔ *add Library*). Gleich danach kann im selben *Java Build Path* Fenster im Reiter Projects mit *Add…* das *LibGDXDemo* Projekt aus dem vorigen Kapitel angegeben werden.

Damit später auch auf dem Android Smartphone die Hardwarebeschleuniger zur Verfügung stehen, fügen wir noch in der AndroidManifest.xml hinter das *application* Tag, vorlegendes ein:

```
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="8" />
```

Damit ist die Konfiguration des libGDX Android Pakets abgeschlossen.

Hello World auf vier Systemen

Nach dem Einrichten der zwei libGDX Projekte können wir damit beginnen ein Programm plattformunabhängig zu programmieren. Dies wird im Folgenden an einem Hello Wold Beispiel gezeigt, das einen Text am Bildschirm ausgibt und ein Bild zeichnet.

Die Klasse *HelloWorld* wird im Java Projekt *LibGDXDemo* in einem beliebigen Package angelegt. Zum Zeichnen brauchen wir noch eine Grafik, die später als Textur geladen wird. Eine Besonderheit von OpenGL dabei ist, dass dessen Breite und Höhe in Pixeln Potenzen von zwei sein müssen. Typische Kantenlängen sind: 64px, 128px oder 256px. Die Grafik im

jpg, png oder gif Format muss im Java Projekt in den Ordner *res* und Android Projekt nach *assests / res* kopiert werden. Anschließend ist folgende Implementierung zu übernehmen:

```
import com.badlogic.gdx.*;
import com.badlogic.gdx.graphics.*;
import com.badlogic.gdx.graphics.VertexAttributes.Usage;
import com.badlogic.gdx.graphics.g2d.*;

public class HelloWorld implements ApplicationListener {
     private SpriteBatch spriteBatch;
     private Sprite      sprite;
     private Texture     texture;
     private BitmapFont  font;

     @Override public void create () {
            font = new BitmapFont();
            texture = new Texture(Gdx.files.internal("res/hska_logo.png"));
            spriteBatch = new SpriteBatch();
            sprite = new Sprite(texture, texture.getWidth(),
                                         texture.getHeight());
     }

     @Override public void render () {
            Gdx.graphics.getGL10().glClear(GL10.GL_COLOR_BUFFER_BIT);
            spriteBatch.begin();
              font.draw(spriteBatch, "Hello World!", 0,600);
              sprite.setPosition(150, 200);
              sprite.draw(spriteBatch);
            spriteBatch.end();
     }

     @Override public void resize (int width, int height) {}
     @Override public void pause () {}
     @Override public void resume () {}
     @Override public void dispose () {}
}
```

Das *SpriteBatch* ermöglicht rechteckige Grafiken auf dem Bildschirm zu zeichnen –So genannten *Sprites*. Ein *Sprite* speichert Position, Größe und Farbe zu einer Textur. Eine Textur enthält dann die Pixelinformationen, die nötig sind um eine Grafik dazustellen. Da OpenGL nicht direkt unterstützt Text auszugeben, behilft man sich mit Bitmap Fonts. Diese Schriften bestehen aus einem Bild in dem alle Zeichen dargestellt sind, und einer Beschreibungsdatei, in der die Position, Größe und Unterschneidung[16] der Zeichen angegeben sind. Der Standardkonstruktor von *BitmapFont* liefert eine in libGDX bereits integrierte Arial 12 Schrift.

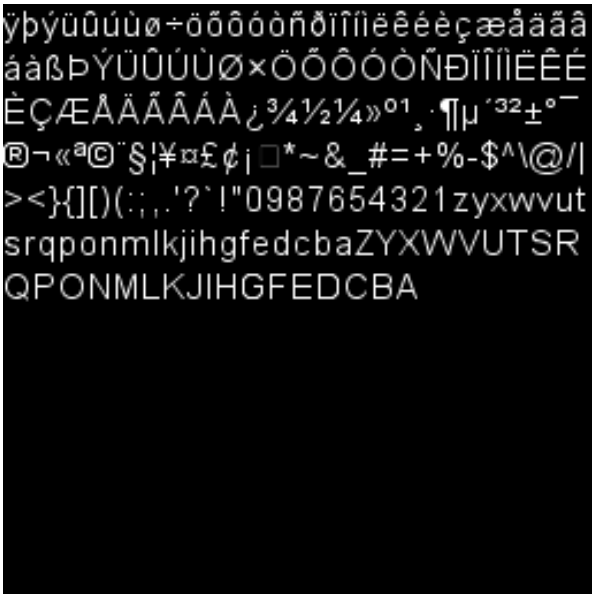


Abbildung 18 Arial 12 BitmapFont von libGDX

[16] Auch Kerning genannt: Verschiebung von Buchstaben um ein angenehmes Schriftbild zu erhalten.

Der erste Aufruf `Gdx.graphics.getGL10().glClear(GL10.GL_COLOR_BUFFER_BIT);` in **`render()`** Löscht den Bildschirminhalt bevor er neu gezeichnet wird. Das ist nötig um auch bewegte Inhalte zu zeichnen, da sonst der Inhalt vom vorherigen Durchlauf noch zu sehen ist und nur übermalt wird. Umgeben von `spriteBatch.begin();` und `spriteBatch.end();` können dann die Zeichenoperationen durchgeführt werden. Die vier Methoden `resize`, `pause`, `resume` und `dispose` werden für dieses einfache Beispiel nicht gebraucht.

Eine main zum starten

Zum starten des HelloWorld Programm fehlt noch der Einstiegspunkt, die main Methode. Diese erstellen wir in einer extra Klasse namens *HelloWorldDesktop*. In dieser müssen wir nur noch eine *LwjglApplication* oder eine *JoglApplication* anlegen, die eine Instanz von *HelloWorld* übergeben bekommt:

```
import com.badlogic.gdx.backends.jogl.JoglApplication;
import com.badlogic.gdx.backends.lwjgl.LwjglApplication;

public class HelloWorldDesktop {

	public static void main (String[] argv) {
		//entweder
		new LwjglApplication(new HelloWorld(), "Hello World", 800,
					480, false);
		//oder
		new JoglApplication(new HelloWorld(), "Hello World", 800,
					480, false);
	}
}
```

Den beiden Startklassen wird beim Aufruf noch eine Auflösung und ob das Programm im Vollbildmodus starten soll mitgeteilt. Für die Android Version müssen wir ebenfalls nur noch eine weitere Klasse erstellen. Diese LibGDXDemo kommt dann aber in das Android Projekt:

```
import com.badlogic.gdx.backends.android.AndroidApplication;
import android.os.Bundle;

public class LibGDXDemo extends AndroidApplication {

	/** Called when the activity is first created. */
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		//setContentView(R.layout.main);
		initialize(new HelloWorld(), true);
	}
}
```

Dabei ist zu beachten, dass die Klasse nicht mehr von *Activity*, sondern von der libGDX Klasse *AndroidApplication* erbt. Die HelloWorld Instanz wird diesmal mit der Methode `initialize(..)` innerhalb von `onCreate` gesetzt. Das Boolean Flag dahinter gibt an, ob wenn verfügbar mit OpenGL ES 2.0 gerendert werden soll.

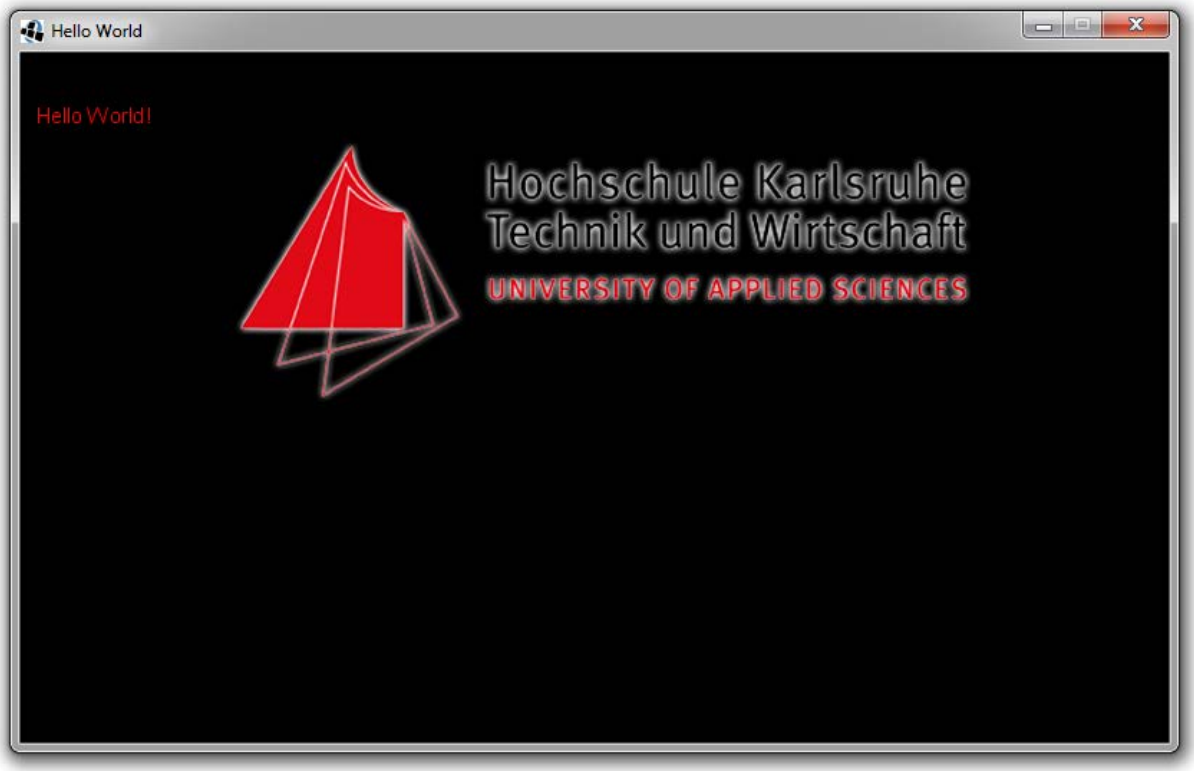


Abbildung 19 Gestartete HelloWorld Desktop Anwendung unter Windows

Literaturverzeichnis

1. **International Data Corporation**, "European Mobile Phone Tracker", *http://www.heise.de/mobil/meldung/IDC-Jedes-dritte-Mobiltelefon-ist-ein-Smartphone-1156244.html*, [Stand: 19,12.2010].

2. **Katrin Mallener**, eco - Verband der deutschen Internetwirtschaft e.V., "Expertenumfrage: Mobile Applications" *http://www.eco.de/dokumente/Expertenumfrage_Mobile_Entertainment_2011.pdf*, [Stand: 02.2011].

3. **Held Micheal**, "Neuer Android Rekord bei App Downloads", *http://mobiles-internet.buemo.net/2011/04/neuer-android-rekord-bei-app-downloads/*, [Stand: 16.04.2011].

4. **Gartner, Inc.**, "Forecast: Mobile Communications Devices by Open Operating System, Worldwide, 2008-2015", *http://www.androidapptests.com/androids-weg-zur-weltherrschaft-alle-neuen-zahlen-studien-und-statistiken-im-uberblick.html*, [Stand: 05.04.2011].

5. **Oracle**, "Java SE Downloads", *http://www.oracle.com/technetwork/java/javase/downloads/index.html*, [Stand: 22.04.2011].

6. **Android Developer**, "Platform Versions", *http://developer.android.com/resources/dashboard/platform-versions.html* [Stand: 01.05.2011].

7. **International Data Corporation**, "European Mobile Phone Tracker", *http://www.heise.de/mobil/meldung/IDC-Jedes-dritte-Mobiltelefon-ist-ein-Smartphone-1156244.html*, [Stand: 19,12.2010]. [Online]

8. **Google Inc.**, "Android Developers", *http://developer.android.com/guide/topics/fundamentals.html*, [Stand: 10.04.2011].

9. **Eclipse Foundation**, "Eclipse Downloads", *http://eclipse.org/downloads/*, [Stand: 01.04.2011].

10. **Mario Zechner**, "libGDX", *http://libgdx.badlogicgames.com* [Stand: 10.05.2011].

11. **Khronos Group**, "OpenGL ES Overview", *http://www.khronos.org/opengles/*,[Stand: 10.05.2011].

12. **eco - Verband der deutschen Internetwirtschaft e.V.**, "Mobile Betriebssysteme: Windows Phone 7 auf absteigendem Ast?" *http://www.eco.de/verband/202_8912.htm*, [Stand: 17.03.2011].

13. **Katrin Mallener**, eco - Verband der deutschen Internetwirtschaft e.V., "Expertenumfrage Mobile Entertainment", *http://mobile.eco.de/files/2011/04/Expertenumfrage_Mobile_Applications_22.pdf*, [Stand: 28.04.2011].