

Softwareentwicklung für Android

Fachseminar
Wintersemester 2008/09
Carsten Taubert
Matrikelnummer.: 253631

Inhaltsverzeichnis

1 Vorwort.....	3
2 Voraussetzungen zum Entwickeln von Android Anwendungen.....	4
2.1 Standard Development Kit.....	4
2.2 Entwicklungsumgebung.....	4
3 Programmieren von Android Anwendungen.....	5
3.1 Lebenszyklus eines Android Anwendung.....	5
3.2 Hello Android.....	6
3.3 Hello Android mit XML-Template.....	7
4 Speichern von Daten.....	9
4.1 Preferences.....	9
4.2 Dateien.....	10
4.3 SQLite Datenbank.....	10
4.4 Netzwerk.....	11
5 Veröffentlichen einer Anwendung.....	12
5.1 Versionierung.....	12
5.2 Signierung.....	12
6 Emulator.....	13
7 Unterschied zu Java MicroEdition.....	14
7.1 Grundsätzliches	14
7.2 Installation der Entwicklungsumgebung.....	15
7.3 Vergleich der Lebenszyklen.....	15
7.4 Hello World in JavaME.....	16
8 Abbildungsverzeichniss.....	17
9 Quellen.....	18

1 Vorwort

In dieser Ausarbeitung soll es um das grundlegende Konzept der Softwareentwicklung für Mobiltelefone mit Android Betriebssystem gehen. Zur Zeit dieser Ausarbeitung gab es noch kein Endgerät mit Android auf dem deutschen Markt zu kaufen. Damit werden hier nur Entwicklungsschritte unter Zuhilfenahme eines Emulator aufgezeigt.

Ebenso soll diese Ausarbeitung keine deutsche Übersetzung der Google Android Dokumentation (<http://developer.android.com/>) darstellen auch wenn sich Überschneidungen natürlich durch das angeschnitten Themengebiet ergeben.

2 Voraussetzungen zum Entwickeln von Android Anwendungen

Um erfolgreich Software für das Android Betriebssystem entwickeln zu können benötigt man nur zwei Komponenten:

- das Developement Kit
- eine Entwicklungsumgebung

Beides wird nachfolgend näher beschrieben.

2.1 Standard Development Kit

Um Anwendungen für Android entwickeln zu können bedarf es zuerst nur dem Android Developement Kit. Dieses liegt in der Version 1.1 für die Betriebssysteme Windows, Linux und MacOS zum download unter <http://developer.android.com/sdk/> bereit.

Ebenso muss das Standard Java Developement Kit ab Version 5 installiert sein.

Dies steht ebenso zum freien download bereit unter

<http://java.sun.com/javase/downloads/index.jsp>

2.2 Entwicklungsumgebung

Empfohlene Entwicklungsumgebung ist Eclipse. Dies steht unter

<http://www.eclipse.org/downloads/> zu freien Verfügung.

Um die Android API in Eclipse nutzen zu können muss das Android Developement Tool Plugin nachinstalliert werden. Eine genaue Anleitung hierfür findet sich unter

http://developer.android.com/sdk/1.1_r1/installing.html#installingplugin.

3 Programmieren von Android Anwendungen

Programmiert wird generell in Java. Dabei ist es möglich fast die komplette Java API der Standard Edition zu nutzen.

Um eine Java Anwendung zu einem Android Programm werden zu lassen muss lediglich die Startklasse von ***android.app.Activity*** erben. Das heißt ein ausführbare Android Anwendung benötigt keine statische ***main***-Methode.

Beispielanwendung ohne Ausgabe:

```
/* Paketname der Klasse */
package net.doncarsten.android;
/* benötigte Imports */
import android.app.Activity;
/* Name der Klasse */
public class EmptyClazz extends Activity {
}
```

3.1 Lebenszyklus eines Android Anwendung

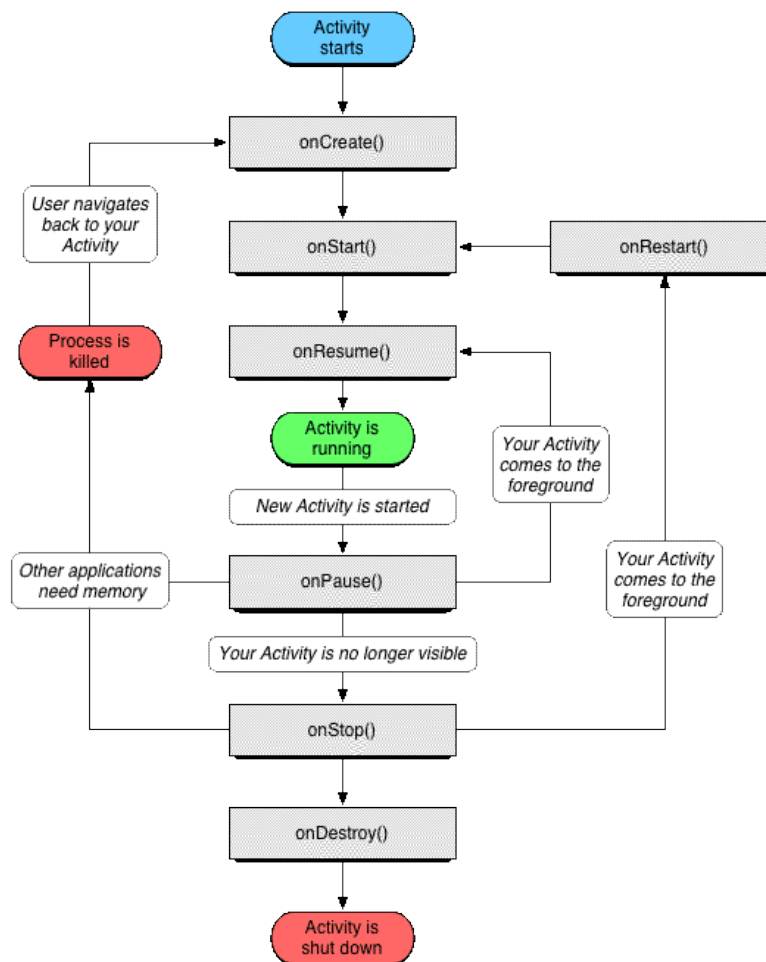
Das Activity Objekt stellt folgende Methoden zur Verfügung die bei Bedarf überschrieben werden können:

- onCreate()
- onRestart()
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroy()

Dabei spielt sich der Lebenszyklus der Anwendung zwischen *onCreate()* und *onDestroy()* ab. Während der sichtbare Lebenszyklus (auch wenn die Anwendung im Hintergrund läuft durch die Methoden *onStart()* und *onStop()* beschreiben wird.

Ist die Anwendung im Vordergrund, also wird aktiv mit ihr gearbeitet wird zusätzlich die Methode *onResume()* aufgerufen. Beim passivieren der Anwendung, sie wird nur noch im Hintergrund ausgeführt, zusätzlich die Methode *onPause()*.

Die zusammenhänge werden im Bild noch einmal verdeutlicht:



[Lebenszyklus einer Anwendung]

3.2 Hello Android

Um eine Ansage zu erzwingen benötigt man **Views**. Dies sind alle zeichenbare Objekte. Auch ein View ist ein JavaObject von dem geerbt werden kann.

Um Text dazustellen benötigt man einen **TextView**. In diesem wird wie in Java üblich per setter-Methode ein Text gespeichert der automatisch auf dem Display des Mobiltelefons erscheinen soll.

Nun muss der **TextView** nur noch mit dem dem **Activity** Objekt verknüpft werden.

Dies geschieht alles in der überschriebenen onCreate() Methode um es beim Aufruf des Programms zu initialisieren.

Beispielanwendung Hello Android:

```
package net.doncarsten.android;
import android.app.Activity;

public class HelloWorld extends Activity {
    /* Überschreiben der onCreate Methode */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* TextView initialisieren */
        TextView tv = new TextView(this);
        /* Text definieren */

        tv.setText("Hello, Android");
        /* TextView mit Activity verknüpfen */
        setContentView(tv);
    }
}
```

3.3 Hello Android mit XML-Template

Die Android API bietet einen zweiten Weg Anwendungen zu designen. In diesem wird die Logikschicht von der Präsentationsschicht getrennt. Dies geschieht durch die Zuhilfenahme von XML Dokumenten in denen das Design gespeichert wird.

Um also im oberen Beispiel den *TextView* aus dem Javacode zu entfernen wird eine XML-Datei benötigt in der es den Tag *<TextView>* gibt.

main.xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/label"
    android:text="Hello, Android"
/>
```

Dieser TextView enthält neben den Koordinaten auch den Inhalt es Objektes. Also „Hello Android“. Ebenso wurde zum späteren Manipulieren eine ID gesetzt, hier *“label”*. Die XML-Datei muss nun nur noch mit dem Activity Objekt verknüpft werden:

Beispielanwendung Hello Android:

```
package net.doncarsten.android;
import android.app.Activity;
public class HelloWorld extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* XML-Datei mit Activity verknüpfen */
        setContentView(R.layout.main);
    }
}
```

Dabei wird eine weitere Klasse genutzt: die R-Klasse. Hier verweist R.layout.main auf layout/main.xml. Das heißt, die Klasse R.java im Projektverzeichnis, verweist auf alle Ressourcen die ebenso in das Projekt eingebunden sind.

R.java:

```
public final class R {  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
}
```

Um jetzt nachträglich den Text aus der XML-Datei zu manipulieren muss nur nach der zuvor definierten ID gesucht werden. Dies geschieht über die Methode ***findViewById()***. Der Text kann danach mit Hilfe der ***setText()***-Methode geändert werden.

Manipulation des Views

```
/* TextView aus XML-Datei auslesen */  
TextView msgTextView = (TextView) findViewById(R.id.label);  
/* Neuen Text einsetzen */  
msgTextView.setText("Hello, World");
```


4 Speichern von Daten

Generell gibt es vier verschiedene Arten Daten aus einer Anwendung heraus für den späteren gebrauch zu speichern:

- Preferences
- Dateien
- SQLite Datenbank
- Netzwerk

4.1 Preferences

Dies stellt die simpelste und meist verwendete Art dar Daten zu speichern. Genutzt wird hier das einfach *Key/Value* verfahren. Das heißt, man weist dem zu speichernden String einen eindeutigen Schlüssel zu und speichert durch Aufruf einer Java Methode.

Wichtig ist hierbei noch, zu bestimmen wer Zugriff auf die Daten haben darf.

Hierzu wird ein bestimmter Modus gesetzt.

MODE_PRIVATE

Nur die Anwendung die den String speichert darf ihn auch lesen.

MODE_WORLD_READABLE

Alle Anwendungen haben leserechte.

MODE_WORLD_WRITEABLE

Alle Anwendungen dürfen den String auch manipulieren.

Nun muss nur noch das Preferences Objekt im gewählten Modus durch ***getPreferences()*** geladen werden und man kann Daten mit Hilfe der ***putString()***-Methode speichern.

Speichern:

```
/* Preferences laden */
SharedPreferences settings = this.getSharedPreferences(MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit();
/* Key/Value definieren */
editor.putString("myKey", "myValue");
/* speichern */
editor.commit();
```

Analog dazu funktioniert das Laden. Nach dem Laden des Preferences Objekts über die Methode ***getString()*** auf die Daten zugreifen. Hierbei ist der „***defaultValue***“ wichtig. Dieser wird als Text gesetzt, sollte es noch keinen String im Speicher zu dem gesuchten Key geben.

Laden:

```
/* Preferences laden */
SharedPreferences settings = this.getSharedPreferences(MODE_PRIVATE);
/* String laden */
String elizaText = settings.getString("myKey", "defaultValue");
```

4.2 Dateien

Da die volle Java API zur Verfügung steht ist es natürlich auch möglich Werte in eine Datei zu speichern. Die Syntax unterscheidet sich also nicht von der eines normalen Java Programms.

Java Code zum schreiben einer Datei

```
/* WriterObjekt laden */
BufferedWriter out = new BufferedWriter(
    new OutputStreamWriter(
        new FileOutputStream( "MeineTextDatei.txt" ) ) );
/* Text der gespeichert werden soll*/
String s = "Text";
/* Text in Datei schreiben */
out.write( s, 0, s.length() );
/* Datei schliessen*/
out.close();
```

4.3 SQLite Datenbank

Die komfortabelste und mächtigste Möglichkeit Daten zu speichern besteht durch Zuhilfenahme der internen SQL Datenbank. Diese bietet die Möglichkeit für jede Anwendungen Tabellen zu definieren und so Daten nicht nur zu speichern, sondern auch zu Verknüpfen.

Da Tabellen erst erzeugt werden müssen ist dies natürlich aufwendiger und nicht für zu empfehlen um sehr wenige Dinge zu speichern. Um Daten zu speichern, muss also zunächst eine Datenbank erzeugt werden und Tabellen angelegt werden.

```
/* Datenbank erzeugen (keine Fehler falls DB schon vorhanden) */
this.createDatabase(MY_DATABASE_NAME, 1, MODE_PRIVATE, null);
Log.i(tag, "database created");

/* Datenbank öffnen */
myDB = this.openDatabase(MY_DATABASE_NAME, null);

/* Tabelle erzeugen */
try {
myDB.execSQL("CREATE TABLE IF NOT EXISTS "
    + MY_DATABASE_TABLE
    + " (User TEXT, Passwd TEXT) "
    + ";");
}
catch (Exception ex) {}
```

Das Einfügen von Daten funktioniert in gewohnter SQL-Syntax. Hier wird in eine Tabelle ein User mit zugehörigem Passwort gespeichert.

```
/* Datensatz einfügen. */
myDB.execSQL("INSERT INTO "
             + MY_DATABASE_TABLE
             + " (User, Passwd)"
             + " VALUES ('abc', 'def');");
```

Um jetzt Daten abzufragen wird ein **Cursor**-Objekt genutzt. So ist es möglich einfach durch alle Treffer des Querys durch zu iterieren. Hier wird einfach das erste Objekt ausgelesen.

```
/* Query der Ergebnisse in Cursor ablegen. */
Cursor c = myDB.rawQuery("SELECT UserNm, PassWd" +
                        " FROM " + MY_DATABASE_TABLE
                        + ";",
                        null);

/* Indizes der Spalten speichern */
int userColumn = c.getColumnIndex("User");
int passwdColumn = c.getColumnIndex("Passwd");

/* Cursor an ersten Treffer setzen */
c.moveToFirst();

/* Username in Variable speichern */
String firstName = c.getString(userColumn);
/* Passwort in Variable speichern */
String pw = c.getString(passwdColumn);
```

4.4 Netzwerk

Da die Standard Java API auch die Möglichkeit bietet Daten über das Netzwerk, und damit auch über das Internet, zu senden, kann dies auch als Speichermöglichkeit betrachtet werden. Das heißt, es werden Daten einfach an eine andere Applikation geschickt, die sich um die Persistierung kümmert. Schnittstelle dafür bieten das Package *java.net*.

5 Veröffentlichen einer Anwendung

Zwei Dinge sind unablässig um eine Android Anwendung zu veröffentlichen: Versionierung und Signierung. Nur wenn man beide erfüllt, darf die Anwendung später über den Android Market verbreitet werden und lässt sich auf einem Endgerät installieren.

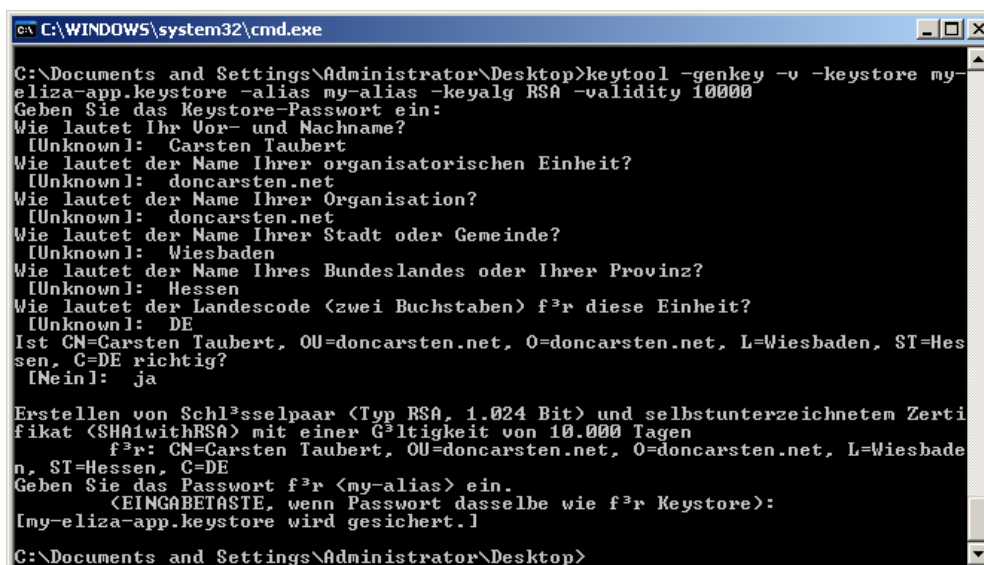
5.1 Versionierung

Versionierung geschieht im eigentlichen nur über die aus Java SE bekannt manifest-Datei. Im Unterschied zur Standard Edition ist die manifest-Datei zu einer echten wohlgeformten XML-Datei gereift. Versionierung soll es ermöglichen den Endbenutzer schnell auf Updates hinzuweisen. So muss nur noch der **android:versionCode** nach Änderungen überprüft werden um eine neue Version festzustellen.

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.doncarsten.android"
    android:versionCode="1"
    android:versionName="1.0.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".FormExample"
            android:label="@string/app_name" >
            ...
        </activity>
    </application>
</manifest>
```

5.2 Signierung

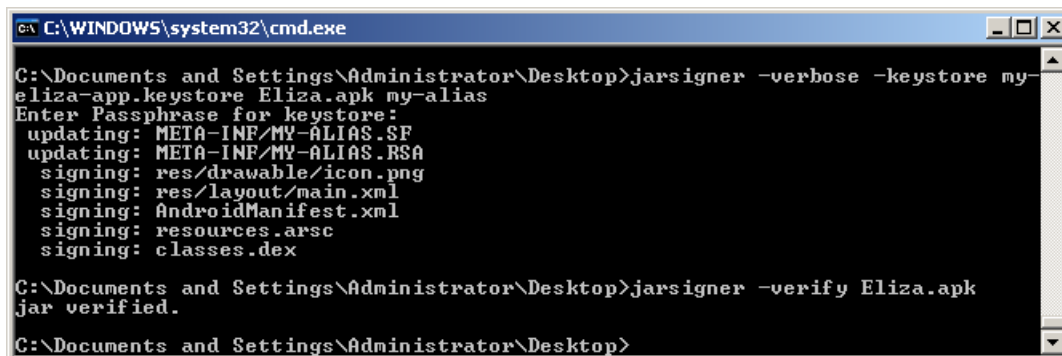
Um eine Anwendung zu Signieren benötigt man einen Schlüssel. Diesen muss man sich nicht von einer CA (*Certificate Authority*) ausstellen lassen, sondern kann ihn sich auch selbst unter Zuhilfenahme von Tools wie keytool, das Bestandteil der Java Standard Edition ist, erstellen.



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\Desktop>keytool -genkey -v -keystore my-
eliza-app.keystore -alias my-alias -keyalg RSA -validity 10000
Geben Sie das Keystore-Passwort ein:
Wie lautet Ihr Vor- und Nachname?
[Unknown]: Carsten Taubert
Wie lautet der Name Ihrer organisatorischen Einheit?
[Unknown]: doncarsten.net
Wie lautet der Name Ihrer Organisation?
[Unknown]: doncarsten.net
Wie lautet der Name Ihrer Stadt oder Gemeinde?
[Unknown]: Wiesbaden
Wie lautet der Name Ihres Bundeslandes oder Ihrer Provinz?
[Unknown]: Hessen
Wie lautet der Landescode (zwei Buchstaben) f³r diese Einheit?
[Unknown]: DE
Ist CN=Carsten Taubert, OU=doncarsten.net, O=doncarsten.net, L=Wiesbaden, ST=Hes-
sen, C=DE richtig?
[Nein]: ja
Erstellen von Schl³sselpaar <Typ RSA, 1.024 Bit> und selbstunterzeichnetem Zerti-
fikat (SHA1withRSA) mit einer G³ltigkeit von 10.000 Tagen
f³r: CN=Carsten Taubert, OU=doncarsten.net, O=doncarsten.net, L=Wiesbade-
n, ST=Hessen, C=DE
Geben Sie das Passwort f³r <my-alias> ein.
(EINGABETASTE, wenn Passwort dasselbe wie f³r Keystore):
[my-eliza-app.keystore wird gesichert.]
C:\Documents and Settings\Administrator\Desktop>
```

[Beispiel für die Nutzung von keytool]

Im nächsten Schritt muss nur noch die erstellte Signatur für die erstellte Anwendung genutzt werden. Auch hierfür gibt es ein kostenfrei Tool: jarsigner.



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\Desktop>jarsigner -verbose -keystore my-
eliza-app.keystore Eliza.apk my-alias
Enter Passphrase for keystore:
updating: META-INF/MY-ALIAS.SF
updating: META-INF/MY-ALIAS.RSA
signing: res/drawable/icon.png
signing: res/layout/main.xml
signing: AndroidManifest.xml
signing: resources.arsc
signing: classes.dex
C:\Documents and Settings\Administrator\Desktop>jarsigner -verify Eliza.apk
jar verified.
C:\Documents and Settings\Administrator\Desktop>
```

[Beispiel für die Nutzung von jarsigner]

6 Emulator

Der Emulator ist Bestandteil des Android Development Kit.

Gestartet wird er automatisch bei Start einer Android Anwendung, aus Eclipse heraus also bei Start einer Java Klasse die von *Activity* erbt.

Ebenso ist es möglich den Emulator aus dem Development Kit Verzeichnis (für Windows: /tools/emulator.exe) heraus zu starten und so das Betriebssystem kennen zu lernen und ohne eigen entwickelte Anwendungen zu testen.



[gestarteter Emulator]

Um nun mit dem gestarteten Emulator kommunizieren zu können, ist es nötig eine Telnet Verbindung aufzubauen. Über diese ist es möglich dem Emulator

- anzurufen
- SMS zu schicken
- Geoinformationen festzulegen
- Akkustand festlegen
- Netzwerkgeschwindigkeit definieren

Möchte man überprüfen, wie die eigene Anwendung reagiert wenn ein Mobilgerät angerufen wird, kann man die über die Telnetverbindung folgendermaßen erreichen:

```
telnet localhost 5554
gsm call <phonenumber>
```

Dies ist besonders wichtig da die eigene Anwendung bei einem Anruf in den Hintergrund verschoben wird, die onPause()-Methode also aufgerufen wird. Die Anwendung sollte also hier auch auf die Pause reagieren (bei Spielen zum Beispiel) und nicht ohne Reaktion weiterlaufen.

Eine SMS wird entsprechend mit folgendem Befehl gesendet:

```
sms send <phonenumber> <text>
```

7 Unterschied zu Java MicroEdition

7.1 Grundsätzliches

Die Idee der Java MicroEdition ist eine Sprache zu haben, mit der man, javatypisch, Anwendungen programmieren kann, die auf allen Endgeräten die JavaME unterstützen, laufen. Hier liegt somit auch schon der größte Unterschied zur Android API. Software für Android ist auf Endgeräten anderer Anbieter (Zum Beispiel: SonyEricsson, Nokia) nicht lauffähig. Jedoch gibt es mittlerweile Portierungsmöglichkeiten.

Die Grundidee einer Javaanwendung der Hard und Softwareunabhängigkeit konnte in JavaME auch nur bedingt umgesetzt werden. Das heißt auch unter JavaME müssen Herstellerabhängige *Packages* verwendet werden um spezielle Fähigkeiten eines Mobiltelefons nutzen zu können (zum Beispiel Bluetooth).

7.2 Installation der Entwicklungsumgebung

Auch für um Anwendungen in Java ME zu entwickeln wird vorher eine SDK benötigt. Dies wird von Sun angeboten. Erste Anlaufpunkt ist <http://java.sun.com/javame/index.jsp>. Die Java SE Installation wird weiter nicht benötigt. Als Entwicklungswerkzeug kann auch hier wie bei Android jeder Editor genutzt werden. Am meisten genutzt auch hier ein Eclipse Plugin. EclipseME. Dies ist frei unter <http://eclipseme.org> verfügbar. Ebenso ist JavaME bei der NetBeans Standard Installation gleich dabei. Dies gibt es ebenfalls kostenlos unter <http://www.netbeans.org/downloads/index.html>.

Java ME bringt gleich vier Emulatoren mit sich:

- pager
- minimum phone
- default grey phone
- default color phone

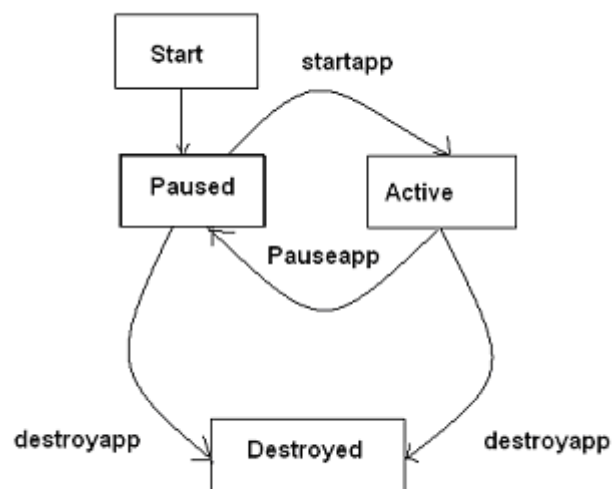
Eine Anwendung wird heutzutage fast ausschließlich für ein „default color phone“ Emulator geschrieben..

7.3 Vergleich der Lebenszyklen

Der Lebenszyklus einer JavaME Anwendung ist einiges simpler als der einer Android Anwendung. Gibt es für Android noch sieben Methoden die überschrieben werden können, sind es bei JavaME nur drei:

- startApp()
- pauseApp()
- destroyApp()

Implementiert werden diese durch Erben von *javax.microedition.midlet.MIDlet*. Die Methoden unter Android *onResume()* oder *onRestart()* fehlen dem *Midlet*, das vergleichbar mit dem *Activity*-Objekt in Android ist. Daraus folgt, dass die nach jeder Unterbrechung immer nur die *startApp()* Methode aufgerufen wird. Es demzufolge nicht möglich, zwischen verschiedenen Ereignissen, wie Passivierung durch und Schließen der Anwendung, beim erneuten Starten, zu unterscheiden.



[Lebenszyklus eine JavaME Anwendung]

7.4 Hello World in JavaME

Als Beispiel für ein Programm das in JavaME entwickelt wurde soll *Hello, World!* dienen. Daran ist auch zu erkennen, dass die Grundfunktionen zu Android gleich sind, meist nur einen anderen Namen haben. So kann man das *Activity* Objekt mit einem *Midlet* vergleichen oder den *TextView* mit einem *Form*.

```
/* Die HelloWorld-Klasse ist ein MIDlet */
public class HelloWorld extends MIDlet {

    /* Display des Handys*/
    Display display;

    /* Form zum positionieren von Elementen */
    Form form;

    /* Ausführung beim Start des Programms*/
    public void startApp() {

        /* neues Form-Objekt instanzieren mit Titel */
        form = new Form("Hello J2ME !!!");
        /* Displayobjekt holen */
        display = Display.getDisplay(this);

        /* das Form-Objekt auf dem Display anzeigen */
        display.setCurrent(form);
    }
}
```


8 Abbildungsverzeichniss

Lebenszyklus einer Anwendung

<http://developer.android.com/guide/topics/fundamentals.html>

Beispiel für die Nutzung von keytool
selbst erstellt

Beispiel für die Nutzung von jarsigner
selbst erstellt

gestarteter Emulator
selbst erstellt

Lebenszyklus eine JavaME Anwendung

http://wiki.forum.nokia.com/index.php/Midlet_basic_lifecycle_and_states

9 Quellen

Developers Guide, Google

<http://developer.android.com/guide/index.html>

Android, Wikipedia

[http://de.wikipedia.org/wiki/Android_\(Plattform\)](http://de.wikipedia.org/wiki/Android_(Plattform))

[http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))

JavaME, Wikipedia

<http://de.wikipedia.org/wiki/JavaME>

<http://en.wikipedia.org/wiki/Javame>

Symbian Developer Network

<http://developer.symbian.com/main/index.jsp>

Eclipse Software

<http://www.eclipse.org/>

Eclipse ME Plugin

<http://eclipseme.org/>

NetBeans

<http://www.netbeans.org/downloads/index.html>

JavaME Wiki for Nokia Devices

<http://wiki.forum.nokia.com/index.php/Category:Java>