Entwicklung Mobiler Applikationen : Android

Roland Hediger

26. Mai 2014

Inhaltsverzeichnis

1. Android GUI

1.1. View Klassen

Die Klasse View ist die Basisklasse der User Interface komponenten.

- UI-Widgets wie Buttn,Textfield,TextView
- LayoutKlassen
- Zeichenöberfläche wie SurfaceView,GLSurfaceView.

UI können om Android auf 2 Arten, Prozedural oder Deklarativ.

Deklarativ Vorteile
- Kurzer in XML,
- leserlicher
- GUI Designers können XML Manipuliere

Performanz von XML GUI Anderoid übersetzt die XML Dateien zur Kompilationszeit in ein komprimiertes binäres Format. Das **Android Asset Packaging Tool - aapt** kompilert Rescourcen in ein binäres Format. Ist in Eclipse Plugin integriert. Das übersetzte binäre Format wird geladen. Verarbeitung zur Runtime ist notwendig.

Eine Activity ist die Applikationskomponente, welche das User Interface zur Verfügung stellt. Über dieses Interface kann ein Benutzer mit der Applikation interagieren. Jede Activity besitzt ein Fenster (Window) in welches das Benutzerinterface gezeichnet wird. Deshalb ist die Methode setContentView() sehr wichtig, denn über diese Methode wird ein geeignetes User-Interface gesetzt, d.h. eine View. Die View-Klasse ist die Basisklasse aller User Interface Komponenten. Diese Komponenten können UI-Widget (Button, EditText, ...), Layout-Klassen (LinearLayout, RelativeLayout, ...) oder Zeichenflächen (SurfaceView, VideoView, ...) sein. Die View-Hierarchie ist wie folgt:

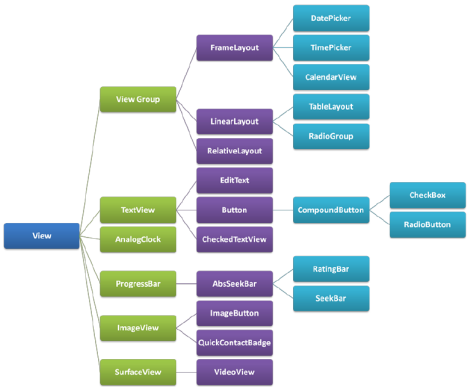

Abbildung 1.1.: figure

1.2. Class R

Die Klasse R wird automatisch vom Eclipse Plugin und aapt generiert und synchron gehalten. die Klasse umfasst alle Resourcen die in den XML Files unter der Folder **res** abgelegt sind. `Hex Zahlen sind Handles auf die eigentliche Resource.` de

Listing 1.1: R Class Example

```
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="ch.fhnw.edu.helloworld.MainActivity$PlaceholderFragment" >
<TextView android:id="@+id/mytext"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello_world" />
</RelativeLayout>
```

1.3. Layout

Ein Layout ist ein Container für mehrere Child Objekte und erlaubt die Positionierung dieser Objekte innerhalb des Containers.

FrameLayout Positioniert alle Child objekte oben links. (Tabbed Views/Image Switcher)

LinearLayout ChildObjekte in einer Reihe oder Kolonne. Meist verwendete Layout.

RelativeLayout ChildObjekte relativ zuenander vor allem in Formular eingesetzt.

TableLayout Analog Html Table.

1.4. Widgets

Gibt eine grosse breite davon..

Listing 1.2: Widget Beispiel

Widget Deklarativ

```
<TextView android:id="@+id/status"
android:layout_height="wrap_content"
android:textSize="10sp"
android:gravity="right"
android:layout_width="match_parent"
android:textColor="#ffff00" />
```

id Die neue Resource ID ßtatus"wird erzeugt - kann im Programm über r.status zugegriffen werden.

gravity Platzierung

SurfaceView eignet sich um eigene Szenen zu zeichnen. Über die klasse SurfaceHolder kann die Zeichenfläche verändert werden.
Erhohlt über getHolder()
Dazu mussen in der eignen Klasse diverse Callback Methoden implementiert werden:

```
  Callback Methoden implementiert werden:
... implements SurfaceHolder.Callback
surfaceChanged(), surfaceCreated(),
surfaceDestroyed()
```

Surface View und Render Thread Das zeichnen im eigenen Thread auslagern. Der Thread wird in `Surfacecreated` gestartet und in `SurfaceDestroyed` gestoppt. Zum zeichnen braucht es in **Canvas**. Canvas stellt verschiedene methoden zur Verfügung, um Zeichen-Primitive wie Kreis Rechteck, Linie,use darzustellen.

Touch Screen Jede View kennt verschiedene OnXXEvent() Methoden z.B :

- onTouchEvent(MotionEvent event)

  Die Klasse MotionEvent umfasst verschiedenste informationen wie x und y Koordinaten des Event. Der Typ des Event z.b des Event ACTION_UP, ACTION_DOWN

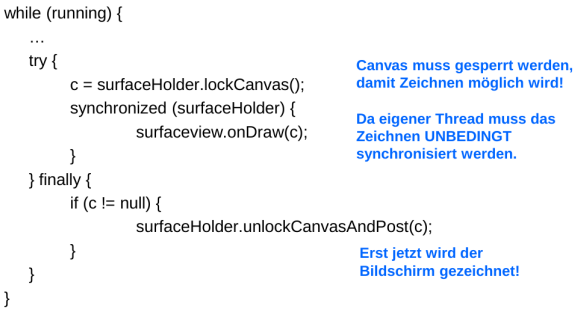

Abbildung 1.2.: figure

1.5. Aufbau eines Games

Spiele sind in den meisten Fällen nicht event bassiert, sondern die laufen ständig. Sie zeichnen die Speilwelt permanent neu, hohlen sich neue Benutzereingaben und simulieren die Spielwelt. Ein Seperater Thread muss die Applikation permanent neu Zeichnen : Game Loop.

Listing 1.3: Game Loop

```
while( !done ) {
processInput( )
simulateWorld( )
renderWorld( )
}
```

2. Intents

2.1. Einführung

Das Android-Betriebssystem verwendet einen asynchronen Mechanismus namens Intent für den Austausch von Nachrichten. Dadurch lassen sich im laufenden Betrieb anfallenden Aufgaben mit einer dazu passenden Android-Activity koppeln. Man kann Intents dazu verwenden, um

- eine Activity innerhalb derselben Anwendung aufzurufen,
- eine Activity innerhalb einer anderen Anwendung aufzurufen,
- einen Service zu starten,
- eine Applikation über Systemereignisse zu informieren.

Der Einsatz von Intents bietet sich auch dann an, wenn mit Hilfe des Betriebssystems eine passende Activity gesucht wird, die eine bestimmte Aufgabe übernehmen soll, so z.B. das Abspielen eines Videos (Youtube vs. Video App). Die folgenden drei Zeilen Programmcode starten beispielsweise die Dialer Applikation, wobei die zu wählende Nummer automatisch im Eingabefeld erscheint

```
 Uri number = Uri.parse("tel:2125551212");
Intent dial = new Intent(Intent.ACTION_DIAL, number);
startActivity(dial);
```

Wenn man sich eine Activity als einen Grundstein vorstellt auf dem alle Android Anwendungen aufbauen, dann stellen Intents den Klebstoff dar, der diese Activities miteinander verbindet. Dadurch kann der Anwender flexibel von einer Activity zur nächsten wechseln

2.2. Details

Ein Intent (android.content.Intent) besteht aus zwei Teilen:

1. die Aktion, die ausgeführt werden soll, und
2. die Daten, die verarbeitet werden sollen

Wenn man einen Intent erstellt und verschickt, weist man das Betriebssystem an eine vorher definierte Aufgabe an eine bestimmte Activity zu delegieren. So ruft man zum Beispiel mit dem Intent, der die Aktion ACTION_VIEW festlegt und als Daten die URL einer Webseite enthält, die Browser Activity auf. Es ist nun diese Activity welche die geforderte Kombination aus Aktion und Daten verarbeitet.

```
 Uri address = Uri.parse("http://www.perggurl.org");
Intent surf = new Intent(Intent.ACTION_VIEW, address);
startActivity(surf);
```

Das Android Betriebssystem sucht dabei nach einer passenden Activity für die im Intent festgelegten Kombination aus Aktion und Daten (hier eine URL) und wird bei der Browser Anwendung fündig. Android lädt nun den Browser in den Vordergrund, die aktuelle Activity wird auf einen Stack geschoben und bleibt inaktiv im Hintergrund. Der Browser lädt die gewünschte Webseite. Wenn der Browser beendet wird, aktiviert Android die Activity die den Intent generiert hat - und die im Stack zuoberst liegt.

2.3. Manifest

Android führt genau Buch über alle auf einem Gerät installierten Anwendungen sowie über alle Activities, die im Android-Manifest deklariert sind. Die <activity> Definition in der Manifest-Datei bestimmt, welche Aktionen und Datentypen eine Activity verarbeiten kann. Diese Fähigkeit wird über den Intent-Filter beschrieben. Das Android-System nutzt diese Informationsangaben um einen Intent mit einer entsprechenden Activity zu verknüpfen. Bei einem Intent-Request wählt das System immer die Activity, die am besten dazu geeignet ist, den Request zur Laufzeit zu verarbeiten. Wenn es nun mehrere Activities geben sollte die zu einem Intent passen, dann entscheidet der Anwender schlussendlich über einen Dialog welche Applikation zum Einsatz kommen soll.

Was passiert, wenn keine **passende Activity auf dem Gerät installiert ist?** Normalerweise wird eine Exception ausgelöst. Man kann als Entwickler aber überprüfen, ob Activities auf dem System existieren, die für bestimmte Intents geeignet sind. Dazu kommen Methoden der PackageManager- Klasse zum Einsatz, wie z.B. queryIntentActivities() oder queryBroadcastReceivers().

Es gibt eine Reihe von Standard-Aktionen, wie z.B. ACTION_VIEW, ACTION_EDIT oder ACTION_SEND, die auf der Android Plattform universell einsetzbar sind. Entwickler können natürlich auch eigene Activities und Intents schaffen. Durch den Einsatz massgeschneiderten Intents lassen sich komplexe Programme entwickeln.

2.4. Expliziter und Impliziter Events

startActivity(Intent intent) wird ein **Impliziter** Intent eingesetzt, der alle Activities anspricht, die den Intent ëdu. intro.HELLOWORLD"verarbeiten können.

```
startActivity(new Intent("edu.intro.HELLOWORLD"));
```

startActivity(new Intent(Context context, Class<?> clazz))

Will der Applikationsentwickler den Intent Empfänger genauer spezifizieren, so muss er einen expliziten Intent einsetzen. Hier wird genau festlegt welche Activity gestartet werden soll. Dieser explizite Einsatz des Intents wird durch den Klassennamen ausgedrückt. Dieser Intent teilt der Android Plattform mit, dass eine Activity namens MyOtherActivity gestartet werden soll. Die entsprechende Klasse MyOtherActivity muss innerhalb der gleichen Anwendung implementiert sein und sie muss einen Eintrag in der Android-Manifest Datei der Anwendung haben.

```
startActivity(new Intent(getApplicationContext(), MyOtherActivity.class));
```

startActivityForResult(Intent intent, int requestCode)

Erwartet man von der aufgerufenen Activity eine Antwort, zum Beispiel soll der Contact Picker Informationen über den ausgewählten Kontakt zurückgeben, so steht dem Entwickler die Methode startActivityForResult() zur Verfügung. Die aufrufende Activity kann eine Ergebnisrückgabe einfordern, indem die Activity mittels der startActivityForResult() gestartet wird. Das Ergebnis kann dann in der Callback Methode onActivityResult()der aufrufenden Activity verarbeitet werden:

```
...
startActivityForResult(intent, requestCode);
...
public void onActivityResult(int requestCode, int resultCode, Intent data) {
...
}
```

2.5. Daten übergeben mit Intents

Intents lassen sich auch dazu verwenden, zusätzliche Informationen zwischen Activities - und folglich zwischen Anwendungen - zu übermitteln. Dazu fügt man dem Intent weitere Eigenschaft hinzu, sogenannte Extras. Ein Extra besteht aus einer Kombination Bezeichnung und Wert. Man kann bestimmte Extra unter der Verwendung dieser Bezeichnung auslesen und schreiben, indem man die entsprechenden put() und get() Methoden verwendet, die in der Intent Klasse zur Verfügung stehen. Bei der Übermittelung von zusätzlichen Informationen zwischen den Activities können beliebige weitere Eigenschaften definiert werden. Es hat sich die Konvention eingebürgert, die Paketnamen der Anwendung als Präfix für die Bezeichnung des Extra zu benutzen. Hier ein Beispiel:

```
 Intent intent = new Intent(getApplicationContext(), GameActivity.class);
Intent.putExtra("com.mygame.LEVEL", 7);
startActivity(intent);
Intent callingIntent = getIntent();
int level = callingIntent.getIntExtra(("com.mygame.LEVEL", 1);
```

3. Responsive Design

Das Aufkommen der Tablets führte zu einer weiteren Fragmentierung der mobilen Systeme. Insbesondere der viel grössere Bildschirm zwang die Entwickler Konzepte zu finden, welche einerseits erlauben, dass die gleiche Applikation auf den unterschiedlichen Gerätetypen lauffähig bleibt, und andererseits dass die vorhandenen Hardware-Ressourcen sinnvoll genutzt werden. Mit dem Responsive Design handelt es sich nun um die technische Umsetzung eines solchen Konzepts. Der grafische Aufbau einer „responsiven“ Seite erfolgt anhand der Möglichkeiten des jeweiligen Gerätes, mit dem die Seite betrachtet wird. Dies betrifft insbesondere die Anordnung und Darstellung einzelner Elemente, wie beispielsweise Navigation, Seitenspalten und Texte.

3.1. Fragments

Seit Android 3.0 lassen sich Activites aus mehreren Fragments zusammensetzen. Diese Fragmente stellen autonome Content-Bereiche bzw. Klassen dar, müssen aber immer innerhalb einer Activity ausgeführt werden. Sie können deshalb genau wie Activities ein Layout besitzen und übernehmen somit die Funktion die früher eine Activity hatte. Fragments besitzen einen umfangreicheren Lebenszyklus als die Activities. Er ist aber mit dem Lebenszyklus der Activity eng verzahnt, da das Fragment an die Host-Activity gebunden ist. Die einzelnen Lebenszyklusmethoden für die Erzeugung eines Fragments bis zu dessen Anzeige lauten:

```
 onAttach (Activity)
onCreate(Bundle)
onCreateView(LayoutInflater, ViewGroup, Bundle)
onActivityCreated(Bundle)
onStart()
onResume()
```

Früh wird die Methode onAttach aufgerufen, nämlich wenn das Fragment der Activity zugeordnet wird. onCreateView muss eine View zurückgeben. Hier können Initialisierungen speziell am Layout der View, also der Oberfläche des Fragments, vorgenommen werden. Ein Zugriff auf das Layout der Activity darf hier nicht erfolgen, da dieses Layout zum Zeitpunkt noch nicht existiert. onActivityCreated wird aufgerufen, nachdem die onCreate-Methode der Activity durchlaufen wurde, zu der das Fragment gehört. In dieser Methode steht somit die vollständige View-Hierarchie zur Verfügung, also auch die der Activity darüber. Dadurch liesse sich beispielsweise auch auf Views der Activity zugreifen. Insbesondere eignet sich diese Methode für dynamische Anpassungen des Layouts, beispielsweise wenn Zustände im Bundle gespeichert wurden. Generell kann diese Methode für Initialisierungen genutzt werden, da sie recht spät im Lebenszyklus des Fragments und seiner Activity aufgerufen wird.

3.2. Resourcen

drawable-XXX: Die drawable Ordner enthalten Bilder Ressourcen in unterschiedlicher Auflösung. Die App fordert z.B. die Ressource @drawable/icon an, und auf einem hochauflösendem Gerät wird dann vom System automatisch die aus dem drawable-hdpi Verzeichnis genommen und auf einen niedrigauflösendem die ldpi-Version. Bei einem Prototyp kann man sich auf ein drawable- Verzeichnis beschränken, dann skaliert Android die Ressource auf die konkret passende Grösse. Der Nachteil dabei ist, dass die skalierten Grafiken bisweilen überproportioniert wirken. Oder wenn man nur grosse Grafiken ablegt verbraucht ein kleines Gerät viele Ressourcen um die (zu) grosse Grafik erst zu laden und dann runter zu skalieren.

layout-XXX: Hier sind die Layout Beschreibungen als XML-File abgelegt. Zusätzliche zum Default Ordner layout können weitere Ordner angelegt werden, die bei einer bestimmten Konstellation gelesen werden. Ein eigenes Layout in der LandscapeOrientierung wird im Ordner layout-land abgelegt.

values-XXX: Hier sind XML Files hinterlegt, welche einfache Werte wie z.B. Strings, Zahlen oder Farben enthalten. Auch diese Werte können an unterschiedliche Konstellation gebunden werden wie minimale Auflösung (values-sw600dp) oder API Versionen (values-v11) oder Bildschirmgrösse (values-large).

Um Duplikationen von Layout Definitionen zu verhindern, kann man über diese values-Ordner und einem Files refs.xml Referenzen erstellen, um einer bestimmten Layout Definition einen anderen Werte zuzuschreiben. Mehr dazu in den folgenden Aufgaben.

3.3. Responsive Design for Tablet

1. Layout anpassen:

Listing 3.1: MainActivity

```
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent" >
<fragment android:name="ch.fhnw.edu.helloworld.MainFragment"
android:id="@+id/main_container"
android:layout_width="match_parent"
android:layout_height="match_parent" />
</LinearLayout>
```

Listing 3.2: MainActivityTwoPane

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent" >
<fragment android:name="ch.fhnw.edu.helloworld.MainFragment"
android:id="@+id/main_container"
android:layout_width="wrap_content"
android:layout_height="match_parent" />
<FrameLayout
android:id="@+id/helloworld_container"
android:layout_width="match_parent"
android:layout_height="match_parent" >
</FrameLayout>
</LinearLayout>
```

2. Code Anpassungen:
   Das erste Element entspricht dem Fragment aus activity_main.xml.Für das zweite Fragment, hier soll ja das HelloWorld-Image eingeblendet werden, wird mittels FrameLayout der notwendige Platz geschaffen. Da das entsprechende Fragment erst bei einem Button-Klick gezeigt werden soll, darf hier nicht schon das resultierende Layout definiert werden. WICHTIG: Der Container für das 2. Fragment hat die ID helloworld_container. Mit dieser ID helloworld_container kann in der MainActivity gearbeitet werden. In der onCreate() Methode von der MainActivity muss entschieden werden, ob man im Portrait oder Landscape Modus ist. Im Landscape Modus ist die View mit der ID helloworld_container vorhanden, im Portrait Modus nicht. Das kann man nutzen indem man folgende Abfrage startet: if (findViewById(R.id.helloworld_container) != null)

3. Landscape Layout Referenzen:
   Damit beim Wechsel in den Landscape Mode das Layout activity_main_twopane geladen wird, muss die MainActivity anstelle activity_main dieses neue Layout laden. Dazu muss man eine alternative Ressource erstellen und den Namen activity_main mit activity_main_twopane "überschreiben". Dies kann über ein File layout-land/refs.xml realisiert werden:

```
<resources>
<item name="activity_main" type="layout">@layout/activity_main_twopane</item>
</resources>
```

WICHTIG: Über den Ordner layout-land wird dem Android System signalisiert, dass im Landscape Modus die Resources aus diesem Ordner gelten, u.a. wird die Referenz von activity_main auf das neue Layout gesetzt.

3.4. Auswechselung von Fragments

```
 FragmentManager fm = getSupportFragmentManager();
Fragment f = fm.findFragmentById(R.id.helloworld_container);
if (f == null) {
FragmentTransaction trx = fm.beginTransaction();
trx.add(R.id.helloworld_container, new MessageFragment());
trx.commit();
}
```

4. Effiziente Netwerk Synchronization

4.1. Probleme

- Wie kann ein eiffizienter batterieschönender Datenaustausch realisiert werden?
- Behandelung von Unterbruch der Verbindung

4.2. Radio State Machine

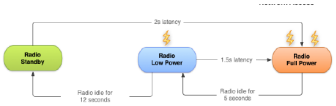

Abbildung 4.1.: figure

Full Power Eine Verbindng ist aktiv. Höchstmögliche Rate.

Low Power Übergang bei 50% der Batterieleistung

Standby Keine Verbindung aktiv, batterieschönend.

4.2.1. Probleme

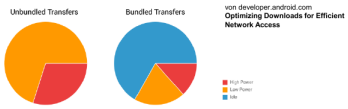

Abbildung 4.2.: figure

Eine App tauscht alle 18 Sekunden Daten aus. Für den Datenaustausch braucht die App 1 Sekunde. Das bedeutet, dass nie in den Standby Modus gewechselt werden kann (1 (Transfer) + 5 (Idle) + 12 (Low Power) = 18). Es kann nie in den batterieschonenden Standby Modus geschaltet werden. Wird die Datenübertragung gebündelt, so genügen pro Minute 3 Sekunden. Das führt dazu, dass 40 Sekunden im Standby Modus gewartet werden kann, da nur noch über 20 Sekunden (3 (Transfer) + 5 (Idle) + 12 (Low Power) = 20) die anderen Zustände gebraucht werden.

4.2.2. Ansätze

Prefetch Anzahl Verbindungen,Batterieverbrauch,Latzenzzeit,Kommunikationsbreite und Downloadzeit reduzieren. Wie viel soll aber im vorraus geladen werden. Welche Daten? Welche Rate? Server API gestaltung?. `Beispiel Ansatz:` Verbindungsaufbau alle 2 bis 5 Minuten Datenvolumen zwischen 1 und 5 MB

Koordination mit mehrere Apps Normalerweise werden mehrere Apps das Netzwerk für Kommunikationsaufgaben nutzen z.B. Gmail, Evernote, WhatsApp, ...
Herausforderung:
Wie können diese verschiedenen Apps koordiniert werden, damit das Netzwerk optimal genutzt wird?

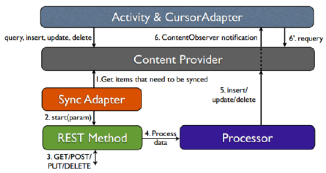

Abbildung 4.3.: figure

4.3. Pattern Content Provider und Sync Adapter

Framework Datentransfer automatisiert. Sync über verschiedene Apps koordinieren.

Sync Adapter ist Plugin für den Android Sync Manager sind Background Services, die vom Sync Manager gestartet werden, wenn z.B. Netzwerk vorhanden ist und Sync Request ansteht Sync Manager verwaltet eine Queue von Sync Adapter mit ihren Synchronisationsaufgaben.

5. Background Tasks

5.1. UI Thread

Ein GUI Framwork ist normallierweise auf einem Main Thread basiert um Events aus Input (Keyboard, Maus, ...) und für Output (Screen, Window, ...) sequentiell abarbeiten zu können,um Deadlocks einfach vermeiden zu können. Beim Start wird für eine Android Applikation ein eigener Prozess mit einem einzigen UI Thread hochgefahren. Der UI Thread ist für die sichtbaren Bestandteile einer Applikation zuständig.

5.2. Herausforderungen Single Thread

Reaktionsverhalte n auf der User Input ist entschiedend für die Akzeptanz einer Applikation. Aufwändige Berechnungen im UI Thread lassen view einfrieren. Das Android System überprüft die Reaktionszeit einer Applikation und wird bei schlechtem Antwortverhalten mit einen entsprechenden Dialog (Applikation not Responding) konfrontiert.

5.2.1. Triggers für ANR

- Window Manager : Keine Reaktion innerhalb 5 sec.
- Broadcastreciever kann Event innerhalb 10 sec verarbeiten.

5.2.2. Folgerungen

Jede Methode im UI-Thread soll nur das Notwendigste ausführen. Alle anderen Operationen wie Netzwerkzugriff, Datenbankzugriff oder umfangreiche Berechnungen sollen in Child Threads oder Services ausgelagert werden. UI Thread soll nie blockiert werden mit Thread wait oder Thread sleep.

5.3. Varianten

5.3.1. Worker Thread

Vorteile: Konstrukt aus Java Welt. Einfache Interaktion mit UI Thread über Methode runonUIThread().
Nachteile:

- Explizites Lifecycle Management
- Sync muss unbedingt beachtet werden.
- Unterscheidung wann im UI Thread vs wann im Worker Thread ist nicht immer einfach.

5.3.2. Async Tasks

Vorteile: klare Trennung der Methoden zwischen UI Thread und Worker Task. Einfgache und Klare Interaktion mit dem UI Thread auch zur Laufzeit des Worker Threads. Keine Thread Management notwendig. **Nachteile:** Neues Kontrukt - Übergabe der Parameter an verschiedene Methoden.

5.3.3. Service and Broadcastreciever

Vorteile: Klare Trennung.Einfache Kommunikation von Background Service zu Activity. Service mit einem eigenen Lifecycle d.h Uunabhängig von der Activity. **Nachteile:**Einwegkommunikation vom Service zur Activity

5.4. Service und Messenger + Handler

Vorteile: Klare Trennung. Zweiwegkommunikation zwischen Service und Activity.keine Synchronisationsprobleme dank Einsatz von Message-Queues in den Handler. **Nachteile:**Komplexe Kommunikationsinfrastruktur zwischen Service und Activity.

6. Background Tasks II

Das Android UI gleicht konzeptionell den gängigen GUI Frameworks. Es ist event-driven, besteht aus geschachtelten GUI Komponenten und vor allem ist es single-threaded. Denn man hat bei der Entwicklung der gängigen GUI Frameworks schnell erkannt, dass mit einem multi-threaded Ansatz Deadlocks fast nicht zu vermeiden sind, da ein GUI auf Events von unterschiedlichen Quellen reagieren muss. Ein GUI mit einem einzigen Thread, der sowohl Input (Touch Screen, Keypad, etc.) und Output (Display, etc.) besitzt ist viel einfacher zu implementieren, da alle anstehenden Events sequentiell verarbeitet werden können. Das GUI ist single-threaded, normale Applikation sind in der Regel aber multi-threaded. Denn ein umfangreicher, aufwändiger Arbeitsschritt darf das GUI nicht blockieren. Solche Arbeiten sind asynchron, d.h. ausserhalb des GUI-Threads auszuführen, so dass das GUI schnell und reaktionsfähig bleibt. Ein Android gibt es nun zwei typische Fehlerdialoge die der Benutzer bei schlecht geschriebenen Anwendungen zu sehen bekommen kann. Der eine ist der Force Close (FC) (linkes Bild) und der andere ist der Äpplication Not RespondingDialog (ANR) (rechtes Bild).

6.1. Variante 1 Background Thread - runonUIThread

```
 public class MainActivity extends Activity {
public static final String TAG = "HelloWorldAsynTask";
3 private TextView helloWorldView;
private ColorThread colorTask;
private RGBColor actColor = new RGBColor();
private Runnable updateBackground = new Runnable() {
#6
8 @Override
public void run() {
synchronized (actColor) {
helloWorldView.setBackgroundColor(
#4
13 Color.rgb(actColor.r,actColor.g, actColor.b));
Log.d(TAG, actColor.r + " " + actColor.g + " " + actColor.b);
}
}
};
18 @Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
helloWorldView = (TextView) findViewById(R.id.txtView);
23 helloWorldView.setText("Hello World");
}
public void start(View view) {
colorTask = new ColorTask();
colorTask.start();
28 #2
#3
}
public void stop(View view) {
colorTask.stopTask();
33 colorTask = null;
}
class ColorTask extends Thread {
private boolean isRunning = true;
#1
38 @Override
public void run() {
while (isRunning) {
synchronized (actColor) {
actColor.r = (int) Math.round(Math.random() * 255);
43 #4
actColor.g = (int) Math.round(Math.random() * 255);
actColor.b = (int) Math.round(Math.random() * 255);
}
runOnUiThread(updateBackground);
48 try {
```

```
#5
Thread.sleep(1000);
} catch (InterruptedException e) {
isRunning = false;
}
}
}
public void stopTask() {
#7
isRunning = false;
}
}
class RGBColor {
int r;
int g;
int b;
}
}
```

6.2. Async Tasks

```
 public class MainActivity extends Activity {
public static final String TAG = "HelloWorldAsynTask";
private TextView helloWorldView;
private AsyncTask<Void, RGBColor, Void> colorTask;
private void updateHelloWorldView(RGBColor c) {
helloWorldView.setBackgroundColor(Color.rgb(c.r,c.g, c.b));
Log.d(TAG, c.r + " " + c.g + " " + c.b);
}
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
helloWorldView = (TextView) findViewById(R.id.txtView);
helloWorldView.setText("Hello World");
}
public void start(View view) {
colorTask = new ColorTask();
colorTask.execute();
#2
#3
}
public void stop(View view) {
colorTask.cancel(true);
colorTask = null;
}
class ColorTask extends AsyncTask<Void, RGBColor, Void> {
private boolean error;
#1
private RGBColor color = new RGBColor();
@Override
protected void onPreExecute() {
#4
error = false;
}
@Override
protected Void doInBackground(Void... params) {
while (!error && !isCancelled()) {
#5
#6
try {color.r = (int) Math.round(Math.random() * 255);
color.g = (int) Math.round(Math.random() * 255);
color.b = (int) Math.round(Math.random() * 255);
publishProgress(color);
#7
Thread.sleep(1000);
} catch (InterruptedException e) {
error = false;
}
}
return null;
}
@Override
protected void onProgressUpdate(RGBColor... values) {
#8
```

```
updateHelloWorldView(values[0]);
#9
}
}
class RGBColor {
int r;
int g;
int b;
}
}
```

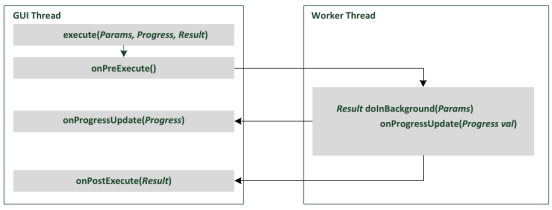

Abbildung 6.1.: figure

6.3. Service und Broadcastreciever

Die Service Klasse ist ein weiteres Konstrukt von Android, um Aufgaben im Hintergrund laufen zu lassen. Genau wie das Activity ist der Service eine Klasse, die von Android zur Laufzeit bereitgestellt wird und erweitert werden kann. Das bedeutet dass ein Service im File AndroidManifest.xml eingetragen werden muss, damit Android diesen Dienst erkennen kann. Wichtig ist die folgende Bemerkung im API zur Klasse Service: Note that services, like other application objects, run in the main thread of their hosting process. This means that, if your service is going to do any CPU intensive (such as MP3 playback) or blocking (such as networking) operations, it should spawn its own thread in which to do that work."Da der Service einen eigenen Life-Cycle hat und unabhängig von einer Activity laufen kann, ist die Kommunikation zwischen Service und Activity aufwändiger. Eine elegante Variante nutzt den Broadcast. Broadcasts sind im Prinzip systemweit in Form von Intents versandte Nachrichten, die von allen Android-Komponenten des Typs Broadcast Receiver empfangen werden können, die sich für den jeweiligen Nachrichtentyp registriert haben.

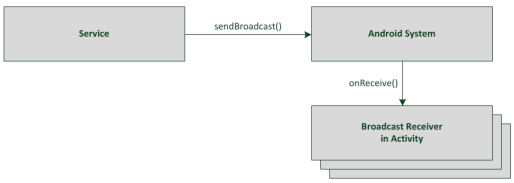

Abbildung 6.2.: figure

```
public class MainActivity extends Activity {
public static final String TAG = "HelloWorld-MainActivity";
private TextView helloWorldView;
private BroadcastReceiver receiver = new BroadcastReceiver() {
@Override
public void onReceive(Context ctx, Intent intent) {
RGBColor color = (RGBColor) intent.getSerializableExtra(
ColorService.COLOR_VALUE);
if (color != null) {
updateHelloWorldView(color);
}
}
};
private void updateHelloWorldView(RGBColor c) {
helloWorldView.setBackgroundColor(Color.rgb(c.r,c.g, c.b));
Log.d(TAG, c.r + " " + c.g + " " + c.b);
}
#1
#2
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
helloWorldView = (TextView) findViewById(R.id.txtView);
helloWorldView.setText("Hello World");
}
@Override
protected void onResume() {
super.onResume();
LocalBroadcastManager.getInstance(this).registerReceiver(
receiver, new IntentFilter(ColorService.NOTIFICATION));
}
@Override
protected void onPause() {
super.onPause();
LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
}
public void start(View view) {
startService(new Intent(this, ColorService.class));
}
public void stop(View view) {
stopService(new Intent(this, ColorService.class));
}
#3
#4
#5
```

```
 public class ColorService extends Service {
private static String TAG = "HelloWorld-ColorServiceBroadcast";
public static final String COLOR_VALUE = "COLOR";
public static final String NOTIFICATION = "ch.fhnw.edu.helloworld";
private Timer timer = null;
@Override
public void onCreate() {
Log.d(TAG, "onCreate() called");
}
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
startColoring();
return Service.START_NOT_STICKY;
}
@Override
public IBinder onBind(Intent arg0) {
return null;
}
@Override
public void onDestroy() {
super.onDestroy();
stopColoring();
Log.d(TAG, "onDestroy() called");
}
public void startColoring() {
if (timer == null) {
timer = new Timer();
timer.schedule(new ColorTask(this), 0, 1000);
}
}
public void stopColoring() {
if (timer != null) {
timer.cancel();
timer = null;
}
}
private class ColorTask extends TimerTask {
private RGBColor color = new RGBColor();
private Service service;
public ColorTask(Service service) {
this.service = service;
}
#1
#2
@Override
public void run() {
color.r = (int) Math.round(Math.random() * 255);
color.g = (int) Math.round(Math.random() * 255);
```

```
color.b = (int) Math.round(Math.random() * 255);
Intent intent = new Intent(NOTIFICATION);
intent.putExtra(COLOR_VALUE, color);
LocalBroadcastManager.getInstance(service).sendBroadcast(intent);
Log.d(TAG, "color is " + color);
}
};
}
#3
#4
#5
```

6.4. Messenger und Service

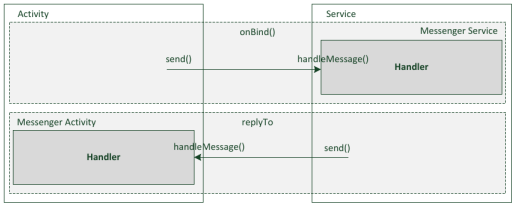

Abbildung 6.3.: figure

Bind to Color Service

```
protected void onResume() {
super.onResume();
bindService(new Intent(this, ColorService.class), svcConn,
Context.BIND_AUTO_CREATE);
}
protected void onPause() {
super.onPause();
if (serviceMessenger != null) {
unbindService(svcConn);
}
}
```

```
private ServiceConnection svcConn = new ServiceConnection() {
public void onServiceConnected(ComponentName className, IBinder service) {
serviceMessenger = new Messenger(service);
}
public void onServiceDisconnected(ComponentName className) {
serviceMessenger = null;
}
};
```

svc instanzieren von messenger in Oncreate

```
myMessenger = new Messenger(new IncomingHandler(this));
...
static class IncomingHandler extends Handler {
private final WeakReference<MainActivity> mainActivity;
public IncomingHandler(MainActivity activity) {
this.mainActivity = new WeakReference<MainActivity>(activity);
}
@Override
public void handleMessage(Message msg) {
switch (msg.what) {
case ColorService.MSG_SET_VALUE:
RGBColor color = (RGBColor) msg.getData().getSerializable(
ColorService.COLOR_VALUE);
mainActivity.get().updateHelloWorldView(color);
break;
default:
super.handleMessage(msg);
}
}
}
```

```
                                    public void start(View view) {
Message msg = Message.obtain(null, ColorService.MSG_START_SERVICE);
msg.replyTo = myMessenger;
try {
serviceMessenger.send(msg);
} catch (RemoteException e) {
Log.e(TAG, e.getMessage());
}
}
```

Start Service mit Message

Color Service

Sending Message

```
                                                Message msg = Message.obtain();
msg.what = ColorService.MSG_SET_VALUE;
Bundle data = new Bundle();
data.putSerializable(COLOR_VALUE, color);
msg.setData(data);
this.messenger.send(msg);
```

```
                                        static class IncomingHandler extends Handler {
private final WeakReference<ColorService> service;
public IncomingHandler(ColorService service) {
this.service = new WeakReference<ColorService>(service);
}
@Override
public void handleMessage(Message msg) {
switch (msg.what) {
case MSG_START_SERVICE:
service.get().mainMessenger = msg.replyTo;
service.get().startColoring();
break;
case MSG_STOP_SERVICE:
service.get().stopColoring();
break;
default:
super.handleMessage(msg);
}
}
}
```

7. Persistenz

Write Amplifikation Faktor als Kenngrosse. Von Write-Amplification spricht man, wenn ein Solid State Drive deutlich mehr Daten schreiben muss, als sich eigentlich ändern.

7.1. Ausgangslage

SSDs sind in Speicherblöcke (512 KB) unterteilt. Ein Speicherblock besteht aus Pages zu 4 oder 8 KB. Einzelne Pages können beschrieben werden. Aber Pages lassen sich erst beschreiben, nachdem sie gelöscht worden sind. Einzelne Page können nicht gelöscht werden. Nur komplette Blöcke können gelöscht werden.

7.2. Read Write Cycle

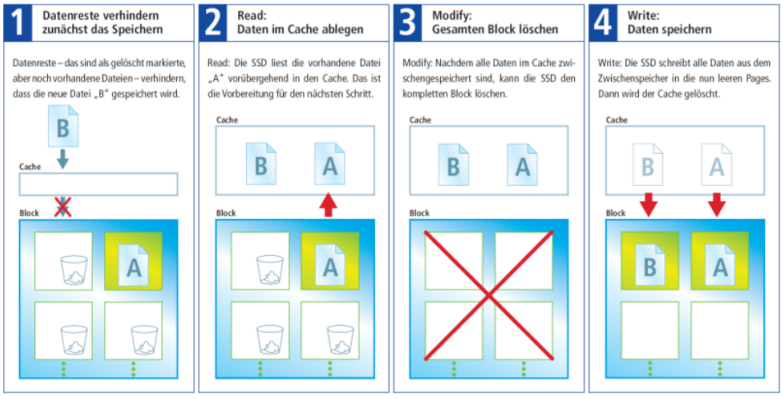

Abbildung 7.1.: figure

7.3. Worst Case

Situation:
SSD-Speicherblock (512 KB) zu 128 Pages (4 KB)
127 Pages mit gültigen Daten
1 Page ungültig
Neue Nutzdaten à 4 KB (= 1 Page) speichern
Folge:
127 Pages (508 KB) lesen
Nutzdaten (1 Page = 4 KB) ergänzen
128 Pages (512 KB) schreiben
Amplification Faktor: 512 KB / 4 KB = 128

7.4. Fazit

Eine geringer Write-Amplification-Faktor ist eine Kennzahl für die Effizienz der Speicher- und Speicherverwaltungsalgorithmen der SSD. Die beste Leistung erzielt eine SSD im fabrikneuen Zustand. Bei vollem Speicher kann es passieren, dass die SSD erst einen ganzen Block von 512 KByte lesen, anschliessend um die neuen Daten ergänzen und alles zusammen wieder zurückschreiben muss

8. Persistenz II

8.1. Key Value

Primitive Datentypen in Form von key/value-Paaren können dauerhaft mit der Klasse SharedPreferences in einer Datei persistiert werden. In der Regel werden solche SharedPreferences eingesetzt, um Benutzereinstellungen oder Zustände des Programms zu speichern. Die Daten werden dabei in einem Preference File abgelegt, das nur der Applikation zur Verfügung steht. Das File kann aber auch lokal zu einer entsprechenden Activity gehalten werden, um z.B. explizit die Zustände dieser Activity zwischenspeichern zu können, oder das Preference File steht der gesamten Applikation zur Verfügung. Mit folgendem Code, die Methode getSharedPreferences() ist in jeder Activity vorhanden, werden die Preferences aus einem benannten File ausgelesen:

```
SharedPreferences settings = getSharedPreferences("MyPrefsFile", 0);
boolean silent = settings.getBoolean("silentMode", false);
```

Das Speichern wird über einen entsprechenden Editor realisiert:

```
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("silentMode", mySilentMode);
editor.commit();
```

8.2. Preferece Activity

```
 <?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
<ListPreference android:title="Choose a robot"
android:summary="This preference allows to select an robot type"
#1
#2
#3
android:key="robottype"
android:entries="@array/robotTypes" #4
#5
android:entryValues="@array/robotIDs" />
</PreferenceScreen>
```

---

values in values/array.xml.

```
 public class Settings extends PreferenceActivity
@Override
{
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
addPreferencesFromResource(R.layout.settings);
}
}
```

8.3. Loading Bitmaps

Internal/External Storage Da dem Android-System ein Linux-Kernel zugrunde liegt, überrascht es nicht, dass man es auch mit einem Linux-Dateisystem zu tun haben. Als Basis-Dateisystem bezeichnet man das fest installierte Speichermedium des Geräts. Das Basis-Dateisystem lässt sich um Speicherkarten (SD-Karten) erweitern, deren Speicherplatz nicht an das Gerät gebunden ist. Mit der Installation einer Anwendung auf dem Android-Gerät wird automatisch ein privates Anwendungsverzeichnis /data/data/packagename.der.anwendung/ erstellt. In dieses Verzeichnis darf nur die Anwendung selbst Schreiboperationen ausführen. Unterhalb des privaten Anwendungsverzeichnisses werden z.B. alle Datenbanken der Anwendung abgelegt. Die Zugriffsmethoden auf das Dateisystem werden vom android.content.Context bereitgestellt. Hier findet man Methoden wie getDir(), getCacheDir(), getFilesDir(), openFileInput(), openFileOutput(), usw. Dateien können aber auch als Ressourcen in der Anwendung selber abgelegt werden. Zum Beispiel soll als Begrüssung das Bild

text_welcome.png gezeigt werden. Dieses Bild wird in den Ordner /res/drawable gelegt. Da dieser res-Ordner vom Android-System verwaltet wird, wird automatisch ein Key generiert und in der R-Klasse aufgenommen. Mit

```
    Resources res = context.getResources();
2 Bitmap greetingBitmap = BitmapFactory.decodeResource(res, R.drawable.text_welcome);
  greetingBitmap = Bitmap.createScaledBitmap(greetingBitmap,
  viewWidth, viewHeight, true);
  canvas.drawBitmap(greetingBitmap, 0, 0, null);
```

8.4. SQLITE

In Android fehlt JDBC. Deshalb muss ein anderer Weg gewählt werden. Die Klasse SQLiteOpenHelper stellt eine entsprechende Funktionalität bereit. Man muss eine Unterklasse von SQLiteOpenHelper implementieren. In dieser Unterklasse wird definiert, wie die Datenbank erstellt und bei einer Versionsänderung (der Applikation) migriert werden soll. In der Unterklasse überschreibt man deshalb folgende Methoden von SQLiteOpenHelper:
public void onCreate(SQLiteDatabase db) Wird aufgerufen, wenn die Datenbank bisher nicht existiert. Enthält typischerweise Anweisungen zum Erzeugen der Datenbank-Relationen.
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) Wird aufgerufen, wenn beim Öffnen der Datenbank eine höhere Versionsnummer angegeben wird, als beim letzten Öffnen. Enthält Anweisungen zum Upgrade der Datenbank auf newVersion.

Handler in ColorService `SQLiteDatabase db = dbHelper.getReadableDatabase();`

```
SQLiteDatabase db = dbHelper.getWriteableDatabase();
Über diese beiden Methoden erhält man den Zugriff auf die Datenbank. Zum Beispiel wie folgt:
private void insertOneHundredRecords() {
SQLiteDatabase db = dbHelper.getWritableDatabase();
#1
for (int i = 0; i < 100; i++) {
ContentValues values = new ContentValues();
#2
values.put("Id", i);
values.put("HelloString", "Hello World"); #3
db.insert(SAMPLE_TABLE_NAME, null, values); #4
}
dbHelper.close();
#5
}
```

Teil I.

Code

9. Layout

Listing 9.1: Layout1

```
package ch.fhnw.edu.helloworld;

import android.content.Context;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class HelloWorldLayout extends RelativeLayout {
  public HelloWorldLayout(Context context) {
    super(context);

      // android:layout_width="match_parent"
      // android:layout_height="match_parent"
    LayoutParams lpView = new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.
        MATCH_PARENT);
    this.setLayoutParams(lpView);

    // android:paddingBottom="@dimen/activity_vertical_margin"
    int bottom = (int) getResources().getDimension(R.dimen.activity_vertical_margin);
      // android:paddingLeft="@dimen/activity_horizontal_margin"
    int left = (int) getResources().getDimension(R.dimen.activity_horizontal_margin);
      // android:paddingRight="@dimen/activity_horizontal_margin"
    int right = (int) getResources().getDimension(R.dimen.activity_horizontal_margin);
      // android:paddingTop="@dimen/activity_vertical_margin"
    int top = (int) getResources().getDimension(R.dimen.activity_vertical_margin);
    this.setPadding(left, top, right, bottom);

    TextView tv = new TextView(context);
    // android:layout_width="wrap_content"
    // android:layout_height="wrap_content"
    lpView = new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
    tv.setLayoutParams(lpView);
    // android:text="@string/hello_world" />
    tv.setText(getResources().getString(R.string.hello_world));

    this.addView(tv);
  }
}
```

Listing 9.2: Layout2

```
package ch.fhnw.edu.helloworld;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.app.ActionBarActivity;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

public class MainActivity extends ActionBarActivity {
  private final String KEY_MYVALUE= "value";
  private int value;

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

    Log.d("HelloWorld", "onCreate() called " + savedInstanceState);

```

```
    if (savedInstanceState == null) {
      getSupportFragmentManager().beginTransaction()
          .add(R.id.container, new PlaceholderFragment()).commit();
    }

  }

  @Override
  protected void onRestoreInstanceState(Bundle savedInstanceState) {
    value = savedInstanceState.getInt(KEY_MYVALUE);
    super.onRestoreInstanceState(savedInstanceState);
  }


  @Override
  protected void onSaveInstanceState(Bundle outState) {
    outState.putInt(KEY_MYVALUE, value);
    super.onSaveInstanceState(outState);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }

  @Override
  protected void onStart() {
    super.onStart();
    Log.d("HelloWorld", "onStart() called");
  }

  @Override
  protected void onResume() {
    super.onResume();
    Log.d("HelloWorld", "onResume() called, value is " + value);
    if (value == 0) {
      value = 1111;
      Log.d("HelloWorld", "value set to " + value);
    }
  }

  @Override
  protected void onPause() {
    super.onPause();
    Log.d("HelloWorld", "onPause() called");
  }

  @Override
  protected void onStop() {
    super.onStop();
    Log.d("HelloWorld", "onStop() called");
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    Log.d("HelloWorld", "onDestroy() called");
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
      return true;
    }
    return super.onOptionsItemSelected(item);
  }

  /**
   * A placeholder fragment containing a simple view.
   */
```

```
    public static class PlaceholderFragment extends Fragment {

      public PlaceholderFragment() {
103   }

      @Override
      public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
          savedInstanceState) {
        //View rootView = inflater.inflate(R.layout.fragment_main, container, false);
108     View rootView = new HelloWorldLayout(this.getActivity());
        return rootView;
      }
    }

113 }
```

10. Intents

Listing 10.1: MainActivity

```
package ch.fhnw.edu.helloworld;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.app.ActionBarActivity;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

public class MainActivity extends ActionBarActivity {
  private final String KEY_MYVALUE= "value";
  private int value;

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

    Log.d("HelloWorld", "onCreate() called " + savedInstanceState);

    if (savedInstanceState == null) {
      getSupportFragmentManager().beginTransaction()
          .add(R.id.container, new PlaceholderFragment()).commit();
    }

  }

  @Override
  protected void onRestoreInstanceState(Bundle savedInstanceState) {
    value = savedInstanceState.getInt(KEY_MYVALUE);
    super.onRestoreInstanceState(savedInstanceState);
  }


  @Override
  protected void onSaveInstanceState(Bundle outState) {
    outState.putInt(KEY_MYVALUE, value);
    super.onSaveInstanceState(outState);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }

  @Override
  protected void onStart() {
    super.onStart();
    Log.d("HelloWorld", "onStart() called");
  }

  @Override
  protected void onResume() {
    super.onResume();
    Log.d("HelloWorld", "onResume() called, value is " + value);
    if (value == 0) {
      value = 1111;
      Log.d("HelloWorld", "value set to " + value);
    }
```

```
    }

    @Override
    protected void onPause() {
      super.onPause();
      Log.d("HelloWorld", "onPause() called");
    }

    @Override
    protected void onStop() {
      super.onStop();
      Log.d("HelloWorld", "onStop() called");
    }

    @Override
    protected void onDestroy() {
      super.onDestroy();
      Log.d("HelloWorld", "onDestroy() called");
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
      // Handle action bar item clicks here. The action bar will
      // automatically handle clicks on the Home/Up button, so long
      // as you specify a parent activity in AndroidManifest.xml.
      int id = item.getItemId();
      if (id == R.id.action_settings) {
        return true;
      }
      return super.onOptionsItemSelected(item);
    }

      public void onButtonClick(View v) {
        // call activity using an implicit intent
          Intent intent = new Intent("ch.fhnw.edu.helloworld.HELLOWORLD");
        // call activity using an explicit intent
//        Intent intent = new Intent(this, MessageActivity.class);

          startActivity(intent);
      }

    /**
     * A placeholder fragment containing a simple view.
     */
    public static class PlaceholderFragment extends Fragment {

      public PlaceholderFragment() {
      }

      @Override
      public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
          savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_main, container, false);
        return rootView;
      }
    }

}
```

Listing 10.2: MessageActivity

```
package ch.fhnw.edu.helloworld;

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

public class MessageActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.message);
    }
}
```

Listing 10.3: MessageActivityv2

```
package ch.fhnw.edu.helloworld.v2;

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

public class MessageActivity extends Activity {

  /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
//        requestWindowFeature(Window.FEATURE_NO_TITLE);
        MessageFromBitmap message = new MessageFromBitmap(this);
        setContentView(message);
    }

}
```

Listing 10.4: MessageActivityBitmap

```
package ch.fhnw.edu.helloworld.v2;

import ch.fhnw.edu.helloworld.R;
import ch.fhnw.edu.helloworld.R.drawable;
import android.app.Activity;
import android.content.Context;
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.view.View;


public class MessageFromBitmap extends View {
  private HelloMessage helloMessage;

  public MessageFromBitmap(Activity context) {
    super(context);
    helloMessage = new HelloMessage(context);
  }

  @Override
  protected void onDraw(Canvas canvas) {
    helloMessage.draw(canvas);
  }

  @Override
  protected void onSizeChanged(int width, int height, int oldw, int oldh) {
    helloMessage.adjustTextLayout(width, height);
  }
}

class HelloMessage {
  private Bitmap helloBitmap;
  private int centerX;
  private int centerY;

  public HelloMessage(Context context) {
    Resources res = context.getResources();
    helloBitmap = BitmapFactory.decodeResource(res, R.drawable.helloworld);
  }

  void draw(Canvas canvas) {
    canvas.drawBitmap(helloBitmap, centerX, centerY, null);
  }

  public void adjustTextLayout(int viewWidth, int viewHeight) {
    centerX = (viewWidth/2) - (helloBitmap.getWidth()/2);
    centerY = (viewHeight/2) - (helloBitmap.getHeight()/2);
  }
}
```

11. Responsive Design

Listing 11.1: mainactivitylayout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <fragment android:name="ch.fhnw.edu.helloworld.MainFragment"
        android:id="@+id/main_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Listing 11.2: mainactivitytwopane

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <fragment android:name="ch.fhnw.edu.helloworld.MainFragment"
        android:id="@+id/main_container"
        android:layout_width="wrap_content"
        android:layout_height="match_parent" />

    <FrameLayout
        android:id="@+id/helloworld_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

    </FrameLayout>

</LinearLayout>
```

Listing 11.3: fragment main

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    tools:context="ch.fhnw.edu.helloworld.MainActivity$PlaceholderFragment" >

  <TextView
    android:gravity="center"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:text="@string/welcome" />

  <LinearLayout android:orientation="vertical"
    android:gravity="bottom"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onButtonClick"
        android:text="@string/press" />

  </LinearLayout>

</LinearLayout>
```

Listing 11.4: fragment message

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:gravity="center">

  <ImageView android:src="@drawable/helloworld"
      android:contentDescription="@string/hello_world"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />

</LinearLayout>
```

Listing 11.5: MainActivity

```
package ch.fhnw.edu.helloworld;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.view.View;

public class MainActivity extends FragmentActivity {
  private boolean isTwoPane;

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

    if (findViewById(R.id.helloworld_container) != null) {
      isTwoPane = true;
    } else {
      isTwoPane = false;
    }
  }

    public void onButtonClick(View v) {
      if (isTwoPane) {
        FragmentManager fm = getSupportFragmentManager();
        Fragment f = fm.findFragmentById(R.id.helloworld_container);
        if (f == null) {
          fm.beginTransaction().add(R.id.helloworld_container, new MessageFragment()).
              commit();
        }
      } else {
        Intent intent = new Intent(this, MessageActivity.class);
          startActivity(intent);
      }
    }

}
```

Listing 11.6: MainFragment

```
package ch.fhnw.edu.helloworld;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class MainFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
        savedInstanceState) {
      View view = inflater.inflate(
          R.layout.fragment_main,
```

```
            container,
            false); //!!! this is important
        return view;
    }
}
```

Listing 11.7: MessageActivity

```
package ch.fhnw.edu.helloworld;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MessageActivity extends FragmentActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.fragment_message);
    }
}
```

Listing 11.8: MessageFragment

```
package ch.fhnw.edu.helloworld;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class MessageFragment extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
      savedInstanceState) {
    return inflater.inflate(R.layout.fragment_message, container, false);
  }


  @Override
  public void onDestroy() {
    super.onDestroy();
    Log.d("HelloWorld", "fragment destroyed");
  }

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("HelloWorld", "fragment created");
  }
}
```

12. Background Tasks

Listing 12.1: ColorService

```
1  package ch.fhnw.edu.helloworld;

   import java.lang.ref.WeakReference;
   import java.util.Timer;
   import java.util.TimerTask;
6
   import android.app.Service;
   import android.content.Intent;
   import android.os.Bundle;
   import android.os.Handler;
11 import android.os.IBinder;
   import android.os.Message;
   import android.os.Messenger;
   import android.os.RemoteException;
   import android.util.Log;
16
   public class ColorService extends Service {
     private static String TAG = "HelloWorld-ColorService";

     public static final int MSG_STOP_SERVICE = 0;
21   public static final int MSG_START_SERVICE = 1;
     public static final int MSG_SET_VALUE = 2;
     public static final String COLOR_VALUE = null;

     private Messenger myMessenger;
26   private Messenger mainMessenger;
     private Timer timer = null;

     @Override
     public void onCreate() {
31     myMessenger = new Messenger(new IncomingHandler(this));
     }

     @Override
     public IBinder onBind(Intent arg0) {
36     Log.d(TAG, "onBind() called");
       return myMessenger.getBinder();
     }

     @Override
41   public void onDestroy() {
       stopColoring();
       Log.d(TAG, "onDestroy() called");
       super.onDestroy();
     }
46
       static class IncomingHandler extends Handler {
       private final WeakReference<ColorService> service;

         public IncomingHandler(ColorService service) {
51       this.service = new WeakReference<ColorService>(service);
       }

           @Override
           public void handleMessage(Message msg) {
56             switch (msg.what) {
                   case MSG_START_SERVICE:
                     service.get().mainMessenger = msg.replyTo;
                     service.get().startColoring();
                       break;
61                 case MSG_STOP_SERVICE:
                     service.get().stopColoring();
                     break;
                   default:
                       super.handleMessage(msg);
```

```
66              }
            }
        }

    private void startColoring() {
71      if (timer == null) {
          timer = new Timer();
          timer.schedule(new ColorTask(mainMmessenger), 0, 1000);
        }
    }
76
    private void stopColoring() {
      if (timer != null) {
        timer.cancel();
        timer = null;
81    }
    }

    private class ColorTask extends TimerTask {
      private Messenger messenger;
86    private RGBColor color = new RGBColor();

      public ColorTask(Messenger messenger) {
        this.messenger = messenger;
      }
91
      @Override
      public void run() {
        if (this.messenger != null) {
          try {
96          color.r = (int) Math.round(Math.random() * 255);
            color.g = (int) Math.round(Math.random() * 255);
            color.b = (int) Math.round(Math.random() * 255);

            Message msg = Message.obtain();
101         msg.what = ColorService.MSG_SET_VALUE;
            Bundle data = new Bundle();
            data.putSerializable(COLOR_VALUE, color);
            msg.setData(data);
            this.messenger.send(msg);
106       } catch (RemoteException e) {
            Log.e(TAG, e.getMessage());
          }
        }
        Log.d(TAG, "color is " + color);
111   }
    };
  }
```

Listing 12.2: MainActivity

```
  package ch.fhnw.edu.helloworld;
2
  import java.lang.ref.WeakReference;

  import android.app.Activity;
  import android.content.ComponentName;
7 import android.content.Context;
  import android.content.Intent;
  import android.content.ServiceConnection;
  import android.graphics.Color;
  import android.os.Bundle;
12 import android.os.Handler;
  import android.os.IBinder;
  import android.os.Message;
  import android.os.Messenger;
  import android.os.RemoteException;
17 import android.util.Log;
  import android.view.View;
  import android.widget.TextView;

  public class MainActivity extends Activity {
22  public static final String TAG = "HelloWorld-MainActivity";
    private TextView helloWorldView;
    private Messenger serviceMessenger;
```

```
  private Messenger myMessenger;

27  private void updateHelloWorldView(RGBColor c) {
      helloWorldView.setBackgroundColor(Color.rgb(c.r,c.g, c.b));
      Log.d(TAG, c.r + " " + c.g + " " + c.b);
    }

32  @Override
    protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      helloWorldView = (TextView) findViewById(R.id.txtView);
37    helloWorldView.setText("Hello World");

      myMessenger = new Messenger(new IncomingHandler(this));
    }

42  @Override
    protected void onResume() {
      super.onResume();
      bindService(new Intent(this, ColorService.class), svcConn, Context.BIND_AUTO_CREATE);
    }
47
    @Override
    protected void onPause() {
      super.onPause();
      if (serviceMessenger != null) {
52      unbindService(svcConn);
      }
    }

    private ServiceConnection svcConn = new ServiceConnection() {
57    public void onServiceConnected(ComponentName className, IBinder service) {
        serviceMessenger = new Messenger(service);
        Log.d(TAG, "service is connected");
      }

62    public void onServiceDisconnected(ComponentName className) {
        serviceMessenger = null;
        Log.d(TAG, "service is disconnected");
      }
    };
67
    public void start(View view) {
      Message msg = Message.obtain(null, ColorService.MSG_START_SERVICE);
      msg.replyTo = myMessenger;
      try {
72      serviceMessenger.send(msg);
      } catch (RemoteException e) {
        Log.e(TAG, e.getMessage());
      }
    }
77
    public void stop(View view) {
      Message msg = Message.obtain(null, ColorService.MSG_STOP_SERVICE);
      try {
        serviceMessenger.send(msg);
82    } catch (RemoteException e) {
        Log.e(TAG, e.getMessage());
      }
    }

87  static class IncomingHandler extends Handler {
      private final WeakReference<MainActivity> mainActivity;

      public IncomingHandler(MainActivity activity) {
        this.mainActivity = new WeakReference<MainActivity>(activity);
92    }

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
97              case ColorService.MSG_SET_VALUE:
                  RGBColor color = (RGBColor) msg.getData().getSerializable(ColorService.
                      COLOR_VALUE);
                  mainActivity.get().updateHelloWorldView(color);
```

```
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }
}
```

Listing 12.3: RGBColor

```
package ch.fhnw.edu.helloworld;

import java.io.Serializable;

@SuppressWarnings("serial")
public class RGBColor implements Serializable {
  int r;
  int g;
  int b;

  @Override
  public String toString() {
    return "r="+r+", g="+g+", b="+b;
  }
}
```

13. Content Provider

Listing 13.1: DBHelper

```
package ch.fhnw.edu.mad;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper {
  public static final String DATABASE_NAME = "HelloWorldDB";
  public static final String DATABASE_TABLE_NAME = "HelloTable";
  private static final int DATABASE_VERSION = 1;

  public DBHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE IF NOT EXISTS " + DATABASE_TABLE_NAME +
               " (Id INT, HelloString VARCHAR(255));");
  }

  @Override
  public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
    // TODO Auto-generated method stub
  }

}
```

Listing 13.2: MainActivity

```
package ch.fhnw.edu.mad;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteStatement;
import android.net.Uri;

public class HelloWorldContentProvider extends ContentProvider {
  public static final String AUTHORITY = "ch.fhnw.edu.mad.helloworld";
  private static final String CONTENT_AUTHORITY = "content://" + AUTHORITY;
  private static final Uri CONTENT_WORDS_URI = Uri.parse(CONTENT_AUTHORITY + "/words");
  private static final int Words = 1;
  private static final int Word_ID = 2;

  private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MATCH);

  static {
    sURIMatcher.addURI(AUTHORITY, "words", Words);
    sURIMatcher.addURI(AUTHORITY, "words/#", Word_ID);
  }

  private DBHelper dbHelper;

  @Override
  public String getType(Uri uri) {
    int match = sURIMatcher.match(uri);
    switch (match) {
    case Words:
      // Using Android's vendor-specific MIME format for multiple rows
```

```
        return "vnd.android.cursor.dir/vnd.ch.fhnw.edu.mad.helloworld.word";
      case Word_ID:
        // Using Android's vendor-specific MIME format for single row
        return "vnd.android.cursor.item/vnd.ch.fhnw.edu.mad.helloworld.word";
    default:
      return null;
    }
  }

  @Override
  public Uri insert(Uri uri, ContentValues values) {
    int match = sURIMatcher.match(uri);
    switch (match) {
    case Words:
      SQLiteDatabase db = dbHelper.getWritableDatabase();
      long newId = db.insert(DBHelper.DATABASE_TABLE_NAME,null,values);
      Uri rowUri = ContentUris.withAppendedId(CONTENT_WORDS_URI, newId);
      dbHelper.close();
      return rowUri;
    }
    return null;
  }

  @Override
  public int bulkInsert(Uri uri, ContentValues[] values) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    String sql = "INSERT INTO " + DBHelper.DATABASE_TABLE_NAME + " VALUES (?,?);";
    SQLiteStatement statement = db.compileStatement(sql);
    db.beginTransaction();
    for (int i = 0; i < 100; i++) {
      statement.clearBindings();
      statement.bindLong(1, values[i].getAsLong("Id"));
      statement.bindString(2, values[i].getAsString("HelloString"));
      statement.execute();
    }
    db.setTransactionSuccessful();
    db.endTransaction();
    dbHelper.close();
    return 100;

//    return super.bulkInsert(uri, values);
  }

  @Override
  public boolean onCreate() {
    dbHelper = new DBHelper(this.getContext());
    return true;
  }


  @Override
  public int update(Uri arg0, ContentValues arg1, String arg2, String[] arg3) {
    // TODO Auto-generated method stub
    return 0;
  }

  @Override
  public Cursor query(Uri uri, String[] projection, String selection,
      String[] selectionArgs, String sortOrder) {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public int delete(Uri arg0, String arg1, String[] arg2) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    int nr = db.delete(DBHelper.DATABASE_TABLE_NAME, null, null);
    dbHelper.close();
    return nr;
  }

}
```

Listing 13.3: HelloWorldContentProvider

```
package ch.fhnw.edu.mad;

import android.app.Activity;
import android.content.ContentValues;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;

public class MainActivity extends Activity implements OnClickListener {
  private static String TAG = "HelloWorld";
  private Uri wordsUri = Uri.parse("content://ch.fhnw.edu.mad.helloworld/words");

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    findViewById(R.id.standard_insert_button).setOnClickListener(this);
    findViewById(R.id.bulk_insert_button).setOnClickListener(this);
  }

  private void insertOneHundredRecords() {
    for (int i = 0; i<100; i++) {
       ContentValues values = new ContentValues();
       values.put("Id", i);
       values.put("HelloString", "Hello World");
       getContentResolver().insert(wordsUri, values);
    }
  }

  private void bulkInsertOneHundredRecords() {
    ContentValues[] items = new ContentValues[100];
    for (int i = 0; i < 100; i++) {
      ContentValues item = new ContentValues();
      item.put("Id", i);
      item.put("HelloString", "Hello World");
      items[i] = item;
    }
    getContentResolver().bulkInsert(wordsUri, items);
  }

  @Override
  public void onClick(View v) {
    int nr = getContentResolver().delete(wordsUri, null, null);
    Log.d(TAG, nr + " records deleted");
    long startTime = System.currentTimeMillis();
    if (v.getId()==R.id.standard_insert_button) {
      insertOneHundredRecords();
    } else {
      bulkInsertOneHundredRecords();
    }
    long diff = System.currentTimeMillis() - startTime;
    ((TextView)findViewById(R.id.exec_time_label)).setText(Long.toString(diff)+"ms");
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
  }

}
```

14. Sensors

Listing 14.1: Main

```
package edu.mad;

import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.graphics.Color;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.TextView;

public class Main extends Activity implements SensorEventListener {
  private float mLastX, mLastY;
  private SensorManager mSensorManager;
  private Sensor mAccelerometer;
  private final float NOISE = (float) 5.0;
  private View myView;
  private TextView textView;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.main);
    textView = (TextView) findViewById(R.id.textView);
    myView = (View) findViewById(R.id.view);
    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    listSensors();
  }

  private void listSensors() {
    List<Sensor> sensorList = mSensorManager.getSensorList(Sensor.TYPE_ALL);
    StringBuilder sensorString = new StringBuilder("Sensors:\n");
    for (Sensor sensor : sensorList) {
        sensorString.append(sensor.getName()).append(", \n");
    }
    textView.setText(sensorString);
  }

  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL
        );
  }

  protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // can be safely ignored for this demo
  }

  @Override
  public void onSensorChanged(SensorEvent event) {
    float x = event.values[0];
```

```
      float y = event.values[1];

67    float deltaX = Math.abs(mLastX - x);
      float deltaY = Math.abs(mLastY - y);
      if ((deltaX > NOISE) || (deltaY > NOISE)) {
        myView.setBackgroundColor(Color.GREEN);
      } else {
72      myView.setBackgroundColor(Color.RED);
      }
      mLastX = x;
      mLastY = y;
    }
77
  }
```