AKG
App-Entwicklung für Android

Schrittzähler App

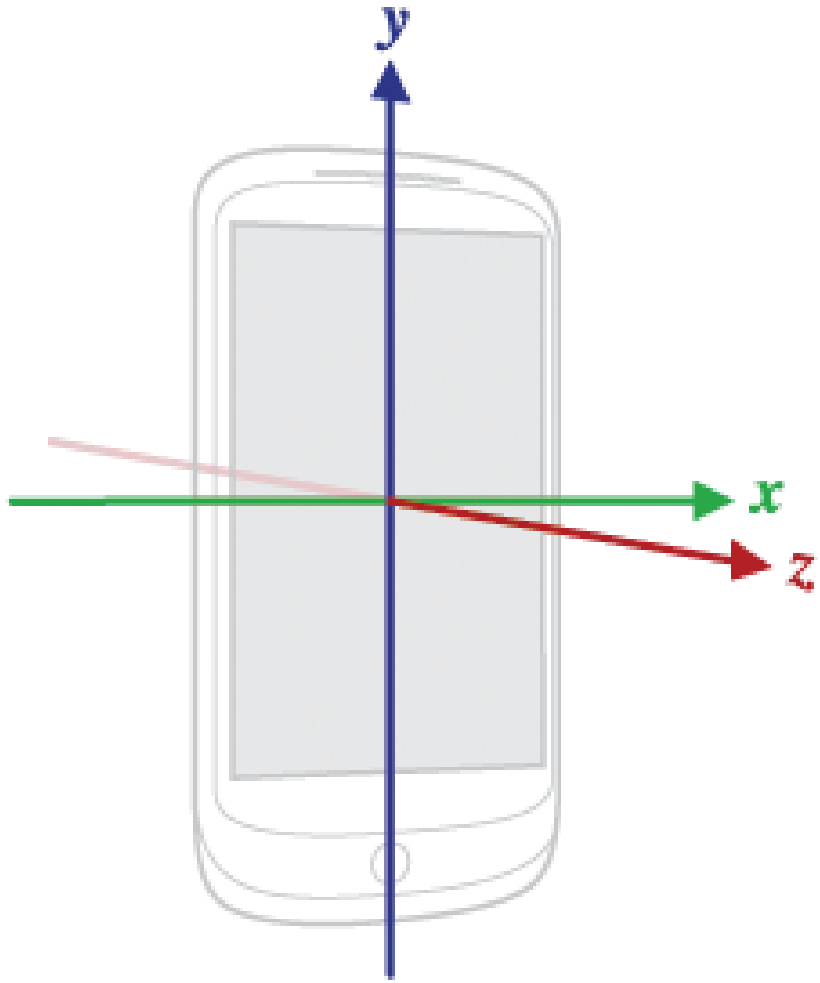
Prof. Dr. Paul Rawiel

Ein Schrittzähler

- Was für Sensoren werden gebraucht?
  - Beschleunigungssensoren
- was misst ein Beschleunigungssensor?
  - Beschleunigungen auf 3 Achsen, die senkrecht zueinander stehen
  - was wird gemessen, wenn
    - das Smartphone bewegungslos auf dem Tisch liegt?
    - es im Auto auf der Autobahn bei Tempo 120 km/h mitfährt?
    - es sich im freien Fall befindet?
- was folgt hieraus?
- was kann man mit Beschleunigungssensoren sonst noch machen?

Koordinatensystem

- Sensorkoordinatensystem des Smartphones
- unabhängig von der Orientierung des Bildschirms



Ein Schrittzähler

- Neues Projekt mit leerer Activity
- Projektname: Schrittzaehler
- Packagename: de.akgaea.schrittzaehler
- Auswahl Minimum SDK API

→Projekt erstellen

Projektstruktur wie bekannt

Layout vorbereiten

- das bereits vorhandene Textfeld
  - vergrößern
  - besser platzieren
  - Text-Attribute anpassen
- wird für die Anzeige der Schrittzahl verwendet
- Was brauchen wir sonst noch?
  - einen Button zum Zurücksetzen
- später
  - einen Button, um den Schrittzähler zu starten
  - einen Button, um ihn zu stoppen

Eintrag ins Manifest

- Screen Orientation auf Portrait setzen
- warum?
  - was passiert, wenn das Smartphone gedreht wird?
  - → Activity startet neu
  - → Schrittzählung startet wieder bei 0

```
<activity android:name=".MainActivity"
          android:label="@string/app_name"
          android:screenOrientation="portrait">
           …
```

MainActivity.java

- ähnlich wie beim Kompass
- private Attribute für
  - einen SensorManager
  - einen Sensor
  - eine Integer Variable für die Schritte
  - eine TextView
- in onCreate
  - textView auf das Element aus dem Layout setzen
  - textView = (TextView)findViewById(R.id.txtSchritte);
- neue Methode: aktualisierenAnzeige
  - setzen des Textes von textView auf schritte

MainActivity onCreate

- aufrufen von `aktualisiereAnzeige` in onCreate
  - Anzeige beim Start ist dadurch 0
- was muss in onCreate noch passieren?
  - onClickListener mit Button verbinden
  - sensorManager setzen
  - sensor setzen
  - textView setzen (siehe vorherige Folie)
- die onClickMehode

```
if (v.getId() == R.id.bzurueck){
        schritte = 0;
        aktualisierenAnzeige();
        }
```

SensorEventListener

- Es fehlt noch die Funktionalität, um Schritte zu zählen
- eine neue Klasse `ErschuetterungsListener`
  - übernimmt die Rolle des SensorEventListener

```
public class ErschuetterungsListener implements
SensorEventListener {
 @Override
    public void onSensorChanged(SensorEvent event) {}


@Override
    public void onAccuracyChanged(Sensor sensor, int
accuracy) {}
```

Der ErschütterungsListener

- Methode `onAccuracyChanged()` bleibt leer
- Methode `onSensorChanged` muss einen Schritt erkennen
- in dem Fall muss die MainActivity informiert werden und die Schrittzahl um 1 erhöhen
- Problem?
  - Kommunikation
  - über einen `Handler`

Handler

- Auf einem Smartphone passieren eine Menge Dinge gleichzeitig
- Was tut das Android-System dabei am meisten?
  - warten, bis etwas geschieht
  - Eingang einer Nachricht
  - klingeln
  - …
- Ereignisorientierte Verarbeitung
- → Ereigniswarteschlangen, sogenannte eventqueues
- taucht ein event auf, wird es (möglichst bald) verarbeitet

Handler

- der Handler erlaubt Zugriff auf Warteschlange
- man kann selbst Ereignisse erzeugen und in die Warteschlange stellen

```
private Handler handler;
```

Konstruktor:

```
public ErschuetterungsListener(Handler h){
        handler = h;
    }
```

Handler

in der MainActivity.java

```
private ErschuetterungsHandler erschuetterungsHandler =
new ErschuetterungsHandler();
    private class ErschuetterungsHandler extends Handler
{
        @Override
        public void handleMessage(Message msg) {
            schritte++;
            aktualisierenAnzeige();
        }
    }
```

Handler

- neue Klasse ErschuetterungsHandler ist eine lokale Klasse innerhalb der MainActivity
- → Zugriff auf Attribute und Methoden der Klasse
- Der ErschuetterungsListener kennt die Klasse noch nicht
- → übergeben als Konstruktor-Parameter

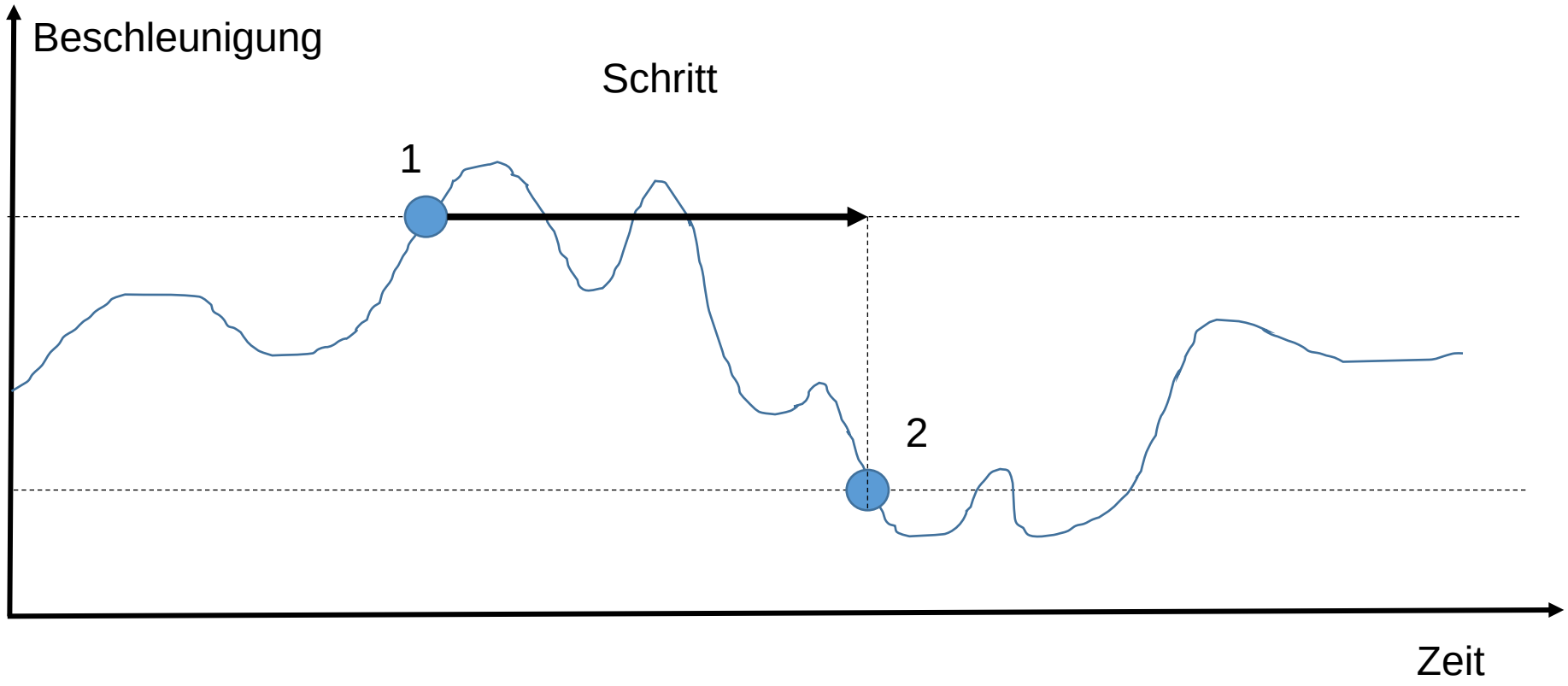
```
private ErschuetterungsHandler handler = new
ErschuetterungsHandler();

    private ErschuetterungsListener listener = new
ErschuetterungsListener(handler);
```

Erschütterungen

- in `onResume()`
  - verdrahten des Sensors mit dem Listener
  - `sManager.registerListener(listener, sensor, SensorManager.SENSOR_DELAY_GAME);`
- in `onPause()`
  - abmelden des Listeners
  - `sManager.unregisterListener(listener);`

Erkennen eines Schrittes



Erkennen eines Schritts

- warten auf die zu einem Schritt gehörenden Einzelereignisse
- wann beginnt ein Schritt
- wann endet ein Schritt
- Schwellwerte
  - am Punkt 1 der Graphik wird ein Schwellwert überschritten und der Schritt begonnen
  - am Punkt 2 wird der untere Schwellwert unterschritten und der Schritt beendet
  - die dazwischen liegenden Überschreitungen des oberen Schwellwerts müssen ignoriert werden

Erkennen eines Schrittes

```
public void onSensorChanged(SensorEvent event) {
 float y = event.values[2];
         if(!schrittBegonnen){
            if(y > schwellwert){
                schrittBegonnen = true;
                handler.sendEmptyMessage(1);
            }
        }else{
            if(y < -schwellwert){
                schrittBegonnen=false;
            }
        }
```

Das war‘s …

- Der Schrittzähler kann ausprobiert werden
- aber
  - was passiert, wenn die App in den Hintergrund geht?

Hintergrund-Services

- bisher: Programmlogik in Activities
- andere Möglichkeit: *Services*
  - Dienste, die im Hintergrund laufen
  - unabhängig von der Benutzeroberfläche
  - ein Schrittzähler muss nicht ständig aktiv sein, sondern kann auch im Hintergrund laufen
- Eine neue Service-Klasse
- Eigene Klasse, die von der Basisklasse Service erbt

Schrittzähler-Service

```
public class SchrittzaehlerService extends Service {
    private SensorManager sManager;
    private Sensor sensor;
     …

@Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

Schrittzähler-Service

- die Attribute, die bisher in der Activity verwendet wurden, kommen jetzt hierher
- inklusive ErschuetterungsHandler
- Services haben einen Lebenszyklus ähnlich wie Activities
  - onCreate
  - onDestroy

Schrittzähler-Service

```
@Override
    public void onCreate() {
        super.onCreate();
        sManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
        sensor = sManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
        sManager.registerListener(listener, sensor, SensorManager.SENSOR_DELAY_GAME);
    }

    @Override
    public void onDestroy() {
        sManager.unregisterListener(listener);
        super.onDestroy();
    }
```

Service anmelden im Manifest

- Im Manifest muss der Service durch folgenden Eintrag angemeldet werden:

```
<application
…
   <service android:name=".SchrittzaehlerService"/>
</application>
```

Service steuern

- Services kann man starten und stoppen
- zwei Buttons
  - start
  - stop
  - (wie zu Anfang schon erwähnt)
- verknüpfen der Buttons mit dem Listener in onCreate der Activity

```
findViewById(R.id.bzurueck).setOnClickListener(this);

findViewById(R.id.los).setOnClickListener(this);

findViewById(R.id.halt).setOnClickListener(this);
```

onClick-Methode

- muss erkennen, welcher Button gedrückt wurde

```
 if (v.getId() == R.id.bzurueck){
}
if(v.getId() == R.id.los){
  startService(new Intent(this,SchrittzaehlerService.class));
}
if(v.getId()R.id.halt){
  stopService(new Intent(this,SchrittzaehlerService.class));
}
```

Austausch zwischen Service und Activity

- ändern des bisher `private` Attributs `schritte` in der Service-Klasse zu

```
public static int schritte = 0;
```

- wegen public, kann die Activity-Klasse darauf zugreifen
- wegen static existiert es auch, wenn der Service nicht läuft

```
if (v.getId() == R.id.bzurueck){
                SchrittzaehlerService.schritte = 0;
                aktualisierenAnzeige();
        }
```

statischer Handler für Datenaustausch

- in der MainActivity

```
private EreignisHandler ereignisHandler = new
EreignisHandler();
private class EreignisHandler extends Handler {
  @Override
  public void handleMessage(@NonNull Message msg) {
      textView.setText(Integer.toString(msg.what));
  }
}
```

statischer Handler für Datenaustausch

- im Service

```
public static Handler ereignisHandler;
private class ErschuetterungsHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            schritte++;
            if(ereignisHandler != null){
                ereignisHandler.sendEmptyMessage(schritte);
            }
        }
    }
```

Activity und Service

- Service kann weiterlaufen, auch wenn Activity gestoppt wurde
- → Abfrage `if(ereignisHandler!=null)` sehr wichtig
- wenn Activity neu gestartet wird
- → in onResume

```
textView.setText(Integer.toString(SchrittzaehlerServic
e.schritte));
```

- Außerdem in der Activity nach dem Start des Service und in onResume

```
SchrittzaehlerService.ereignisHandler=ereignisHandler;
```

Das war‘s …

- Der Schrittzähler kann ausprobiert werden