

Standortbezogene Dienste, Sensoren und Kamera



Inhaltsübersicht

2

1. Einführung in die Entwicklung mobiler Anwendungen
2. Erste grafische Oberflächen und Benutzerinteraktionen
3. Weiterführende Konzepte mobiler Plattformen
4. Standortbezogene Dienste, Sensoren und Kamera
5. Dauerhaftes Speichern von Daten (Persistenz)
6. Responsive Design, Weiterführende Interaktionsmuster
7. Asynchrone Verarbeitung



Sensoren in Android

- Android Geräte besitzen unterschiedliche Sensoren, welche Bewegungs-, Lage- und Umweltinformationen zur Verfügung stellen können.
- Unterschieden werden
 - **Umweltsensoren (z.B. Temperatur, Umgebungslicht, Feuchtigkeit, ...)**
z.B. zur Anpassung der Displayhelligkeit an die Umgebung
 - **Bewegungssensoren (z.B. Beschleunigungs- und Neigungssensoren, ...)**
z.B. zur Nutzung der Geräteneigung zur Interaktion und Lageerkennung
 - **Lagesensoren (z.B. Magnetfeldsensoren, ...)**
z.B. zur Orientierung
- Die Ausstattung mit Sensoren ist gerätespezifisch und unterscheidet sich stark in Bezug auf Anzahl und Präzision
 - Android Framework bietet einen vereinfachten und standardisierten Zugriff auf diese Sensordaten

Sensoren in Android

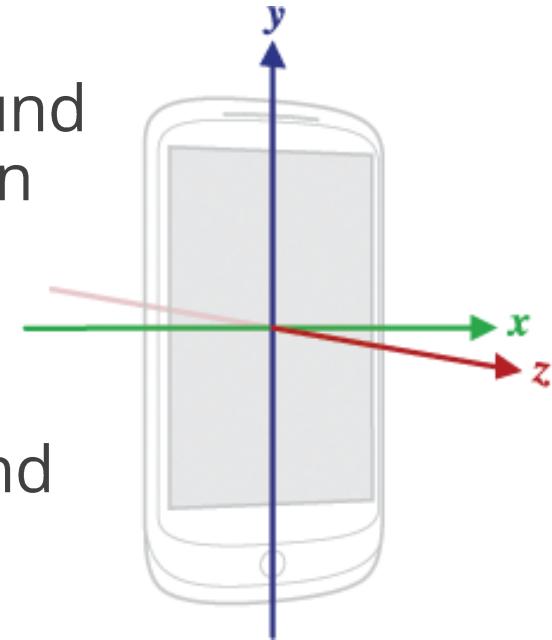
4

- Die Arbeit mit Sensoren benutzt folgende API und Klassen
 - **SensorManager**: stellt zentral den Zugriff auf alle Sensoren bereit. Liefert Informationen über Defaultsensoren und deren Verfügbarkeit
 - **Sensor**: ist die Klasse, die einen spezifischen Sensor kapselt; ermöglicht das Auslesen von Sensorinformationen und das Verknüpfen mit anderen Klassen
 - **SensorEventListener**: sind Klassen, die auf Sensorevents registriert werden und so Änderungen der Sensordaten aufnehmen und weiterverarbeiten können
 - **SensorEvent**: kapselt alle Informationen zu einem sensorspezifischen Event, also die Sensordaten, den Zeitpunkt, die Genauigkeit des Messwertes und eine Referenz auf den Sensor, der die Daten erzeugt hat

Arbeiten mit Sensoren

5

- Sensoren liefern Vektoren von Messwerten, welche vom Anwendungsentwickler interpretiert und möglicherweise noch bereinigt werden müssen.
- Genaues Verständnis über die gelieferten Messwerte benötigt tiefergehendes Wissen über Physik und Elektrotechnik.
- Alle Sensordaten werden als dreidimensionale Vektoren [x,y,z] wiedergegeben
- Beispiel Lagesensor



Interpretation von Sensoren

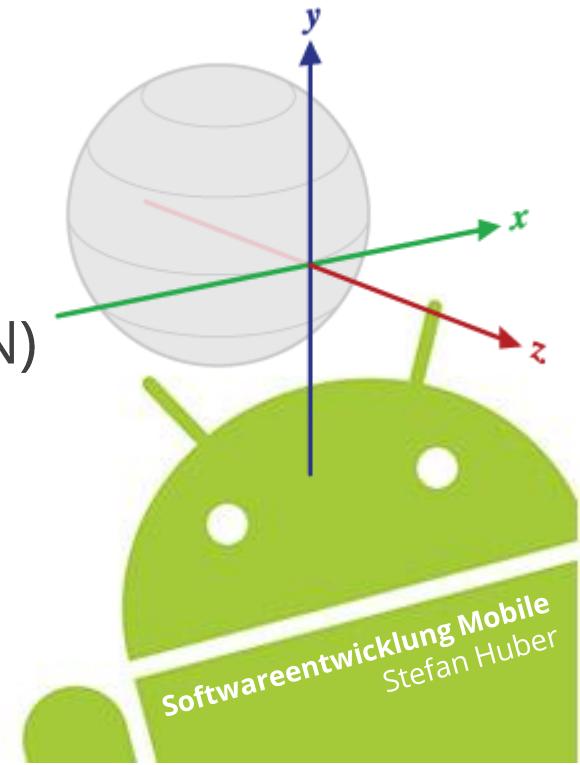
6

- Accelerometer (TYPE_ACCELEROMETER)
 - Der Accelerometer misst die Beschleunigung eines Gegenstands mittels der einwirkenden Gravitations Kraft. In der Ruheposition liefert der Accelerometer deshalb $1g = 9,81 \text{ m/s}^2$ in der z-Achse, was genau die Erdanziehungskraft wiederspiegelt. Im freien Fall würde der Accelerometer 0 m/s^2 liefern.
 - Zu Nutzung der Werte muss deshalb meistens die Gravitationskraft herausgerechnet werden. Der LINEAR_ACCELEROMETER implementiert dies bereits Softwareseitig.
 - Die Werte des Vektors [x,y,z] repräsentieren positive/negative Beschleunigung entlang der entsprechenden Achse.
- Möglichkeiten: Messen der Geschwindigkeit, wenn man aus Ruheposition startet.

Interpretation von Sensoren

7

- Gyroskop (TYPE_GYROSKOP)
 - Das Gyroskop misst die Veränderung der Rotation in rad/s.
 - Die Veränderung wird positiv bei einer Rotation gegen den Uhrzeigersinn.
 - Die Rotationen sind analog nach dem Vektor [x,y,z] zu interpretieren.
- Abstandssensor (TYPE_PROXIMITY)
 - Misst den Abstand vom Gerät zu einem Objekt. In den meisten Fällen kann nur nah oder fern erkannt werden.
- Orientierungssensor (TYPE_ORIENTATION)
 - Ausrichtung des Geräts anhand Azimuth, Pitch und Roll



Abfragen des SensorManagers

8

Abfrage des SensorManagers innerhalb einer Activity.

Instanz des Sensormangers erfragen.
Die Konstante SENSOR_SERVICE spezifiziert dabei den entsprechenden Systemdienst.

```
SensorManager sm = (SensorManager) getSystemService(Activity.SENSOR_SERVICE);  
  
List<Sensor> sensors = sm.getSensorList(Sensor.TYPE_ALL);
```

Abfrage einer Liste aller Sensoren vom Typ ALL. In diesem Fall sind das alle Sensoren.

Arbeiten mit Sensordaten

Implementieren des SensorEventListener
um Sensordaten zu verarbeiten.

```
public class MainActivity extends Activity implements SensorEventListener {

    private Sensor s;
    private SensorManager sm;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // Initialisierungen ...

        sm = (SensorManager) getSystemService(SENSOR_SERVICE);
        s = sm.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int acc) {
        // Dieses Event ist in diesem Fall nicht wichtig für uns .
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        // Nun können die Werte des Sensors genutzt und interpretiert werden
        float[] values = event.values;
    }
}
```

Eine Instanz des Umgebungssensors vom Typ Licht erfragen

Bei jeder Veränderung der Sensordaten wird das Event getriggert



Arbeiten mit Sensordaten

10

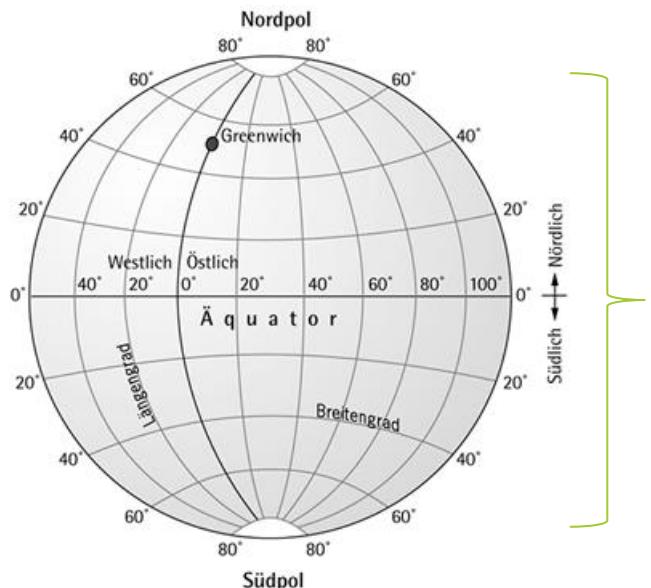
Registrieren und Abmelden des SensorEventListener innerhalb der Activity Lebenszyklus Callbacks. Dies ist wichtig, damit nicht unnötig Strom verbraucht wird.

```
@Override  
protected void onResume() {  
    super.onResume();  
    sm.registerListener(this, s, SensorManager.SENSOR_DELAY_NORMAL);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    sm.unregisterListener(this);  
}
```

Der SensorEventListener wird am SensorManager registriert. Dabei wird der Sensor und die Frequenz der Änderungsaufrufe eingestellt.

Abmelden des SensorEventListeners





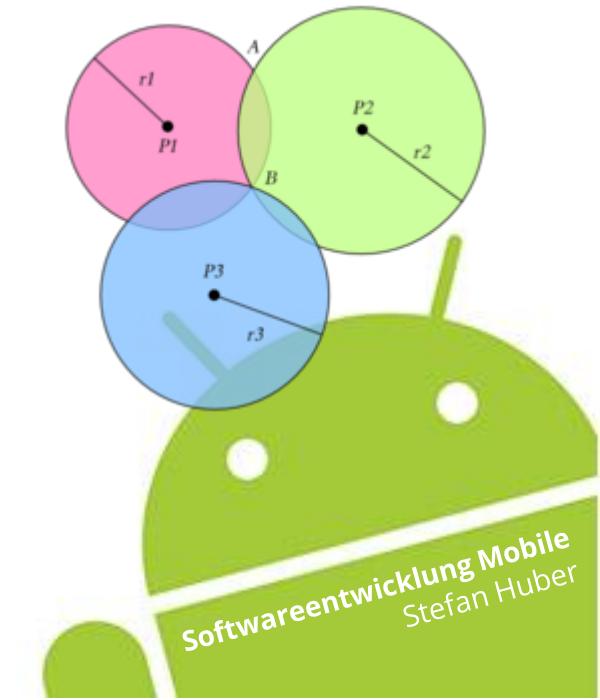
180° W – 180° O

- Jeder Ort auf der Erdoberfläche kann über Längen u. Breitengrade spezifiziert werden
- Beispiel:
 - FH Kufstein: 47° 35' 2.69" N, 12° 10' 24.46" O
- In Geographischen Informationssystemen (GIS) werden Fließkommazahlen verwendet
 - Latitude: von -90.00 bis +90.00
 - Longitude: von -180.00 bis +180.00

Positionsbestimmung durch Trilateration

12

- Durch Trilateration kann die Position eines Gerätes bestimmt werden
- GPS Satelliten kommunizieren mit GPS Receivern mittels elektromagnetischen Wellen (Lichtgeschwindigkeit)
- Die Entfernung von Satellit und Receiver kann durch die Dauer der Nachrichtenübertragung determiniert werden
- Im Schnittpunkt dreier Satelliten kann die Position des Gerätes ermittelt werden



Positionsbestimmung über Wifi bzw. CellID

13

- Google (auch Apple u. Microsoft) loggen alle Geokoordinaten von Wifi Access Points
 - SSIDs bzw. Mac-Adressen + die Position der Geräts werden an Google gesendet (Fingerprint)
 - Dies kann über die Einstellungen des Gerätes deaktiviert werden
- Jeder Mobilfunkmast verfügt über eine eigene CellID, welche Auskunft über seine Position zulässt (zB freie Datenbank: <http://opencellid.org>)
- Bei diesen Positionsbestimmungsmethoden kommt ebenfalls Trilateration zum Einsatz

Auswahl eines geeigneten LocationProvider über Criteria Objekte

14

Kriterium	Wertebereich
Accuracy	ACCURACY_FINE, ACCURACY_COARSE
Power Requirement	POWER_LOW, POWER_MEDIUM, POWER_HIGH
Altitude (Höhe)	ACCURACY_LOW, ..._MEDIUM, ..._HIGH
Bearing (Kompass)	ACCURACY_LOW, ..._MEDIUM, ..._HIGH
Speed	ACCURACY_LOW, ..._MEDIUM, ..._HIGH
Cost Allowed	True, False

Erstellen eines entsprechenden Criteria Objekts

```
LocationManager lm = (LocationManager) getSystemService(LOCATION_SERVICE);  
  
Criteria criteria = new Criteria();  
criteria.setAccuracy(Criteria.ACCURACY_COARSE);  
criteria.setPowerRequirement(Criteria.POWER_MEDIUM);  
  
String provider = lm.getBestProvider(criteria, true);
```

Laden des LocationManagers aus den System Services

Abfrage eines entsprechenden Location Providers
Kann null sein!

Registrieren eines LocationListener am LocationManager

15

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    text = (TextView) findViewById(R.id.text);  
  
    LocationManager lm = (LocationManager) getSystemService(LOCATION_SERVICE);  
  
    Location last = lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);  
    updateWithNewLocation(last);  
  
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000, 5, listener);  
}  
  
public void updateWithNewLocation(Location loc) {  
    if (loc != null) {  
        Log.i(TAG, "updateWithNewLocation called with loc: " + loc.getLatitude() + ", " + loc.getLongitude());  
        text.setText("Location: " + loc.getLatitude() + ", " + loc.getLongitude());  
    } else {  
        Log.i(TAG, "Location was null...");  
    }  
}
```

Die letzte gespeicherte Location des Providers abfragen

Einen Location Listener registrieren für Location Updates

Arbeiten mit dem LocationListener

```

public class MyLocationListener implements LocationListener {
    public void onLocationChanged(Location location) {
        double lat = location.getLatitude();
        double lng = location.getLongitude();

        // ... Anwendung der Location Informationen
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {
        // Behandeln von Statusänderungen des LocationProviders

        switch (status) {
            case LocationProvider.TEMPORARILY_UNAVAILABLE:
            case LocationProvider.OUT_OF_SERVICE:
            case LocationProvider.AVAILABLE:
        }
    }

    public void onProviderEnabled(String provider) {
        // LocationProvider wurde aktiviert
    }

    public void onProviderDisabled(String provider) {
        // LocationProvider wurde deaktiviert
    }
}

```

Reaktion auf eine Veränderung der Position

Auf die Verfügbarkeit von LocationProvidern kann nicht vertraut werden!

LocationProvidern können z.B. vom Benutzer jederzeit aktiviert oder deaktiviert werden

Indoor Positioning Systems (IPS)

17

- Für IPS existiert kein Standard wie etwa GPS.
- Es existieren unterschiedliche Lösungsansätze, Kommerzielle aber auch freie
- Mögliche Anwendungsfelder
 - Krankenhäuser
 - Einkaufscenter, Indoor Navigation
Produktwerbung
 - Sport
 - Hotels, Tourismus, Museumsguide
 - Indoor Parking
 - ...
- Ausgewählte Beispiele:
 - iBeacons
 - RedPin



Kommerzielles Beispiel: iBeacon

18

- Apple Trademark
- funktioniert über Bluetooth,
iBeacons senden Pushnachrichten
an Geräte
- Geräte können über Signalstärke
die Distanz dazu bestimmen.
- Vorteile
 - Einfache Handhabung
 - iBeacons haben eine Batterielaufzeit von ca. 2 Jahren
- Nachteile
 - Bluetooth muss aktiviert sein
 - Positionsbestimmung über iBeacons
alleine ist nur schwer möglich



OSS Beispiel: RedPin

19

- Nutzung von bestehenden Infrastrukturen: Wifi, CellID, Bluetooth
- Nutzung von Fingerprints: Raum ID + Signalstärken und ID der Umgebungssignale
- Vorteile
 - Kostengünstig, durch die Nutzung bestehender Infrastrukturen
- Nachteil
 - Trainingsaufwand
 - viele Ungenauigkeitsfaktoren durch dynamische Veränderungen der Signalstärke



Abfrage der Cell Position in GSM Netzen

20

- Folgende Informationen werden benötigt:
 - Mobile Country Code (MCC): Land zb Österreich: 232
 - Mobile Network Code (MNC): Netzbetreiber zb A1: 01
 - Location Area Code (LAC): Gruppierung von GSM Masten
 - CELLID: Identifikation des jeweiligen Mastens

```
final TelephonyManager tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);  
  
tm.listen(new PhoneStateListener() {  
  
    @Override  
    public void onCellLocationChanged(CellLocation loc) {  
        if (loc instanceof GsmCellLocation) {  
            GsmCellLocation gloc = (GsmCellLocation) loc;  
            // Verarbeiten der Location Informationen  
        }  
    }  
}, PhoneStateListener.LISTEN_CELL_LOCATION);
```

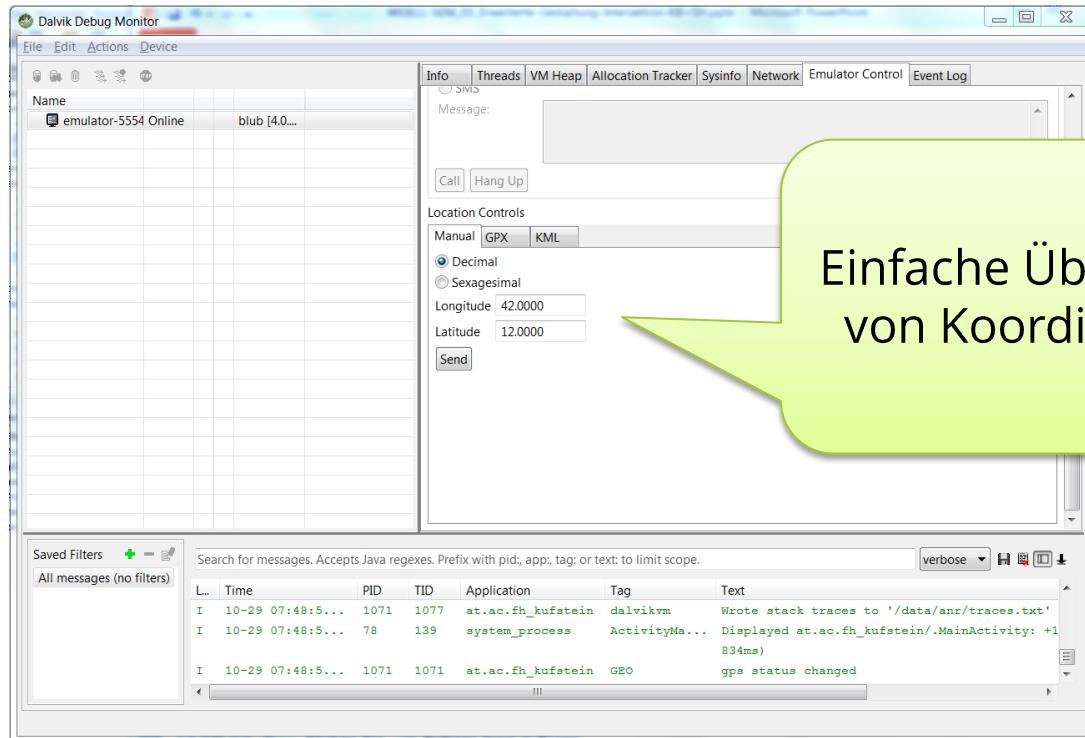
Abfragen des
TelephonyManage
r

Registrieren eines
PhoneStateListener, welcher
bei Änderungen der GSM
Zelle ein Event triggert

Emulation von Positionsdaten

21

- Der Dalvik Debug Monitor (DDM) befindet sich im Tools Verzeichnis des Android SDK
- Verwenden Dalvik Debug Monitors zur Übergabe von Dummy Koordinaten an den Emulator



- Batterie versus Genauigkeit
 - Je höher die gewünschte Genauigkeit, desto höher der Akkuverbrauch
- Zeit bis zum ersten Ergebnis
 - Die Startup Zeit von LocationProvidern kann beträchtlich sein (negativer Einfluss auf User Experience)
 - Mit ***getLastKnownLocation()*** kann diesem Problem entgegengewirkt werden
- Aktualisierungsrate
 - Hohe Aktualisierungsraten haben negative Auswirkungen auf die Akkulaufzeit
- Provider Verfügbarkeit überprüfen
 - Provider können jederzeit ausfallen (z.B. manuell vom Benutzer, Tunnel, Hochhaus, ...)
 - Verfügbarkeit über LocationListener behandeln!



- Installation der Google Play Services über den SDK Manager
 - Im Projekt müssen die Google Play Services als Library hinzugefügt werden
- Generieren eines API-KEY (siehe nächste Folie)
- Anpassungen im Android Manifest
 - Permissions
 - INTERNET: Zum Download der Karteninformationen
 - NETWORK_STATE
 - WRITE_EXTERNAL_STORAGE: Zum Cachen von Tiles
 - ACCESS_LOCATION: Kann genutzt werden um aktuelle Position zu ermitteln.
 - Maps Metainformationen innerhalb des Application Tags
 - API-KEY
 - Version
- Arbeiten mit Google Maps!

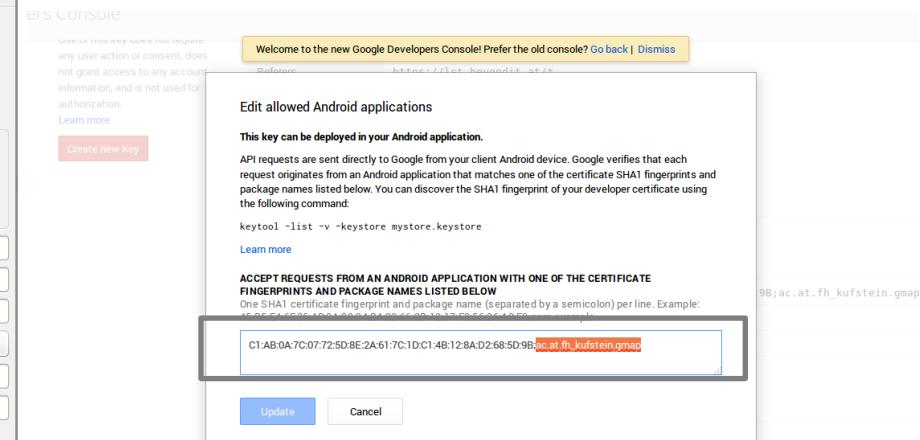
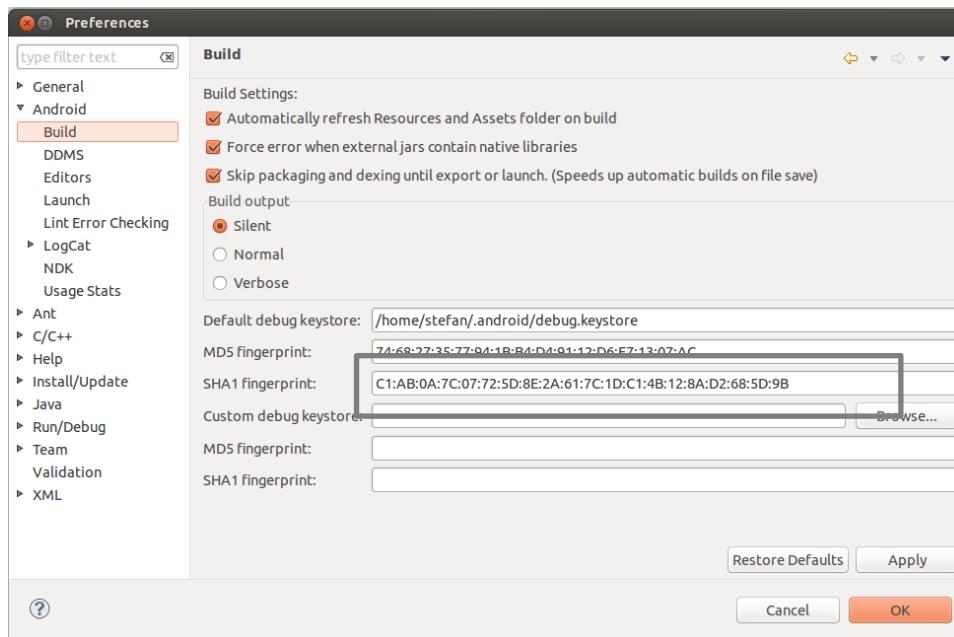
Details: <https://developers.google.com/maps/documentation/android/start>



Google Map API-KEY generieren

24

- Debugging Key aus Eclipse lesen (Window > Preferences > Android > Build)
- Neues Projekt in Google API Konsole erstellen
- Debugging Key + Projekt Packagename eintragen



Bsp: Setzen eines Markers mit Google Maps v2

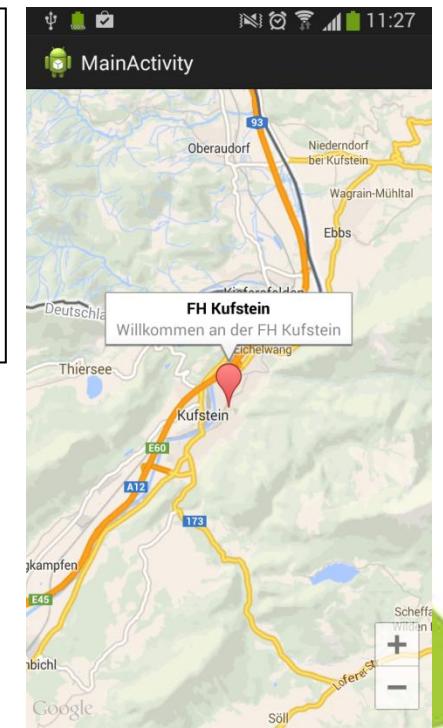
25

Abfragen des MapFragments über die Activity. Das MapFragment enthält eine Referenz auf die GoogleMap

```
MapFragment frag = (MapFragment) getSupportFragmentManager().findFragmentById(R.id.map);
GoogleMap map = frag.getMap();

MarkerOptions mos = new MarkerOptions();
mos.title("FH Kufstein")
    .snippet("Willkommen an der FH Kufstein")
    .position(new LatLng(47.583907,12.172931));

map.addMarker(mos);
```



1. Setzen von MarkerOptions: Title, Kurzes Text Snippet, Koordinaten
2. Setzen des Markers

Benutzung der Kamerafunktion über Intents

26

- Die Benutzung der Kamerafunktion erfolgt am einfachsten über ein Delegationsmodell:
 - Activity „ruft“ eine andere Activity mit der Anweisung ein Foto zu machen (MediaStore.ACTION_IMAGE_CAPTURE)
 - Kopplung über einen (impliziten) Intent
 - Ergebnis wird dann in der eigenen Activity weiter verarbeitet
 - Unterscheidung von zwei Fällen:
 1. Übergabe der Bilddaten im Event oder
 2. Übergabe der Bilddaten über externe Datei



Kamerafunktion über Intents

27

- Aufruf der „Kamera-Activity“ über einen Intent

```
/**  
 * Nutzt einen Intent zur Erzeugung eines Bildes (mit Hilfe der Kamera-App).  
 * Dabei wird nur eine verkleinerte Version (Thumbnail) zurückgeliefert  
 */  
public void takeThumbnail(View _v) {  
    startActivityForResult(new Intent(MediaStore.ACTION_IMAGE_CAPTURE),  
        TAKE_PICTURE);  
}  
  
/**  
 * Nutzt ebenfalls einen Intent zur Erzeugung eines Bildes. Diesmal wird aber  
 * noch ein URI auf das lokale Dateisystem mitgegeben, der verwendet wird  
 * um das Bild abzuspeichern. In diesem Fall wird kein Thumbnail zurückgeliefert  
 */  
public void takePicture(View _v) {  
  
    //Anlegen einer Datei und Referenz mit Hilfe eines URI  
    File file = new File(Environment.getExternalStorageDirectory(), "test.jpg");  
    outputFileUri = Uri.fromFile(file);  
  
    // Intent erzeugen und den URI als Metadatum mitgeben.  
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
    intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);  
    startActivityForResult(intent, TAKE_PICTURE);  
}
```

Fall 1 (Daten im Event)

Fall 2 (Daten in externer Datei,
Spezifikation des Dateinamens
über ein zusätzliches Meta-
Datum („Extra“))

EventMethode, die nach Abarbeiten des aufgerufenen Intents gerufen wird

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent _data) {  
    Log.v(LOG_LABEL, "Result available");  
  
    if (_requestCode != TAKE_PICTURE || _resultCode != RESULT_OK) {  
        Log.v(LOG_LABEL, "No Photo taken");  
        return;  
    }  
  
    if (_data != null) {  
        Log.v(LOG_LABEL, "we got a thumbnail");  
        Bitmap photo = (Bitmap) _data.getExtras().get("data");  
        Log.v(LOG_LABEL, "Thumbnail Dimensions:  
            + " Height:" + photo.getHeight()  
            + " Width:" + photo.getWidth());  
        myImageView.setImageBitmap(photo);  
    } else {  
        Log.v(LOG_LABEL, "we got a full image");  
    }  
}
```

Eventdaten enthalten das Bild

Bilddaten werden in der im Layout vorbereiteten ImageView angezeigt

Übergabe der Bilddaten über Event (aus der Datei)

29

```
Log.v(LOG_LABEL, "we got a full image");

BitmapFactory.Options bmOptions = new BitmapFactory.Options();

bmOptions.inJustDecodeBounds = true;
BitmapFactory.decodeFile(outputFileUri.getPath(), bmOptions);

int scaleFactor = Math.min(
    myImageView.getHeight() / bmOptions.outHeight,
    myImageView.getWidth() / bmOptions.outWidth);

bmOptions.inJustDecodeBounds = false;
bmOptions.inSampleSize = scaleFactor;
bmOptions.inPurgeable = true; // saving memory, if needed

Bitmap bitmap = BitmapFactory.decodeFile(outputFileUri.getPath(), bmOptions);
Log.v(LOG_LABEL, "Image Dimensions: "
    + " Height:" + bmOptions.outHeight
    + " Width:" + bmOptions.outWidth);
myImageView.setImageBitmap(bitmap);
}
```

Lesen der Bilddimensionen aus der Datei

Skalierung auf die Größe des ImageViews

Lesen und Anzeigen des Bildes in der vorbereiteten ImageView

“Take Away” für diese Einheit

30



- Welche Sensoren gibt es und wie können sie verwendet werden?
- Welche Möglichkeiten gibt es zur Positionsbestimmung (Indoor u. Outdoor)?
- Arbeiten mit der Kamera