Broadcasts

Broadcasts stellen systemweite Nachrichten dar. Die Nachrichten werden versandt und jeder BroadcastReceiver, der sich für den entsprechenden Nachrichtentyp interessiert, kann darauf reagieren. Broadcasts implementieren in Android also das allgemeine *Publish-Subscribe Design Pattern*.

Die Idee dahinter ist, dass sich Sender und Empfänger nicht "kennen" müssen, um miteinander kommunizieren zu können. Es benötigt nur ein einmaliges "Anmelden" des Empfängers beim Senders. Nun erhält der Empfänger laufend die Nachrichten und kann beliebig darauf reagieren.

Broadcast Nachrichten in Android können entweder Systemnachrichten oder eigene Nachrichten sein.

Beispielsweise:

- Bootvorgang ist abgeschlossen
- Batteriestand niedrig
- Flugmodus aktiviert
- etc.

Broadcasts von Betriebssystem werden als *statische* Broadcasts, eigene Broadcasts als *dynamische* bezeichnet. Beide Typen von Broadcasts können mithilfe von *BroadcastReceivern* aufgefangen und bearbeitet werden.

Dynamische Broadcasts

Mithilfe dynamischer Broadcasts ist es z.B. möglich, Daten vom Service, das im Hintergrund läuft, an die Activity zu senden. Dieser dynamische Broadcast wird vom Service erzeugt und läuft nur solange, solange auch das zugehörige Service bzw. Activity läuft.

Um einen Broadcast zu versenden wird wiederum ein Intent-Objekt benötigt.

```
private void sendMessage(int counter) {
   Intent intent = new Intent(getPackageName() + ".counter");
   intent.putExtra("counter", counter);
   sendBroadcast(intent);
}
```

Hier sollte man darauf achten, dass der Name des Intent eindeutig ist. Mithilfe des Package-Namens kann dies erreicht werden.

Möchte man nun in einer Activity auf diesen Broadcast reagieren, so muss ein entsprechender `BroadcastReceiver` implementiert werden. Dies kann in Form einer inneren Klassen innerhalb der Activity umgesetzt werden - so kann man auch auf die UI-Komponenten der Activity einfach zugreifen.

Die Verarbeitung der Daten erfolgt in der Klasse `MyBroadcastReceiver` innerhalb der Methode `onReceive`, welche aufgerufen wird, sobald eine passende Broadcast Nachricht abgefangen wird. Im `Intent` Objekt können mögliche Werte übergeben werden.

```
private class MyBroadcastReceiver extends BroadcastReceiver {

    private static final String TAG = MyBroadcastReceiver.class.getSimpleName();

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "onReceive: entered");
        int counter = intent.getIntExtra("counter", -1);
        Toast.makeText(getApplicationContext(), "recent counter value: " + counter,
    }
}
```

Um den Kommunikationsfluss zu garantieren, muss der BroadcastReceiver noch für die Activity registriert werden. Dazu benötigt man eine Instanz der Klasse `MyBroadcastReceiver` innerhalb der Activity.

```
private MyBroadcastReceiver receiver = new MyBroadcastReceiver();
private IntentFilter filter = new IntentFilter("net.eaustria.broadcastdemo.counter"
```

Der `IntentFilter` ist dafür zuständig, nur jene Nachrichten herauszufiltern, die für die jeweilige Activity interessant sind. Der Parameter (in diesem Fall `net.eaustria.broadcastdemo.counter`) muss genau jener Bezeichnung entsprechen, die als Name für den Intent vergeben wurde.

Ähnlich wie schon beim GPS Provider gesehen, registriert man den Receiver in der Methode `onResume` und meldet ihn in der Methode `onPause` wieder ab. Zum Anmelden (Registrieren) des Receiver muss zusätzlich auch noch ein Objekt vom Typ `ContentFilter` mitübergeben werden, das für die Auswahl der Broadcastnachrichten zuständig ist.

```
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(receiver, filter);
}

@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```

Statische Broadcasts

Um systemweite Broadcasts verarbeiten zu können, wird der Receiver direkt in der Manifest Datei definiert. Auf diese Weise kann der Receiver auch auf Nachrichten reagieren, wenn die App nicht gerade gestartet ist.

Folgende Schritte sind erforderlich:

1. Eigene Klasse von `BoradcastReceiver` ableiten.
2. Logik innerhalb der Methode `onReceive` implementieren.
3. Klasse als Receiver im Manifest eintragen.
4. Jene Systemereignisse mithilfe von `intent-filter` definieren, auf die der Receiver lauschen soll.

Natürlich können die Schritt auch in anderer Reihenfolge abgearbeitet werden. Diese Reihenfolge erscheint jedoch logisch….

Die Klasse wird nun nicht als innere Klasse, sondern als eigenständige Klasse definiert (jedoch wieder von `BroadcastReceiver` abgeleitet).

```
public class MyStaticReceiver extends BroadcastReceiver {
    private static final String TAG = MyStaticReceiver.class.getSimpleName();

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "onReceive: emtered");
        String action = intent.getAction();
        Log.d(TAG, "onReceive: Action: " + action);
    }
}
```

Nun kann die Receiver Klasse im Manifest eingetragen werden:

```
<receiver android:name=".MyStaticReceiver" />
```

Wie eingangs erwähnt, muss nun noch mittels `Content-Filter` definiert werden, auf welche Broadcasts gelauscht werden soll. Eine Liste mit allen systemweiten Android-Broadcasts findet man unter:
https://developer.android.com/reference/android/content/Intent.html#constants

Jene Ereignisse, bei denen ein Broadcast versendet wird, sind durch eine `Broadcast Action` gekennzeichnet.`

```
<receiver android:name=".MyStaticReceiver">
   <intent-filter>
       <action android:name="android.intent.action.AIRPLANE_MODE" />
       <action android:name="android.intent.action.BATTERY_LOW" />
   </intent-filter>
</receiver>
```

Für jeden Receiver können natürlich auch mehrere Actions definiert werden, auf die er lauschen sollte. Benötigt man für die Action besondere Berechtigungen, so müssen diese auch im Manifest eingetragen werden.

Content-Filter programmatisch definieren

Es gibt Actions, die nicht über das Manifest registriert werden können. Diese müssen im Quellcode mittels Aufruf der Methode `registerReceiver` angefordert werden. Die Action `android.intent.action.TIME_TICK`, die jede volle Minute ausgelöst wird, gehört etwa zu jenen Actions.

```
registerReceiver(new MyStaticReceiver(),
               new IntentFilter(Intent.ACTION_TIME_TICK));
```