

## Mobile Programming

# Android 5 – Services, Broadcast Receiver



[kaspar.vongunten@hslu.ch](mailto:kaspar.vongunten@hslu.ch)



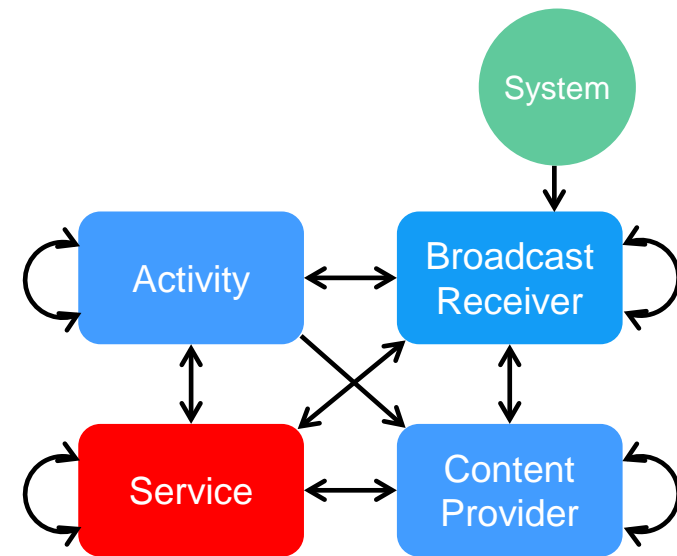
# Inhalt

- Services
  - Komponente ohne UI
  - Für «wahrnehmbare» Hintergrundaktivitäten
- WorkManager
  - Für generelle Arbeiten im Hintergrund
- Broadcast Receiver
  - Empfänger von (System-)
  - Interner «Messenger»

**Oh, happy day: Relativ  
wenig Stoff heute!**

# Service-Komponente

- Ursprünglich in Android zur Kapselung und Erledigung von Hintergrundarbeiten eingeführt
- Seit API 26 (Android 8, Oreo) stark eingeschränkt, wegen System-performance Issues (zu viele Services in zu vielen Apps)
- Nun nur noch als Spezialfall «Foreground»-Service empfohlen (und zum Export von App-Logik)
  - Klassischer Fall: Music Player
- Für frühere Anwendungsbereiche wird nun Verwendung von *WorkManager* empfohlen
  - Z.B. Location Update, Background Sync, etc.



Vorsicht:  
potentielles  
Sicherheitsrisiko

# Android: Service-Konzept

<https://developer.android.com/guide/components/services.html>  
<https://developer.android.com/reference/android/app/Service.html>

Achtung: Eingeschränkte  
Verwendung ab API 26!

`startService()` - Auftrag  
für einen Service erteilen

- Was bietet ein Service?
  - Eine Möglichkeit, dem System mitzuteilen, dass eine gewisse Arbeit im Hintergrund erledigt werden soll
  - Eine Möglichkeit, gewisse Funktionalität (API) zu exportieren und anderen Applikationen anzubieten
- Was ist ein Service **nicht**?
  - Kein separater Worker-Thread (per se)
  - Kein eigener Prozess (nur wenn so definiert)
- Lang andauernde Operationen & Nebenläufigkeit?
  - Ein Service kann (und sollte) einen eigenen Thread starten, um lang andauernde Operationen zu erledigen

`bindService()` – Öffnet  
stehende Verbindung für  
Kommunikation mit Service

# Lebensarten eines Services + Lifecycle Methods (1)

## ■ Zwei Lebensarten

Kombination möglich!

**Ungebundener Service**  
Service verbleibt im Zustand  
RUNNING bis explizit beendet

### ■ Aufruf durch **startForegroundService(...)**

- `onCreate()`: Bei Erzeugung
- `onStartCommand()`: Auftragsbehandlung
- `onDestroy()`: Bei Beendung (durch Service selbst, durch Applikation oder durch System)

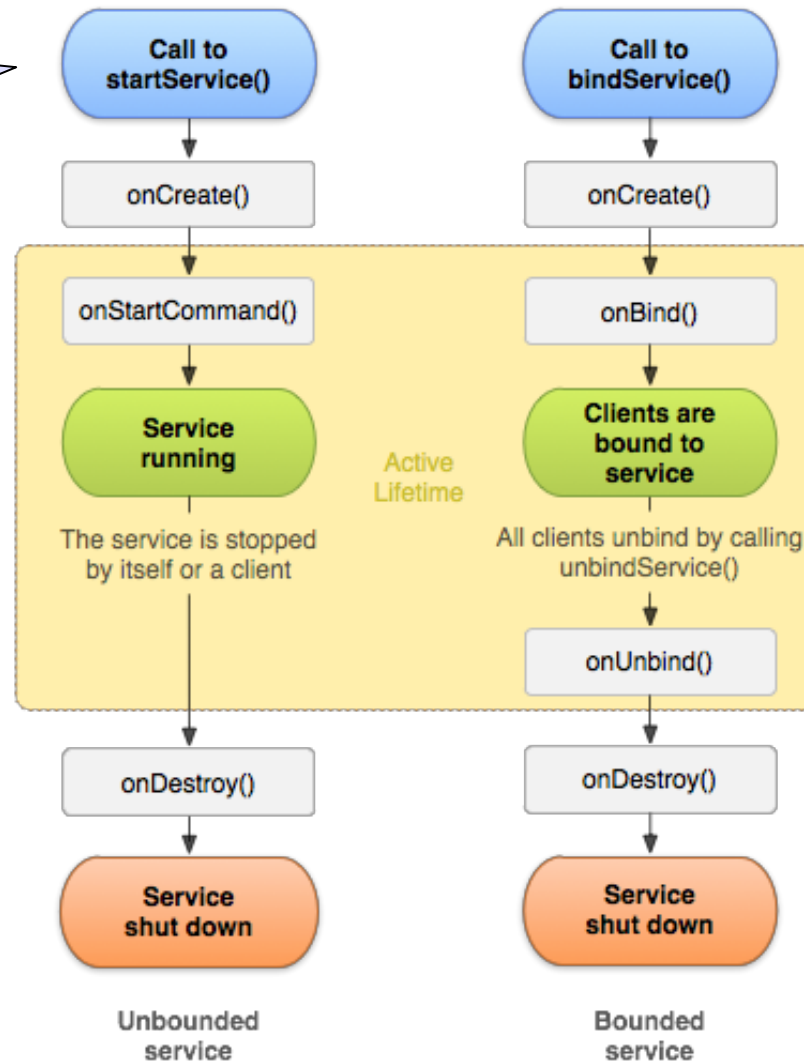
### ■ Aufruf durch **bindService(...)**

**Gebundener Service**  
Service verbleibt nur so lange  
im Zustand RUNNING wie  
Bindings existieren

- `onCreate()`: s.o.
- `onBind()`: Wenn Komponente Verbindung herstellt
- `onUnbind()`: Wenn Komponente Verbindung beendet
- `onDestroy()`: s.o.

# Lebenszyklus eines Service + Lifecycle Methods (2)

Ab API 26:  
startForegroundService()

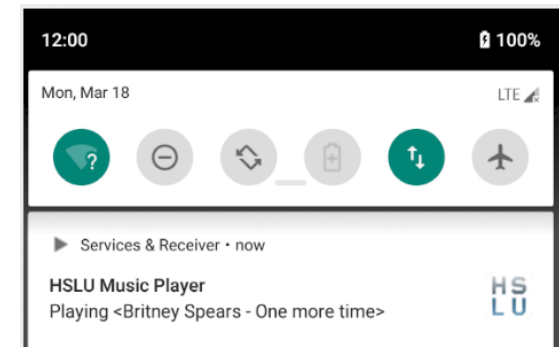
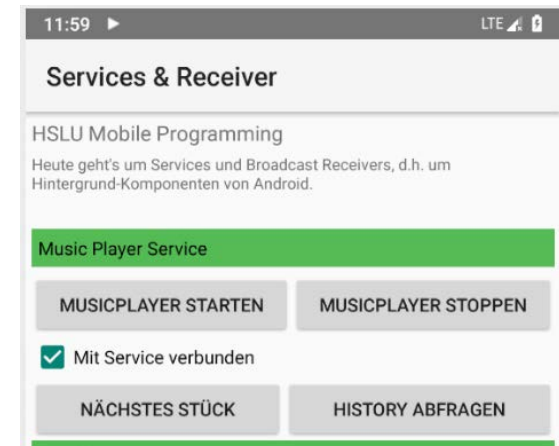


# Demo: Foreground Service

- Für Hintergrund-Arbeiten, die für Benutzer wahrnehmbar sind, z.B. *Music Player*
- Zeigt Notification an, während er läuft
- Benötigt Permission  
FOREGROUND\_SERVICE
- Interaktion mit / Steuerung Service oft über Binding

«Normal Permission»,  
d.h. wird automatisch vergeben

Folgt später



# Foreground Service: Music Player Service (1)

```
public class DemoMusicPlayerService extends Service {
```

```
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        startPlayer();  
        return Service.START_NOT_STICKY;  
    }
```

Wird bei startService() aufgerufen.  
Achtung: *main Thread*!

```
    private void startPlayer() {  
        if (playThread != null && playThread.isAlive()) return;  
        startPlayThread();  
        startForeground(NOTIFICATION_ID, createNotification("Playing..."));  
    }
```

```
    @Override  
    public void onDestroy() {  
        stopPlayThread();  
        stopForeground(true);  
    }
```

Wird bei stopService() aufgerufen

Es gibt keinen onStopService() Hook, nur onDestroy().

Macht aus Bg-Service einen Fg-Service

Muss innerhalb von 5s nach Start erfolgen



# Foreground Service: Music Player Service (2)

```
<manifest>
...
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
...
<service
    android:name=".service.DemoMusicPlayerService"
    android:description="@string/musicplayerservice_desc"
    android:exported="false" />
...
</manifest>
```

```
new NotificationCompat.Builder(this, CHANNEL_ID)
    .setContentTitle("HSLU Music Player")
    .setTicker("HSLU Music Player")
    .setContentText(someText)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .setOngoing(true)
    .setSmallIcon(android.R.drawable.ic_media_play)
    .setLargeIcon(...)
    .setWhen(System.currentTimeMillis())
    // .setContentIntent(pendingIntent) // öffne Kontroll-Activity
    // .addAction(...) // Notification-Actions hinzufügen
    .build();
```

Channel muss 1x für App  
angelegt werden, am  
besten in Main Activity  
(siehe Docs)

Notification zuoberst,  
kann von User nicht  
weggeklickt werden

# Foreground Service: Music Player Service (3)

## ■ Service starten

```
public void startPlayerService(View v) {  
    startService(new Intent(this, DemoMusicPlayerService.class));  
}
```

Kann natürlich auch noch  
Parameter mitgeben im Intent!

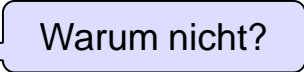



## ■ Service stoppen

```
public void stopPlayerService(View v) {  
    stopService(new Intent(this, DemoMusicPlayerService.class));  
}
```

# Allgemeines Muster

- Service wird bei App-Start oder aufgrund Event als Foreground-Service gestartet und bleibt alive
- Startet einen Worker-Thread oder Thread-Pool, der aktiv bleibt (zu Beginn ggf. idle)
- Mittels `bindService()` kann *synchron* kommuniziert werden

# Service: STICKY oder NOT\_STICKY?

- Was soll mit Service passieren, wenn das System den Applikationsprozess zerstört und später wieder herstellt?
  - `onStartCommand()` retourniert gewünschte Verhaltensweise
  - Nicht wichtig für gebundene Services! 
- Mögliche Rückgabewerte von `onStartCommand()` 
  - `START_STICKY`: Service soll nach Wiederherstellung automatisch gestartet werden - `onStartCommand()` wird erneut aufgerufen, aber ohne Intent 
  - `START_NOT_STICKY`: Service wird nicht automatisch neu gestartet nach Wiederherstellung 
  - `START_REDELIVER_INTENT`: Wie `START_STICKY`, aber der ursprüngliche Intent wird noch einmal ausgeliefert, damit parametrisierte Reinitialisierung möglich

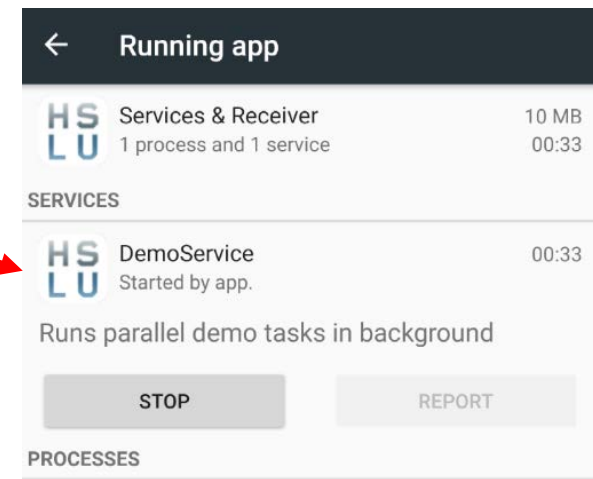
# Tipp: Laufende Services auf Gerät anzeigen

User können laufende Services auf Gerät anzeigen und manuell stoppen:

- Developer Mode freischalten
- Settings > Developer options > Running services
- Deshalb: In Service-Deklaration im Manifest `android:description` verwenden, um Service zu beschreiben (nur String-ref erlaubt!)

Settings > About  
Phone > Build # >  
Tap 7 times

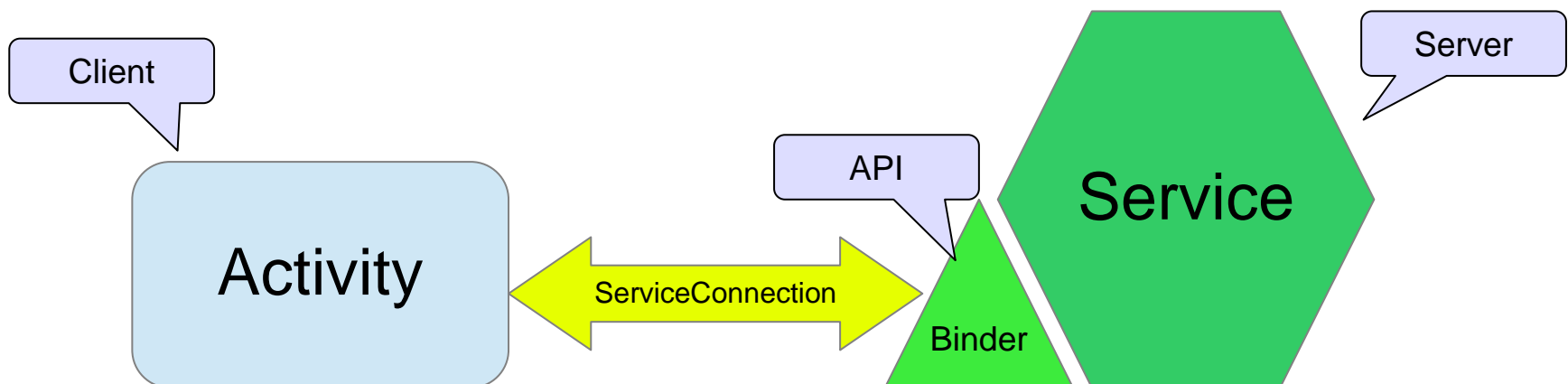
```
<service
    android:name=".service.DemoService"
    android:exported="false"
    android:description="@string/demoservice_desc"/>
<service
    android:name=".service.DemoIntentService"
    android:exported="false"
    android:description="@string/intenservice_desc"/>
```



# Gebundene Services

## Ein Service kann gebunden werden

- Mittels `bindService(intent, connection, flag)`
- Client kommuniziert mit Service über `ServiceConnection`
- Damit kann Funktionalität einer App exportiert werden
  - Insbesondere mit einem „Remote Service“ Hier **nicht** behandelt
- Bindung lösen mit `unbindService(connection)`



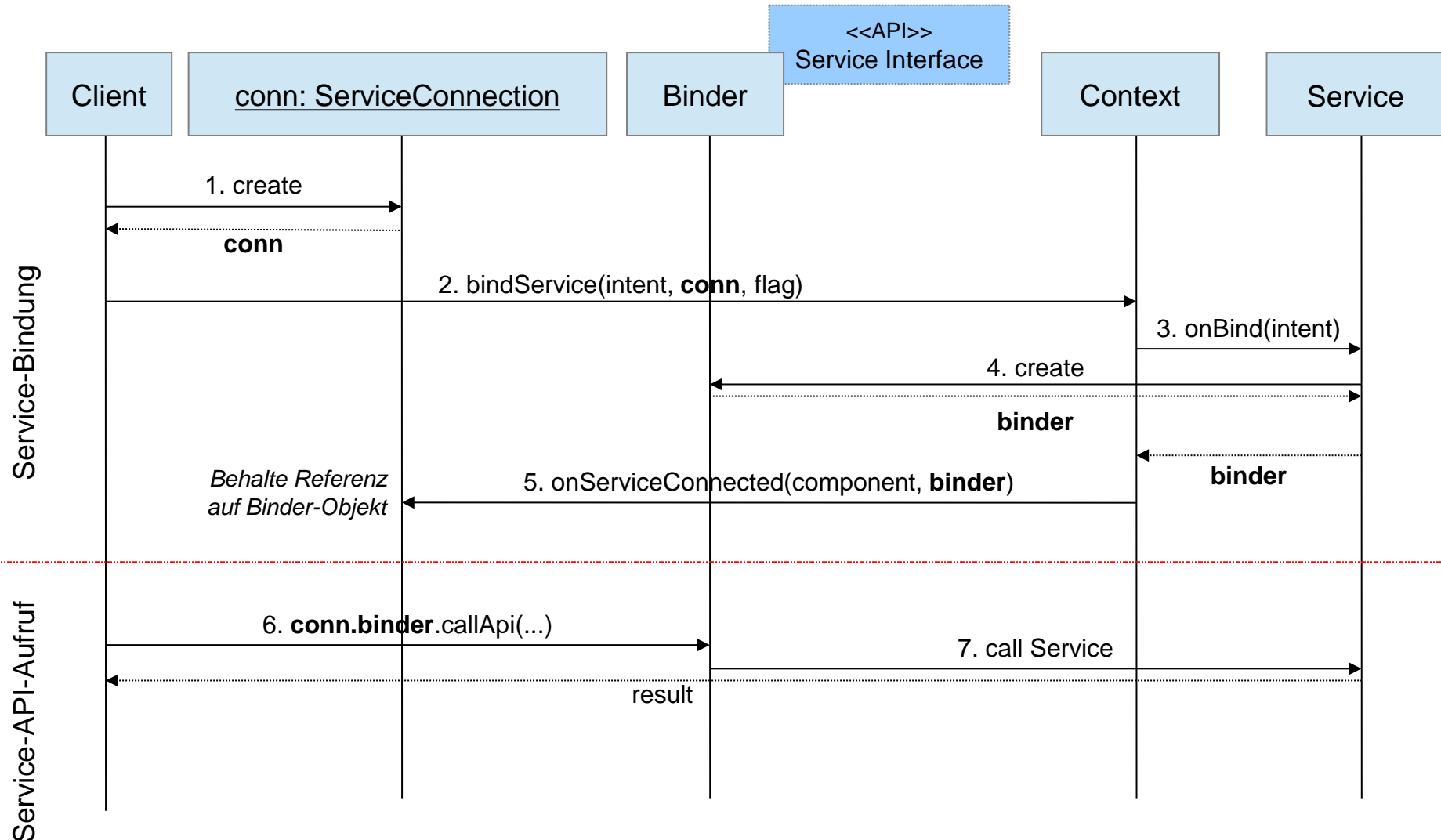
# Gebundene Services: Involvierte Klassen

- Service Interface (MusicPlayerApi)
  - Definiert API des Services
- Binder (MusicPlayer.MusicPlayerApiBinder)
  - Implementiert Service Interface, wird dem Client bei erfolgreicher Verbindung übergeben (Service-Stub / -Handle)
- Service Connection (MusicPlayerConnection)
  - Definiert Callbacks für erfolgreiche Verbindung oder verlorene Verbindung, erhält Binder-Objekt (=API) bei Erfolg
- Service (MusicPlayerService)
  - Implementiert `onBind(intent)` und gibt Binder-Objekt zurück
- Client (MainActivity)
  - Ruft `bindService(intent, connection, flag)` resp. `unbindService(connection)` auf

Klassenname in Beispiel-Code, siehe spätere Folie (Übung 5)

Callback-Handler

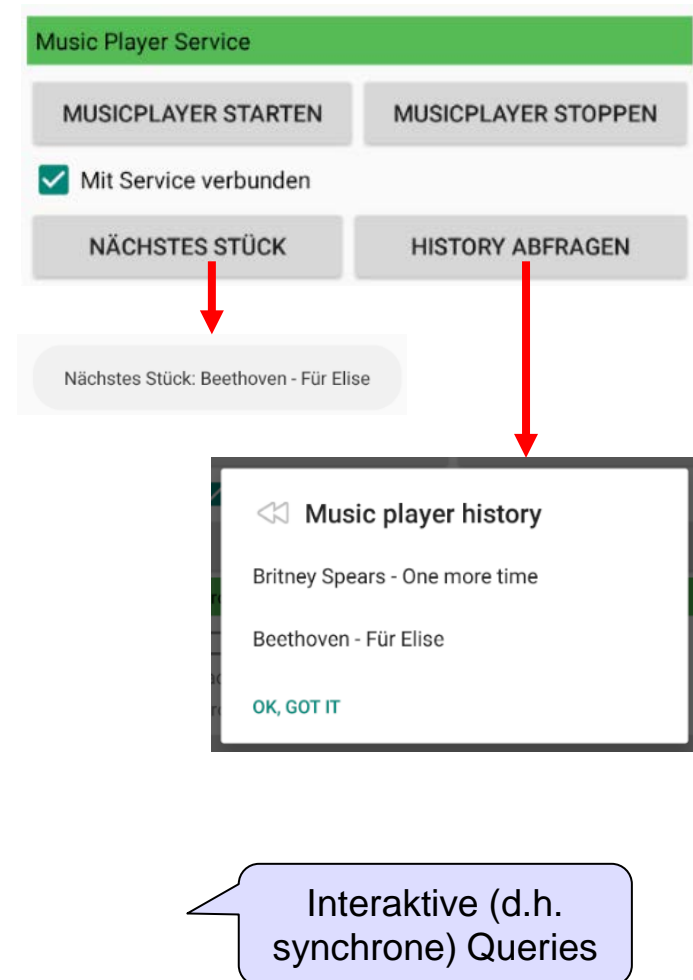
# Ablauf: Service-Bindung & Service API-Aufruf





# Demo: gebundener Service (Siehe Übung 5)

- `MusicPlayerService`
  - Kann gebunden werden
    - d.h. implementiert `onBind()`
  - Ablauf siehe Sequenz-Diag. vorangehende Folie!
- `MusicPlayerApi`
  - Bietet 3 Methoden an:
    - `playNextItem()`
    - `getHistory()`



# Gebundener Service: Music Player Service (4)

```
public interface MediaPlayerApi {  
    String playNext();  
    List<String> queryHistory();  
}
```

Service API Definition

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

Gibt Binder-Instanz  
zurück (=API Instanz)

```
private class MediaPlayerApiImpl extends Binder implements MediaPlayerApi {  
    @Override  
    public String playNext() {  
        return DemoMusicPlayerService.this.playNext();  
    }  
  
    @Override  
    public List<String> queryHistory() {  
        return DemoMusicPlayerService.this.queryHistory();  
    }  
}
```

Service API  
Implementierung

# Gebundener Service: Client (5)

Verbinden

```
private void bindServiceToThisActivity() {  
    if (!isServiceBoundToThisActivity()) {  
        Intent intent = new Intent(this, MediaPlayerService.class);  
        serviceConnection = new MediaPlayerConnection();  
        bindService(intent, serviceConnection, BIND_AUTO_CREATE);  
    }  
}
```

```
private void unbindServiceFromThisActivity() {  
    if (isServiceBoundToThisActivity()) {  
        unbindService(serviceConnection);  
        serviceConnection = null;  
    }  
}
```

Verbindung  
trennen

```
public void playNextClicked(View v) {  
    if (isServiceBoundToThisActivity()) {  
        MediaPlayerApi api = serviceConnection.getMediaPlayerApi();  
        api.playNext();  
    }  
}
```

API aufrufen

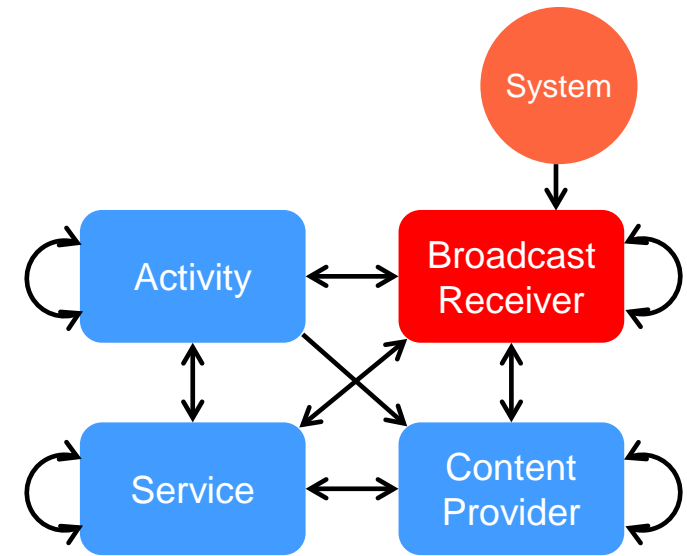


<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Broadcast Receiver

# Broadcast Receiver

- Broadcasts sind Nachrichten
  - App-interner «Message Bus»
  - Alle Komponenten können Broadcasts verschicken und sich für Empfang registrieren
  - System verschickt ebenfalls Nachrichten bei gewissen Events (App installiert, Timer, ...)
  - **Seit API 26 (Android 8, Oreo) stark eingeschränkt**, wegen Performance (zuviele Handler, zuviele Msg)
  - Nun werden nur noch sehr wenige Events global verteilt
- z.B. «OnBootCompleted», gehen wir nicht näher drauf ein



# Broadcasts verschicken

D.h. können Daten mit sich tragen

- Broadcasts werden als Intents verschickt
- Implizit (in App) via `LocalBroadcastManager`  
`.getInstance(this).sendBroadcast(intent);`
- Explizit (an andere App) über
  - `Context::sendBroadcast(intent)`
- Empfang von Broadcasts: Receiver (Empfänger)
  - Empfänger werden dynamisch im Code registriert
    - `registerReceiver(receiver, filter)`
  - Können auch statisch im Manifest definiert werden
    - Tag: `<receiver ...>`

Aufruf erfolgt auch wenn App nicht gestartet! Nur für explizite Broadcasts.

# Globaler Broadcast Receiver (1)

Don't: Keine AsyncTasks!  
Keinen Service binden!  
Keinen Dialog anzeigen!

Do: Activity starten,  
Service starten,  
Notification schicken, ...

mit hoher  
Priorität!

- BR ist immer nur so lange aktiv, wie Bearbeitung der empfangenen Nachricht braucht
  - Sonst inaktiv, wird vom System gelöscht!
  - (Erneute) Erzeugung „on demand“
  - Nur für Empfang von expliziten Broadcasts geeignet
- BR hat kein UI
  - Notifications benutzen für Kommunikation mit Benutzer...
  - ...oder Service starten für Hintergrund-Aufgaben

# Globaler Broadcast Receiver (2)

## ■ Im Manifest

```
<receiver android:name=".BootCompletedReceiver">  
  <intent-filter>  
    <action android:name="android.intent.action.BOOT_COMPLETED" />  
  </intent-filter>  
</receiver>
```

XML Intent-Filter

## ■ Dedizierte Broadcast Receiver-Klasse

```
public class BootCompletedReceiver extends BroadcastReceiver {  
  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        // do something, when boot has completed, e.g. start a service...  
    }  
}
```

## ■ Expliziten Broadcast an andere App versenden

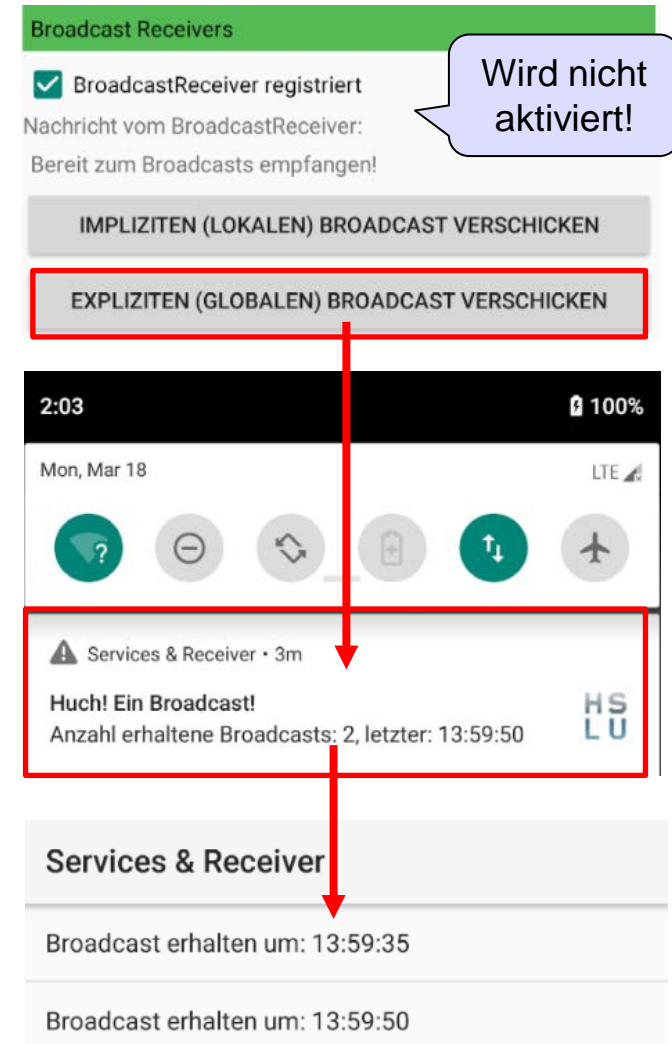
```
Intent broadcastIntent = new Intent("ACTION_MY_BROADCAST");  
broadcastIntent.setPackage("ch.hslu.mobpro.other");  
sendBroadcast(broadcastIntent);
```

Wichtig:  
Empfänger-ID!



# Demo: Statischer BR (globale Message)

- App registriert Broadcast statisch Receiver im Manifest
- Versand globaler Broadcast adressiert an App via Package-ID
- Empfängt nur die expliziten Events und zeigt diese mit Notification an
  - So konfiguriert, dass Activity mit Details angezeigt wird bei Klick auf Notification



# Lokale Broadcasts: «App Message-Bus»

## ■ BR erzeugen & registrieren im Code

```
downloadCompleteListener = new BroadcastReceiver() {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Toast.makeText(this, "Got it!", LENGTH_SHORT).show();  
    }  
};
```

Hinweis: Muss  
nicht eine innere  
Klasse sein

Filter auf Action-Code

```
IntentFilter filter = new IntentFilter("mobpro.DOWNLOAD_COMPLETE");  
LocalBroadcastManager.getInstance(this)  
    .registerReceiver(downloadCompleteListener, filter);
```

## ■ Nachricht versenden (Emitter)

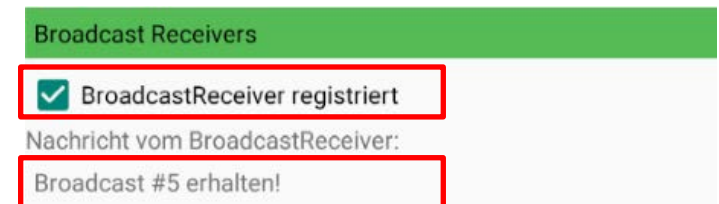
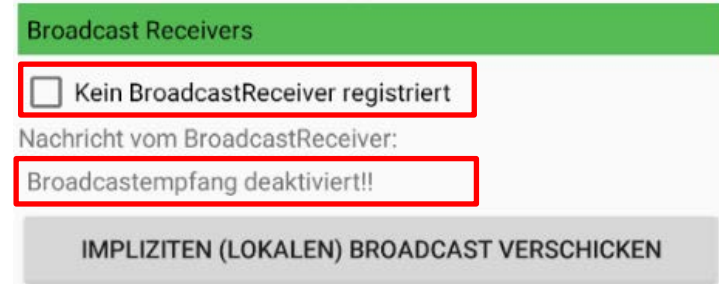
```
Intent downloadComplete = new Intent("mobpro.DOWNLOAD_COMPLETE");  
downloadComplete.putExtra("file", "Terminator2.mp4");  
LocalBroadcastManager.getInstance(this).sendBroadcast(downloadComplete);
```

## ■ BR deregistrieren (wenn nicht mehr benötigt)

```
LocalBroadcastManager.getInstance(this)  
    .deregisterReceiver(downloadCompleteListener);
```

# Demo: dynamischer BR (on/off, Übung 5)

- MainActivity registriert eigenen Broadcast Receiver auf ACTION\_MY\_BROADCAST
  - On/Off-CheckBox
  - Registrierung im Code
    - `(un)registerReceiver(...)`
- Eigener Broadcast Receiver (z.B. anonyme Klasse) zählt Anzahl empfangene Broadcasts und zeigt diese auf MainActivity an
  - ☞ Empfang ausschliesslich, wenn BR registriert!





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

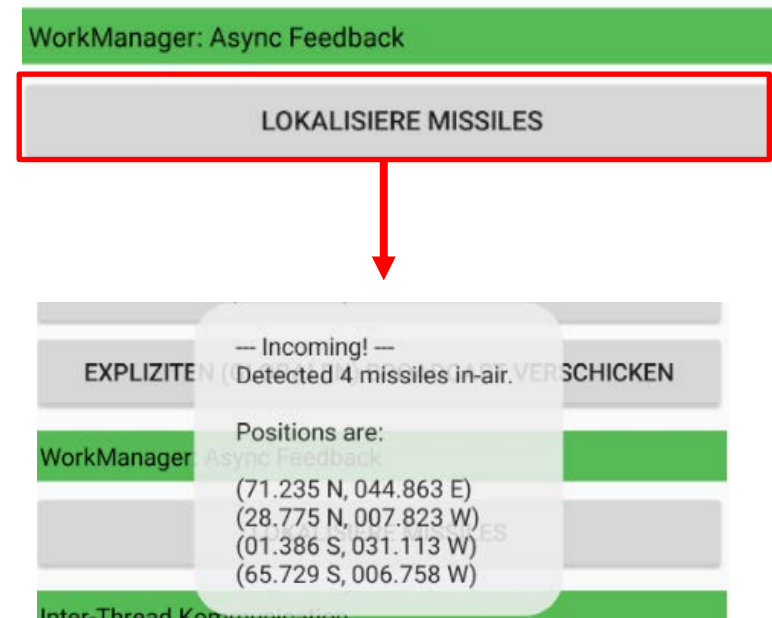
## WorkManager + Broadcasts

# WorkManager für Hintergrundtasks

- Erinnere letzte Vorlesung: repetitive oder einmalige Background-Tasks, die aufschiebbar sind, sollten via WorkManager erledigt werden
- Wie findet die App heraus, wenn der Task abgeschlossen ist?
- Z.B. mittels lokalem Broadcast
  - Verarbeitung in diesem Fall nur, wenn App noch läuft, resp. Receiver noch registriert ist
  - z.B. um Liste heruntergeladener Files zu refreshen

# Demo: WorkManager

- Lang andauernder Task: Missile-Positionen erkennen
- Worker task wird an WorkManager übergeben
- Sobald Positionen bestimmt sind, werden diese mittels Broadcast zurückgemeldet
- Broadcast-Receiver hört auf «Localize Missiles» Action und zeigt Resultate in Toast



# Worker mit Resultat-Event

WorkRequest  
erstellen und erfassen

```
public void startBackgroundTaskClicked(View view) {  
    OneTimeWorkRequest getLocationTask =  
        new OneTimeWorkRequest.Builder(LocalizeMissilesWorker.class).build();  
    WorkManager.getInstance().enqueue(getLocationTask);  
}
```

```
public static class LocalizeMissilesWorker extends Worker {  
    public LocalizeMissilesWorker(Context context, WorkerParameters params) {  
        super(context, params);  
    }  
}
```

Arbeit definieren

@Override

```
public Result doWork() {  
    Log.i("LocalizeMissilesWorker", "Getting location of missile");  
    // determine number and location of missiles (long time)  
  
    Intent result = new Intent("mobpro.ACTION_LOCALIZE_MISSILES");  
    result.putStringArrayListExtra("missilePositions", positions);  
    LocalBroadcastManager.getInstance(getApplicationContext()).sendBroadcast(result);  
  
    return Result.success(); // or Result.failure() or Result.retry()  
}
```

Broadcast-Event mit  
Resultat verschicken

Erfolg/Misserfolg melden

# WorkManager: Weitere Informationen

- Benötigte Gradle-Dependency:

```
implementation "androidx.work:work-runtime:2.0.0-rc01"
```

- Das gezeigte Background-Work Beispiel mit einem Worker funktioniert selbstredend auch mit Threads
- Worker kann auch als wiederkehrender Task registriert werden oder mit einem anfänglichen Delay
- Arbeiten können auch in Graphen mit Abhängigkeiten definiert werden
- Für mehr Info siehe *How-To Guides* unter <https://developer.android.com/topic/libraries/architecture/workmanager>





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Übung 5

# Zur Übung 5

- Eigener gebundener Service
  - Simuliert Music Player
  - Starten & stoppen
  - Inkl. Service-API
- Eigener Broadcast Receiver
  - Dynamische Registration im Code
  - Eigene Broadcast-Action
- Optional: WorkManager verwenden

