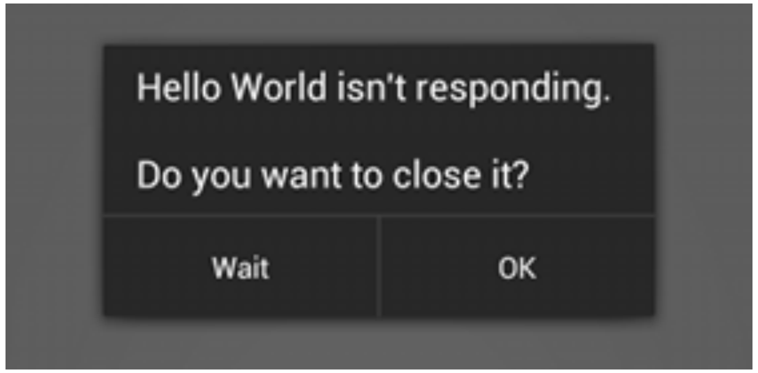
Day 3

Schedule

- **Last Time**
  - Android UI Resources
  - AdapterView
- **Today**
  - Android Threading
  - AsyncTask
  - Service
- **If time**
  - RecyclerView as replacement for AdapterView

Threading

ANR – Application not responding


Hello World isn't responding.

Do you want to close it?

Threading

Overview

- Single process
- Single UI Thread
- Every Android Component is running in UI Thread
- Create thread like in normal java, but ….

```java
Thread t = new Thread(new Runnable() {

    @Override
    public void run() {
        // Do long running operation here
    }
});

t.start();
```

```
10-24 14:25:32.195: E/AndroidRuntime(1906): FATAL EXCEPTION: Thread-8310-24 14:25:32.195: E/AndroidRuntime(1906): android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:4746)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:823)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.view.View.requestLayout(View.java:15468)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.view.View.requestLayout(View.java:15468)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.view.View.requestLayout(View.java:15468)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.view.View.requestLayout(View.java:15468)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.widget.RelativeLayout.requestLayout(RelativeLayout.java:318)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.view.View.requestLayout(View.java:15468)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.widget.TextView.checkForRelayout(TextView.java:6313)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.widget.TextView.setText(TextView.java:3567)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.widget.TextView.setText(TextView.java:3425)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at android.widget.TextView.setText(TextView.java:3400)10-24 14:25:32.195: E/AndroidRuntime(1906): 	at ch.schoeb.day3_demo_threading.MainActivity$1$1.run(MainActivity.java:25)
```

Threading

UI Access

- Access to UI Elements from background thread not allowed
  - Even if it seems to work sometimes

- Possibilities to “dispatch”
  - Handler.post(Message)
  - runOnUiThread(new Runnable(…)) on Context

Threading

AsyncTask

- Simplified background-worker
- Executed in own thread (from thread pool, or by custom Executors)
- Callback-Methods on UI-Thread
- Abstract basic class

- Do not use AsyncTask extensively!! It is useful for small examples but not for production code, there are more elegant solutions.

Threading

AsyncTask – class structure

```
public class Worker extends AsyncTask<Params, Progress, Result>
{
       protected Result doInBackground(Params... param1) {
              publishProgress(Progress)
              return theResult;
       }

       protected void onPreExecute() {
       }

       protected void onPostExecute(Result result) {
       }

       protected void onProgressUpdate(Progress... values) {
       }
}
```

UI THREAD

```
new Worker().execute(Params param1);
```

Threading

AsyncTask – flaws

- AsyncTask is tightly bound to a particular Activity
  - If an activity is destroyed or its configuration is changed (e.g. rotation or language changes) the AsyncTask holds possibly an old callback to update the UI -> NullPointerException

Threading

Alternatives

- There are a lot of very good libraries that make the use of AsyncTasks superfluous
  - **RxJava** provides thread support (choose on which thread work is done and on which thread it is reported)
  - REST with Retrofit (also compatible with RxJava)
  - Background Jobs (e.g. Synchronization with a backend) using **Android JobManager** (or GcmNetworkManager)

Components

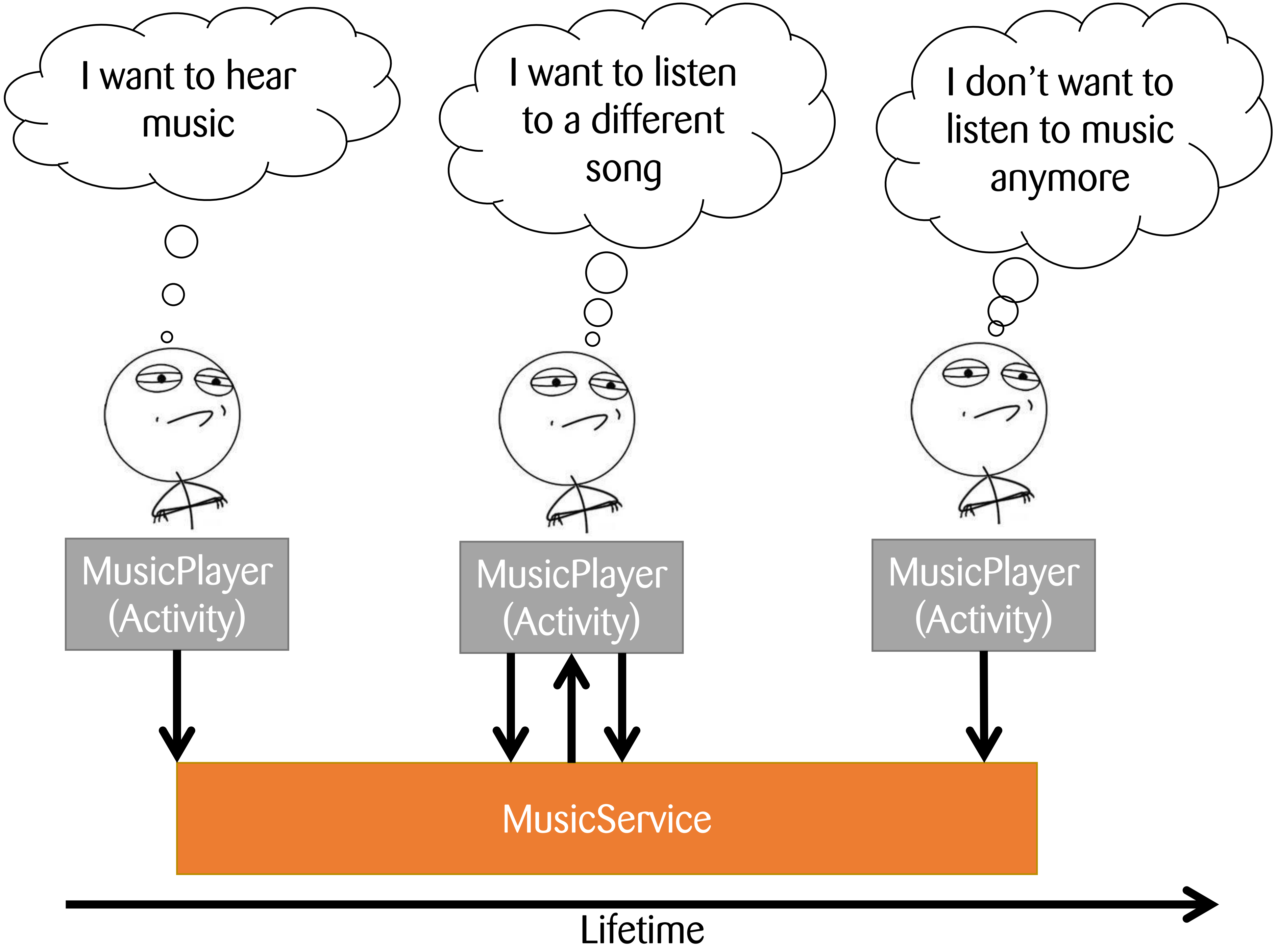
Activity	**Service**
ContentProvider	BroadcastReceiver

Android Service

Definition

- Component for long running operations
  - Playing music, long running web operations, data fetching, …

- Types of Services
  - **Bound Services**
    Exists as long any other component is bound to it
    Start use bindService(…)
  - **Started Service**
    Exists as long as nobody stops it
    Start use startService(…)


I want to hear music
I want to listen to a different song
I don't want to listen to music anymore
MusicPlayer (Activity)
MusicPlayer (Activity)
MusicPlayer (Activity)
MusicService
Lifetime

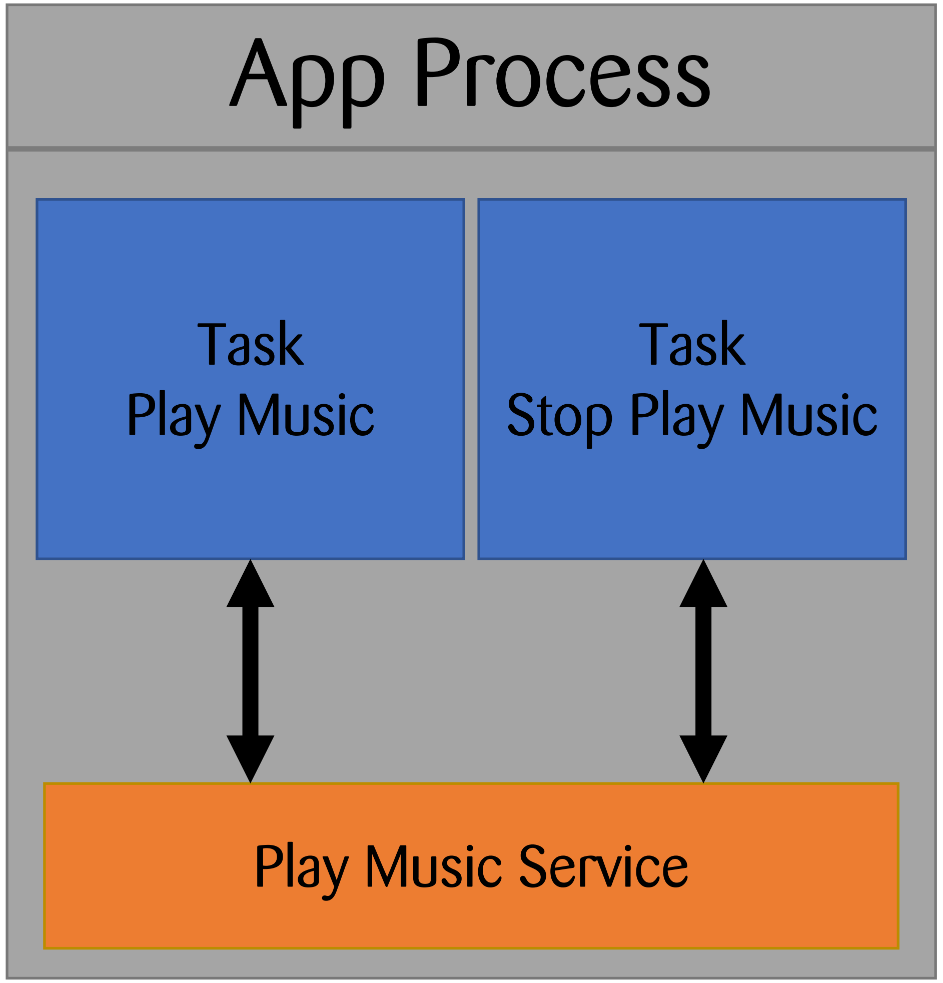
Android Service

Technical implementation

- Class extending Service
- Register in Manifest

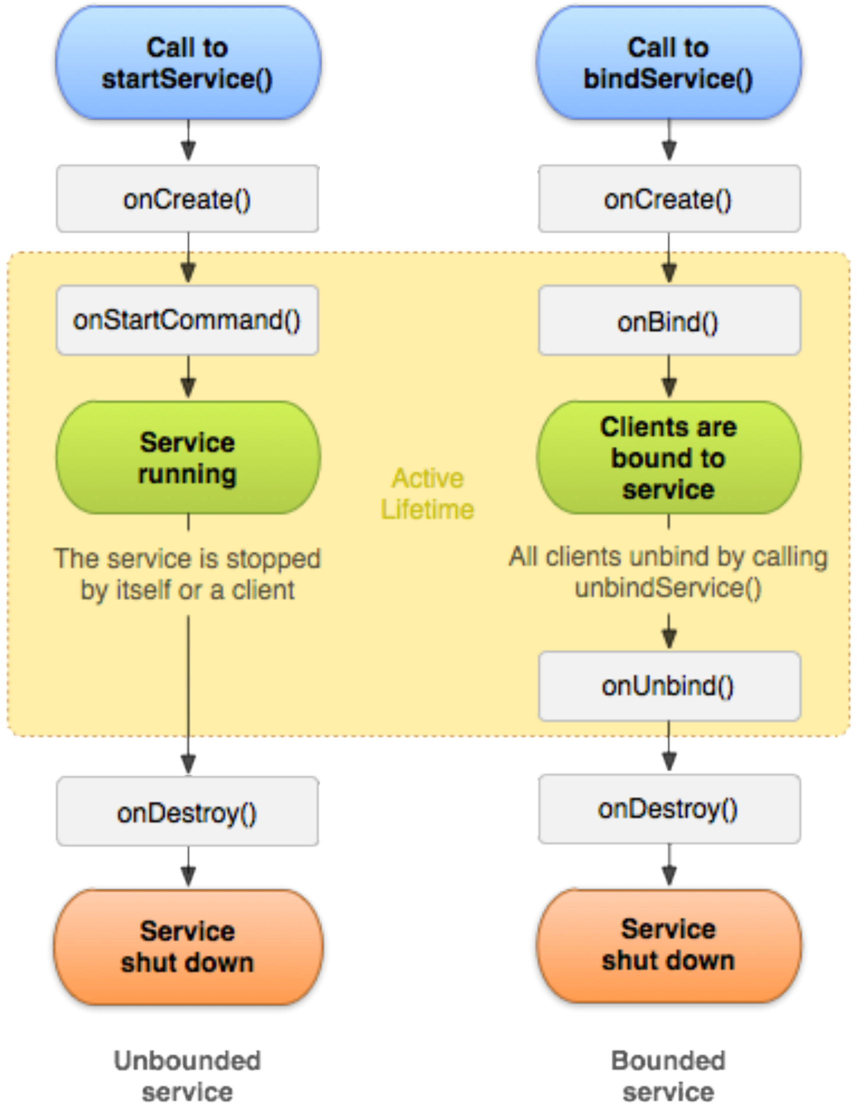
- Runs on UI Thread
- Service runs only once

- Stop the service
  - stopService(Intent)
  - stopSelf() in the service



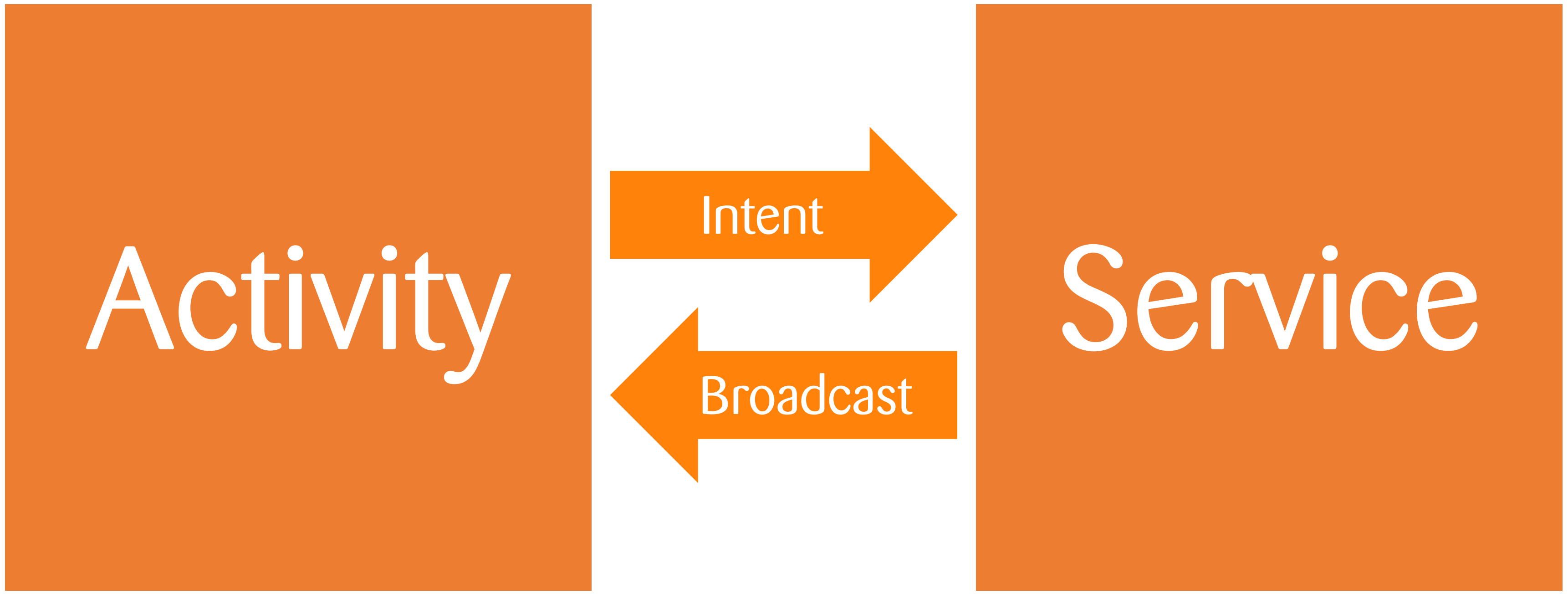
Android Service

Lifecycle



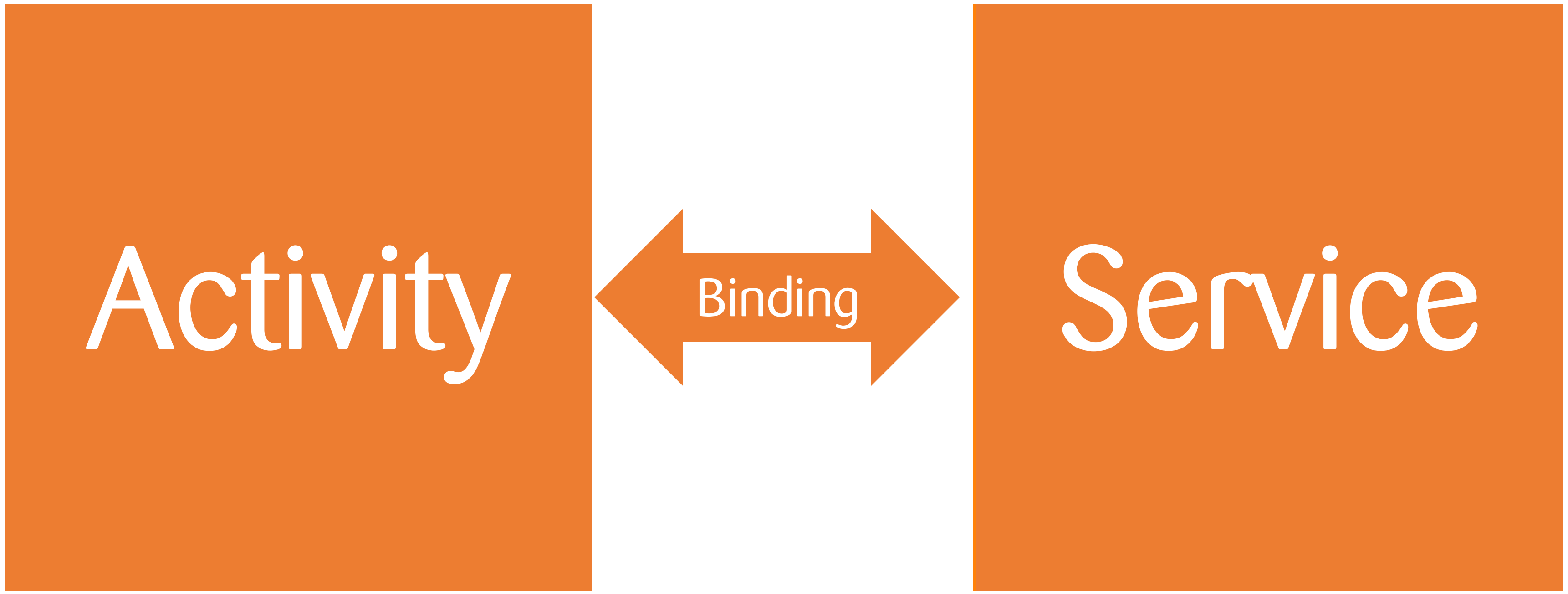
Service

Communication – Started Service



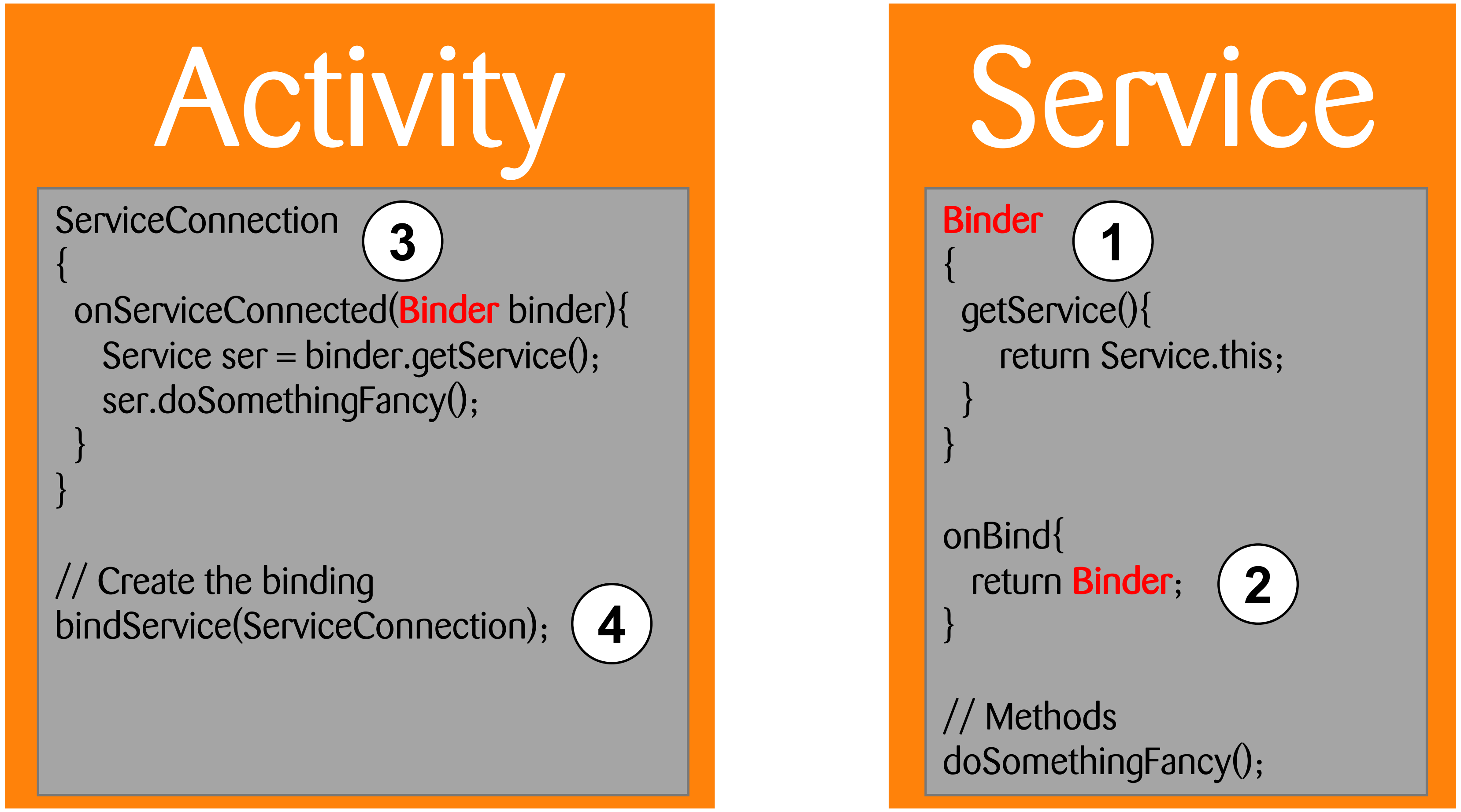
Service

Communication – Bound Service



Service

Binding Service Big Picture



```
ServiceConnection          3
{
  onServiceConnected(Binder binder){
     Service ser = binder.getService();
     ser.doSomethingFancy();
  }
}

// Create the binding
bindService(ServiceConnection);     4
```

```
Binder          1
{
  getService(){
     return Service.this;
  }
}

onBind{
   return Binder;          2
}

// Methods
doSomethingFancy();
```

Service

Bind to Service

- Create a Binder in the Service

```
public class MyServiceBinder extends Binder {   1
       public MyBindableService getService() {
              return MyBindableService.this;
       }
}

private final IBinder binder = new MyServiceBinder();

@Override
public IBinder onBind(Intent intent) {
       return binder; 2
}
```

Service

Bind to Service

- Create a ServiceConnection in your Activity

```
private ServiceConnection serConn= new ServiceConnection() {

@Override
public void onServiceConnected(ComponentName name, IBinder binder){
       MyServiceBinder customBinder = (MyServiceBinder)binder;
       MyBindableService service = customBinder.getService();
}                                                               ③

@Override
public void onServiceDisconnected(ComponentName name) {

}
};
```

Service

Bind to Service

- Use bindService() and unbindService() to connect/disconnect

```
@Override
protected void onResume() {
        super.onResume();
        Intent intent = new Intent(this, MyBindableService.class);
        bindService(intent, serCon, Context.BIND_AUTO_CREATE);
}

@Override
protected void onPause() {
        super.onPause();
        unbindService(serCon);
}
```

4

Service

Combine Started & Bound Service

- Use startService() to start a service indefinitely
- Use bindService() to connect to the started Service

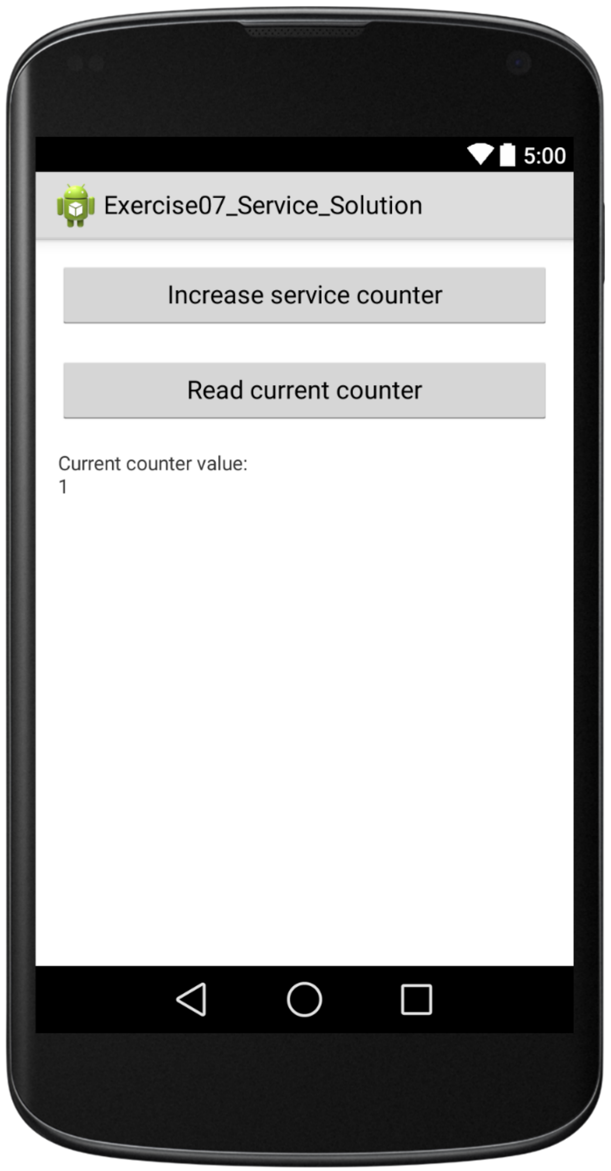
Exercise07_Service

Exercise07_Service

- Increase service counter
  → increase counter

- Read current counter
  → Read counter from service and
  display below

- Counter should keep value even when
  the app is closed (using the back button or wipe
  It out)



Current counter value:
1

Aufgaben:

1. Analysiere das AndroidManifest.xml und verstehe wie Services definiert werden
2. Implementation Bindable Service
   1. In der CounterService-Klasse muss erst ein Binder implementiert werden der in onBind(…) zurückgegeben werden kann (siehe slides)
   2. Erstelle eine ServiceConnection in der MainActivity. In der onServiceConnected-Methode müssen wir uns nun die Service-Instanz welche wir über den Binder erhalten in der MainActivity-Klasse merken.
   3. Benutze nun die lifecycle-methoden der Activity um den Service zu binden:
      - **I.** **onCreate()** → startService()
      - **II.** **onResume()** → bindService(…)
        BenutzeContext.BIND_AUTO_CREATE-flag damit der service gestartet wird falls er noch nicht vorhanden ist. das
      - **III.** **onPause()** → unbindService(…)

Verwendede API Klassen:

- Activity
  - Wir benötigen die lifecycle-Methoden zum binden eines services
- Service
  - BindableService ist ein normaler Service
- IntentService
  - CustomIntentService ist ein IntentService → Jobs werden in einem Hintergrund-Thread ausgeführt