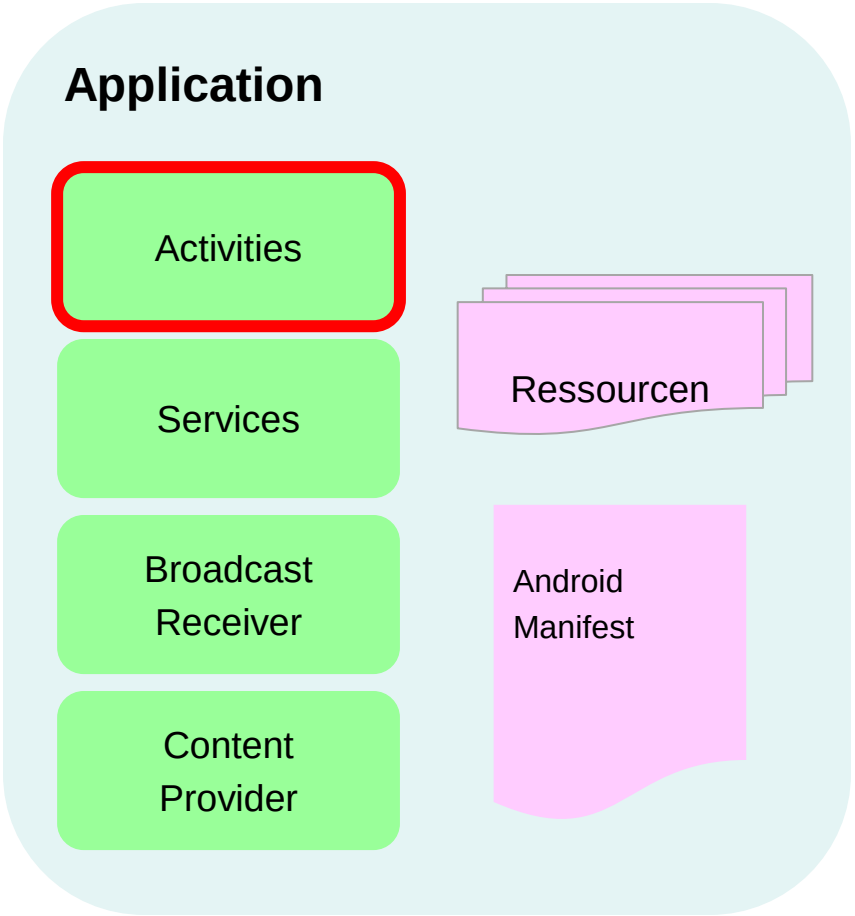
living place
hamburg

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

WP SmartHome
Aktivitäten

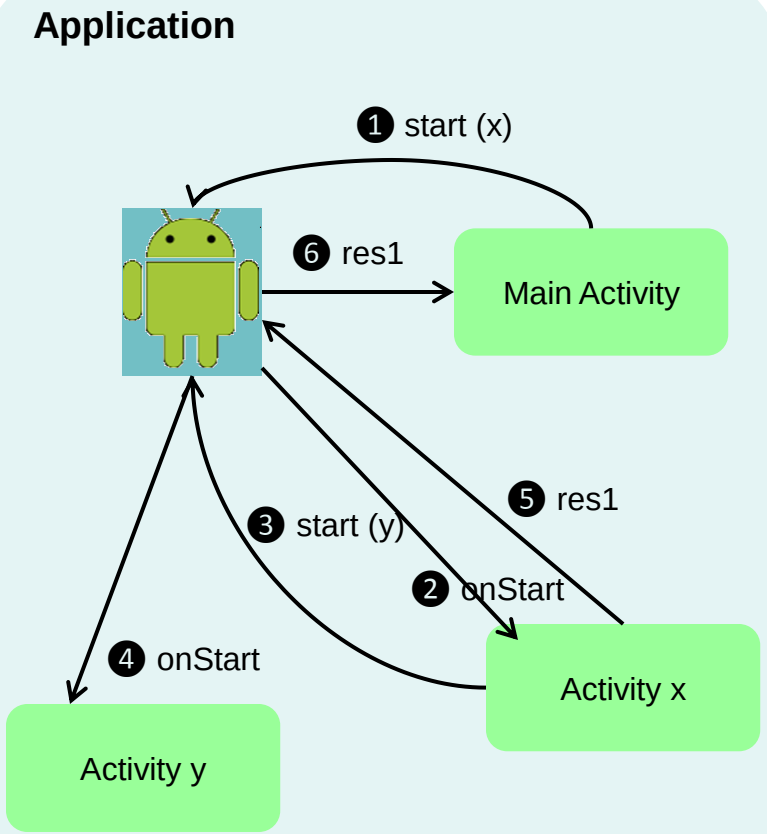
Prof. Dr. Ing. Birgit Wendholt

Komponenten einer Android Applikation



Aktivitäten, Applikationen, Backstack, Lifecycle

- Eine Activity ist eine Komponente, die (in der Regel) mit dem Benutzer über eine UIF kommunzieren kann.
- Eine **Applikation** besteht aus mehreren lose gekoppelten Aktivitäten.
- Eine Applikation hat eine **Hauptaktivität (main activity)**, die gestartet wird, wenn die Applikation startet.
- Aktivitäten kommunizieren über Intents. Eine Aktivität beantragt beim Android System das Starten einer anderen Aktivität mittels eines **Intents.**

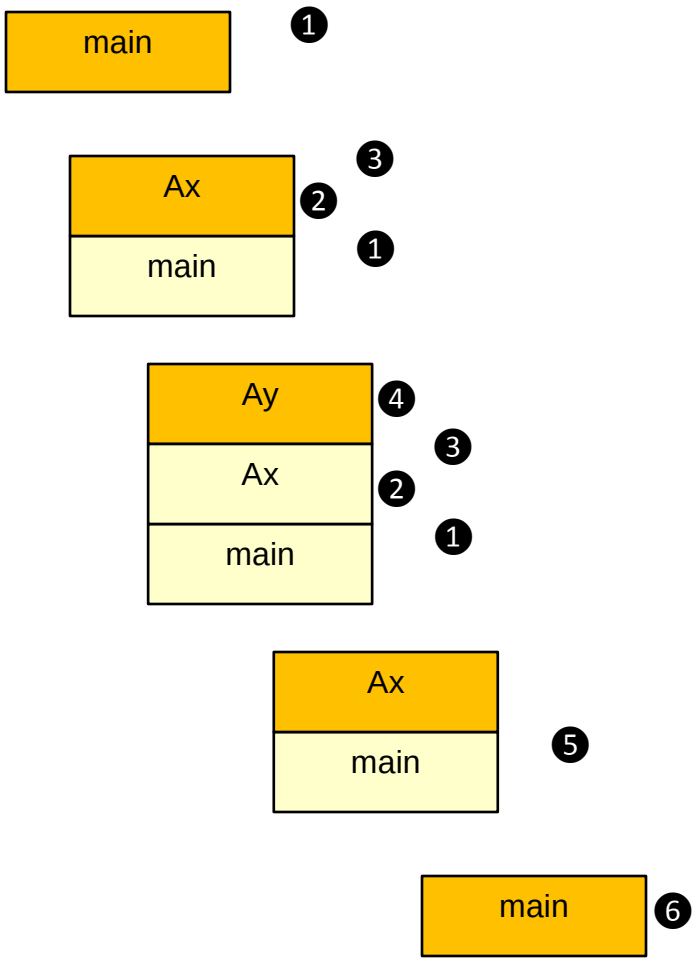


Aktivitäten, Applikationen, Backstack, Lifecycle

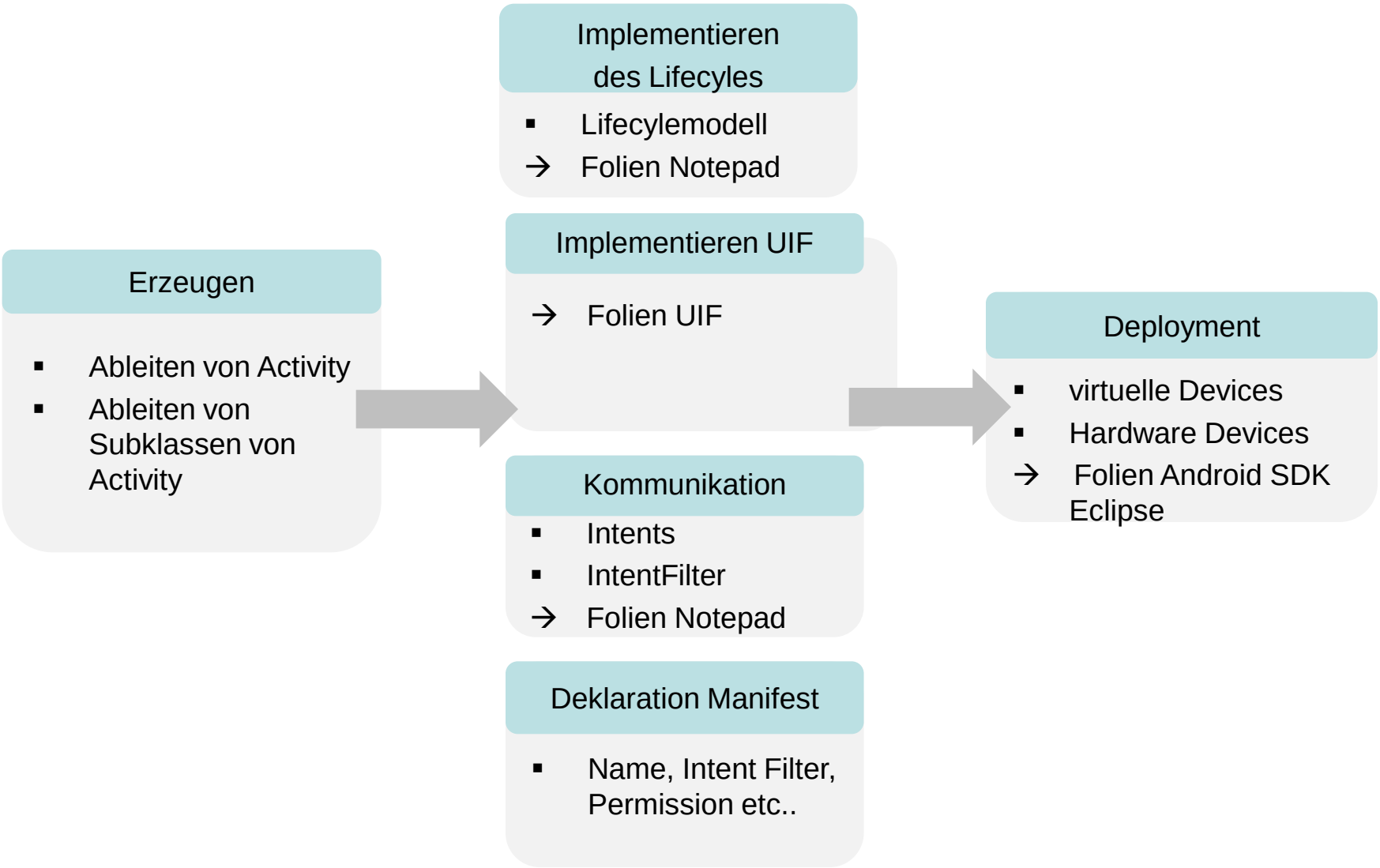
- Wenn Android eine Aktivität startet, wird die aktive gestoppt auf den Aktivitäten Stack (**backstack**) gelegt. Die startende Aktivität wandert ebenfalls auf den **backstack** und erhält den Benutzerfokus.

- Verlässt der Benutzer eine Aktivität, entfernt Android diese vom Stack, zerstört sie und reaktiviert die vorhergehende Aktivität.

- Die Kontrolle des Zustandswechsels einer Aktivität liegt beim Android System. Aktivitäten werden über den Zustandswechsel mittels **Lifecycle**-Methoden (Callback Methoden) informiert. Wird z.B. eine Aktivität gestoppt, wird die Methode ***onStop*** der Aktivität aufgerufen.



Implementieren einer Aktivität



Erzeugen einer Aktivität

- → Ableiten von Activity oder
- → einer der existierende Subklassen von Activity (z.B. ListActivity, TabActivity)

```
public class MyHelloAndroidActivity
    extends Activity {
    ...
}
```

```
public class Notepadv1 extends
    ListActivity {
}
```

```
public class ProcessorTabActivity
    extends TabActivity {
    ...
}
```

Erzeugen einer Aktivität

→ Implementieren der Lifecycle Methoden der Aktivität

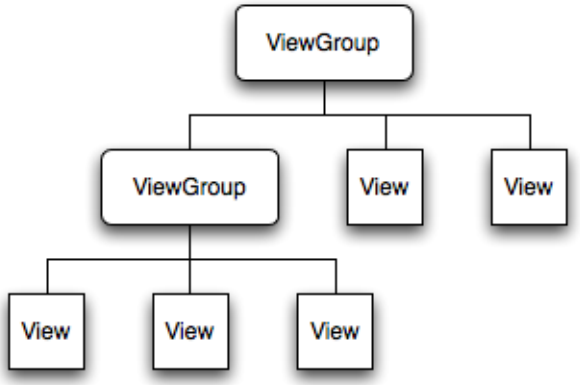
→ wichtigste Methode onCreate():
- muss überschrieben werden
- wird aufgerufen, wenn die Aktivität erzeugt wird.
- Initialisieren der Aktivität
- Initialisieren des UIF der Aktivität mit setContentView().

- weitere Lifecycle Methoden

```
public class MyHelloAndroidActivity
    extends Activity {

    @Override
    public void onCreate(Bundle
                savedInstanceState) {

        super.onCreate(
                savedInstanceState);

        setContentView(
                R.layout.main);

    }
}
```

Das UIF einer Aktivität

- Das UIF einer Aktivität ist eine Hierarchie von Views und ViewGroups.



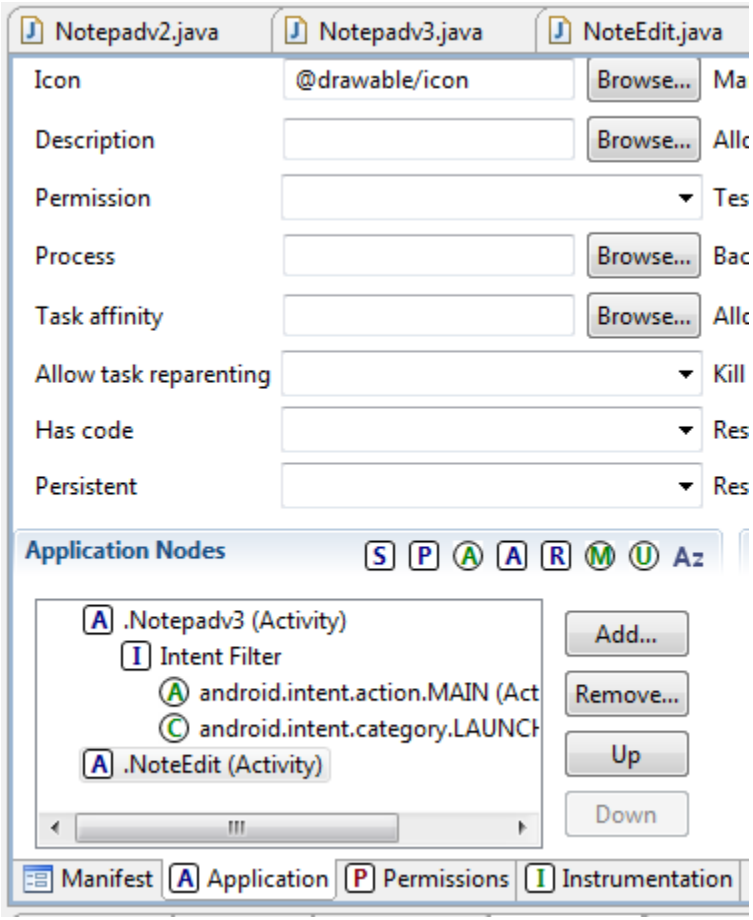

```
<?xml version="1.0" encoding="utf-8"?>
<TextView android:id="@+id/text1"
   xmlns:android="http://schemas.android.
         com/apk/res/android„
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>

public void onCreate(Bundle
         savedInstanceState) {
    ...
    setContentView( R.layout.main);
}
```

- Erzeugen eines UIF
  - als XML Layout-Datei, deren Referenz der Methode setContentView() übergeben wird
  - programmatisch durch explizites Erzeugen von View-Objekten

```
public void onCreate(Bundle
         savedInstanceState) {
    ...
   TextView tv = new TextView(this);
   tv.setText("Hello Android World");
   setContentView(tv);
}
```

Deklaration der Aktivität im Manifest

- Bevor Android Aktivitäten starten kann, müssen diese im Manifest der Applikation definiert werden.
- Im Manifest können verschiedene Eigenschaften der Aktivität definiert werden.
  - Intent Filter
  - **notwendiges** Attribut: Klassenname
  - Label für die Aktivität,
  - Icon für die Aktivität
  - Theme oder Style für das UI



Kommunikation zwischen Aktivitäten

- Aktivitäten können andere Aktivitäten starten, indem sie dem Android System die Absicht (**Intent**) mitteilen eine Aktivität zu starten.

- Der Intent wird einer der ***start*** Methoden (***startActivity/startActivity ForResult***) beim Aufruf übergeben.

- Unterschieden werden:

  – **explizite** Intents, die eine Aktivität direkt adressieren.

  – **implizite** Intents mit einer Aktion, die von Android gegen die Intent Filter der registrierten Aktivitäten gemachted wird.

```
Intent i = new Intent(this,
                      NoteEdit.class);
i.putExtra(NotesDbAdapter.KEY_ROWID, id);
startActivityForResult(i, ACTIVITY_EDIT);
```

```
Intent intent = new
    Intent(Intent.ACTION_SEND);

intent.putExtra(Intent.EXTRA_EMAIL,
                      recipientArray);
startActivity(intent);
```

Intents

- Intent Nachrichten sind das Kommunikationsmedium für Activitäten, Services und Broadcast Receiver innerhalb und zwischen Applikationen.

- Ein Intent Objekt, ist eine passive Datenstruktur, die eine abstrakte Beschreibung enthält über
  - die Operation, die auszuführen ist
  - ein Ereignis, das aufgetreten ist und abonniert wurde (im Falle von Broadcast Receivern)

- Android sucht passende Aktivitäten Services und Broadcast Receiver und instantiiert diese ggf.

- Aufbau eines Intents:
  - **Komponentenname** ComponentName Objekt: voll qualifizierter Klassenname des Empfängers (**optional**)
  - Namen einer **Aktion** oder eines Ereignisses (→ Konstanten in Intent): bestimmt wie die Daten und Extra Felder des Intents aufgebaut sind.
  - **Daten:** die URI oder der MIME-Typ der Daten für die Aktion. Z.B. für ACTION_VIEW eine **http: URI**, für ACTION_CALL eine **tel: URI**
  - **Kategorie:** CATEGORY_ LAUNCHER, etc.
  - **Extras:** Schlüssel-Wert Paare, für beliebige Informationen, die zwischen Komponenten transportiert werden. Einige Aktionen benötigen spezielle Extras, z.B. ACTION_TIMEZONE_CHANGED benötigt das Extra „timezone“

Intent Resolution

explizite Intents

- Android liefert den Intent an eine Instanz der Zielklasse aus.
- Alle weiteren Informationen des Intents werden ignoriert.

implizite Intents

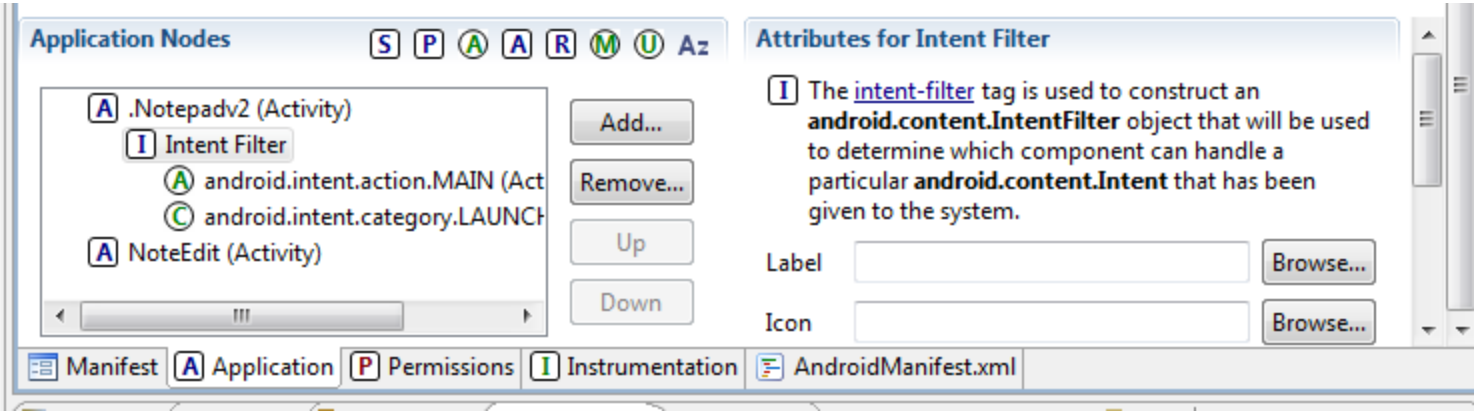
- Android sucht nach der Komponente, die am Besten für die Behandlung des Intents geeignet ist.
- Der Inhalt des Intents wird mit den **Intent Filtern** von Komponenten verglichen.
- 3 Aspekte des Intents werden beim Abgleich verwendet: **Aktion**, **Daten** (URI und Datentyp), **Kategorie**.

Intent Filter

- Intent Filter beschreiben, welche Arten von Intents eine Komponente empfängt.

→ mehrere Intent Filter pro Komponente möglich.

- **Beispiel 1**: Jede Applikation, die mit dem Android SDK erzeugt wird, enthält eine Stub Aktivität mit dem Intent Filter
  - für die Aktion "**main**" → Aktivität wird beim Starten der Applikation gestartet
  - und die Kategorie "**launcher**" (Aktivität wird im Launcher platziert.)



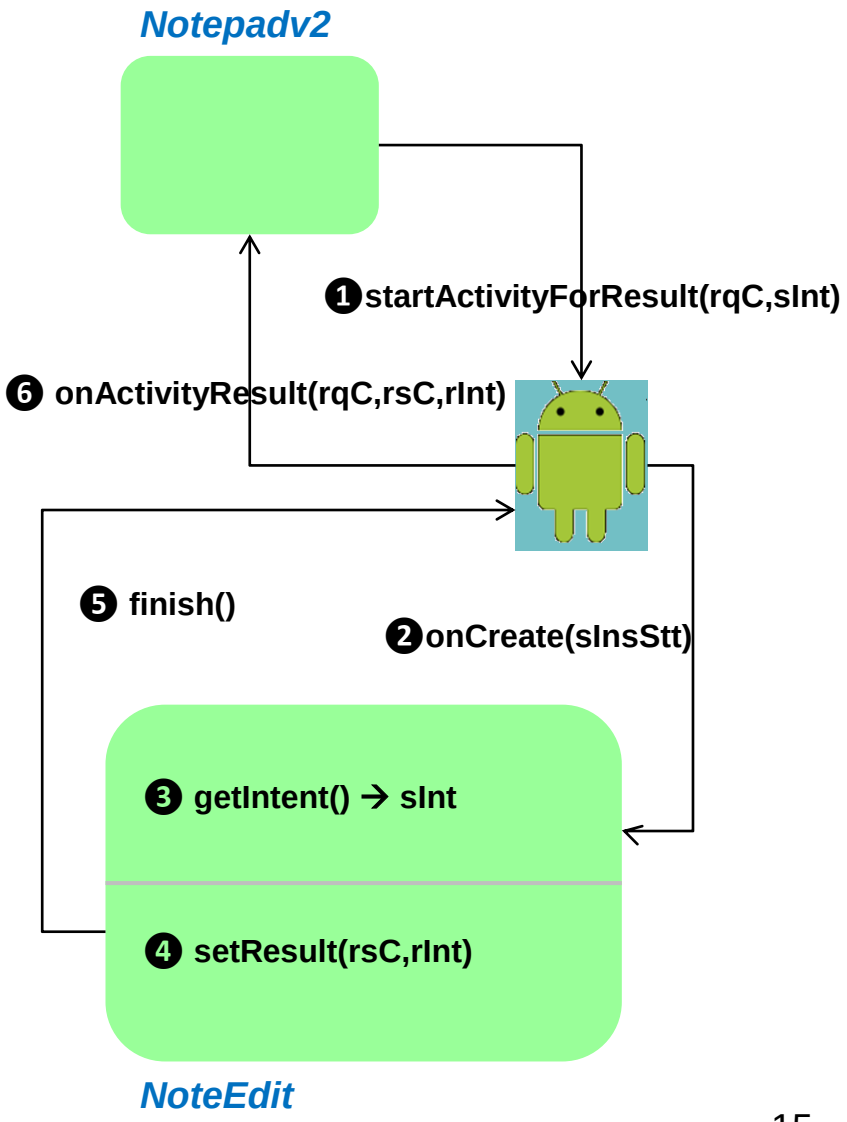
Intent Filter

- **Beispiel 2**: ***NoteEditor*** Aktivität definiert zwei Intent Filter, einen für das Editieren einer Notiz und einen für das Erstellen einer Notiz. (→ Note Pad Example).

```
<activity android:name="NoteEdit">
  <intent-filter android:label="@string/resolve_edit">
    <action android:name="android.intent.action.VIEW"></action>
    <action android:name="android.intent.action.EDIT"></action>
    <action android:name="com.android.notepad.action.EDIT_NOTE"></action>
    <category android:name="android.intent.category.DEFAULT"></category>
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.INSERT"></action>
    <category android:name="android.intent.category.DEFAULT"></category>
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note"></data>
  </intent-filter>
</activity>
```

Starten einer Aktivität mit Ergebnis

- Aktivitäten die ein Ergebnis liefern sollen, werden mit startActivityForResult() gestartet.
- Kommunikation zwischen Aktivitäten ist asynchron.
- Eine Aktivität, die ein Ergebnis erwartet, muss die Callback-Methode onActivityResult() implementieren.
- Android ruft die Methode onActivityResult() mit dem Ergebnis der gestarteten Aktivität auf.
- **Beispiel**: Aufruf der ***NoteEdit*** Aktivität aus der ***Notepadv2*** Aktivität zum Editieren oder Erstellen von Notizen.



Starten einer Aktivität

- ***Notepadv2*** ruft in der Methode ***createNote*** die ***NoteEdit*** Aktivität mit dem Request-Code ACTIVITY_CREATE auf.
- ***Notepadv2*** ruft im ClickListener die ***NoteEdit*** Aktivität mit dem Request-Code ACTIVITY_EDIT auf und befüllt die Extras des Intents mit einer Referenz auf die Notiz und den Daten der Notiz.

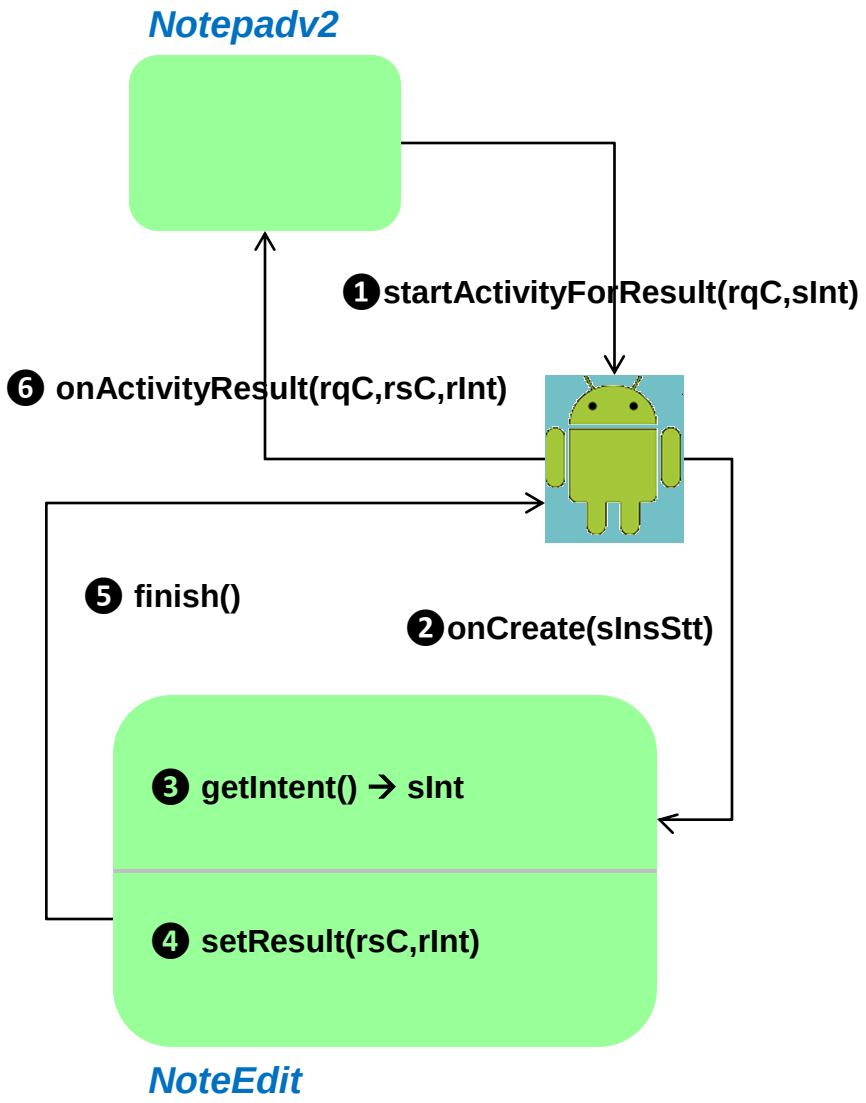
```
public class Notepadv2 extends ListActivity {

 private void createNote() {
    Intent i = new Intent(this, NoteEdit.class);
    startActivityForResult(i, ACTIVITY_CREATE);
 }
 @Override
 protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Cursor c = mNotesCursor;
    c.moveToPosition(position);
    Intent i = new Intent(this, NoteEdit.class);
    i.putExtra(NotesDbAdapter.KEY_ROWID, id);
    ... // weitere Extras der Notiz
    startActivityForResult(i,ACTIVITY_EDIT);
 }
}
```

Auswerten des Ergebnisses


```
public class Notepadv2 extends ListActivity {
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    Bundle extras = intent.getExtras();

    switch (requestCode) {
    case ACTIVITY_CREATE:
          String title = extras.getString(NotesDbAdapter.KEY_TITLE);
          String body = extras.getString(NotesDbAdapter.KEY_BODY);
          mDbHelper.createNote(title, body);
          fillData();
          break;
    case ACTIVITY_EDIT:
          Long mRowId = extras.getLong(NotesDbAdapter.KEY_ROWID);
          if (mRowId != null) {
             String editTitle = extras.getString(NotesDbAdapter.KEY_TITLE);
             String editBody = extras.getString(NotesDbAdapter.KEY_BODY);
             mDbHelper.updateNote(mRowId, editTitle, editBody);
          }
          fillData();
          break;
}}
```

Auswerten des Ergebnisses

- Wenn im Beispiel die ***NoteEdit*** Aktivität endet, ruft Android auf der ***Notepadv2*** Aktivität die Methode ***onActivityResult*** auf und übergibt dabei:
  - ***requestCode***: derselbe Code, der von der ***Notepadv2*** Aktivität beim ***start ActivityForResult*** übergeben wurde → Zuordnung von Ergebnissen zu Aufrufen
  - ***resultCode***: zeigt an, ob der Aufruf mit dem ***requestCode*** erfolgreich war (=0)
  - ***intent***: Intent der von der ***NoteEdit*** Aktivität erzeugt wird und als Ergebnis gesetzt wird. Im Beispiel sind in dem Ergebnis-Intent die Änderungen an der Notiz enthalten.



Beenden einer Aktivität

- Eine Aktivität wird durch Aufruf ihrer finish() Methode "beendet".
- Die Methoden sollte nur dann aufgerufen werden, wenn der Benutzer nicht wieder in diese Aktivität gelangen soll.
- Vor Aufruf von ***finish*** muss eine Aktivität, wenn sie ein Ergebnis liefert, dieses mit ***setResult*** gesetzt haben.
- ***setResult*** Parameter:
  - ***resultCode***: eine der Konstanten für Ergebniscodes in Aktivitäten (z.B. RESULT_OK)
  - ***intent***: die Daten, die mit dem Ergebnis transportiert werden sollen.

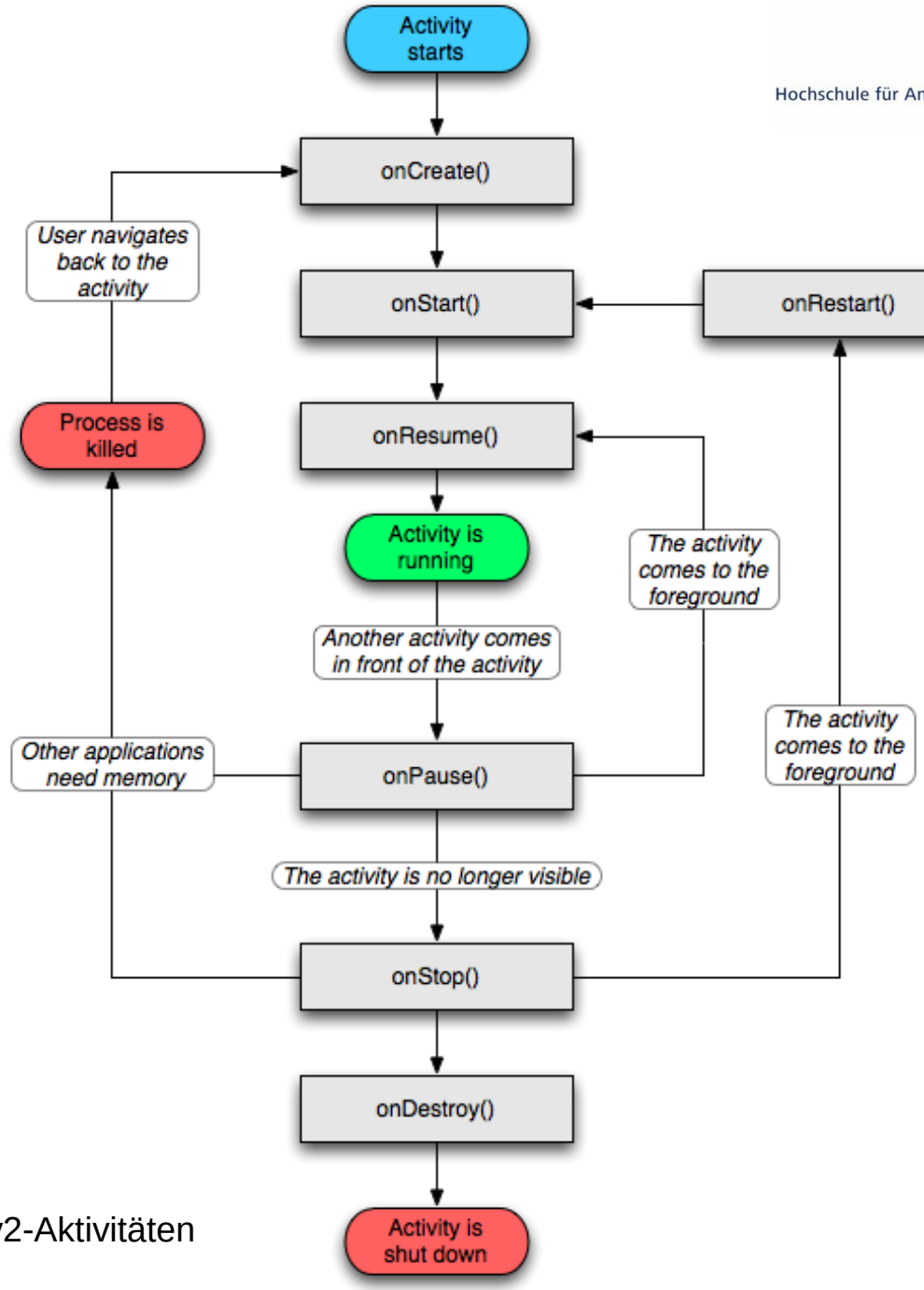
Auszug aus der ***NoteEdit*** Aktivität

```
Bundle bundle = new Bundle();
bundle.putString(
    NotesDbAdapter.KEY_TITLE,
    mTitleText.getText().toString());
bundle.putString(NotesDbAdapter.KEY_BODY,
    mBodyText.getText().toString());

if (mRowId != null) {
    bundle.putLong(NotesDbAdapter.KEY_ROWI
    D, mRowId);
}

Intent mIntent = new Intent();
mIntent.putExtras(bundle);
setResult(RESULT_OK, mIntent);
finish();
```

Lifecycle Management von Aktivitäten

- Aktivitäten können nicht selber darüber bestimmen, wann sie aktiviert werden und den Benutzerfokus erhalten.
- Das Android System entscheidet, wann eine Aktivität den Fokus erhält, gestoppt wird etc.
- **Grund**: Android muss die Grundfunktionalität des Gerätes (Smartphones) immer sicher stellen:
  - eingehende Anrufe / Nachrichten haben Priorität vor allen anderen Aktivitäten
  - kritische Zustände von Ressourcen des Gerätes (Battery, Speicher etc.) müssen behandelt werden. → Aktivitäten werden gestoppt
- **Grund**: Aus Effizienz und Ressourcen Gründen wird ein Modell benötigt, das Aktivitäten im Hintergrund betreibt und dabei den Zustand der Aktivität erhält bis sie erneut den Fokus erhält.

➔ Lifecyclemodell für Aktivitäten gemanaged durch das Android System


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Lifecycle von Aktivitäten

- Die drei wesentlichen Zustände:
  - **resumed:** Aktivität arbeitet im Vordergrund und hat Benutzerfokus. (aka "running".)
  - **paused:** Eine andere Aktivität ist sichtbar oberhalb der eigenen Aktivität. Die eigene wird teilweise oder ganz verdeckt. Eine pausierende Aktivität ist vollständig lebendig, bleibt im Speicher und ist mit dem Windowmanager verbunden.
  - **stopped:** Aktivität wird vollständig von einer andere Aktivität verdeckt. Die Aktivität ist im “background”. Eine gestoppte Aktivität ist vollständig lebendig, bleibt im Speicher ist aber nicht mehr mit dem Window Manager verbunden.

- Eine Aktivität, die im Zustand “paused” oder “stopped” ist, kann aus dem System entfernt werden, indem die finish() Methode aufgerufen oder der Linux Prozesses beendet wird.
- Wenn eine Aktivität dann neu geöffnet wird, muss sie vollständig neu erzeugt werden.

Lifecyle Callbacks

- Bei jedem Zustandswechsel ruft Android eine entsprechende Lifecycle-methode der Aktivität (Lifecycle Callback) auf.
- Lifecycle Methoden sind Hook-methoden mit einer Standardimple-mentierung in der Klasse ***Activity*** und allen Subklassen von ***Activity*** der Android Bibliothek.
- Überschreiben dieser Methoden in eigenen Aktivitäten sollten daher immer die Implementierung der Superklasse aufrufen.

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this acti
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is no
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Lifecycle Callbacks und ihre Bedeutung

Methode	Bedeutung	„Killable after?“	Nächste
onCreate	Aufruf beim Erzeugen. Methode für Initialisierung von Views, Adaptern etc. Parameter: letzter Zustand der Aktivität.	nein	onStart
onRestart	Aufruf, wenn eine gestoppte Aktivität erneut gestartet wird.	nein	onStart
onStart	Aufruf, bevor die Aktivität sichtbar wird.	nein	onResume onStop
onResume	Aufruf vor der Interaktion mit dem Benutzer. Aktivität liegt oben auf dem **backstack**.	nein	onPause
onPause	Aufruf, wenn Android dabei ist eine andere Aktivität in den Zustand “resumed” zu versetzen. **Verwendung**: Speichern von Änderungen, Anhalten von Animationen,etc.	ja	onResume onStop
onStop	Aufruf, wenn die Aktivität nicht mehr sichtbar ist. Sie wird zerstört oder von einer andern Aktivität verdeckt.	ja	onRestart onDestroy
onDestroy	Aufruf, bevor die Aktivität zerstört wird, entweder als Folge des Aufrufs von finish() oder durch das System, um Speicher frei zu räumen.	ja	nichts

Bedeutung von „Killable After“

- zeigt an, ob das Android System den (Linux) Prozess der Aktivität zerstören kann, wenn die Methode beendet ist.
- gilt für onPause(), onStop(), und onDestroy()
- gilt bereits für onPause(), die Methode die vor den zwei anderen durchlaufen wird

→ onPause() ist die letzte Methode die garantiert durchlaufen wird, bevor der Prozess der Aktivität zerstört werden kann.

➔ in onPause() sollten daher alle wichtigen Daten persistiert werden.

- die mit „nein“ gekennzeichneten Methoden bedeuten nicht, dass der Prozess einer Aktivität in diesem Zustand nicht zerstört werden kann.
- Unter extremen Bedingungen können Prozesse jederzeit zerstört werden → Processes and Threading

Geschachtelte “Lebenszeiten”

- **entire lifetime:**
  - zwischen onCreate() und onDestroy().
  - Initialisierung der Aktivität in onCreate() (z.B. das Setzen des Layouts)
  - Freigabe aller offenen Ressourcen im onDestroy()
  - **Beispiel**: Eine Aktivität, die in einem separaten Thread Daten über das Netz lädt, sollte den Thread im onCreate() erzeugen und diesen im onDestroy() stoppen.
- **visible lifetime:**
  - zwischen onStart() und onStop()
  - Aktivität ist sichtbar
  - onStop() wird aufgerufen, wenn eine andere Aktivität startet und sichtbar wird.
  - zwischen den Zuständen Verwaltung von Ressourcen für die Anzeige
  - **Beispiel**: Ein BroadcastReceiver, der den Wechsel von Portrait zu Landscape verfolgt, wird im onStart() initialisiert und im onStop() deregistriert.
  - onStart() und onStop() können mehrere Male während der entire lifetime aufrufen werden.

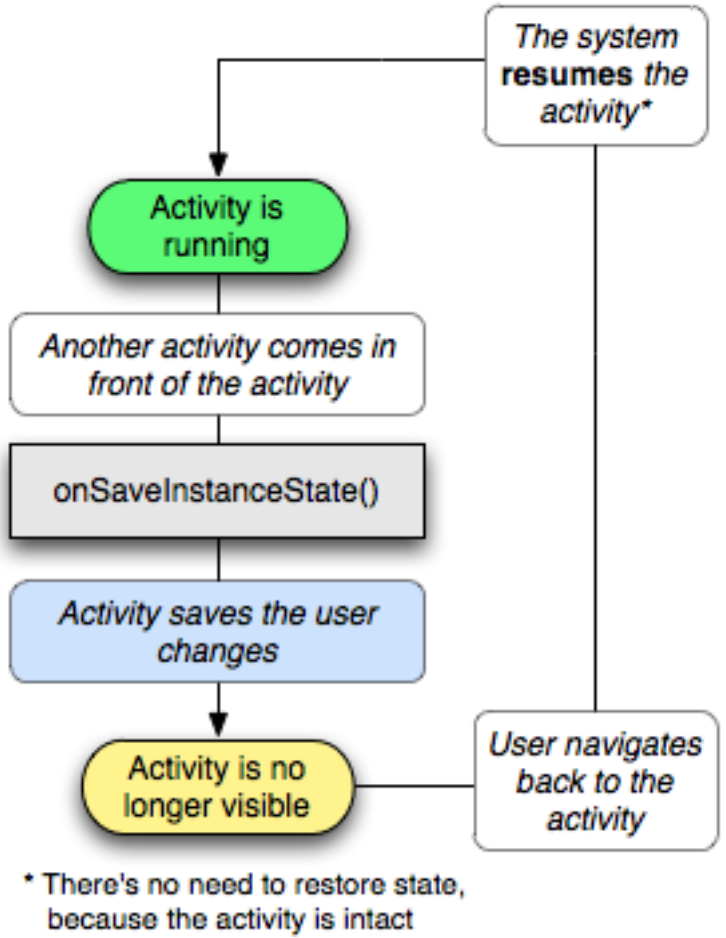
Geschachtelte “Lebenszeiten”

- **foreground lifetime**
  - zwischen onResume() und onPause().
  - Aktivität ist im Vordergrund aller Aktivitäten und hat Eingabefokus.
  - eine Aktivität kann sehr häufig zwischen Vorder- und Hintergrund wechseln.
  - onPause() wird auch dann aufgerufen, wenn das Gerät in den Sleep-Modus geht.
  - sehr häufiger Zustandswechsel ➔ der Source Code in diesen Methoden sollte leichtgewichtig sein.

Zustand einer Aktivität speichern

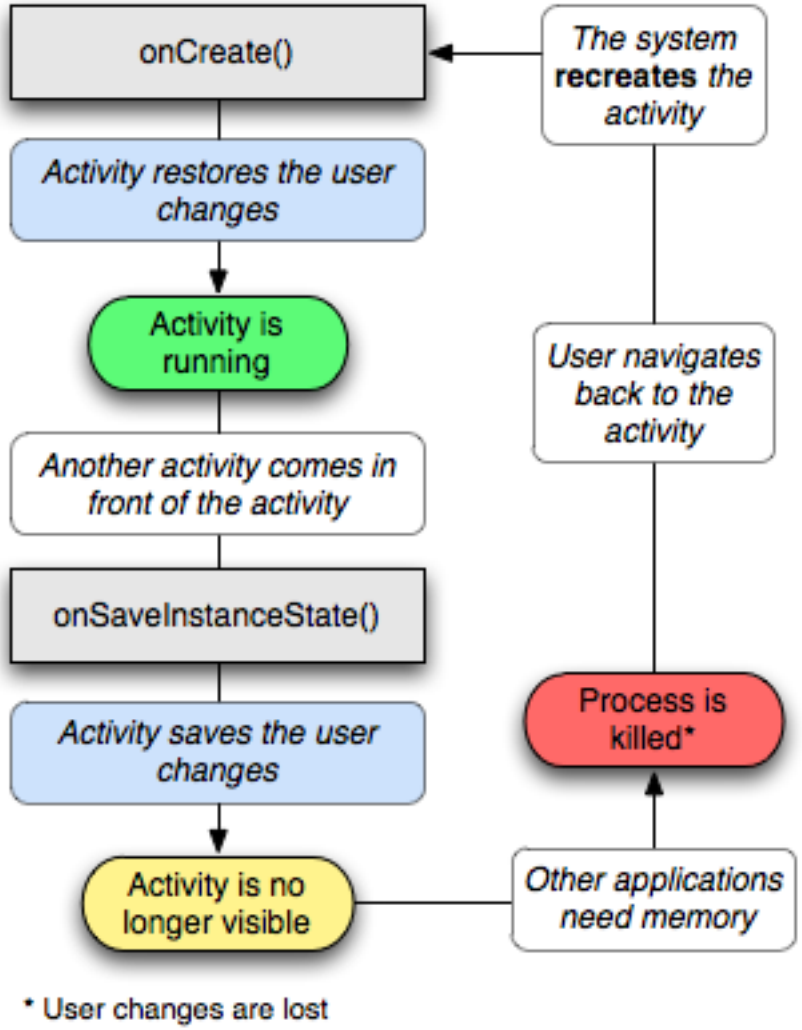
- Aktivitäten im Zustand “paused” or “stopped” behalten ihren Zustand, da das Activity Objekt im Speicher bleibt.
- Lokale Änderungen bleiben erhalten.
- Kommt die Aktivität in den Vordergrund, sind alle Änderungen noch vorhanden.

→ keine Notwendigkeit den Zustand einer Aktivität zu speichern.



Zustand einer Aktivität speichern

- Zerstört das Android System den Prozess einer Aktivität, muss diese erneut erzeugt werden, wenn der Benutzer zu ihr zurück navigiert.
- In diesem Fall muss der letzte Zustand der Aktivität wiederhergestellt werden.
- Zustandsinformationen sollten in der Callback Methode onSaveInstance State() persistiert und beim Neustart rekonstruiert werden.



Zustand einer Aktivität speichern

- onSaveInstanceState() wird vom Android System aufgerufen, bevor eine Aktivität zerstört werden kann.
- onSaveInstanceState() wird mit einem mutable Bundle Objekt aufgerufen, in dem der Zustand der Aktivität in Name-Value Paaren gespeichert wird.
- Danach zerstört Android den Prozess der Aktivität.
- Wird die Aktivität neu gestartet (back Navigation des Benutzer), dann übergibt Android der Methode onCreate() das Bundle, aus dem der letzte Zustand rekonstruiert werden kann.

- Es gibt keine Garantie, dass onSaveInstanceState() aufgerufen wird, bevor eine Aktivität zerstört wird. Z.B. dann nicht wenn der Benutzer die Aktivität explizit mit der BACK Taste beendet.
- Wenn die Methode aufgerufen wird, dann immer vor onStop(), ggf. vor onPause().

➔ onSaveInstanceState() darf nur *transiente* Information speichern.

➔ *Persistente* Zustandsdaten müssen immer in der Methode onPause() gespeichert werden.

Zustand einer Aktivität speichern

- Default-Implementierung von onSaveInstanceState() in Activity
  - ruft onSaveInstanceState() für jede View des Layouts auf. Das erlaubt es allen Views, Informationen über den eigenen Zustand zu speichern.
  - Widgets des Android Frameworks implementieren diese Methode: im UI sichtbare Änderungen werden gespeichert und wiederhergestellt.
  - **Beispiel:** EditText speichert den Text, den der Benutzer eingegeben hat.
  - **Voraussetzung**: Jedes Widget hat eine eindeutige ID.

➔ onSaveInstanceState() in der eigenen Aktivität muss immer die Implementierung der Superklasse aufgerufen.

Koordinieren von Aktivitäten

- Wenn eine Aktivität eine andere startet, durchlaufen beide spezifische Zustände ihres Lifecycles.
- Die erste ist im Zustand “pause” ggf. “stop”, während die zweite erzeugt wird.
- Wenn sich beide Aktivitäten Daten teilen, z.B. über eine SQLlite DB, müssen die Änderungen in der korrekten Methode gespeichert werden.

- **Szenario**: Aktivität A startet Aktivität B
  - Die onPause() Methode von Aktivität A wird ausgeführt.
  - Die onCreate(), onStart() und onResume() Methoden von Aktivität B werden nacheinander ausgeführt Aktivität B hat den Fokus.
  - Sobald Aktivität A nicht mehr auf dem Bildschirm sichtbar ist, wird die onStop() Methode von A ausgeführt.

➔ Wenn Aktivitäten Daten austauschen, dann sollte A die Änderungen in der Methode onPause() und nicht erst in der Methode onStop() schreiben.