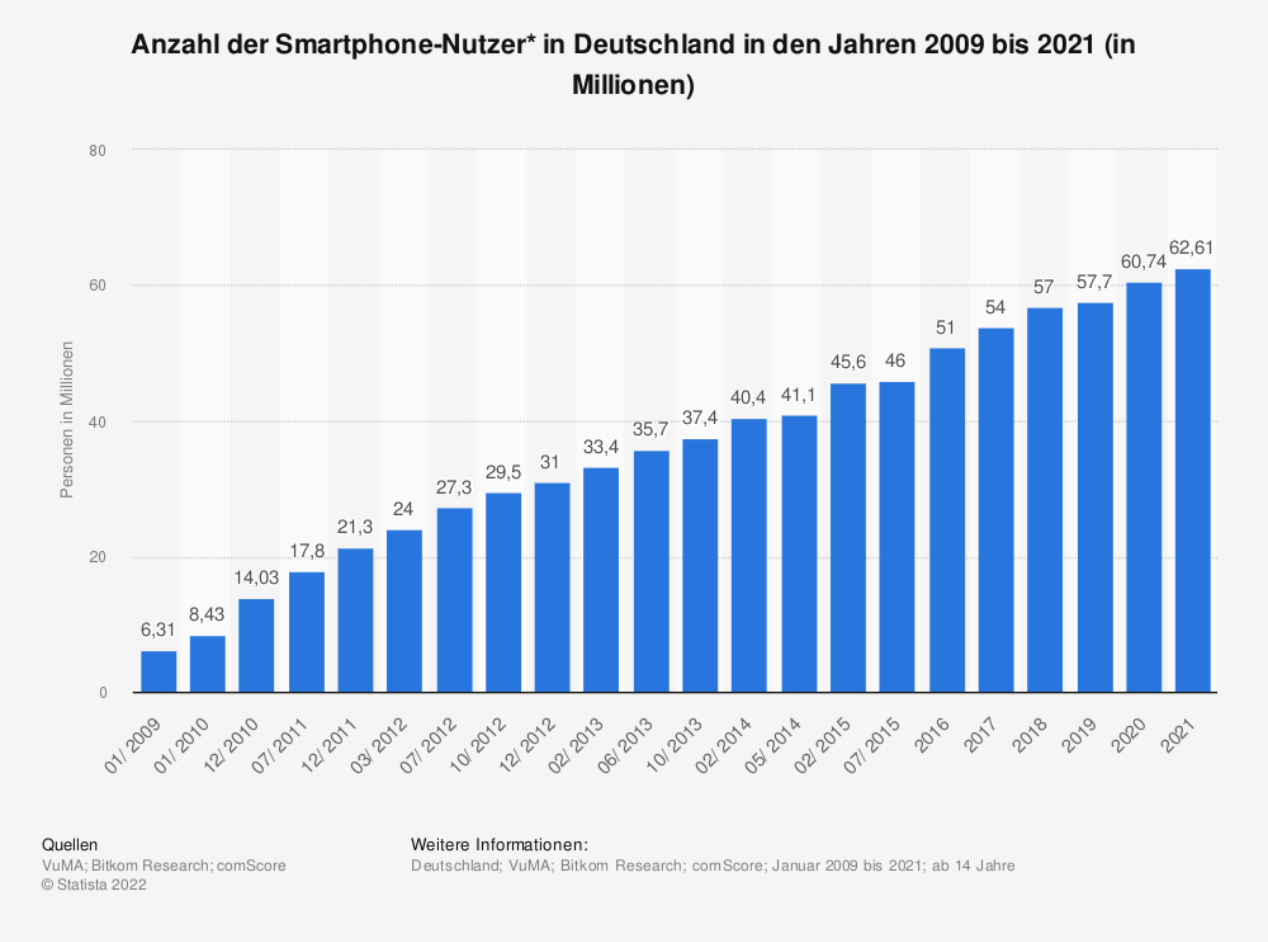
Gibt es die Welt ohne Smartphone?
Gibt es verteilte Software für Smartphone?

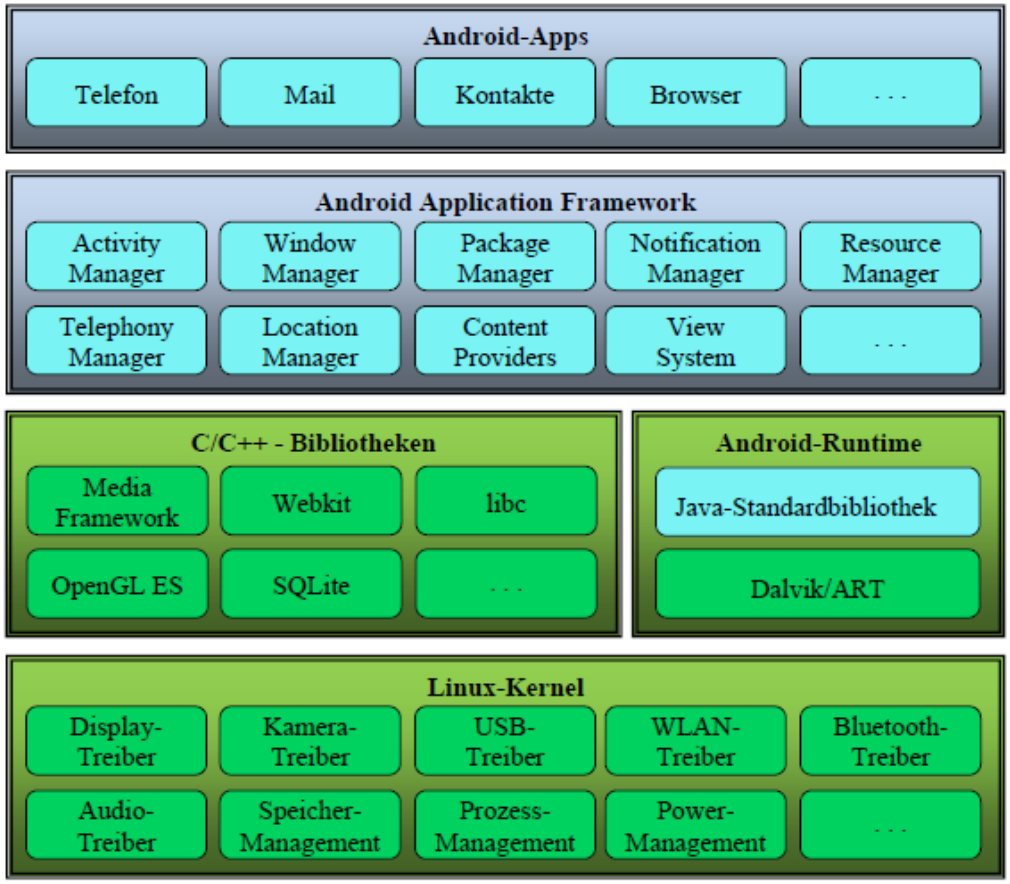


Quelle: Statista

Warum Android?

- Betriebssystem für mobile Geräte wie Smartphones, Netbooks …
- und Software-Plattform zum Entwickeln von Anwendungen
- In Java und Kotlin (seit 2017 offiziell von Google unterstützt, seit 2019 empfohlene Programmiersprache)
- Java und Kotlin sind miteinander kombinierbar

Wo ordnen wir Bytecode ein? oder Android-Architektur



Apps

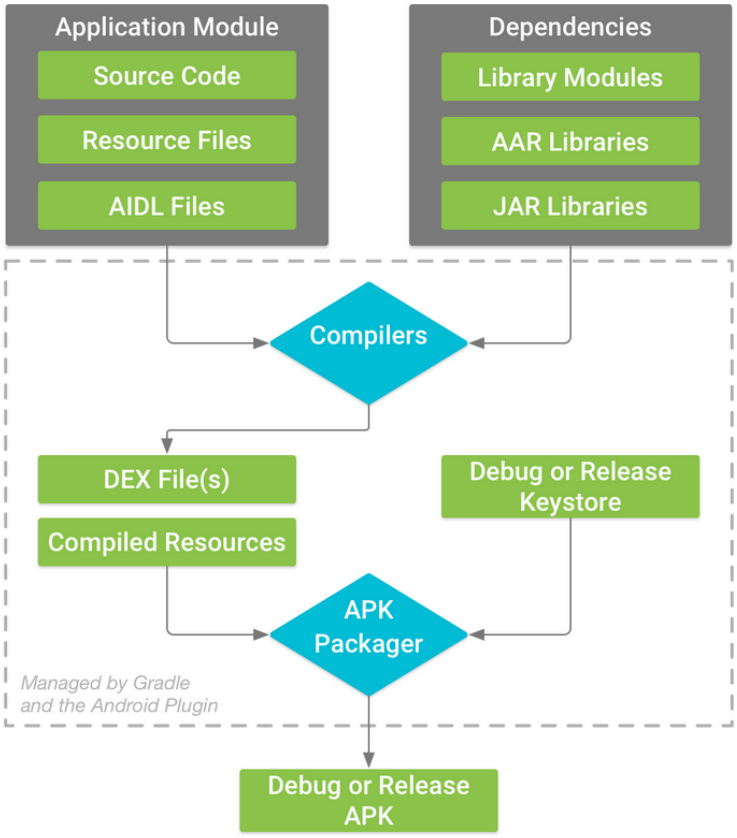
Anwendungsrahmen (Android Application Framework)

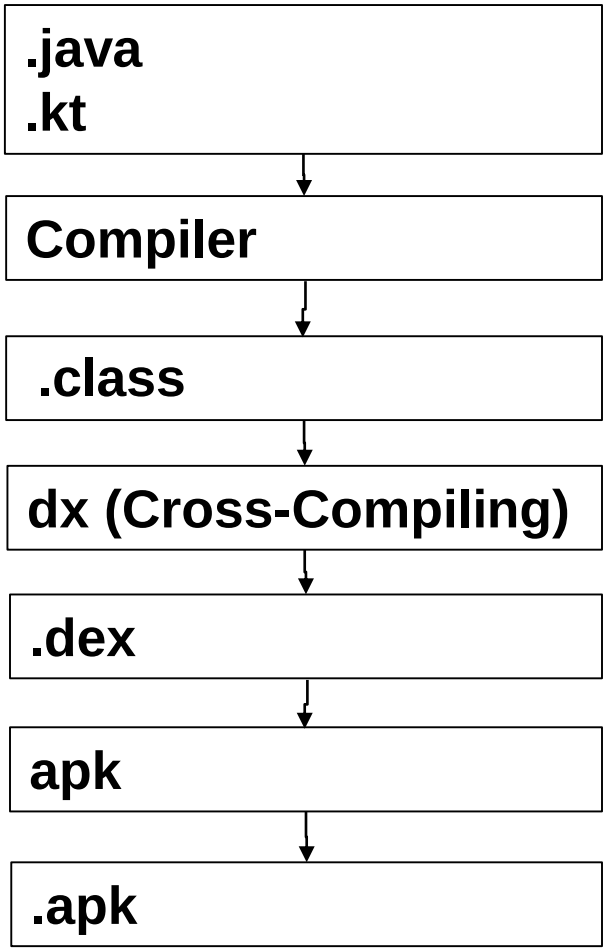
Android Laufzeitumgebung

Basis: Linus-Kernel

Quelle: Universität Trier, Bernhard Baltes-Götz, Einführung in die Entwicklung von Apps für Android 8

Warum wir Java und Kotlin kombinieren können … und Bytecode ist nur der erste Schritt…





Quelle: https://developer.android.com/studio/build

Android Studio

- freie (und offizielle!) Integrierte Entwicklungsumgebung (IDE) von Google und für die Android-Softwareentwicklung
- verfügt über *Instant Run* Funktion
- verwendet Build-Management-Automatisierungs-Tool Gradle

Was ist eine Komponente welche und wie viele gibt es in einer App?

- **Activity (Aktivität):** präsentiert eine Bildschirmseite mit Bedienelementen.

- **Service (Dienst):** führt Aufgaben im Hintergrund aus und hat keine Bedienoberfläche. Er kann von einer anderen Anwendungskomponente *gestartet oder gebunden* werden

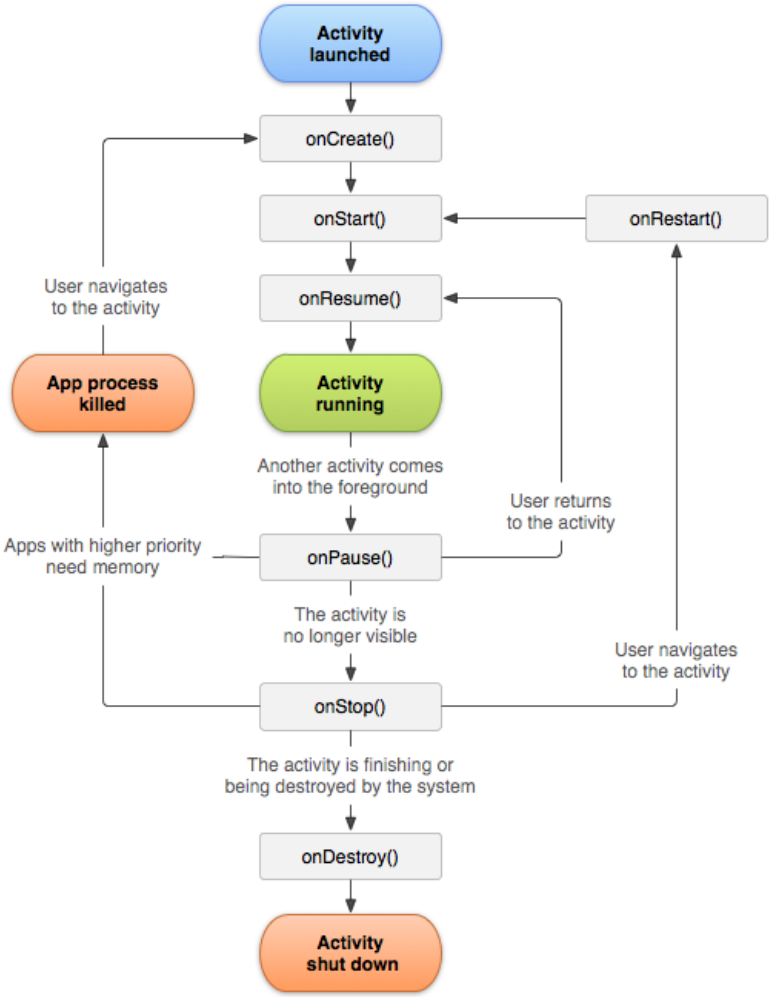
- **Broadcast Receiver (Empfänger von Nachrichten):** kann auf Nachrichten reagieren, die vom System oder von Anwendungen stammen. Er hat keine Benutzeroberfläche und ist nur kurzzeitig aktiv, kann aber Aktivitäten oder Dienste starten.

- **Content Provider (Anbieter von Daten):** verwaltet Daten, abstrahiert darunterliegende Schicht (z. B. eine Datenbank). Er kann über erteilte Berechtigungen die Daten anderen Anwendungen zur Verfügung stellen.

Aktivität

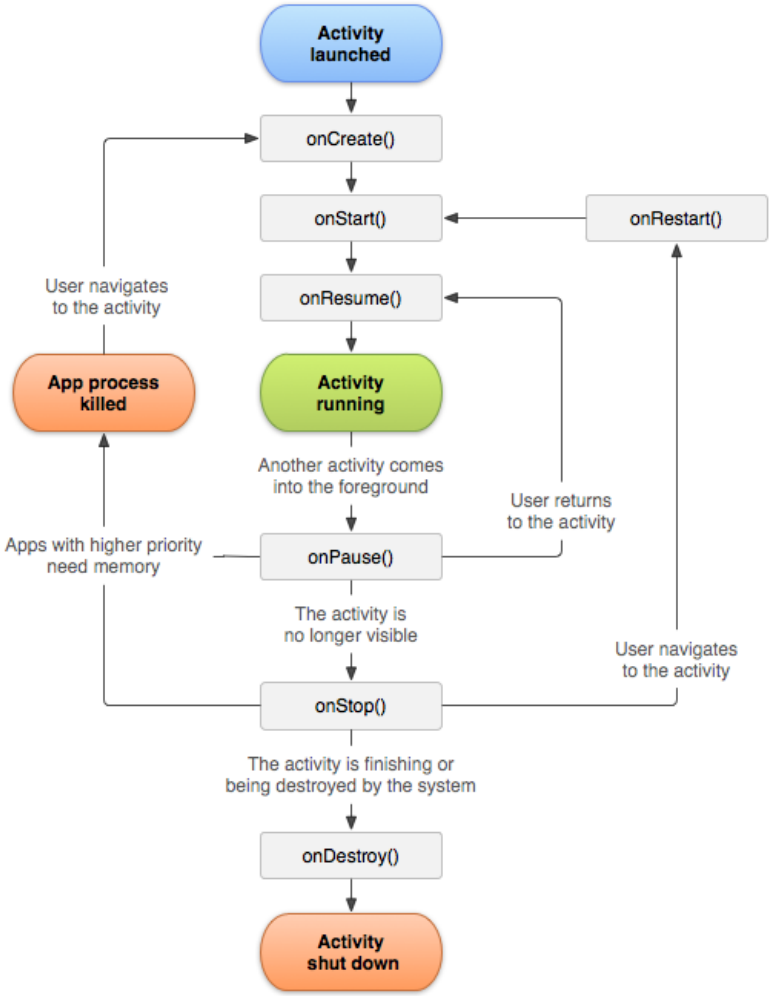
- eine Komponente der Android App
- enthält Bedienelemente (Views) und ermöglicht dadurch das Interagieren mit dem Benutzer
- präsentiert eine Bildschirmseite
- beim App-Start erscheint die Startaktivität
- Aktivität kann weitere Komponenten starten

- Eine Reihe von nacheinander gestarteten Aktivitäten bilden einen Stapel, ***Back Stack,*** nach dem **LIFO** (last in, first out) Prinzip.
- Oberste Aktivität interagiert mit dem Benutzer.
  Die überlagerte Aktivität pausiert oder wird gestoppt, der Zustand ihrer Bedienoberfläche wird gespeichert für den Fall der Rückkehr.
- In Falle der Rückkehr wird die oberste Aktivität entfernt (zerstört, ihr Zustand wird nicht gespeichert).
- Per Home-Schalter wird ein neuer Stapel begonnen und komplette Back Stack (alle Aktivitäten) wird gestoppt.
- Achtung: Alle gestoppte Aktivitäten können im Fall des Speichermangels beendet werden.
- Eine Aktivität besitzt keine main()-Methode, zur Beginn des Lebenszyklus wird onCreate() gestartet.



- Phasenübergänge werden durch Callback Methoden eingeleitet
- Diese werden von System aufgerufen in Abhängigkeit von aktuellen Zustand
- Ob und welche Methoden des Aktivitätslebenszyklus implementiert werden müssen, hängt von der Komplexität der Aktivität ab

Quelle: https://developer.android.com/guide/components/activities/activity-lifecycle

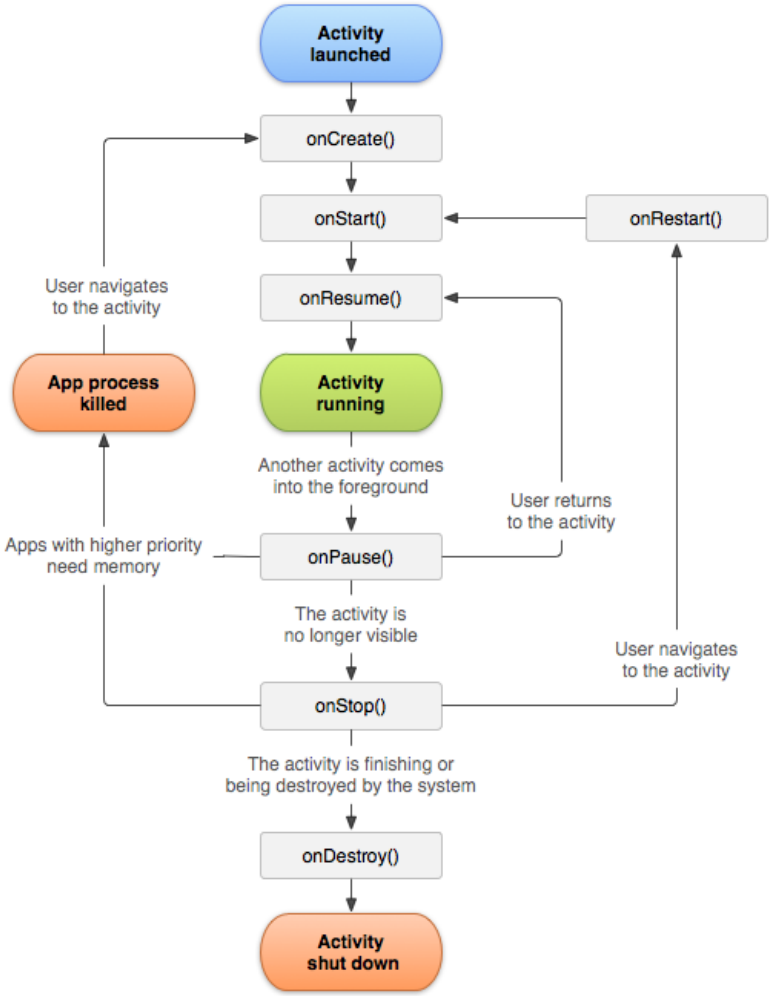


onCreate ()
Erstgestaltung der Oberfläche der Aktivität

- direkt im Quellcode
- mit Hilfe der XML-Layout-Datei (Ressourcen-Datei)

```
@Override
protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

onDestroy ()
wird implementiert, um sicherzustellen, dass alle Ressourcen einer Aktivität freigegeben werden, wenn die Aktivität endgültig zerstört wird



onStart()
enthält die letzten Vorbereitungen der Aktivität, um sichtbar zu werden

onResume()
Aktivität befindet sich oben im Aktivitätsstapel und kann mit dem Nutzer interagieren.

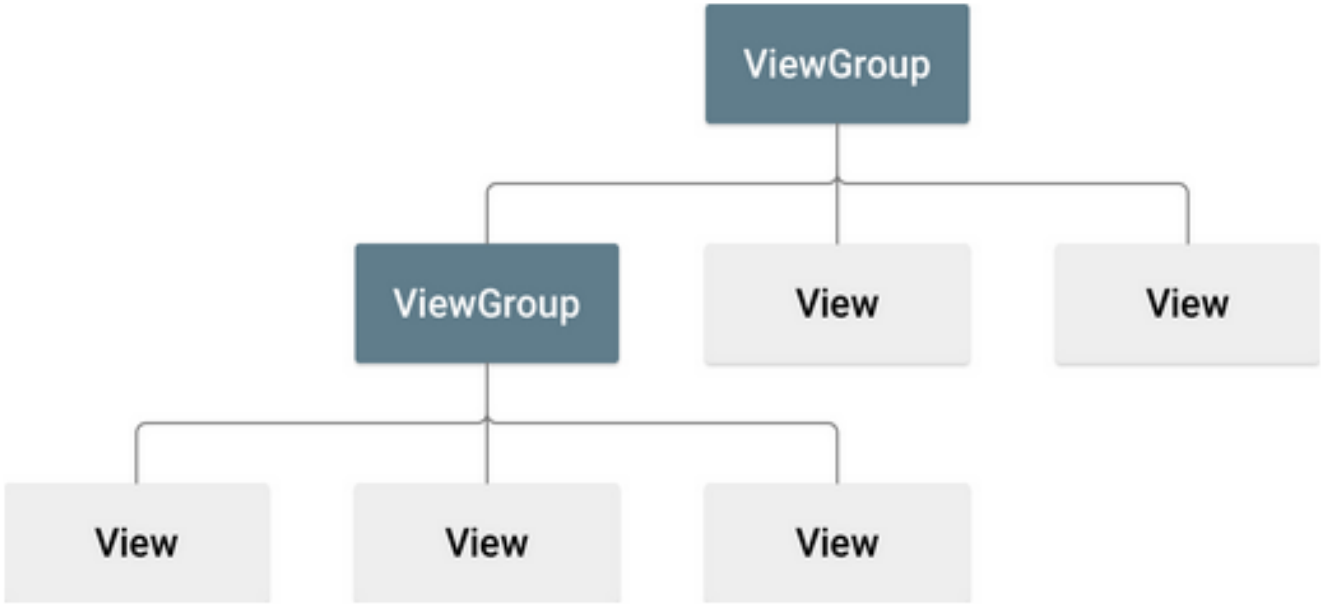
onPause()
Aktivität ist nicht mehr sichtbar (wenigstens teilweise), weil sie durch andere Aktivität verdeckt wird

onStop ()
Aktivität längerer Zeit nicht sichtbar

onRestart ()
wenn die Aktivität mit dem Status "Stoppt" neu gestartet werden soll

UI besteht aus Views und ViewGroups

- mit View kann der Nutzer interagieren,
- ViewGroup ist ein unsichtbarer Container (z.B. Layouts)



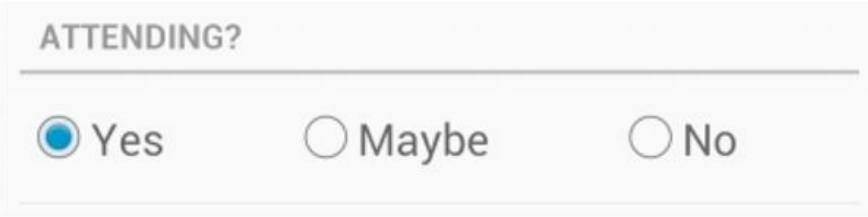
Quelle: https://developer.android.com/guide/topics/ui/declaring-layout

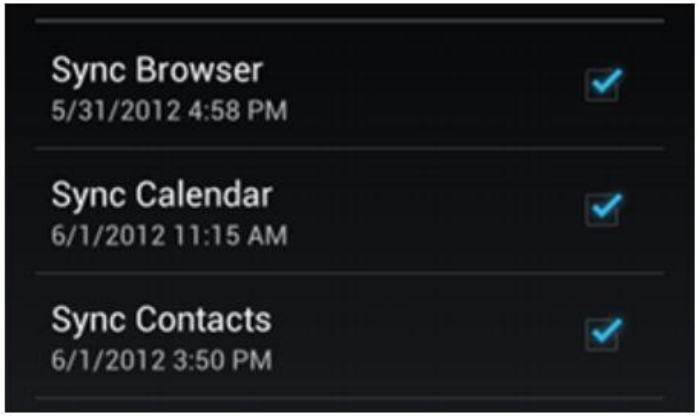
Eine Auswahl der Views:

- TextView: einfache Textausgabe
- Button: Schalter
- EditText: Texteingabe
- CheckBox: Ankreuzfeld
- RadioButton: Auswahlschalter einer RadioGroup
- ImageView: Bild

Paket:

`android.widget`





Quelle: https://developer.android.com

Definition:

- direkt im Quellcode

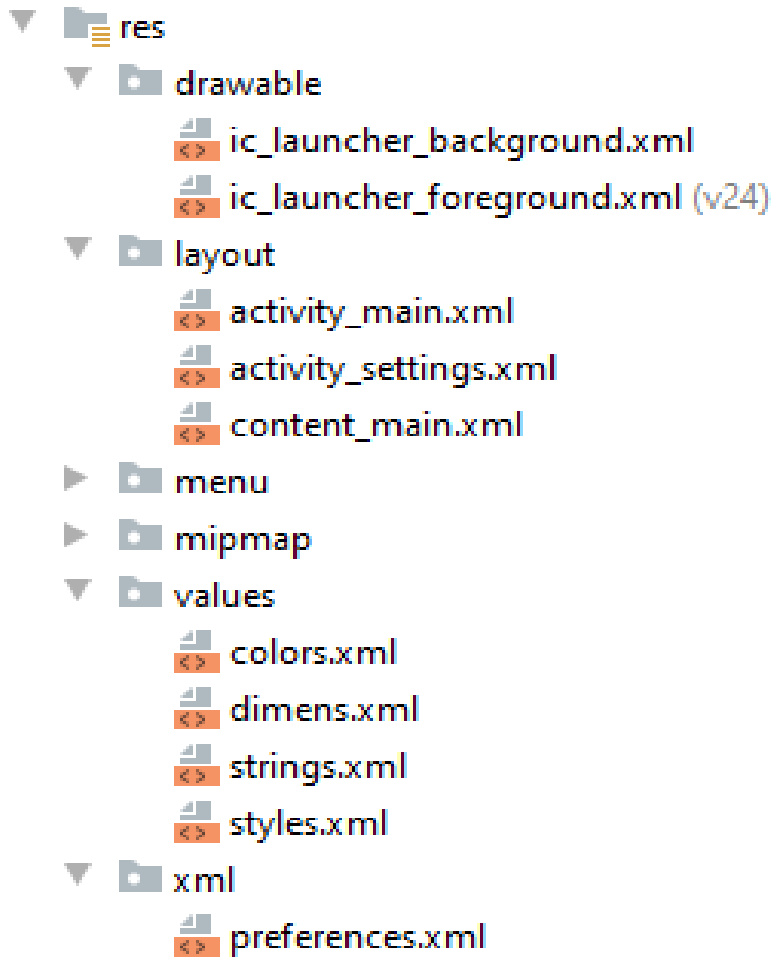
```
TextView textView = new TextView(this);
textView.setText("Hello, I am a TextView");
```

- In der Ressource

```
//kompletes Layout
setContentView(R.layout.activity_main);
//…
TextView textView=null;
//…
textView=(TextView)findViewById(R.id.textview);
textView.setText("Hello, I am a TextView");
```

Ressourcen
sind die zusätzlichen Dateien, die statischen Inhalte enthalten, z. B. Bilder (Icon), Layout-Definitionen, Beschriftungen für die Benutzeroberfläche und mehr.

- werden aus dem Quellcode auslagert und unabhängig verwaltet
- Damit sind die alternativen Ressourcen für bestimmte Gerätekonfigurationen möglich (z.B. je nach Spracheinstellung, Größe des Gerätes, …).
  Die Auswahl der Ressource geschieht zur Laufzeit basierend auf der aktuellen Konfiguration



- befinden sich in den Unterverzeichnissen des Verzeichnis res.
- üblicherweise werden als xml-Dateien definiert
- Bilder, Videos etc. liegen als Binärdateien vor

- Die Ressourcen werden in Binärformat umgewandelt und .apk-Datei hinzugefügt.
- Dabei werden sie automatisch indiziert und können dann über einen Schlüssel (Ressourcentyp plus Ressourcen-ID) angesprochen werden, z.B.

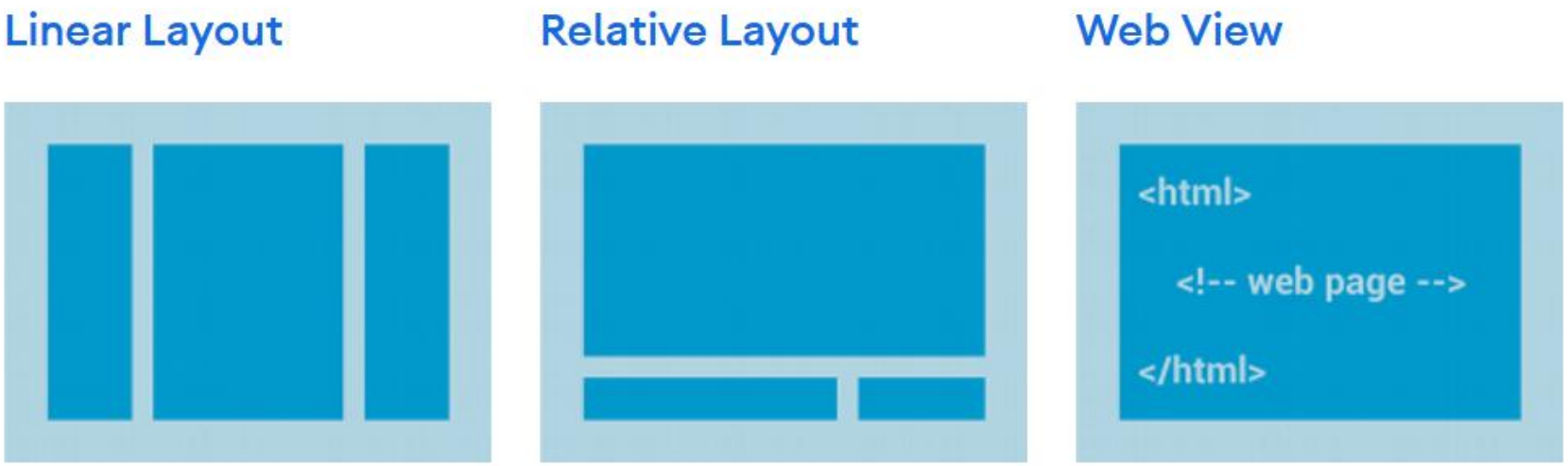
```
R.layout.activity_main
```

Layout Ressource: definiert die Benutzeroberfläche: View-Gruppen und Views

file location: res/layout/*activity_main*.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@drawable/pyramide" />
    </LinearLayout>
```



Linear Layout: organisiert untergeordneten Elemente in einer einzelnen horizontalen oder vertikalen Zeile
Relative Layout: ermöglicht Positionieren von untergeordneten Objekten relativ zueinander (untergeordnetes Objekt A links von untergeordnetem B) oder zum übergeordneten Objekt (ausgerichtet am oberen Rand des übergeordneten Objekts)
Web View: zeigt Web-Seiten

Quelle: https://developer.android.com/guide/topics/ui/declaring-layout

```
<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WebView myWebView = new WebView(this);
        setContentView(myWebView);
        myWebView.loadUrl("https://tu-freiberg.de/");

    }
}
```

Eigenschaften von Views

- Ressourcen-ID: ein eindeutiger Ressourcenname
- Höhe, Breite des Elements als Wert oder Schlüsselwort ("match_parent" oder "wrap_content")
- Abstände etc.

match_parent: übereinstimmend mit dem übergeordneten Element
wrap_content: wie für den Inhalt des Elements erforderlich

relative Einheiten: sp und dp - skalieren gemäß der Pixeldichte
dp entspricht einem Pixel auf einem 160-dpi-Bildschirm,
sp berücksichtigt zudem die systemweite Schriftgröße

absoluten Einheiten: px, pt, in, mm

Textressourcen

- werden in strings.xml verwaltet
- einzelnen Strings und String Arrays möglich

Z.B.

```
<resources>
    <string name="app_name">My beautiful App</string>
    <string-array name="fruits">
        <item>apple</item>
        <item>orange</item>
        <item>other fruit</item>
    </string-array>
</resources>
```

```
String[] array=getResources().getStringArray(R.array.fruits);
String str=getResources().getString(R.string.app_name);
```

Eventhandling

- Reaktion auf Benutzereingaben mit Hilfe der Ereignisbehandlungsmethoden
- Event: Eintreten einer Eingabe, auf welche die App reagieren muss (z.B. Click, LongClick, Touch …)
- Eventquelle: Objekt in dem ein Event entsteht (z.B. Button)
- Eventhandler (Listener): erwartet das Auftreten eines Events, implementiert eine dem Event zugeordnete Methode (z.B. Activity)

Eventhandling

- Klasse View verfügt über eine Reihe von Interfaces, den sogenannten Event-Listener
- Ein Event-Listener enthält jeweils eine einzige Methode, z.B. `onClick (View.OnClickListener)`

```
public class MainActivity extends AppCompatActivity
                                        implements View.OnClickListener  {
//…
    @Override
    public void onClick(View v) {
        //v ist ein View, das geklickt wurde
    }
}
```

- Event-Listener (Handler) werden bei Views (Eventquellen) mit Hilfe der Methoden `view.setOn…Listener()` registriert (eine Möglichkeit, andere Möglichkeit: Registrierung über XML-Attribut)

z.B.

```
Button button=(Button)findViewById(R.id.button);
button.setOnClickListener(this);
```

weitere Methoden:

```
onClick (View.OnClickListener)
onLongClick (View.OnLongClickListener)
onFocusChange (View.OnFocusChangeListener)
onTouch (View.OnTouchListener)
onCreateContextMenu (View.OnCreateContextMenuListener)
```

Die Behandlungsmethoden werden implementiert:

- direkt in der Aktivität (Beispiel 1),
- im anonymen Objekt (Beispiel 2),
- als Lambda-Ausdruck (Beispiel 3).

- direkt in der Aktivität

Beispiel 1

```
public class MainActivity extends AppCompatActivity
                                    implements View.OnClickListener  {
    TextView textview=null;
    Button button=null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textview=(TextView)findViewById(R.id.textview);
        button=(Button)findViewById(R.id.button);
        button.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        textview.setText("Hello, I am a refreshed TextView");
    }
}
```

- im anonymen Objekt

Beispiel 2

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    textview=(TextView)findViewById(R.id.textview);
    button=(Button)findViewById(R.id.button);

    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            textview.setText("Hello, I am a refreshed TextView");
        }
    });
}
```

- als Lambda-Ausdruck

```
Beispiel 3
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    textview=(TextView)findViewById(R.id.textview);
    textview.setText("Hello, I am a TextView");

    button=(Button)findViewById(R.id.button);
    button.setOnClickListener(view->{
                textview.setText("Hello, I am a refreshed TextView");
        });
}
```

Registrierung per XML-Attribut:

```
<Button android:id="@+id/button"
        android:onClick="onButtonClick"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
```

Im Quellcode der Activity :

```
public void onButtonClick(View v) {
    textview.setText("Hello, I am a refreshed TextView");
}
```

- Ereignisbehandlungsmethoden laufen nacheinander im selben Thread (UI-Thread).
- Die Ereignisse werden in eine Warteschlange nach dem FIFO-Prinzip (First In, First Out) eingetragen
- Zeitaufwändige Aufgaben müssen in einen separaten Thread verlagert werden.
- Auf Views kann *nur* vom UI-Thread aus zugegriffen werden, d.h. alle Änderungen der Bedienoberfläche über Ereignisse im UI-Thread abgewickelt werden müssen und Informationstransfer erforderlich ist.

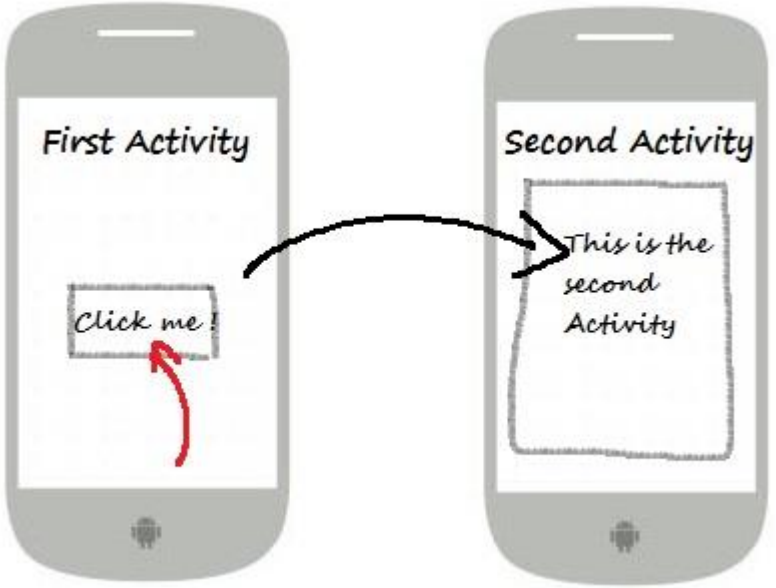
Intent

- Intension/Vorhaben, asynchrone Nachricht
- ermöglicht den Komponenten einer (und verschiedener) Anwendungen untereinander und mit Android-Plattform zu interagieren.

Explizite Intents

- die Empfängerkomponente wird explizit beim Erstellen von Intent benannt

Beispiele:
Starten von bestimmten Activities, Services, Broadcasts



Quelle: http://www.w3big.com/de/android/android-intents-filters.html

Implizite Intents

- Adressieren keine bestimmten Komponenten,
- sondern geben eine Aktion an, die von einer anderen Komponente ausgeführt werden soll.
- Die Empfängerkomponenten entscheiden selbst, ob und welche Intents sie empfangen
- Die Berechtigungen dazu werden über die Intent-Filter im Manifest festgelegt:
  bei einer passenden Komponente empfängt sie das *Intent*-Objekt,
  bei mehreren zeigt das *Android System* einen Auswahldialog.

Beispiele:

Anruf, Map Location, Öffnen einer Webseite

Expliziten Intents:

```
Intent intent = new Intent(this,Main2Activity.class);
```

Parameter:

- Objekt einer von Klasse Context (android.content.Context) abgeleiteten Klasse
- die Zielkomponente,
  die auch mit Hilfe der Methoden der Klasse Intent festgelegt werden kann:

```
setComponent(ComponentName)
setClass(Context, Class)
```

Beispiel:

```
Intent intent = new Intent();
intent.setComponent(new
ComponentName("com.example.myapplication", //package
                "com.example.myapplication.Main2Activity"));
startActivity(intent)
```

Klasse Context

- eine abstrakte Klasse, abgeleitet von Object,
  die Implementierung wird vom Android-System bereitgestellt
- ermöglicht den Zugriff auf anwendungsspezifische Klassen, z.B.

  ermöglicht z. B. das Starten von Aktivitäten, Senden und Empfangen von Intents

```
Intent intent =
     new Intent(getApplicationContext(),Main2Activity.class);
```

- repräsentiert die Schnittstelle zu Informationen der App-Umgebung, z.B.

  erlaubt über die Methode `getSystemService` den Zugang zu Manager-Diensten auf der Systemebene

```
getSystemService(Context.ALARM_SERVICE)
```

liefert einen AlarmManager

```
getSystemService(Context.NOTIFICATION_SERVICE)
```

liefert einen NotificationManager

Übermitteln der (Extra) Daten

Methoden
- putExtra und
- getXXXExtra (getXXXArrayExtra und weitere)

XXX : Int, Long, String etc.

putExtra – Parameter: Schlüssel und Wert,
get-Methoden – Parameter: Schlüssel, Rückgabe: der Wert

Schlüssel muss dem Sender und Empfänger bekannt sein

Beispiel

Sender-Komponente:

```
private final String INT_VALUE="ein Integer Wert";//key definition
private final String STRING_VALUE="ein String";//another key definition

Intent intent = new Intent(this,Main2Activity.class);
intent.putExtra(INT_VALUE,1000);
intent.putExtra(STRING_VALUE,"greetings");
startActivity(intent);
```

Empfänger-Komponente:

```
private final String INT_VALUE="ein Integer Wert";//key definition
private final String STRING_VALUE="ein String";//another key definition

Intent intent=getIntent();
int intWert=intent.getIntExtra(INT_VALUE,0);//default 0
String stringWert=intent.getStringExtra(STRING_VALUE);
Toast.makeText(this,intWert+ " "+stringWert,Toast.LENGTH_SHORT).show();
```

Implizite Intents

Beim Erstellen eines impliziten Intent werden angegeben:

- Aktion (ACTION_VIEW, ACTION_EDIT, ACTION_DIAL etc.) und
- passendes Argument im „URI“ – Format

Standard-Action Angebot variiert je nach Komponentenart
Welche Apps (welche Komponenten) auf Intent reagieren wird im Manifest der jeweiligen Apps festgelegt

```
Intent impIntent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("https://www.tu-freiberg.de/"));
startActivity(impIntent);

Intent impIntent = new Intent(Intent.ACTION_DIAL,
                Uri.parse("tel:123-45678"));
startActivity(impIntent);
```

Weiteres unter: https://developer.android.com/reference/android/content/Intent

Manifest

Im Abschnitt <application> werden alle **Komponenten** mit Eigenschaften deklariert

Eigenschaften:

- Name,
- Beschriftung,
- Symbol,
- UI-Thema etc.

```
<application …>
        <activity
            android:name=".SettingsActivity"
            android:label="@string/title_activity_settings">
        </activity>
        <activity android:name=".MainActivity">
            …
        </activity>
 </application>
```

Deklariert werden

- jede Activity-Klasse mit <activity>,
- jede Service-Klasse mit <service>,
- jeder BroadcastReceiver mit <receiver> und
- jeder ContentProvider mit <provider>.

Intent-Filter festlegen mit

- <action>-Element und
- optional <category>- und <data> -Element

MAIN: Komponente gilt als der Einstiegspunkt der App

Kategorie:
LAUNCHER: Komponente wird im System-Launcher aufgelistet

```
<application …>
  <activity android:name=".MainActivity">
     <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name=
                        "android.intent.category.LAUNCHER" />
     </intent-filter>
   </activity>
</application>
```

Berechtigungen deklarieren

mit <uses-permission> werden Systemberechtigungen angegeben, die der Benutzer erteilen muss

```
<manifest ... >
    <uses-permission
       android:name="android.permission.SEND_SMS"/>
    <uses-permission
       android:name="android.permission.INTERNET"/>

    ...
</manifest>
```

Beispiel

```
<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout_editor_absoluteX="1dp">

    <EditText
        android:id="@+id/textedit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Greetings!" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Los!"
        android:onClick="onButtonClick"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    </LinearLayout>
```

```
public class MainActivity extends AppCompatActivity {
 public class TCPKomm implements Callable<String>{
     //auf der nächsten Folie
}
    EditText textedit=null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textedit=findViewById(R.id.textedit);
    }

    public void onButtonClick(View view) {
        String message = textedit.getText().toString();
        ExecutorService exec = Executors.newSingleThreadExecutor();
        Future<String> future = exec.submit(new TCPKomm(message));
        try{
            String s=future.get();
            textedit.setText(s);
        }
        catch (InterruptedException | ExecutionException ex ) {
            ex.printStackTrace();
        }
        exec.shutdown();
    }
}
```

```
public class TCPKomm implements Callable<String>{

    String text=null;
    public TCPKomm(String message){
        text=message;
    }
    @Override
    public String call() throws Exception {
        Socket socket;
        try{
            socket=new Socket("10.0.2.2",5556);//("139.20.16.XXX",5556);
            PrintWriter pw= new PrintWriter(socket.getOutputStream(),true);
            pw.println(text);
            pw.flush();

            Scanner sca = new Scanner(socket.getInputStream());
            String s = sca.nextLine();
            pw.close();
            sca.close();
            socket.close();
            return s;
        } catch(Exception e){
            System.out.println(e);
            return null;
        }
    }
```

10.0.2.2 - host loopback interface (127.0.0.1)

https://developer.android.com/studio/run/emulator-networking

Hintergrundoperationen

- Läuft im Hintergrund heißt das für den Benutzer sind in der Zeit keine Aktivitäten der App sichtbar
- jede Aufgabe, die länger als ein paar Millisekunden dauert soll an einen Hintergrundthread delegiert werden

Kategorien der Ausführung im Hintergrund:

- Immediate (sofort) ->Threading, Work Manager, Kotlin-Coroutines
- Exact (genau) -> Alarm Manager (+Broadcast Receiver bzw. Service)
- Expedited (beschleunigt) ->Work Manager (setExpedited())
- Deferred (verzögert) ->Work Manager

Lösungen

- Threads (Runnable)
  Informationstransfer über:
    View: post(Runnable action)
    Activity: runOnUiThread(Runnable action)
  s. Beispiel
- ThreadPoolExecutor und Callable-Future-Task
- Kotlin-Coroutines
- Broadcast Receiver
- Service-Komponente
- (Wiederholte) geplante Ausführung mit dem Alarm Manager
- (Wiederholte) geplante Ausführung mit dem Work Manager

Beispiel

```
public class MainActivity extends AppCompatActivity {
    TextView mTextView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = (TextView) findViewById(R.id.mytextview);

        new Thread(new Runnable() {
            @Override
            public void run() {
                Random r=new Random();
                while (r.nextInt(6)+1!=6) {
                    try { sleep(1000);
                    } catch (InterruptedException e) { e.printStackTrace();
                    }
                }
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        mTextView.setText("6!");
                    }
                });
            }
        }).start();
    }
}
```

Coroutine

- ist ein in Kotlin (ab Version 1.3) realisierter Entwurfsmuster
- wird verwendet, um lang andauernder Aufgaben, die andernfalls den Hauptthread blockieren würden, asynchron auszuführen
- Die asynchron auszuführenden Funktionen werden als **suspend** gekennzeichnet und in einem „Ausführungsbereich“ (**CoroutineScope**) ausgeführt
- An den „Ausführungsbereich“ wird ein Dispatcher übergeben
- Verteilung erfolgt über **Dispatcher**, die im Sinne der Nebenläufigkeit wie Thread funktionieren, sind jedoch nicht an einen bestimmten Thread gebunden. Die Ausführung kann in einem Thread unterbrochen und in einem anderen fortsetzen werden.
- Implementierungen von CoroutineScope werden mit dem Coroutine-Builder (Funktionen launch, async usw.) angelegt

CoroutineScope (Interface):
- definiert einen Ausführungbereich für neue Coroutinen.

CoroutineDispatcher (abstrakte Basisklasse):
- wird an Scope übergeben,
- ist eine Implementierung aus der Klasse CoroutineDispatchers oder ein Thread bzw. mit newSingleThreadContext und newFixedThreadPoolContext erstellter Threadpool.
  Ein beliebiger java.util.concurrent.Executor kann als Dispatcher verwendet werden.

Dispatchers: Implementierungen von Dispatcher

Dispatchers.IO - für die Auslagerung von E/A-intensiven Blockierungsvorgängen
Dispatchers.Main – spricht den Hauptthread an
Dispatchers.Unconfined – starten Ausführung bis zur ersten Unterbrechung im Aufrufer-Thread, danach im für die Coroutine vorgesehen Thread
Dispatchers.Default – gemeinsam genutzter Threadpool, wird standartmäßig verwendet, wenn kein anderer explizit angegeben wird

launch, async: Funktionen zum Starten von Coroutine (Coroutine-Builder).

- starten eine neue Coroutine, die gleichzeitig mit dem Rest des Codes arbeitet.

launch: blockiert nicht den main-Thread, der Rest des Codes wartet auf das Ergebnis von Ausführung nicht
async: blockiert den Haupt-Thread am Punkt des Aufrufs der await()-Funktion

```
launch { // inherits context and thus dispatcher
    println("main runBlocking: I'm working in thread ${Thread.currentThread().name}")
}
launch(Dispatchers.Main) { // dispatched to DefaultDispatcher
    println("Main: I'm working in thread ${Thread.currentThread().name}")
}
launch(newSingleThreadContext("MyOwnThread")) { // dispatched to own new thread
    println("newSingleThreadContext: I'm working in thread
                                                    ${Thread.currentThread().name}")
}
```

```
val resultOne = async(Dispatchers.IO) { function1() }
val resultTwo = async(Dispatchers.IO) { function2() }
val resultText = resultOne.await() + resultTwo.await()
```

Beispiel

```
import kotlinx.coroutines.*
class MainActivity : AppCompatActivity() {
    private lateinit var button: Button
    private lateinit var editText: EditText

    private val scope = CoroutineScope(Dispatchers.Default)

    override fun onCreate(savedInstanceState: Bundle?) {  //…
    }

    fun onButtonClick(view: android.view.View) {
        val s=editText.text.toString()
        scope.launch { startKomm(s)
                     }
    }
    private suspend fun startKomm(message:String) {
        val socket: Socket
        try {
            socket = Socket("10.0.2.2", 5556)
            val pw = PrintWriter(socket.getOutputStream(), true)
            pw.println(message)
            pw.flush()
            val sca = Scanner(socket.getInputStream())
            val s:String?= sca.nextLine()?:""
            pw.close()
            sca.close()
            socket.close()
            withContext((Dispatchers.Main)) {  s.also { editText.setText(it) }
            }
        } catch (e: Exception) {       }
    }
```

also: the context object is available as an argument „it“.
The return value is the object itself.

Abhändigkeiten, einzutragen in build.gradle (Module)

```
buildscript {
        ext.kotlin_version='1.4.0'
    }
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.9")
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9")
```

Quellen:

https://kotlinlang.org/docs/coroutines-basics.html#your-first-coroutine
https://kotlinlang.org/docs/coroutine-context-and-dispatchers.html#dispatchers-and-threads
https://www.geeksforgeeks.org/launch-vs-async-in-kotlin-coroutines/

Broadcast Receiver

- ist eine Android-Componente
- Aufgabe: die Nachrichten vom System oder von Anwendungen (von Komponenten der selben App) zu empfangen
  - Android-System sendet Broadcasts, wenn Systemereignisse auftreten.
  - Apps senden Informationen, die andere Apps interessieren könnten.
- besitzt keine Benutzer Oberfläche
- sind in der Regel nur kurz aktiv, können aber Aktivitäten oder Dienste (Services) starten
- Broadcasts werden vom System automatisch an die Apps weitergeleitet, die sich für den Empfang dieser bestimmten Art von Broadcasts angemeldet haben.

"Publish-Subscribe" - Entwurfsmuster

- Absender von Nachrichten (Publisher) senden NICHT direkt an bestimmte Empfänger (Subscriber)
- Absender veröffentlicht Nachrichten ohne zu wissen, welche Abonnenten vorhanden sind
- Empfänger melden Interesse nur für bestimmte Nachrichten ohne zu wissen wer sie sendet

Broadcast-Nachrichten

- sind Intent-Objekte,
- geben die Action und damit die Art des Ereignisses bekannt
- können zusätzliche Informationen enthalten

Datei broadcast_actions.txt

android.accounts.LOGIN_ACCOUNTS_CHANGED
android.accounts.action.ACCOUNT_REMOVED
android.app.action.ACTION_PASSWORD_CHANGED
…
android.intent.action.BATTERY_LOW
android.intent.action.BATTERY_OKAY
android.intent.action.BOOT_COMPLETED
…
android.intent.action.DATA_SMS_RECEIVED
android.intent.action.DATE_CHANGED
…
android.intent.action.DOWNLOAD_COMPLETE
android.intent.action.DOWNLOAD_NOTIFICATION_CLICKED
…
android.telephony.action.SIM_APPLICATION_STATE_CHANGED
android.telephony.action.SIM_CARD_STATE_CHANGED
android.telephony.action.SIM_SLOT_STATUS_CHANGED

C:\Users\xxx\AppData\Local\Android\Sdk\platforms\android-29\data

Broadcast Receiver Klasse

- wird von der abstrakten Klasse `android.content.BroadcastReceiver` abgeleitet
- muss die `onReceive(Context,Intent)`-Methode implementieren

```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver is receiving
        // an Intent broadcast.
        Log.i("MyReceiver", "onReceive");
    }
}
```

- **Manifest**

```
<receiver android:name=".MyBroadcastReceiver"
          android:exported="true">
   <intent-filter>
      <action android:name=
             "android.intent.action.BOOT_COMPLETED"/>
     <action android:name=
             "android.intent.action.INPUT_METHOD_CHANGED"/>
    </intent-filter>
</receiver>
```

android:exported: if true, the broadcast receiver can receive messages from sources outside its application

Im Context registrierte Receiver:

- empfangen Broadcasts, solange ihr Registrierungskontext (z.B. Aktivität) gültig ist.
- Zum Registrieren des Receivers ist ein Objekt der Receiver-Klasse und der IntentFilter zu erstellen.
- Das Registrieren erfolgt mit der Methode `registerReceiver`
- Das Beenden mit `unregisterReceiver`

```
BroadcastReceiver receiver= new MyBroadcastReceiver();
IntentFilter filter = new IntentFilter
        (ConnectivityManager.CONNECTIVITY_ACTION);
filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);

registerReceiver(receiver, filter);

unregisterReceiver(receiver);
```

Methoden der Context Klasse:

```
registerReceiver(BroadcastReceiver receiver,
   IntentFilter filter, String broadcastPermission,
   Handler scheduler)
```

zum Registrieren von Broadcast-Intents, Ausführungskontext Thread *schedule oder null*

```
unregisterReceiver(BroadcastReceiver receiver)
```

zum Entfernen einer Registrierung

- Das Anmelden und Abmelden des Empfangs soll konform gestaltet werden
  - in onCreate() und onDestroy()
  - in onResume() und onPause ()

- Vermeiden Sie den unnötigen Broadcast-Empfang, um den Systemaufwand zu verringern.

Empfangsberechtigungen

- Berechtigen zum Empfangen von bestimmten Broadcasts
- Werden definiert im Manifest für die gesamte App (1) oder für einzelne Receiver (2)

Beispiel 1:

```
<uses-permission
           android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<application … > … </application>
```

Beispiel 2:

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

Empfangsberechtigungen

für die einzelnen Receiver-Komponenten im Quellcode möglich (3):

Beispiel 3:

```
MyBroadcastReceiver receiver = new MyBroadcastReceiver();
IntentFilter filter = new IntentFilter
        (ConnectivityManager.CONNECTIVITY_ACTION);
filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);

registerReceiver(receiver, filter,
       Manifest.permission…., null );
```

Broadcasts versenden (Methode der Klasse Context)

- sendBroadcast(Intent) – sendet Nachricht an alle Receiver in einer undefinierten Reihenfolge (Normal Broadcast)
- sendOrderedBroadcast(Intent, String) – reicht die Nachricht von einem Empfänger zum anderen in einer festgelegten Reihenfolge weiter oder veranlasst den kompletten Abbruch.
- und weitere

```
Intent intent= new Intent();
intent.setAction("com.example.broadcast.MY_NOTIFICATION");
intent.putExtra("data", "was auch immer");
sendBroadcast(intent);
```

- Berechtigungen beim Senden festlegen

```
sendBroadcast(new Intent("com.example.NOTIFY"),
                        Manifest.permission.SEND_SMS);
//erfoldert beim Empfänger
<uses-permission android:name="android.permission.SEND_SMS"/>
```

- Verwenden Sie bevorzugt die Kontextregistrierung gegenüber der Manifest-Deklaration.
- Senden Sie vertrauliche Informationen nicht mit impliziten Intents, denn die Informationen können von jeder App gelesen werden, die sich für den Empfang registriert.
- Beachten Sie, dass auch böswillige Sendungen an den Receiver Ihrer App verschickt werden können.
- Beachten Sie wegen Konflikte mit anderen Apps, dass die Broadcast-Aktionen globale Bezeichnungen haben.
- Vermeiden Sie die „normalen“ Hintergrund-Threads von einem Broadcast-Empfänger aus zu starten. Nach Beendung von onReceive() können sie aus Speicherplatzgründen vom System abgebrochen werden.

Beispiel 1 Receiver

```
public class MainActivity extends AppCompatActivity {
    public final static  String MY_ACTION_BROADCAST_ID="My Action Broadcast Id";
    public final static  String DATA_KEY="data key";

    TextView textView=null;
    class MyReceiver extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
            String data=intent.getStringExtra(DATA_KEY);
            textView.setText(data);
        }
    }

    MyReceiver receiver=new MyReceiver();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView=(TextView)findViewById(R.id.empfangen);
        IntentFilter filter= new IntentFilter(MY_ACTION_BROADCAST_ID);
        getApplicationContext().registerReceiver(receiver, filter);
    }

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

Beispiel 1
Sender

```
public class MainActivity extends AppCompatActivity {
    public final static  String MY_ACTION_BROADCAST_ID="My Action Broadcast Id";
    public final static  String DATA_KEY="data key";

    EditText editText=null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editText=(EditText)findViewById(R.id.edittext);
    }

    public void onClickSend(View view) {
        Intent broadcastI = new Intent(MY_ACTION_BROADCAST_ID);
        String zusenden=editText.getText().toString();
        broadcastI.putExtra(DATA_KEY,zusenden);
        this.sendBroadcast(broadcastI);
    }
}
```

Beispiel 2

```
public class ChargerConnectedReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (Intent.ACTION_POWER_CONNECTED.equals(action)) {
            context.startService(new Intent(MyService.ACTION_POWER_CONNECTED));
        }
        else if (Intent.ACTION_POWER_DISCONNECTED.equals(action)) {
            context.startService(new Intent(MyService.ACTION_POWER_DISCONNECTED));
        }
    }
}
```

```
<receiver android_name=".ChargerConnectedReceiver">
    <intent-filter>
        <action android_name="android.intent.action.ACTION_POWER_CONNECTED"/>
        <action android_name="android.intent.action.ACTION_POWER_DISCONNECTED"/>
    </intent-filter>
</receiver>
```

Quelle: https://camposha.info/android-examples/android-broadcastreceiver/

Beispiel 2

```
public class MyActivity extends AppCompatActivity {
    private ChargerConnectedReceiver myChargerConnectedReceiver;

    @Override
    protected void onResume() {
        super.onResume();
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(Intent.ACTION_POWER_CONNECTED);
        intentFilter.addAction(Intent.ACTION_POWER_DISCONNECTED);
        myChargerConnectedReceiver = new ChargerConnectedReceiver();
        registerReceiver(myChargerConnectedReceiver, intentFilter);
    }
    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(myChargerConnectedReceiver);
```

Service

- eine Anwendungskomponente
- besitzt keine Benutzeroberfläche
- kann im Hintergrund langlaufende Vorgänge ausführen

- kann aus einer anderen Komponente oder von System gestartet werden und läuft im gleichen Prozess oder in einem eigenen Prozess

- Start erfolgt mit einem Intent (bevorzugt explizitem Intent)

- kann mit Alarm Manager geweckt werden

Aufgaben:

- langlaufende Vorgänge im Hintergrund ausführen zu lassen und ihre Funktionen anderen Anwendungen zugänglich zu machen.
  Andere Apps (Komponenten) können eine Verbindung zum Service herstellen, um mit ihm zu interagieren.

Einordnung:

- Service ist kein Prozess
- Service ist kein Thread
- Service benötigt ggf. für arbeitsintensive Aufgabe weitere Threads etc.

Methoden der Basisklasse **Service**, sind ggf. zu überschreiben:

onStartCommand()

- wird aufgerufen, wenn Service mit **startService()** startet wird
- der **gestartete** Service läuft solange bis es selbst seine Arbeit beendet oder durch andere Komponente gestoppt wird

onBind()

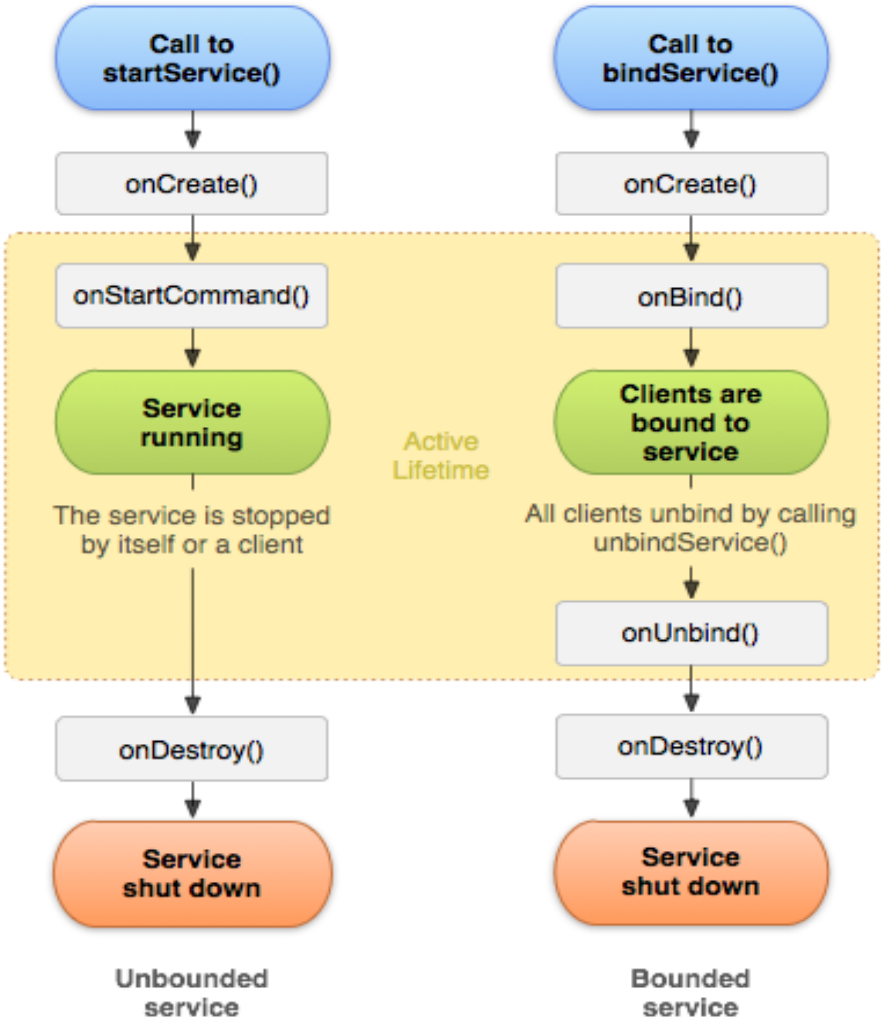
- wird aufgerufen, wenn eine andere Komponente eine Verbindung zum Dienst über **bindService()** herstellt
- der **gebundene** Service stellt eine Schnittstelle bereit, über die Clients mit dem Service kommunizieren
- ist auf jeden Fall zu implementieren
- das Beenden der Verbindung wird clientseitig veranlasst

onCreate()

wird ausgeführt, wenn der Service erstellt wird

onDestroy()

wird aufgerufen auf bevor der Service zerstört wird



Unbounded Services

- bekommt ein **Intent** übergeben,
- wird mit **stopSelf()** oder **stopService()** beendet
- wird benutzt um eine Aufgabe langfristig und wiederholt ohne Kommunikation durchzuführen.

Bounded Service

- die **onBind()**-Methode liefert eine Implementierung von IBinder
- die Verbindung wird gelöst mit **unbindService()** und zerstört, wenn keine Verbindungen mehr existieren.
- wird benutzt um die Aufgabe im Hintergrund unter bestehenden Verbindung durchzuführen.

Quelle: https://developer.android.com/guide/components/services

- Ein Service kann durch Android - System beendet werden und muss evtl. wieder gestartet werden:

  onStartCommand()-Methode liefert einen Integer-Wert zurück

  START_STICKY: onStartCommand() soll noch einmal mit einem null-Intent aufzurufen werden.
  START_NOT_STICKY: Service ist nur dann neu zu starten, wenn das ausstehende Intent noch zu liefern ist
  START_REDELIVER_INTENT: Service ist neu zu erstellen und ein Intent an onStartCommand() zu verschicken

- Gebundene Services werden in der Regel nicht beendet.

```
public class MyService extends Service {
    MediaPlayer mediaPlayer=null;
    public MyService() {}

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

    @Override
    public void onCreate() {
        super.onCreate();
        mediaPlayer=MediaPlayer.create(getApplicationContext(),R.raw.liedchen);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        mediaPlayer.start();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        mediaPlayer.release();
        super.onDestroy();
    }
}
```

Starten und Beenden im MainAktivity:

```
public void onStartClick(View view) {
    Intent intent=new Intent(getApplicationContext(), MyService.class);
    startService(intent);
}

public void onStopClick(View view) {
    Intent intent=new Intent(getApplicationContext(), MyService.class);
    stopService(intent);
}
```

```
public class MyService extends Service {

    class MyBinder extends Binder {
    //Binder is a standard implementation of IBinder
           MyService getService() {
           return new MyService();
       }
    }

    MyBinder binder= new MyBinder();

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

    public String getNews(){
        return "actually nothing";
    }
}
```

```
public class MainActivity extends AppCompatActivity {
    boolean gebunden=false;
    MyService myservice=null;

    private ServiceConnection connection = new ServiceConnection() {
        // ServiceConnection is an interface
        // definiert callbacks für service binding mit bindService()
        //s. nächste Folie
    };

    // onCreate()
    public void onClick(View view) {
        if (gebunden) { String news=myservice.getNews();
                        Toast.makeText(this,news, Toast.LENGTH_LONG).show();}
    }
    @Override
    protected void onResume() {
        super.onResume();
        Intent intent = new Intent(this, MyService.class);
        this.bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }
    @Override
    protected void onPause() {
        super.onPause();
        if (gebunden) { this.unbindService(connection);
                        gebunden = false;
        }
    }
}
```

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

    @Override
    public void onServiceConnected(ComponentName className, IBinder servicebinder) {
        MyService.MyBinder binder = (MyService.MyBinder) servicebinder;
        myservice = binder.getService();
        gebunden = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName arg0) {
        gebunden = false;
    }
};
```

Notifications

- Benachrichtigungen, die kurze, zeitnahe Informationen zu Ereignissen einer App vermitteln, während sie nicht verwendet wird,
- erscheinen in der Statusleiste mit Symbol (s. ImageAsset) , Titel etc.
- durch ihre Auswahl kann eine Aktion auszuführt werden

Figure 1. Notification icons appear on the left side of the status bar

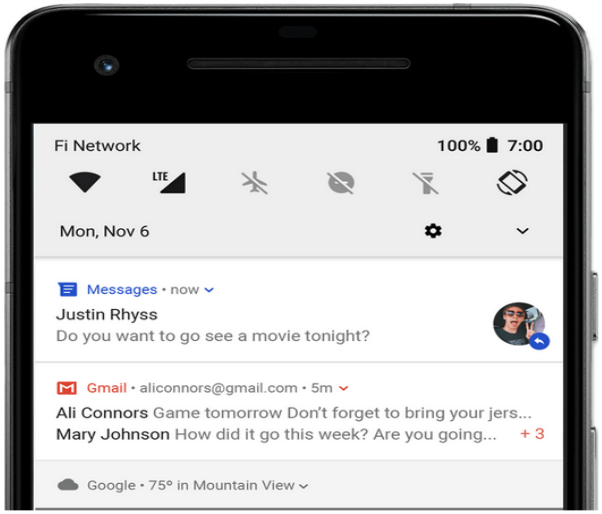


Figure 2. Notifications in the notification drawer

Quelle: https://developer.android.com/guide/topics/ui/notifiers/notifications

Erstellen und Versenden

- Zum Versenden wird Builder-Objekt und Notification Channel benötigt
- Das Versenden erfolgt mit Hilfe des NotificationManagers
- Für jede Notification muss eine eindeutige ID definiert werden
- Kann automatisch durch Auswählen beendet werden (s. andere Eigenschaften)

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_stat_name)
        .setContentTitle("TITEL")
        .setContentText("wichtige Meldung")
        .setAutoCancel(true)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);

NotificationManager notificationManager = getSystemService(NotificationManager.class);
int notificationId=9999;

notificationManager.notify(notificationId, builder.build());
```

Notification–Kanal

- Ab Android 8.0 (API-Ebene 26) werden Benachrichtigungen einem Kanal zugeordnet, um nur bestimmte Benachrichtigungskanäle einer App deaktivieren zu können
- Die Definition eines Nachrichtenkanals erfolgt über NotificationManager und muss vor Notification-Definition stattfinden

```
public final static String CHANNEL_ID="1234567890";
NotificationChannel channel;


channel=new NotificationChannel(CHANNEL_ID, "Name", NotificationManager.IMPORTANCE_DEFAULT);
channel.setDescription("Kanalbeschreibung");
NotificationManager notificationManager = getSystemService(NotificationManager.class);
notificationManager.createNotificationChannel(channel);
```

https://developer.android.com/training/notify-user/build-notification#java

- durch ihre Auswahl kann eine Aktion (Standard-Action) ausgeführt, z. B. eine Aktivität oder ein Service gestartet werden

PendingIntent:

- Ein Verweis auf Intent, das vom System verwaltet wird
- ein Erlaubnis für eine andere App den Vorgang durchzuführen
- existiert unabhängig von Sender-App, d.h. kann von anderen Prozessen auch dann verwendet werden, wenn der Prozess der besitzenden Anwendung beendet wurde
- `getActivity(Context context, int requestCode, Intent intent, int flags, Bundle options)`

```
Intent intent=new Intent(getApplicationContext(),AnotherActivity.class);
PendingIntent launchIntent=PendingIntent.getActivity(getApplicationContext(),0,intent,
PendingIntent.FLAG_IMMUTABLE);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_stat_name)
        .setContentTitle("Titel")
        .setContentText("wichtige Meldung")
        .setContentIntent(launchIntent)
        .setAutoCancel(true)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);

int notificationId=9999;
// notificationId is a unique int for each notification that you must define
notificationManager.notify(notificationId, builder.build());
```

- Es können bis drei weitere Actionen hinzugefügt werden (addAction), z.B. das Starten einen BroadcastReceiver, der einen Job im Hintergrund ausführt
- Außerdem möglich: „replay button“, „progress bar“ und weiteres

```
Intent intent=new Intent(getApplicationContext(),AnotherActivity.class);
PendingIntent launchIntent = PendingIntent.getActivity(getApplicationContext(),0,intent,
                                                        PendingIntent.FLAG_IMMUTABLE);

Intent intentBR=new Intent(this, MyReceiver.class);
intentBR.setAction(MY_ACTION_BROADCAST_ID);
intentBR.putExtra(DATA_KEY,"irgendwas");
PendingIntent pendingIntentBR =
          PendingIntent.getBroadcast(getApplicationContext(),0,intentBR,
                                                       PendingIntent.FLAG_IMMUTABLE);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_stat_name)
        .setContentTitle("Titel")
        .setContentText("wichtige Meldung")
        .setContentIntent(launchIntent)
        .addAction(R.drawable.ic_stat_senden, "Senden",pendingIntentBR)
        .setAutoCancel(true)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

```
public class MainActivity extends AppCompatActivity {
    public final static String MY_ACTION_BROADCAST_ID="My Action Broadcast Id";
    public final static String DATA_KEY="data key";
    //…
    MyReceiver broadcastReceiver;
    IntentFilter filter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //GUI Definition, Notification cannel definition  …

        broadcastReceiver = new MyReceiver();
        filter= new IntentFilter(MY_ACTION_BROADCAST_ID);
        getApplicationContext().registerReceiver(broadcastReceiver,filter);
    }
    @Override
    protected void onDestroy() {
        getApplicationContext().unregisterReceiver(broadcastReceiver);
        super.onDestroy();
    }
    public void onClick(View view) {
        Intent intent=new Intent(getApplicationContext(),OtherActivity.class);
        PendingIntent launchIntent=PendingIntent.getActivity(getApplicationContext(),0,intent, PendingIntent.FLAG_IMMUTABLE);

        Intent intentBR=new Intent(this, MyReceiver.class);
        intentBR.setAction(MY_ACTION_BROADCAST_ID);
        intentBR.putExtra(DATA_KEY,"irgendwas");
        PendingIntent pendingIntentBR=PendingIntent.getBroadcast(getApplicationContext(),0,intentBR, PendingIntent.FLAG_IMMUTABLE);

        //Notification definition und senden
    }
}
```

```
public class MyReceiver extends BroadcastReceiver {
    public final static String MY_ACTION_BROADCAST_ID="My Action Broadcast Id";
    public final static String DATA_KEY="data key";
    @Override
    public void onReceive(Context context, Intent intent) {
        String data=intent.getStringExtra(DATA_KEY);
        Log.i("MyReceiver",data);
    }
}
```

Alarm Manager

- kann generell eine Anwendung zu einem späteren Zeitpunkt starten.
- Dafür wird ein Intent (PendingIntent) vom System gesendet und die festgelegte Zielkomponente gestartet.
- Registrierte Alarme können erhalten bleiben, während das Gerät asleep bzw. ausgeschaltet ist.
- Ein einmaliger bzw. eine wiederholter Start ist mit den Methoden set(), setReapeating(), setExact() etc. möglich.
- Beenden erfolgt mit cancel()
- Ein Parameter der Methoden regelt die Dringlichkeit, z.B.

RTC_WAKEUP weckt das Gerät aus dem asleep-Zustand
RTC wird ausgelöst beim nächsten Aufwachen des Geräts

Beispiel

```
public void onClickStart(View view) {
    Intent intent=new Intent(getApplicationContext(), MyService.class);
    PendingIntent pendingIntent =
        PendingIntent.getService(getApplicationContext(),0,intent,
        PendingIntent.FLAG_IMMUTABLE);

    long firstStart=System.currentTimeMillis();
    long interval=60000;
    AlarmManager alarmManager= (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, firstStart, interval,
                                                                pendingIntent);

    //SimpleDateFormat sdf = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");
    //String dateString = "01-10-2021 10:05:00";
    //Date date = sdf.parse(dateString);
    //Calendar calendar=Calendar.getInstance();
    //calendar.setTime(date);
    //alarmManager.setExact(AlarmManager.RTC_WAKEUP,calendar.getTimeInMillis(),
    //pendingIntent);
}

public void onClickStop(View view) {
    AlarmManager alarmManager= (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    Intent intent=new Intent(getApplicationContext(), MyService.class);
    alarmManager.cancel(PendingIntent.getService(getApplicationContext(),0,intent,
                        PendingIntent.FLAG_IMMUTABLE));
}
```

```
public class MyService extends Service {
    public final static String CHANNEL_ID="1234567890";
    NotificationChannel channel;

    public MyService() { }

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

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        channel=new NotificationChannel(CHANNEL_ID,"mitNotApp",
                                                NotificationManager.IMPORTANCE_DEFAULT);
        channel.setDescription("Kanalbeschreibung");
        NotificationManager notificationManager = getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
                .setSmallIcon(R.drawable.ic_stat_name)
                .setContentTitle("AUFWACHEN!")
                .setContentText("Schlaf nicht, du verpasst was!")
                .setAutoCancel(true)
                .setPriority(NotificationCompat.PRIORITY_DEFAULT);
        int notificationId=9999;
        notificationManager.notify(notificationId, builder.build());
        this.stopSelf();
        return START_STICKY;
    }
}
```

WorkManager API

- ermöglicht zuverlässige Planung asynchroner Aufgaben, die auch dann ausgeführt werden sollen, wenn die App beendet oder das Gerät neu gestartet wird.

- Die genaue Ausführungszeit hängt von den Einschränkungen (z.B. nur wenn WLAN vorhanden, ausreichend Speicher vorhanden etc.) und von Systemoptimierungen.

- Ersetzt alle bisherigen Android-APIs für die Hintergrundplanung

```
Dependencies in build.gradle (Module: app):
def work_version = "2.7.1"
// (for Java)
implementation "androidx.work:work-runtime:$work_version"
```

Definition der Klasse:

```
public class MyWork extends Worker {
    public MyWork(@NonNull Context context, @NonNull WorkerParameters workerParams) {
        super(context, workerParams);
    }

    @NonNull
    @Override
    public Result doWork() {
        for (int i=0;i<10;i++)
        Log.i("Worker",String.valueOf(i));
        Log.i("Worker","doing");
        return Result.success();
    }
}
```

Rückgabe über ListenableWorker.Result: Aufruf von Result.success(), Result.failure() oder Result.retry() zum Erzeugen eines Objektes

Definition von WorkRequest: erstellen und Verschicken zum WorkManager:

```
WorkRequest workRequest = new OneTimeWorkRequest.Builder(MyWork.class).build();
WorkManager.getInstance(getApplicationContext()).enqueue(workRequest);
```

Zustände des Lebenszyklus vom **WorkRequest**, können mit *getState()* von **WorkInfo** abgefragt werden

BLOCKED

CANCELLED

ENQUEUED

FAILED

RUNNING

SUCCEEDED

Einmalige Ausführung ohne und mit zusätzlichen Konfiguration

```
WorkRequest myWorkRequest = OneTimeWorkRequest.from(MyWork.class);
```

```
WorkRequest myWorkRequest = new
        OneTimeWorkRequest.Builder(MyWork.class)
        .build();
```

```
WorkRequest myWorkRequest = new
        OneTimeWorkRequest.Builder(MyWork.class)
        // Additional configuration
        .build();
```

NetworkType - schränkt den Netzwerktyp ein, z.B. nur WLAN (UNMETERED)
BatteryNotLow - wenn auf true gesetzt, erfolgt die Ausführung nicht, wenn das Gerät nicht ausreichend aufgeladen ist (sich im Batteriemodus NotLow befindet)
RequiresCharging - wenn auf true gesetzt, erfolgt die Ausführung nur, wenn das Gerät gerade aufgeladen wird
DeviceIdle - wenn auf true gesetzt, muss das Gerät des Benutzers im Doze-Modus (längere Zeit inaktiv) sein
StorageNotLow - wenn auf true gesetzt, erfolgt keine Ausführung, wenn der Speicherplatz des Benutzers auf dem Gerät zu gering ist

```
Constraints constraints = new Constraints.Builder()
        .setRequiredNetworkType(NetworkType.UNMETERED)
        .setRequiresCharging(true)
        .build();

WorkRequest myWorkRequest =
        new OneTimeWorkRequest.Builder(MyWork.class)
                .setConstraints(constraints)
                .build();
```

Wiederholte Ausführung:

- **Request erstellen**

```
PeriodicWorkRequest saveRequest =
new PeriodicWorkRequest.Builder(SaveImageToFileWorker.class, 1, TimeUnit.HOURS)
           // Constraints
           .build();

PeriodicWorkRequest saveRequest =
new PeriodicWorkRequest.Builder(SaveImageToFileWorker.class,
               1, TimeUnit.HOURS,
               15, TimeUnit.MINUTES)//in den letzten 15 min der Stunde
           .build();
```

- **Ausführung veranlassen**

```
WorkManager.getInstance(getApplicationContext()).
         enqueueUniquePeriodicWork("uniquename",ExistingPeriodicWorkPolicy.KEEP,
         request);
```

- **Ausführung beenden**

```
workManager.cancelWorkById(request.getId()); // by id
workManager.cancelUniqueWork("uniquename"); // by name
```

- **Suchen und beobachten von WorkRequest**

```
WorkRequest request = new OneTimeWorkRequest.Builder(MyWorker.class).build();
workManager.enqueue(request);

LiveData<WorkInfo> livedata =
                    workManager.getWorkInfoByIdLiveData(request.getId());
livedata.observe(this, new Observer<WorkInfo>() {
          public void onChanged(WorkInfo info) {
            //TODO
          }
        });
```

- **Übergabe der Daten:** über Data-Objekt möglich

```
import androidx.lifecycle.Observer;
import androidx.work.Data;
import androidx.work.OneTimeWorkRequest;
import androidx.work.WorkInfo;
import androidx.work.WorkManager;
import androidx.work.WorkRequest;
import androidx.work.Worker;
import androidx.work.WorkerParameters;
```

```
public void onClickLos(View view) {
    int arg=Integer.parseInt(editText.getText().toString());
    Data myData = new Data.Builder()
            .putInt(KEY_ARG, arg)           //public static final String KEY_ARG = "argument";
            .build();
    WorkRequest workRequest =
            new OneTimeWorkRequest.Builder(MyWorker.class)
                    .setInputData(myData)
                    .build();
    WorkManager.getInstance(getApplicationContext())
            .enqueue(workRequest);
    WorkManager.getInstance(getApplicationContext())
            .getWorkInfoByIdLiveData(workRequest.getId())
            .observe(this, new Observer<WorkInfo>() {
                @Override
                public void onChanged(WorkInfo info) {
                    if (info != null && info.getState().isFinished()) {
                        Log.i("Worker","fertig");
                        String myResult = info.getOutputData().getString(KEY_RESULT);
                        textView.setText(String.valueOf(myResult));
                    }
                }                           //public static final String KEY_RESULT = "result";
            });
}
// isFinished() true bei  SUCCEEDED, FAILED und CANCELLED
// if (info.getState()==WorkInfo.State.FAILED)…
```

```
public class MyWorker extends Worker {
    public static final String KEY_ARG = "argument";
    public static final String KEY_RESULT = "result";

    public MyWorker(@NonNull Context context, @NonNull WorkerParameters workerParams) {
        super(context, workerParams);
    }

    @NonNull                                    //Compiler erkennt NullPointerException
    @Override
    public Result doWork() {
        int arg = getInputData().getInt(KEY_ARG, 0);
        if (arg<42) {
            Data output = new Data.Builder()
                    .putString(KEY_RESULT, "zu einfach!")
                    .build();
            return Result.failure(output);
        }

        long result=1;
        for (int i=1;i<=arg;i++) result*=i;

        Data output = new Data.Builder()
                .putString(KEY_RESULT, String.valueOf(result))
                .build();
        return Result.success(output);
    }
}
```

https://developer.android.com/guide/

Bernhard Baltes-Götz, Einführung in die Entwicklung von Apps für Android 8