

Mobil alkalmazásfejlesztés - Aszinkron feladatok, kommunikáció

Dr. Bilicki Vilmos

Szoftverfejlesztés Tanszék



A fóliához felhasznált anyagok:

Google: Android Developer Fundamentals (Version 2), <https://developer.android.com/courses/fundamentals-training/overview-v2>

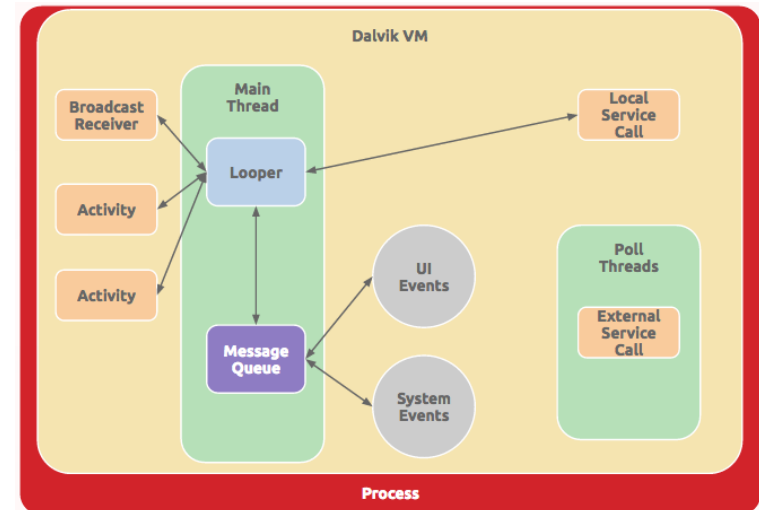
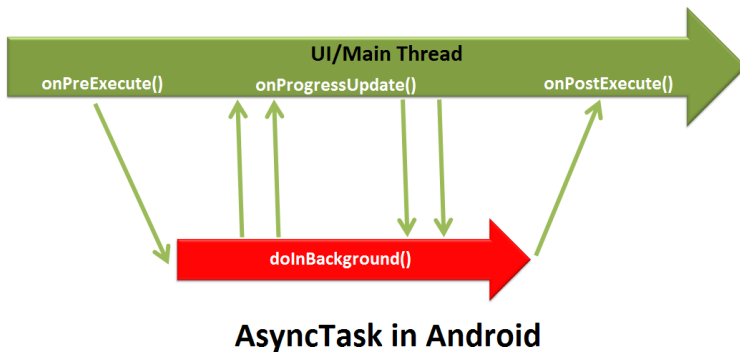
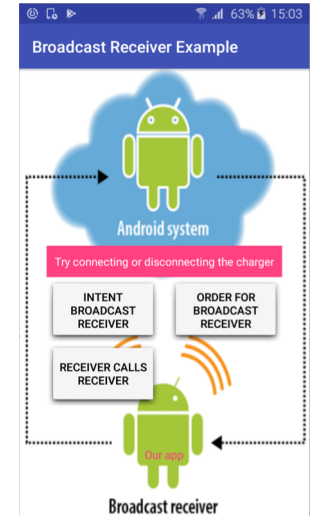
Az előző előadás áttekintése:

- ▶ RecyclerView
 - Komponensek
 - Megvalósítása
- ▶ Egyedi felhasználói élmény
 - Drawables
 - Stílusok
 - Témák
- ▶ Material design
 - Az alapelv
 - Képek
 - Színek
 - Elrendezések
 - Komponensek
- ▶ Adaptív elrendezés erőforrások
 - Adaptív erőforrások
 - Alternatív erőforrások



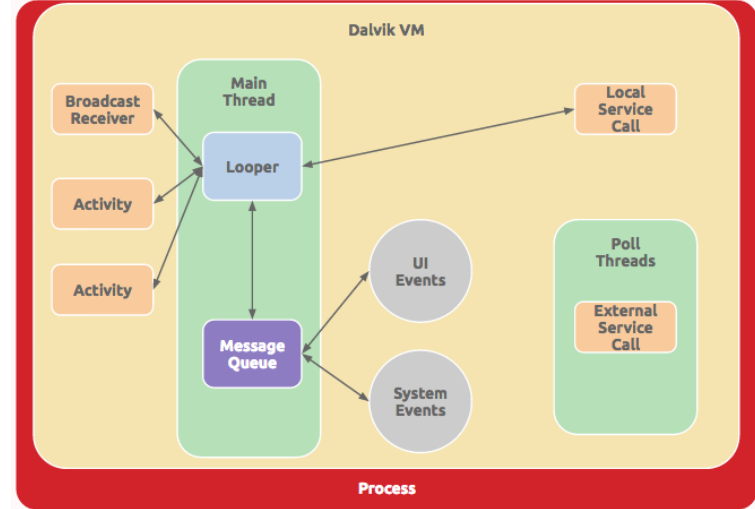
Attekintés

- ▶ Háttér folyamatok
 - AsyncTask és AsyncTaskLoader
 - Internet kapcsolat



Szálkezelés

- ▶ A programban egy független futtatási útvonal
- ▶ A kód soronként van futtatva
- ▶ Az alkalmazások a Java "fő" vagy "UI szál"-jában futnak
- ▶ Kirajzolja a UI-t a képernyőre
- ▶ UI események kezelésével ragál a felhasználói eseményekre



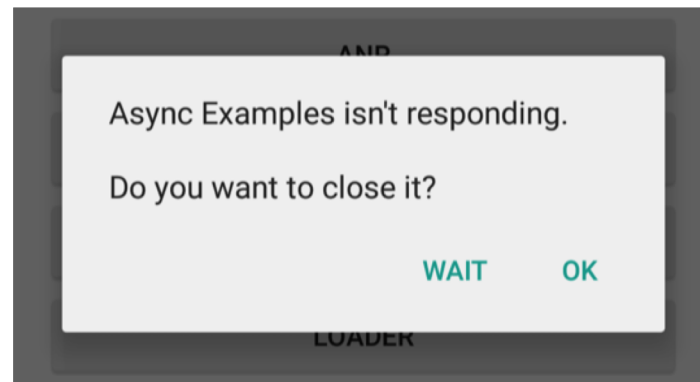
A Fő szálnak gyorsnak kell lennie

- ▶ A HW 16ms-ént frissíti a képernyőt
- ▶ Az UI szálnak 16ms-a van a dolga elvégzéséhez
- ▶ Ha ez tovább tart akkor az alkalmazás akadozik vagy lefagy



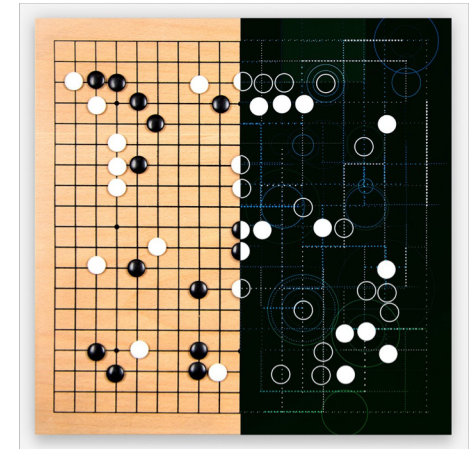
A felhasználók letörlik a nem válaszoló alkalmazásokat

- ▶ Amennyiben a képernyő túl sokat vár adott válaszra akkor nem lesz válaszképes
- ▶ A keretrendszer kiteszi az Application Not Responding (ANR) dialógust



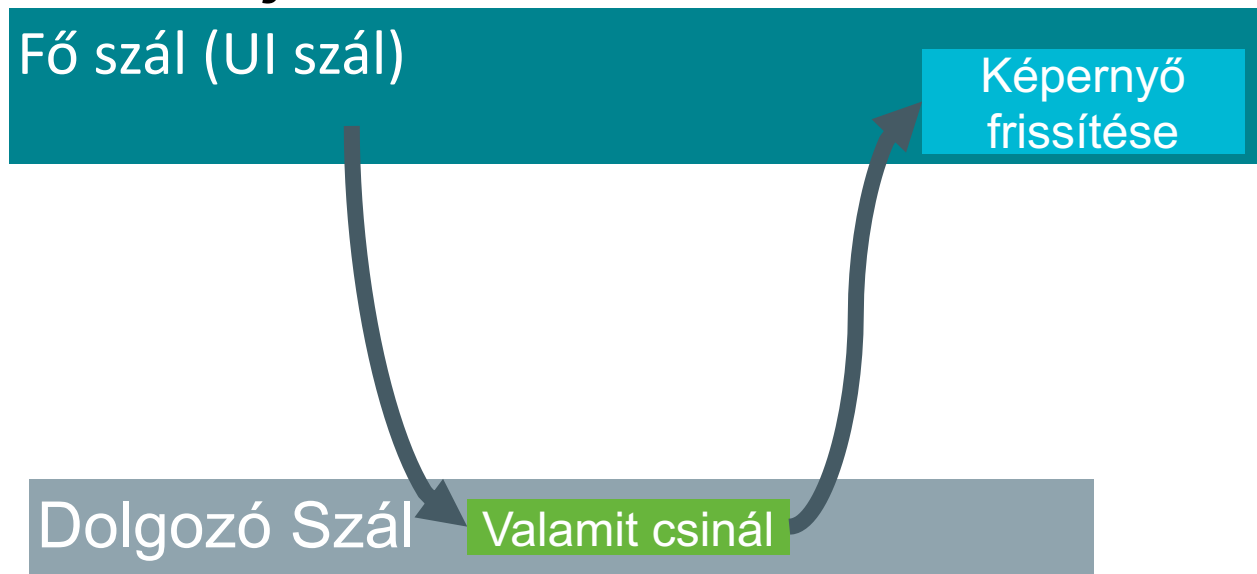
Mi lehet hosszú futási idejű feladat?

- ▶ Hálózati műveletek
- ▶ Hosszú számítások
- ▶ Fájlok fel/le töltése
- ▶ Képek kezelés
- ▶ Adat betöltése

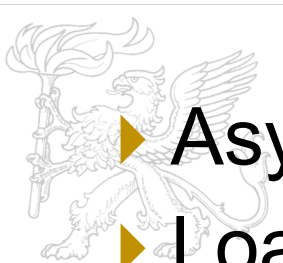


Háttér szálak

- ▶ A hosszú futási idejű feladatokat háttér szálakban futtatjuk



- ▶ AsyncTask
- ▶ Loader Framework
- ▶ Services



Két szabály az Android szálakhoz

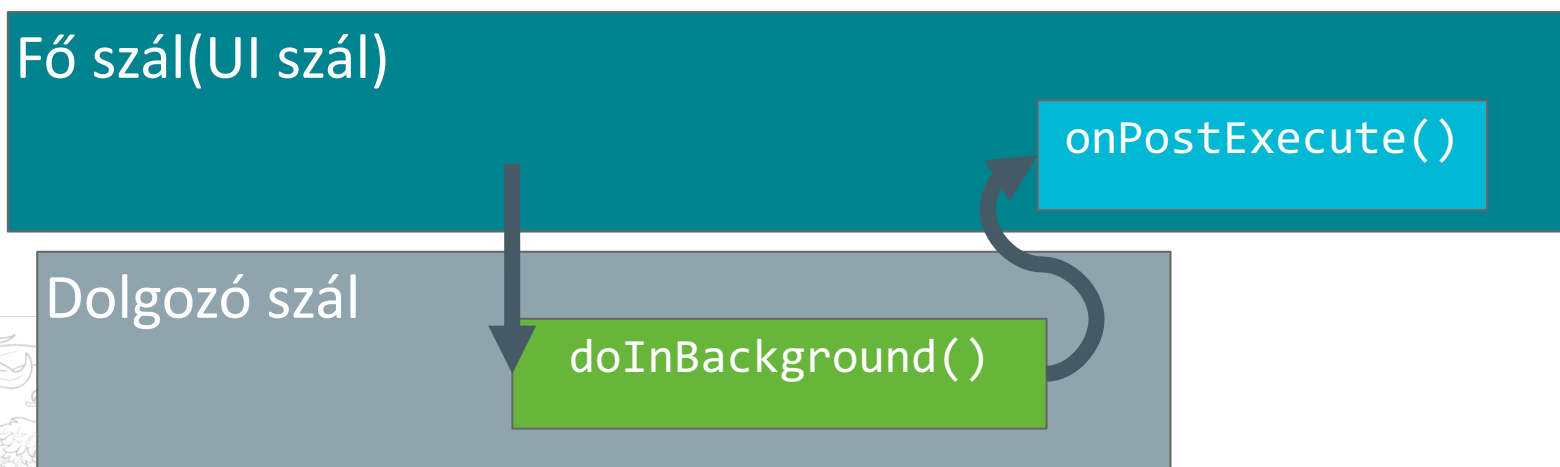
- ▶ Ne blokkoljuk a UI szálát
 - Az egyes képernyőkhöz tartozó munkát fejezzük be kevesebb mint 16ms alatt
 - A lassú nem UI feladatokat futtassuk háttér szálban
- ▶ Tilos az Android UI eszköztárhoz hozzáférni az UI szálon kívül
 - Az UI-hoz hozzáférő munkát csak az UI szálon végezhetjük el





AsyncTask

- ▶ A háttér folyamatok megvalósításához használjuk az AsyncTask-ot



Két metódust kell felülírni

- ▶ `doInBackground()`—a háttérszálban fut
 - A háttérben futó munka helye
- ▶ `onPostExecute()`—a fő szálban fut
 - Eredmények feldolgozása
 - Az eredmények kihelyezése az UI-ba

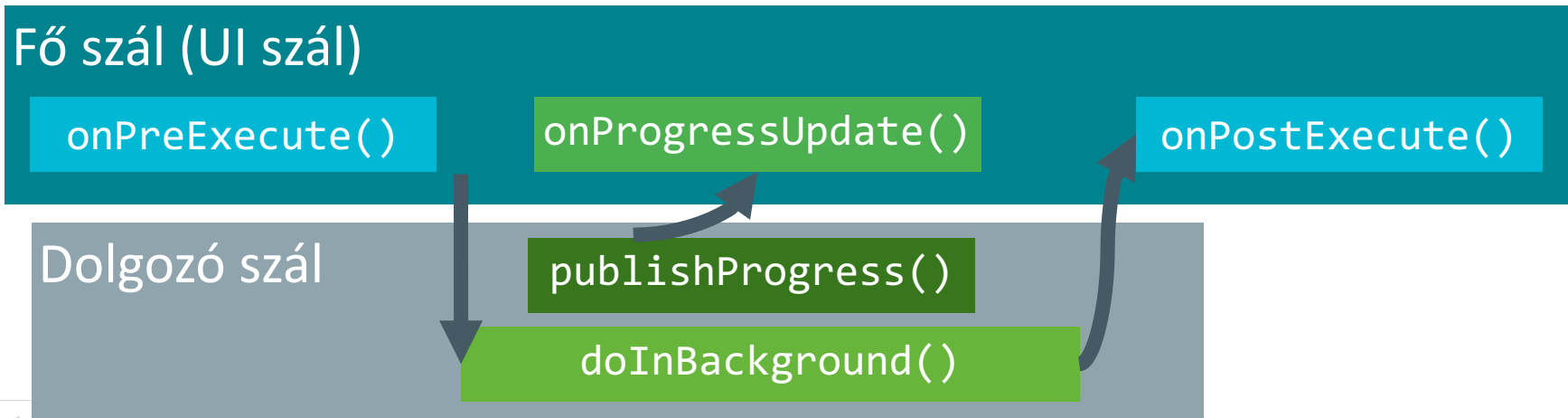


AsyncTask kiegészítő metódusok

- ▶ `onPreExecute()`
 - A fő szálon fut
 - A feladatot állítja össze
- ▶ `onProgressUpdate()`
 - A fő szálon fut
 - A `publishProgress()` hívásokat fogadja a háttér szálból



AsyncTask kiegészítő metódusok



AsyncTask létrehozása

1. AsyncTask-ból alosztály létrehozása
2. Adjuk meg a doInBackground() adattípusát
3. Adattípus megadása a onProgressUpdate() lépésekhez
4. Adattípus megadása onPostExecute() futtatáshoz



```
private class MyAsyncTask extends
    AsyncTask<URL, Integer, Bitmap> {...}
```

MyAsyncTask osztály definíció

```
private class MyAsyncTask
    extends AsyncTask<String, Integer, Bitmap>
{ ... }
```

doInBackground()

onProgressUpdate()

onPostExecute()

- String— lehet egy lekérdezés, fájlnev, URI
- Integer— az előrehaladás százalékos, vagy lépésenkénti aránya
- Bitmap—kihelyezendő kép
- Void ha nem akarunk adatot adni



onPostExecute()

```
protected void onPostExecute() {  
    // display a progress bar  
    // show a toast  
}
```



doInBackground()

```
protected Bitmap doInBackground(String... query) {  
    // Get the bitmap  
    return bitmap;  
}
```



onProgressUpdate()

```
protected void onProgressUpdate(Integer... progress) {  
    setProgressPercent(progress[0]);  
}
```



onPostExecute()

```
protected void onPostExecute(Bitmap result) {  
    // Do something with the bitmap  
}
```



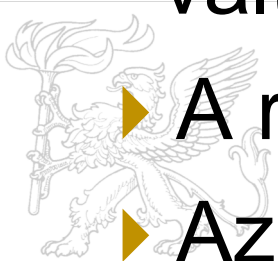
Start background work

```
public void loadImage (View view) {  
    String query = mEditText.getText().toString();  
    new MyAsyncTask(query).execute();  
}
```



Az AsyncTask korlátai

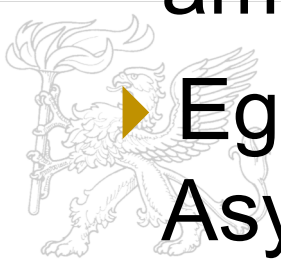
- ▶ Amikor a konfiguráció változik akkor az Activity megsemmisül
- ▶ Az AsyncTask nem kapcsolódhat többé az Activity-hez
- ▶ Új AsyncTask készül minden konfiguráció változás után
- ▶ A régi AsyncTask-ok megmaradnak
- ▶ Az alkalmazás kifogyhat a memóriából





Mikor alkalmazzuk az AsyncTask-ot?

- ▶ Rövid, megszakítható feladatok
- ▶ Olyan feladatoknál ahol nem kell visszajelezni a felhasználónak, képernyőnek
- ▶ Olyan alacsony prioritású feladatok amelyek befejezetlenek is maradhatnak
- ▶ Egyébként használjuk a AsyncTaskLoader-t



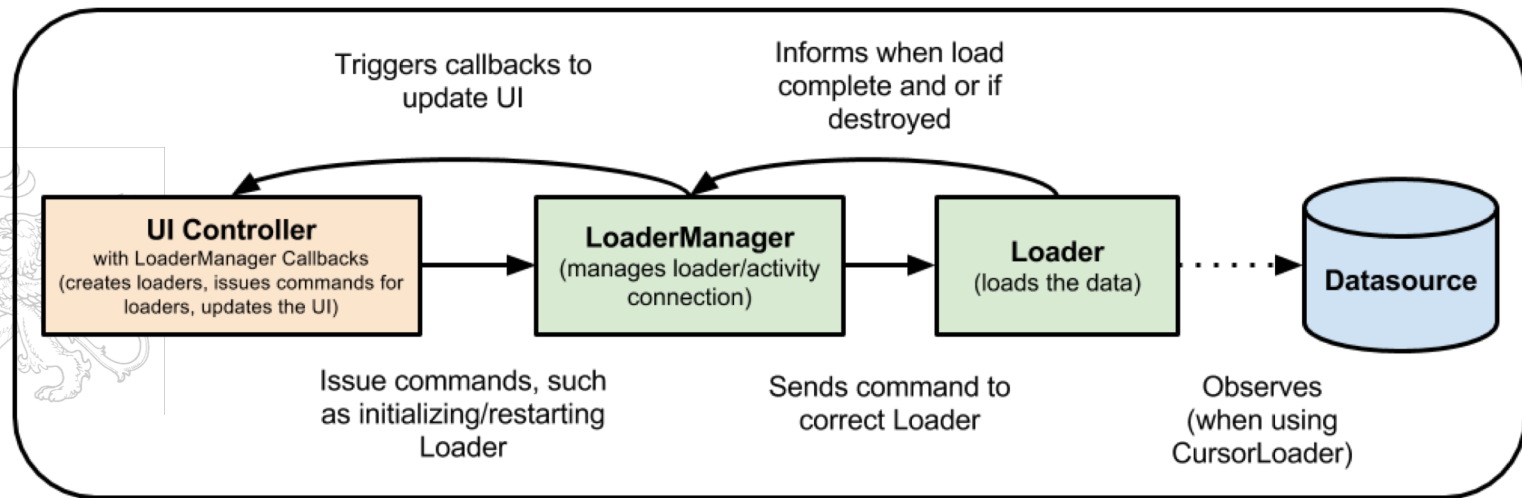
Loader

- ▶ Aszinkron adatbetöltést biztosít
- ▶ Konfiguráció változás után újrapcsolja az Activity-t
- ▶ Figyelheti az adatforrás változásait és az új adatokat küldheti
- ▶ Activity –ben megvalósított Callback-k
- ▶ Több Loader megvalósítás
 - AsyncTaskLoader, CursorLoader



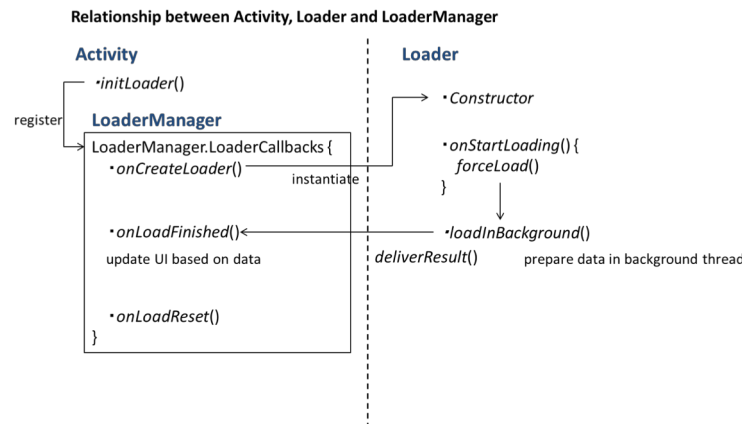
Miért célszerű loader-t használni?

- ▶ Ki lehet szervezni a munkát az UI szálból
- ▶ A LoaderManager kezeli a konfiguráció változásokat
- ▶ A keretrendszer hatékonyan valósítja meg



LoaderManager

- ▶ Callback-ek segítségével menedzseli az egyes loader-eket
- ▶ Több loader-t is kezelhet egyszerre
 - egy loader az adatbázishoz, egy az AsyncTask adathoz, Internet adathoz...



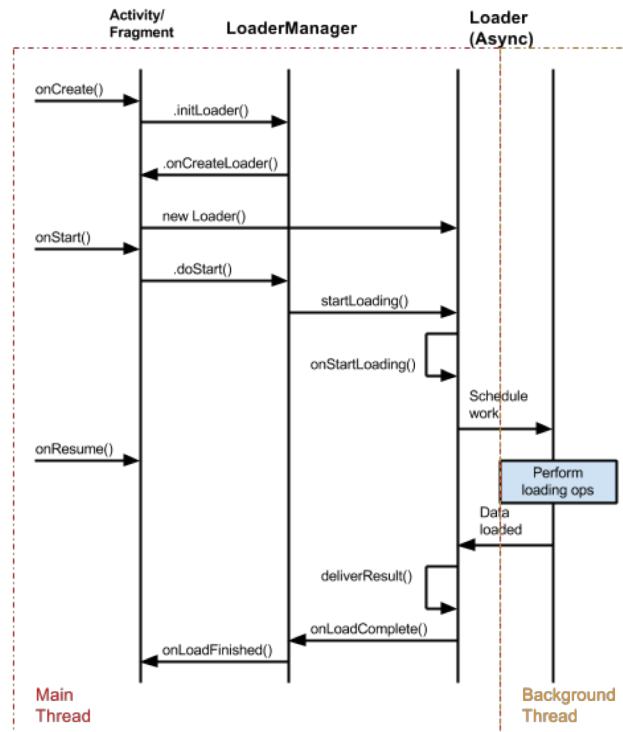
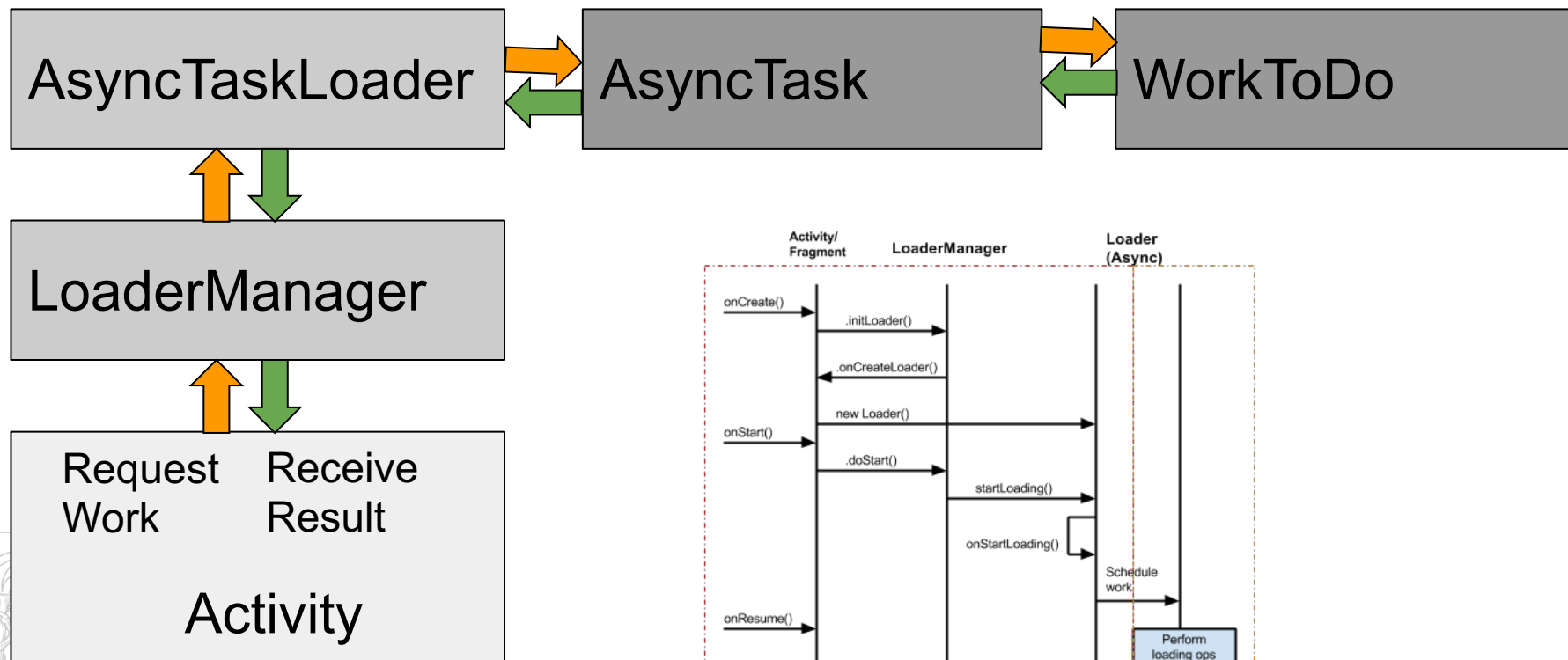
■ Loader létrehozása initLoader()-el

- ▶ Létrehoz és elindít egy loader-t, vagy újrahasonosít egy meglévőt az adattal együtt
- ▶ Használjuk a restartLoader() az adatok törlésére

```
getLoaderManager().initLoader(ld, args, callback);  
getLoaderManager().initLoader(0, null, this);  
getSupportLoaderManager().initLoader(0, null, this);
```



AsyncTaskLoader áttekintés

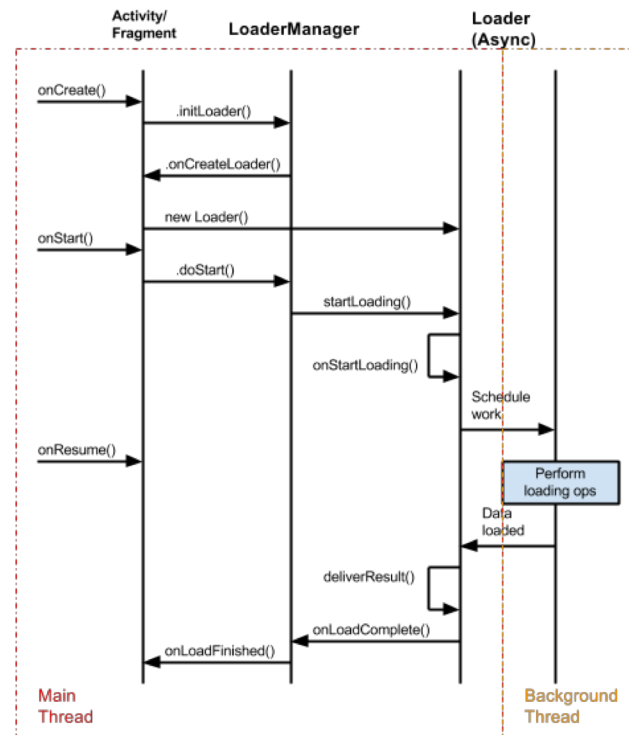


AsyncTask ➡ AsyncTaskLoader

doInBackground() ➡ loadInBackground()
onPostExecute() ➡ onLoadFinished()

► Az AsyncTaskLoader osztály létrehozásának lépései

1. AsyncTaskLoader osztály létrehozása
2. Konstruktor megvalósítása
3. loadInBackground()
4. onStartLoading()



AsyncTaskLoader-ból alosztály

```
public static class StringListLoader
    extends AsyncTaskLoader<List<String>> {

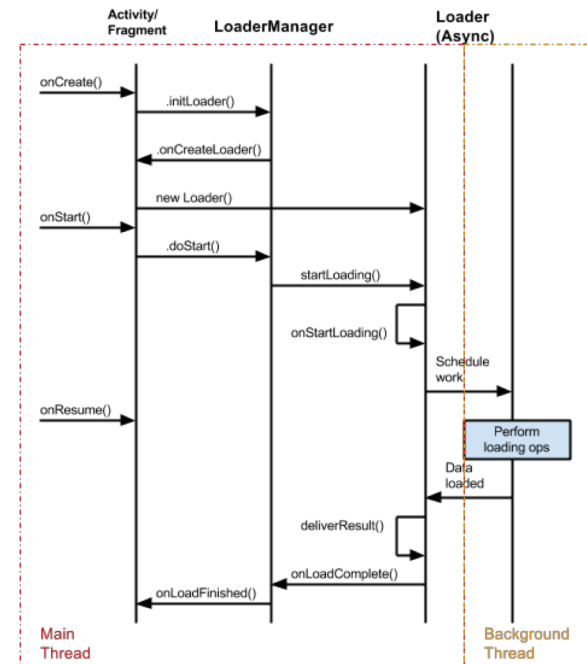
    public StringListLoader(Context context, String queryString) {
        super(context);
        mQueryString = queryString;
    }
}
```





loadInBackground()

```
public List<String> loadInBackground() {
    List<String> data = new ArrayList<String>;
    //TODO: Load the data from the network or from a database
    return data;
}
```

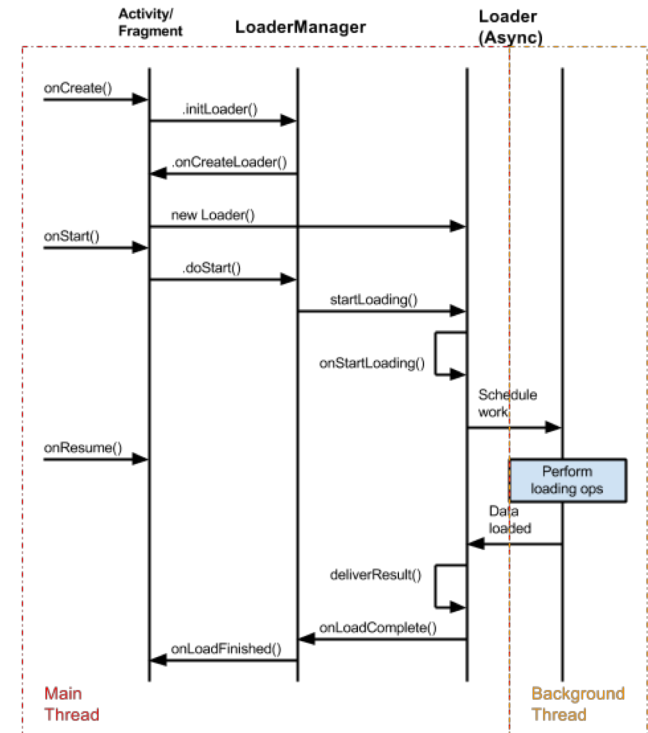


onStartLoading()

Amikor meghívjuk a `restartLoader()`-t vagy `initLoader()`-t, a `LoaderManager` meghívja az `onStartLoading()` callback-et

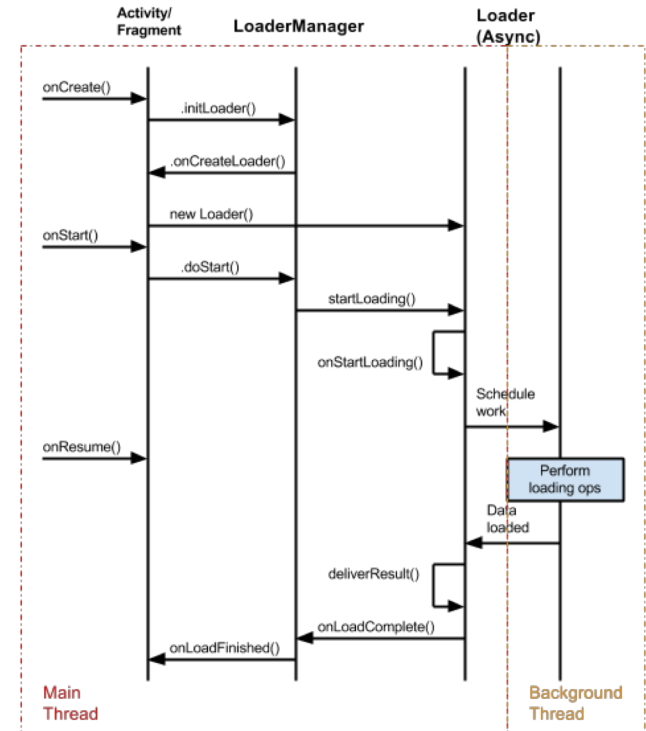
- Gyorsítótárazott adatok ellenőrzése
- Az adatforrás figyelésének elkezdése(amennyiben szükséges)
- Hívjuk meg a `forceLoad()`-ot az adat betöltésére ha nincs adat vagy változások voltak

```
protected void onStartLoading()
{ forceLoad(); }
```



Az Activity-ben valósítsuk meg a szükséges callback-eket

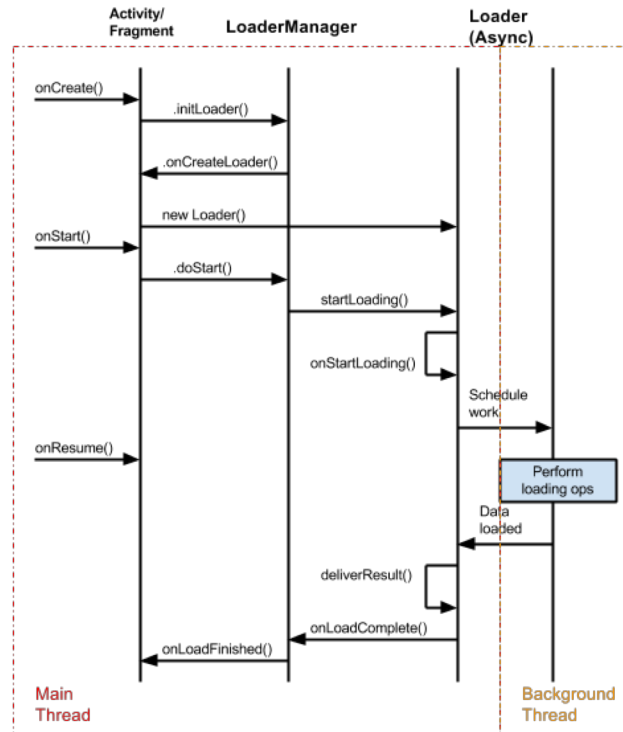
- ▶ onCreateLoader() — Loader létrehozása és visszaadása az adott ID-vel
- ▶ onLoadFinished() — akkor hívja meg, ha az előzőleg létrehozott loader befejezte a munkáját
- ▶ onLoaderReset() — Akkor hívja meg ha az előzőleg létrehozott loader-t nullázták és az adata már nem hozzáférhető





onCreateLoader()

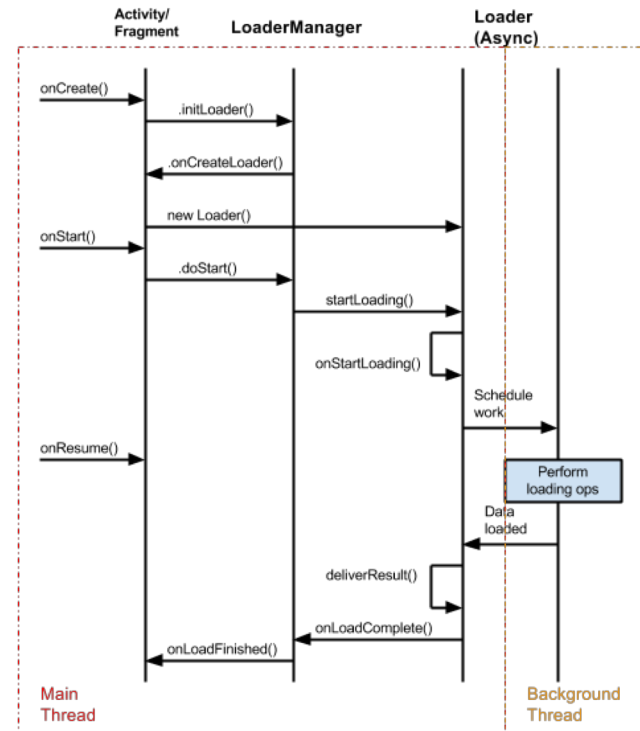
```
@Override
public Loader<List<String>> onCreateLoader(int id, Bundle args) {
    return new StringListLoader(this, args.getString("queryString"));
}
}
```



onLoadFinished()

A `loadInBackground()` eredményei át lettek adva a `onLoadFinished()`-nek amellyel kitehetjük a képernyőre

```
public void
onLoadFinished(Loader<List<String>> loader,
List<String> data) {
    mAdapter.setData(data);
}
```



onLoaderReset()

- ▶ Csak akkor hívják meg, ha a loader nullázódik
- ▶ Leggyakrabban üresen hagyható

@Override

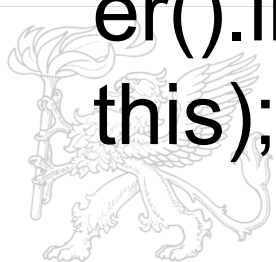
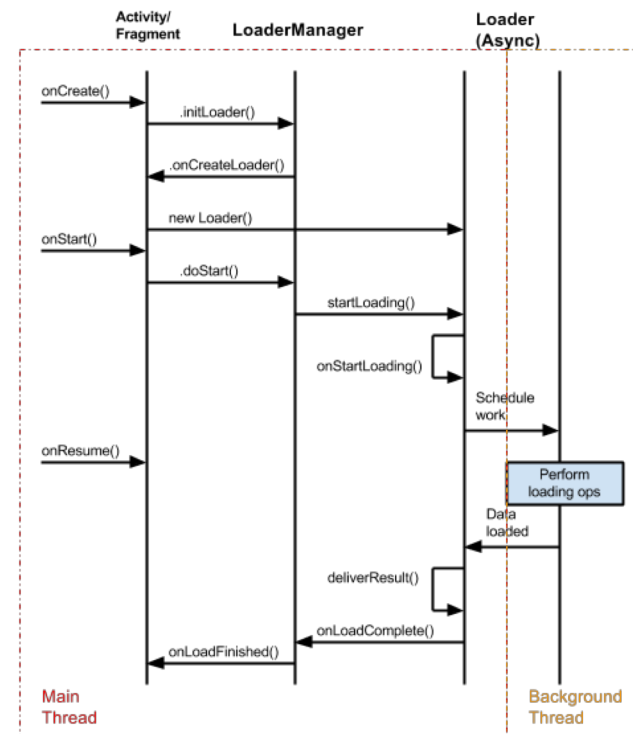
```
public void onLoaderReset(final
LoaderList<String> loader) { }
```



Loader betöltése initLoader()-el

- ▶ Az Activity-ben
- ▶ Használjunk támogató könyvtárakat

getSupportLoaderManager().initLoader(0, null, this);



Internet kapcsolat

► Internet kapcsolat lépései

1. Adjuk hozzá a jogosultságokat az Android Manifest-hez
2. Ellenőrizzük a hálózati kapcsolatot
3. Hozzunk létre egy dolgozó szálát
4. Valósítsuk meg a háttér feladatot
 1. URI létrehozása
 2. HTTP kapcsolat létrehozása
 3. Adat lekérése
 5. Eredmények feldolgozása
 1. Eredmények értelmezése



Jogosultságok az AndroidManifest-ben

Internet

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

Hálózat állapot ellenőrzése

```
<uses-permission  
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
```



Hálózat állapot

► ConnectivityManager

- Az elküldött kérdésekre válaszol
- Értesíti az alkalmazást, ha változik a hálózat állapota

► NetworkInfo

- Az adott típusú interfész státuszát írja le
- Mobil vagy Wi-Fi

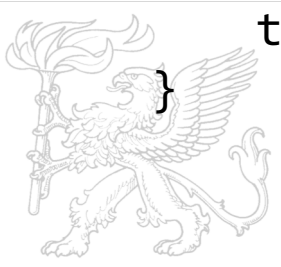


Van-e hálózat?

```
ConnectivityManager connMgr = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

if (networkInfo != null && networkInfo.isConnected()) {
    // Create background thread to connect and get data
    new DownloadWebpageTask().execute(stringUrl);
} else {
    textView.setText("No network connection available.");
}
```



WiFi és Mobil

```
NetworkInfo networkInfo =  
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);  
boolean isWifiConn = networkInfo.isConnected();  
  
networkInfo =  
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);  
boolean isMobileConn = networkInfo.isConnected();
```



Dolgozó szál

- ▶ AsyncTask—rövid, UI-t nem érintő feladat
- ▶ AsyncTaskLoader—hosszú UI-t érintő
- ▶ Background Service—hosszú UI-t nem érintő



Háttér feladat

- ▶ A háttér folyamatban (pl.: `doInBackground()`)
 - Hozzuk létre az URI-t
 - Nyissuk meg a HTTP kapcsolatot
 - Töltsük le az adatot



URI = Uniform Resource Identifier

- ▶ Egy erőforrásokat azonosító karakterlánc
 - file://
 - http:// and https://
 - content://



Példa

[https://www.googleapis.com/books/v1/volumes?
q=angular&maxResults=5&printType=books](https://www.googleapis.com/books/v1/volumes?q=angular&maxResults=5&printType=books)

Konstansok:

```
final String BASE_URL =  
    "https://www.googleapis.com/books/v1/volumes?";  
  
final String QUERY_PARAM = "q";  
  
final String MAX_RESULTS = "maxResults";  
  
final String PRINT_TYPE = "printType";
```



Hozzuk létre az UIR-t

```
Uri builtURI = Uri.parse(BASE_URL).buildUpon()  
    .appendQueryParameter(QUERY_PARAM, "Angular")  
    .appendQueryParameter(MAX_RESULTS, "10")  
    .appendQueryParameter(PRINT_TYPE, "books")  
    .build();  
URL requestURL = new URL(builtURI.toString());
```



Hozzunk létre egy kapcsolatot

- Használjuk a [URLConnection-t](#)
- Külön számban kell létrehozni
- Használunk kell az InputStreams-et és a try/catch blokkokat

```
URLConnection conn =
```

```
(URLConnection) requestURL.openConnection();
```

```
conn.setReadTimeout(10000 /* milliseconds */);
```

```
conn.setConnectTimeout(15000 /* milliseconds */);
```

```
conn.setRequestMethod("GET");
```

```
conn.setDoInput(true);
```



Kapcsolódás, letöltés, zárás

```
conn.connect();  
int response = conn.getResponseCode();  
InputStream is = conn.getInputStream();  
String contentAsString = convertIsToString(is, len);  
return contentAsString;
```

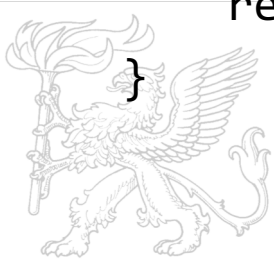
```
} finally {  
    conn.disconnect();  
    if (is != null) {  
        is.close();  
    }  
}
```



Folyam – karakterlánc átalakítás

```
public String convertIsToString(InputStream stream, int len)
    throws IOException, UnsupportedEncodingException {

    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```



Más HTTP klines könyvtárak

- Pl.: [OkHttp](#) , [Volley](#)
- Meghívhatóak a fő szálból
- Kevesebb kód



Volley

```
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                // Do something with response
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {}
        });
queue.add(stringRequest);
```



OkHttp

```
OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()
    .url("http://publicobject.com/helloworld.txt").build();
client.newCall(request).enqueue(new Callback() {
    @Override
    public void onResponse(Call call, final Response response)
        throws IOException {
        try {
            String responseData = response.body().string();
            JSONObject json = new JSONObject(responseData);
            final String owner = json.getString("name");
        } catch (JSONException e) {}
    }
});
```



Eredmények feldolgozása

- Valósítsuk meg az eredmények feldolgozását kezelő metódust (`onPostExecute()`)
- A válasz gyakran JSON vagy XML

Használjunk kisegítő osztályokat

- [JSONObject](#), [JSONArray](#)
- [XMLPullParser](#)—XML kezel



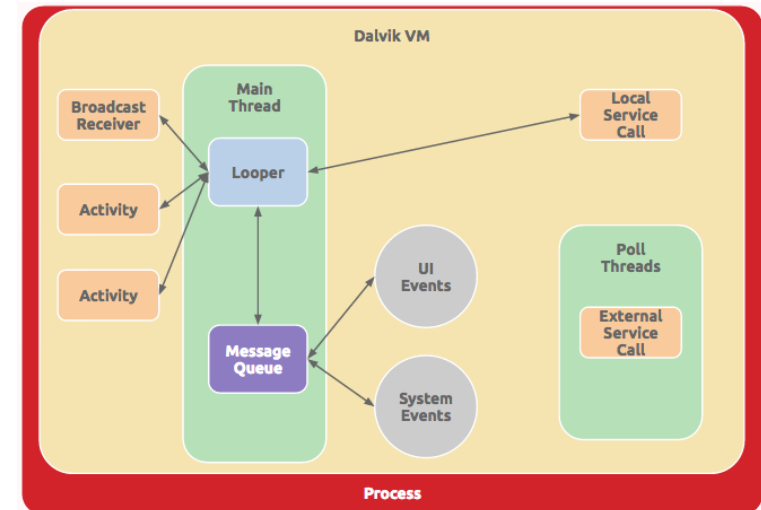
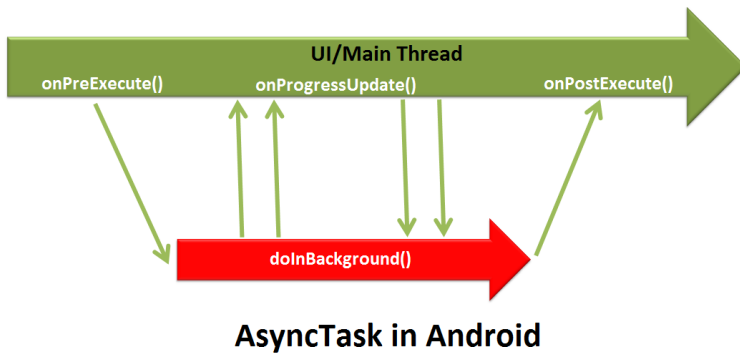
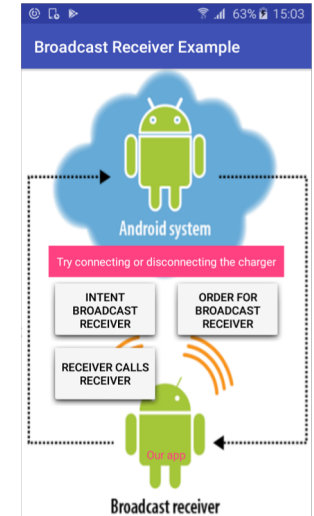
JSONObject alapok

```
JSONObject jsonObject = new JSONObject(response);  
String nameOfCountry = (String) jsonObject.get("country");  
long population = (Long) jsonObject.get("population");  
JSONArray listOfCities = (JSONArray) jsonObject.get("cities");  
  
Iterator<String> iterator = listOfCities.iterator();  
while (iterator.hasNext()) {  
    // do something  
}
```



Attekintés

- ▶ Háttér folyamatok
 - AsyncTask és AsyncTaskLoader
 - Internet kapcsolat



A következő előadás tartalma

- ▶ Üzenetszórás
- ▶ Szolgáltatások
- ▶ Értesítések
- ▶ Riasztások

