2. Az alkalmazások szerkezete

A mobil eszközök sajátosságai okán az Android platformra írt alkalmazás eltéren mködik, mint egy asztali operációs rendszerre írt alkalmazás. Ennek oka elssorban a rendelkezésre álló energia szkössége, de a kijelz kis mérete sem teszi lehetvé, hogy alkalmazásaink ablakát egymás mellé tegyük. Sok mobil platform egyszeren oldja meg ezt a problémát: egyszerre csak egy alkalmazás futását teszik lehetvé.

2.1. Android alkalmazások

Az Android platformon is találunk bven olyan korlátozásokat, amelyek a platform jellegébl adódnak, ne gondoljuk, hogy egy PC platformra megírt Java alkalmazás futni fog Android alatt! Ám az egyik legnagyobb korlátozás mindössze az, hogy a képernyt mindig teljes egészében elfoglalja az alkalmazásunk, leszámítva a mobil eszköz státusz sorát, illetve az alkalmazás "ablakának" fejlécét (természetesen teljes képernys módban ezek is elrejthetek).

A mobil eszközök a többfeladatos mködést se szokták támogatni, s ez Android esetén is trükkösen van megoldva. Egy véletlenül háttérbe taszított alkalmazás jelentsen csökkentheti a mobil eszköz rendelkezésre állását, ezért alapból a programok csak akkor futnak, ha eltérben vannak. Amint háttérbe teszünk egy alkalmazást, az operációs rendszer felfüggeszti a program futását, de a programozó által szolgáltatásként elindított szálat futni hagyja. Ezen túl a platform lehetvé teszi, hogy különféle eseményekre a program elinduljon és reagáljon az eseményre.

Nézzük az alkalmazások - szövegesen már említett - részeit az Android nyelvezetében:

- Activity - a program egy "ablaka"
- Intent - esemény
- Service - háttérben futó szolgáltatás
- Content Provider - adatok kezelése

2.1.1. Az Activity

Az *Activity* osztályok az alkalmazásaink legfontosabb részét adják: egy-egy Activity leszármazott egy-egy képerny a mobil eszköz kijelzjén. Egyszerre mindig csak egy Activity látható, de egy alkalmazáshoz több képernykép tartozhat, amelyeket futás közben - események hatására - szabadon cserélhetünk. Minden programnak kell legyen egy belépési pontja, amely az az Activity leszármazott, amelyet elször mutat meg a felhasználónak.

2.1.2. Az Intent

Egy *Intent* feladata a küls események kezelése. Minden alkalmazás fel tud iratkozni egy-egy - akár a platform által nyújtott, akár magunk által definiált - küls eseményre, mint a bejöv hívás vagy egy idpont bekövetkezte. Le kell származtatnunk egy saját példányt a BroadcastReceiver osztályból, és meg kell mondanunk, hogy milyen eseményre figyeljen. Ezt megtehetjük az *AndroidManifest.xml* állományban, vagy akár a program futása közben is. Ha az alkalmazásunk nem fut, a figyelt események bekövetkeztekor a platform gondoskodik az elindításról.

2.1.3. A Service

Sok alkalmazás igényli, hogy bezárt ablakkal is képes legyen futni a háttérben, ezt szolgáltatásként megteheti, egyszeren kell egy Service osztályból leszármazott példány, így marad a programunknak olyan része, amely a felfüggesztett Activity esetén is fut (gondoljunk például egy média lejátszóra). Minden szolgáltatást addig futtat a platform, amíg azok be nem fejezdnek, s a futó alkalmazások képesek hozzákapcsolódni a szolgáltatásokhoz, így képesek vagyunk vezérelni a háttérben futó szolgáltatást.

2.1.4. A Content Provider

Általában minden alkalmazás tárol adatokat két futás között, hiszen ha felfüggesztés helyett bezárnánk az alkalmazást, akkor elvesznének az addig összegyjtött adatok. A platformon lehetségünk van állományokba vagy SQLite adatbázisba menteni adatokat, és ezt segíti a Content Provider, illetve lehetvé teszi, hogy két alkalmazás adatokat cseréljen egymással.

2.1.5. Alkalmazások támogatása

Az eddig tárgyalt részek elegendek ahhoz, hogy programjainknak legyen felülete, képesek legyenek eseményeket kezelni, illetve adatokat menteni és a háttérben futni. Szükség van ezen túl olyan osztály-könyvtárakra, amelyek támogatást adnak a hardver kezeléséhez:

- Storage - a platform által biztosított fájlrendszer, illetve a küls tárhelyek (SD kártya, stb.) kezelése
- Network - a hálózat kezelése, WebKit alapú böngész komponens
- Multimedia - hangok és videók lejátszása, a hardver által biztosított multimédia lehetségek kihasználása
- GPS - A GPS vev használata
- Phone - A telefon használata

2.2. A forrás szerkezete

Egy Android projekt nagyjából úgy épül fel, mint egy átlagos Java projekt, a Java osztályok a megszokott *java* kiterjesztés fájlokban vannak, a megszokott csomagstruktúrában, s a megszokott módon kell fordítani ezeket a forrásfájlokat. Ha jobban körülnéztünk, akkor látunk egy *AndroidManifest.xml* állományt, amely a projekt **lelke**, itt kell leírni mindent, ami az alkalmazásunkkal kapcsolatos. A példánkban így néz ki:

AndroidManifest.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="hu.javaforum.android">
    <application>
         <activity android:name=".HelloActivity" android:label="HelloActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

A fenti példa az a minimum, amelynek minden alkalmazásban szerepelnie kell, egyszeren definiáljuk azt az Activity osztályt, amely a programunk belépési pontja, illetve azokat a feltételeket, amelyek aktivizálják: jelen esetben nem figyelünk eseményeket, a program indítását a felhasználóra bízzuk.

A Java források mellett találunk egy (általában) *res* nev könyvtárat, amely a programunkhoz csomagolt erforrásokat tartalmazza. Ilyen erforrások a megjelen ikonok, illetve sok egyéb XML fájl, amelyek például leírják a grafikus felületet. Minden olyan dolgot ide kell tennünk, amely nem Java program. Az els példánkban kell legyen itt egy *strings.xml* a *values* könyvtár alatt, amelyben a használt szövegek vannak összegyjtve:

values/strings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <string name="app_name">HelloJavaForum</string>
</resources>
```

A másik fontos XML a *layout* mappában lév *main.xml*, amely az elrendezést hordozza:

layout/main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">"
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello Android"/>
</LinearLayout>
```

Ezt azonban egy laza mozdulattal felülírtuk, amikor a *HelloActivity* nev osztályban közvetlenül adtuk hozzá a *TextView* példányt az Activity területéhez, ezért errl majd a késbbiekben szót ejtünk.

Mint már tudjuk, az Android tervezésekor sok apróságot feláldoztak a gyorsaság és az egyszerség oltárán, egy ilyen "apróság" a memória kezelés és az erforrások elérése, a platform egy *R.java* forrásban gyjti össze ez összes erforrás elérhetségét (ezt automatikusan generálja, nem kell szerkesztenünk!):

R.java

```
package hu.javaforum.android;
public final class R {
    public static final class attr {
    }
    public static final class layout {
        public static final int main=0x7f020000;
    }
    public static final class string {
        public static final int app_name=0x7f030000;
    }
}
```

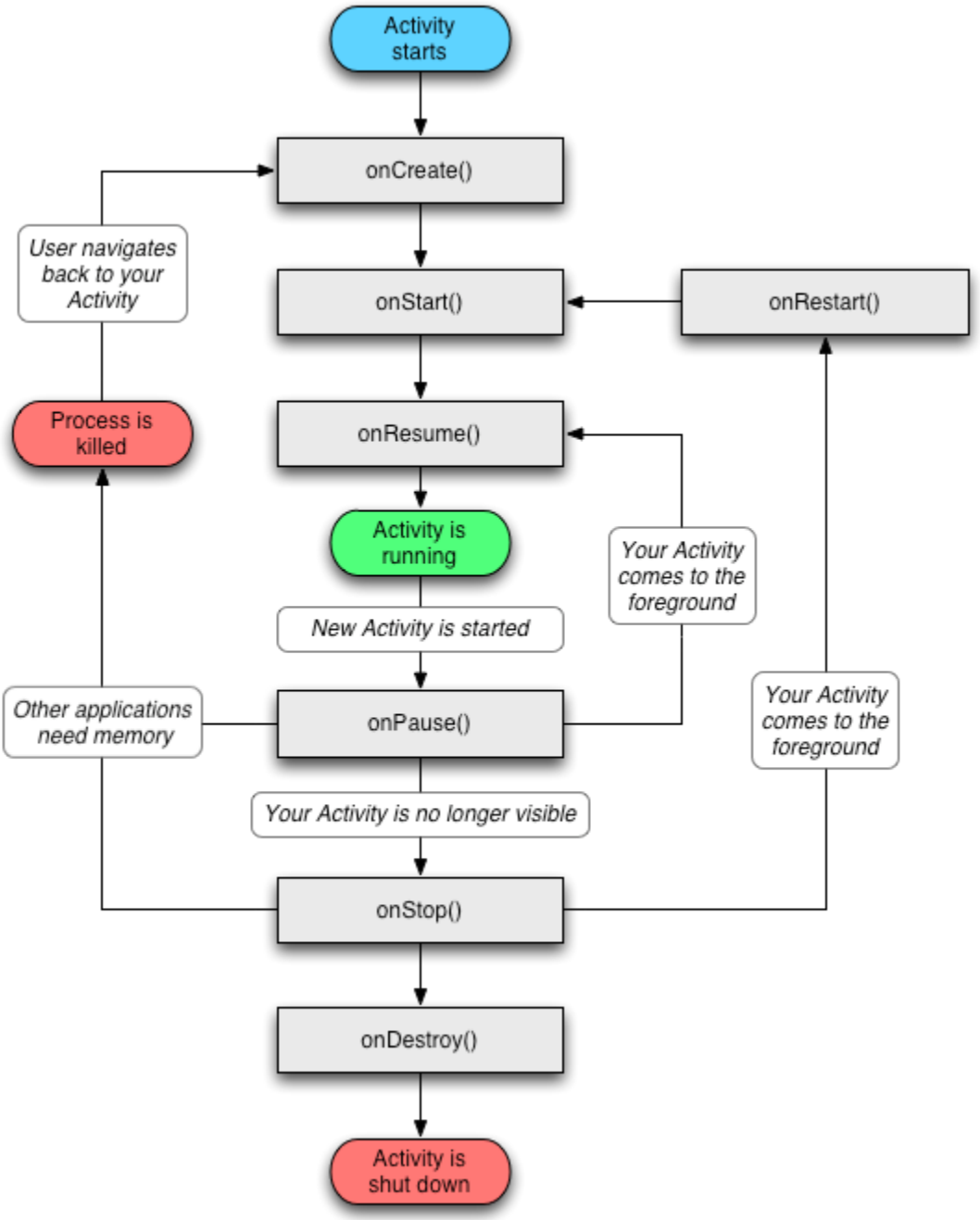
Ezt a Java fájlt használhatjuk arra, hogy az ikonokat, képeket, elrendezéseket vagy szövegeket nem a programba írunk közvetlenül, hanem felhasználjuk az R osztályt, amely mutat az adott erforrásra. Sok esetben találkozunk azzal, hogy nem a Java nyelvben megszokott módon adunk át egy példányra mutató referenciát, hanem magát a referenciát adjuk át, mint memóriacímet.

Ha megtekintjük újfent a *HelloActivity* forrását, akkor láthatjuk, hogy mindössze egy metódust írtunk meg, vagyis definiáltunk felül:

HelloActivity.java

```
package hu.javaforum.android;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class HelloActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView textView = new TextView(this);
        textView.setText("Hello, JavaForum.hu!");
        setContentView(textView);
    }
}
```

Az *onCreate* metódus egy a sok metódus közül, amelyet a platform hív meg: minden Activity példánynak van saját életciklusa és egymástól függetlenül élnek, ennek megértéséhez a legjobb egy folyamatábra:



Kövessük végig az ábrán egy Activity életének állomásait, amelyeket az életciklus különféle állapotai és eseményei hívnak meg:

- **onCreate**: Ez a metódus egyszer hívódik meg, amikor az alkalmazás létrejön. Általában ezen belül hozzuk létre a felhasználói felületet, amelyrl a késbbiekben lesz szó.
- **onStart**: A létrehozott alkalmazásban többször is meghívódhat az *onStart* metódus. Els alkalommal a létrehozás után hívódik meg, a többi alkalommal pedig az *onRestart* metódus után közvetlenül.
- **onResume**: A metódus meghívása közvetlenül az Activity képernyre kerülése eltt történik, s három irányból juthatunk ide:
  - Az alkalmazásunk most jött létre, az *onStart* metódust az *onCreate* elzte meg.
  - Az alkalmazásunkon belül váltottunk vissza egy már létrehozott Activity képernyre, ekkor az elz állapot az *onPause* volt.
  - Az alkalmazásunkat egy másik alkalmazás a háttérbe szorította, ezért az *onStart* metódust egy *onRestart* állapot elzte meg.
- **onPause**: A metódus meghívása eltt létrehoztunk az alkalmazásunkon belül egy új Activity képernyt, és mieltt annak meghívódna az *onCreate* metódusa, az aktuális Activity *onPause* metódusa hívódik meg. Ebbl az állapotból akkor kerülhet újra *onResume* állapotba, ha a meghívott új Activity futása befejezdött.
- **onStop**: Ha egy másik alkalmazás kerül az eltérbe, akkor meghívódik az alkalmazásunk *onStop* metódusa, így értesülünk arról, hogy az alkalmazásunkat háttérbe szorították. Ha újra eltérbe kerül az alkalmazásunk, akkor ezen metódus után hívódik meg az *onResume*.
- **onRestart**: Az *onStop* hívása után kerül meghívásra, ha a háttérbe tett alkalmazásunkat újra eltérbe hívták a körülmények.
- **onDestroy**: Ha bezárják az alkalmazásunkat, akkor meghívódik ez a metódus, amely lefutása után a platform megszünteti az alkalmazást. Ebbe az állapotba csak az *onStop* metódus meghívása után kerülhetünk, de fel kell készülnünk arra, hogy ha elfogy a szabad memória, akkor az operációs rendszer szó nélkül kilheti a háttérbe szorított alkalmazásunkat.

A fenti hét metódus bármelyikét felül tudjuk definiálni az Activity leszármazottban, egyedül az *onCreate* metódusnak van egy Bundle paramétere, amely segítségével az alkalmazásunk ki tudja olvasni az elz futás végén elmentett paramétereket.

Egyelre elégedjünk meg ennyivel, és nézzük meg jobban a felhasználók által látható felületet.