3. Egyszer komponensek

Egy-egy Activity felületére tehetünk egy-egy View-t, amelyre a késbbiekben komponensként hivatkozunk, hiszen az Android felületén a View olyan, mint Swing esetén a JComponent. Egy Activity felületére egy idben csak egy View-t tehetünk, amely az esetek nagy részében ViewGroup, amelybe további View-okat tehetünk, így alakíthatjuk ki az AWT/Swing esetén már megszokott komponensfát.

3.1. A View általánosságban

Mint említettem, a View nagyjából azt tudja, amit a JComponens Swing alatt: ez az osztály minden más komponens se. Az leszármazott Activity példányunk szoros kapcsolatban van egy View osztály példányával, mégpedig a *setContentView* metóduson át:

HelloActivity.java

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

Ha nem adunk meg semmi különleges paramétert, akkor az átadott View pontosan kitölti az Activity által nyújtott területet. Saját komponenseket a View leszármaztatásával tudunk készíteni.

3.2. Komponensek

A felhasználó által látható View-okat nevezzük komponenseknek, ezek közé tartoznak a feliratok, a gombok, a szövegbeviteli mezk és a többi hasonló komponens. Minden komponens a *View* osztályból származik, amelyrl a késbbiekben bvebben is szót ejtek.

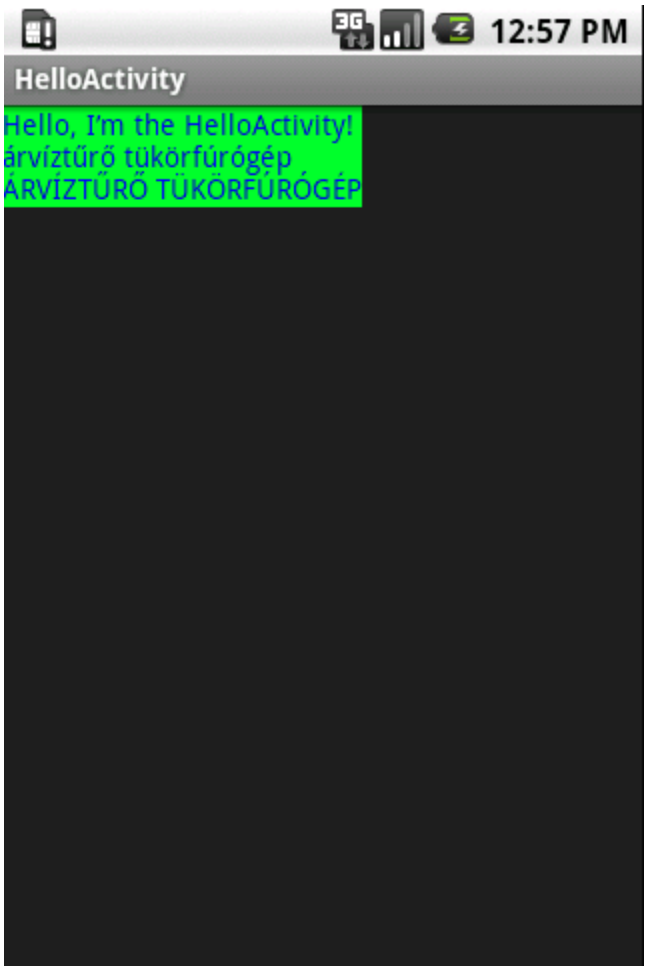
3.2.1. TextView

A TextView komponens feladata egyszer: a számára átadott szöveget írja ki címkék vagy többsoros szövegblokk formájában. Egy szöveget lehet neki átadni - például *StringBuilder* használatával, ugyanis a legtöbb *megszokott*Java API hívás a rendelkezésünkre áll. A programban szerepl LinearLayout-ról kicsit késbb szót ejtünk, egyelre elég annyi, hogy alapesetben a komponenseket egymás mellé pakolja:

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

TextView textView = new TextView(this);
StringBuilder sb = new StringBuilder();
sb.append("Hello, I'm the ");
sb.append(this.getTitle());
sb.append("!\n");
sb.append("árvíztr tükörfúrógép");
sb.append("\n");
sb.append("ÁRVÍZTR TÜKÖRFÚRÓGÉP");
textView.setText(sb.toString());
textView.setTextColor(Color.BLUE);
textView.setBackgroundColor(Color.GREEN);
layout.addView(textView);

setContentView(layout);
```

A program kimenete a mobil képernyjén:



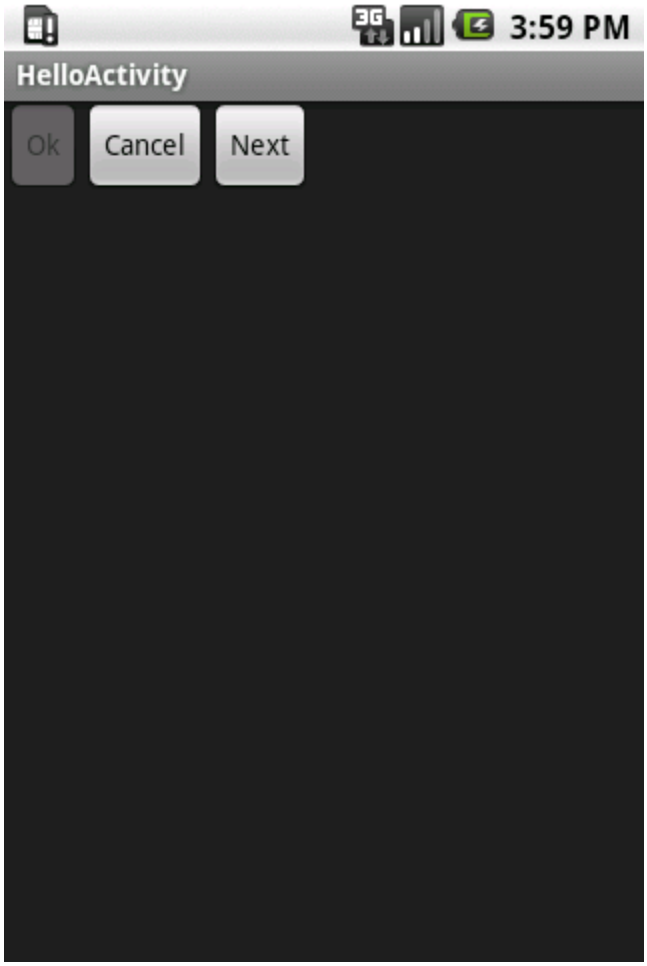
3.2.2. Button

A feliratok után a nyomógombok a legtöbbet használt, s egyben a legsibb felhasználói interfészek. Hozzunk létre három nyomógombot, rendre *Ok*, *Cancel* és *Next* feliratokkal:

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

Button okButton = new Button(this);
okButton.setText("Ok");
okButton.setEnabled(false);
layout.addView(okButton);

Button cancelButton = new Button(this);
cancelButton.setText("Cancel");
layout.addView(cancelButton);

Button nextButton = new Button(this);
nextButton.setText("Next");
layout.addView(nextButton);

setContentView(layout);
```

Az *Ok* gomb le lesz tiltva, tehát nem tudjuk megnyomni. Nézzük ezt a mobil kijelzjén:



3.2.3. ToggleButton

A kapcsolható gomb (kapcsológomb) úgy mködik, hogy állapotát megrzi a lenyomások során, vagyis felváltva lenyomva vagy felengedve marad.

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

ToggleButton releasedButton = new ToggleButton(this);
layout.addView(releasedButton);

ToggleButton pressedButton = new ToggleButton(this);
pressedButton.setTextOn("Incoming");
pressedButton.setTextOff("Incoming");
pressedButton.setChecked(true);
layout.addView(pressedButton);

setContentView(layout);
```

Ha ennek a gombtípusnak nem adunk át szöveget, akkor az *ON* vagy az *OFF* szöveget látja a felhasználó, természetesen a gomb állapotának megfelelen. A komponensnek át tudunk adni saját szöveget is a *textOn* és a *textOff*metódussal, ez hasznos, ha szrként szeretnénk használni a kapcsológombunkat:

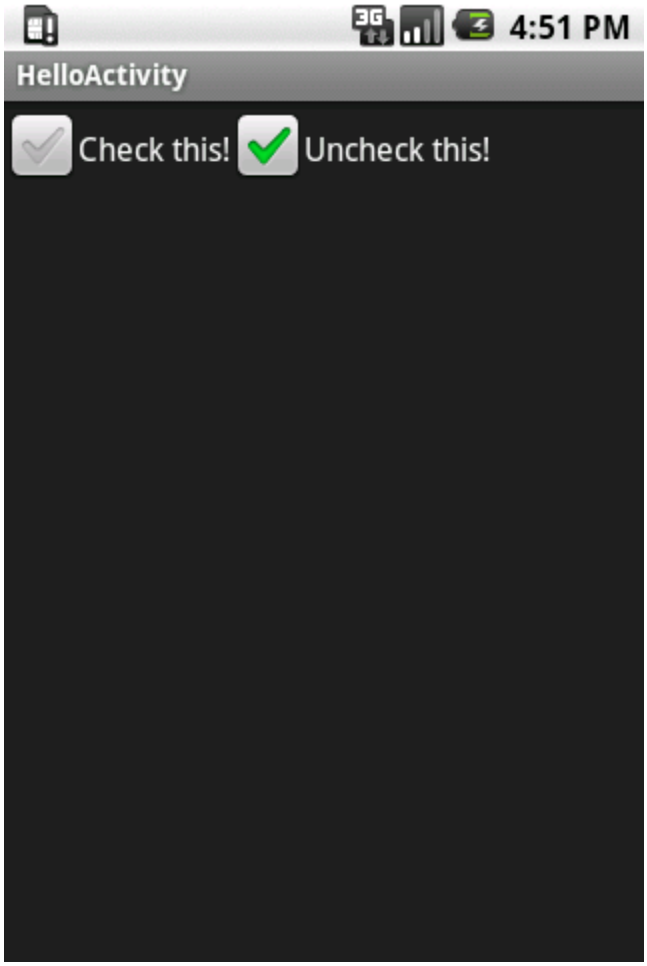
3.2.4. CheckBox

A jelölnégyzet hasonló célokat szolgál, mint a kapcsológomb, de más a megjelenése. Általában ha rövid szöveggel jellemezhet a kapcsolandó dolog, akkor kapcsológombot használunk; ha hosszabb szöveggel kell magyaráznunk, akkor pedig jelölnégyzetet. Ebbl adódik az a különbség is, hogy a jelölnégyzetnek adnunk kell szöveget, amely a ki- vagy bekapcsolható dolgot magyarázza meg; kapcsológomb esetén egy külön TextView kell a magyarázathoz.

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);
CheckBox uncheckedBox = new CheckBox(this);
uncheckedBox.setText("Check this!");
layout.addView(uncheckedBox);

CheckBox checkedBox = new CheckBox(this);
checkedBox.setText("Uncheck this!");
checkedBox.setChecked(true);
layout.addView(checkedBox);

setContentView(layout);
```

Nézzük meg a mobil kijelzjén:



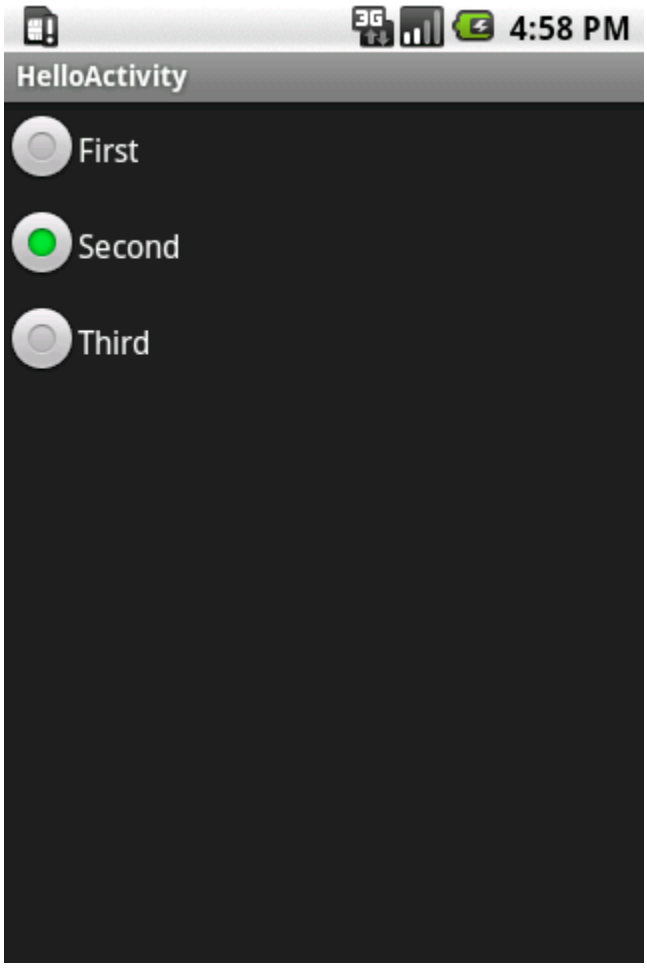
3.2.5. RadioButton

A rádiógombok a régi rádiókon lév gombokról kapták a nevüket, ugyanis csak egy lehet egy idben lenyomva közülük, amint egy újabbat nyomunk le, az elzleg lenyomott gomb jelölése megsznik. Akkor célszer használni ket, ha a felhasználónak választania kell több dolog közül, de csak egy opciót választhat. Ebbl következen a rádiógombok csak csoportban érzik jól magukat, a csoportot egy *RadioGroup* fogja össze:

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

RadioGroup group = new RadioGroup(this);

RadioButton firstButton = new RadioButton(this);
firstButton.setText("First");
group.addView(firstButton);

RadioButton secondButton = new RadioButton(this);
secondButton.setText("Second");
group.addView(secondButton);
secondButton.setChecked(true);

RadioButton thirdButton = new RadioButton(this);
thirdButton.setText("Third");
group.addView(thirdButton);

layout.addView(group);

setContentView(layout);
```

Mint látjuk, a csoporthoz kell hozzáadni a rádiógombokat, majd a csoportot rendeljük hozzá ahhoz a konténerhez, amibe szánjuk a gombokat. Általában célszer a felhasználónak egy alapértelmezett lehetséget adni, de ügyeljünk rá, hogy a csoporthoz való hozzáadás után állítsuk be az alapértelmezetten bejelölt gombot. Nézzük a kijelzn:



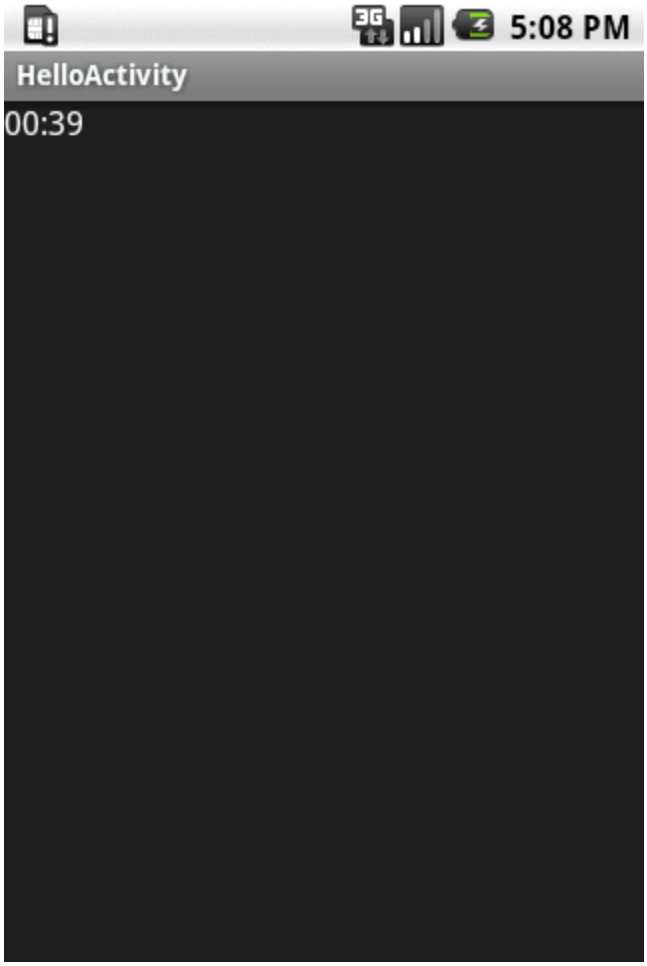
3.2.6. Chronometer

Ha szükségünk lenne eltelt idt mérni, akkor a *Chronometer* egy példányára lesz szükségünk, az alábbi esetben az alkalmazás indítása óta eltelt idt tudjuk kiírni (feltéve, ha az alábbi kódrészletet az *onCreate* metódusba írjuk):

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

Chronometer chronometer = new Chronometer(this);
chronometer.start();

layout.addView(chronometer);

setContentView(layout);
```

A kijelzn egyszeren egy perc:másodpercszámláló jelenik meg, és kezd számolni:



3.2.7. AnalogClock és DigitalClock

Ha ingerünk támadna analóg vagy digitális órát használni különösebb munka nélkül, akkor használhatjuk a kész komponenseket:

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

AnalogClock analogClock = new AnalogClock(this);
layout.addView(analogClock);

DigitalClock digitalClock = new DigitalClock(this);
layout.addView(digitalClock);

setContentView(layout);
```

S mindez a kijelzn:

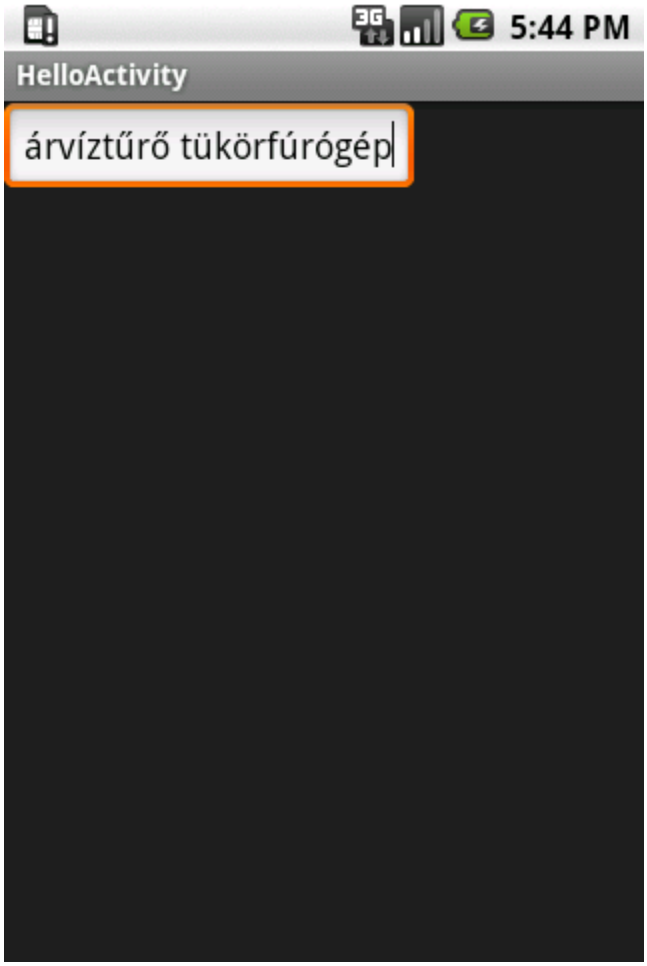
3.2.8. EditText

Felhasználó által gépelt szöveg beviteléhez használhatjuk az *EditText* komponenst, amely képes többsoros bevitelt is kezelni. Ha nem adunk meg méretet, görgetsávot vagy különösebb elrendezést, akkor a benne lév szöveg határozza meg a méretét.

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

EditText editText = new EditText(this);
editText.setText("árvíztr tükörfúrógép");
layout.addView(editText);

setContentView(layout);
```

S mindez a képernyn:



3.2.9. ImageView

Képek megjelenítéséhez az *ImageView* komponenst használhatjuk, egyszeren át kell neki adni a megjelenítend kép referenciáját... ám ilyet még nem csináltunk. A feladat nem nehéz, a *res* könyvtárban létre kell hoznunk egy *drawable* mappát és abba tehetjük a képeket, a platform pedig gondoskodik arról, hogy a következ fordításkor aktualizálja az *R.java* állományt, amelybe belekerül a kép is, a változó neve a kép nevét veszi fel kiterjesztés nélkül:

R.java

```
package hu.javaforum.android;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int logo=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
}
```

Nem kell mást tennünk, mint meghatározni a kép címét:

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

ImageView imageView = new ImageView(this);
imageView.setImageResource(R.drawable.logo);
layout.addView(imageView);

setContentView(layout);
```

A kijelzn pont akkora méretben jelenik meg, amekkora a kép tényleges felbontása:

3.2.10. ImageButton

Az *ImageView* és a nyomógomb keveréke, amely lehetvé teszi, hogy olyan nyomógombot tegyünk ki a kijelzre, amelyben szöveg helyett egy kép van:

Java

```
super.onCreate(savedInstanceState);

LinearLayout layout = new LinearLayout(this);

ImageButton imageButton = new ImageButton(this);
imageButton.setImageResource(R.drawable.logo);
layout.addView(imageButton);

setContentView(layout);
```

A kijelzn a keret mutatja mindössze, hogy gombról van szó:

3.2.11. Egyéb komponensek

A részletezett komponenseken kívül találhatunk még ProgressBar-t, SeekBar-t, *autocomplete* beviteli mezt, és egyéb apróságot, érdemes konzultálni a részletes referencia leírással ez ügyben, mivel a komponensek száma minden egyes kiadott verzióval növekszik... 🙂

3.3. Mindez XML alapokon

A *res* mappában a *layout* alatt található XML fájlokkal eddig nem foglalkoztunk, most vessünk az ott figyel *main.xml* állományra egy hosszabb pillantást:

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 from NetBeans"/>
</LinearLayout>
```

Ezt a tartalmat a NetBeans plugin generálta a projekt létrehozásakor, és a projekt fordításakor ebbl egy Activity által használható elrendezést fog generálni, amely belekerül az R.java fájlba is:

R.java

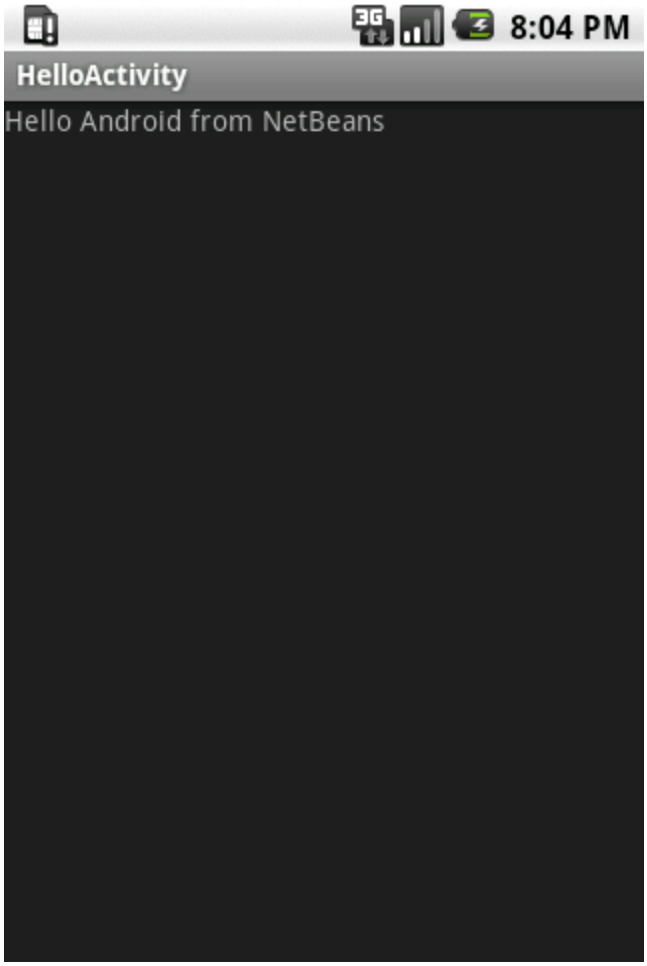
```
public static final class layout {
    public static final int main=0x7f030000;
}
```

S erre az erforrásra a programunkban tudunk hivatkozni, mint View:

HelloActivity.java

```
package hu.javaforum.android;

import android.app.Activity;
import android.os.Bundle;

public class HelloActivity extends Activity
{

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

A programot futtatva igen ers összefüggést kell látnunk a *main.xml* és a képerny között:



A *res* mappa alatti elemekre is tudunk hivatkozni az XML állományból, ehhez a szöveg értékét @ jellel kell kezdenünk és az *R.java* állomány neveit kell használnunk. Módosítsuk a *strings.xml* állományt az alábbiak szerint:

strings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <string name="app_name">HelloJavaForum</string>
    <string name="hello">Hello, I\'m the HelloJavaForum!</string>
</resources>
```

Majd fordítsuk le a projektet, hogy a környezet legenerálja az újabb *R.java*forrást:

R.java

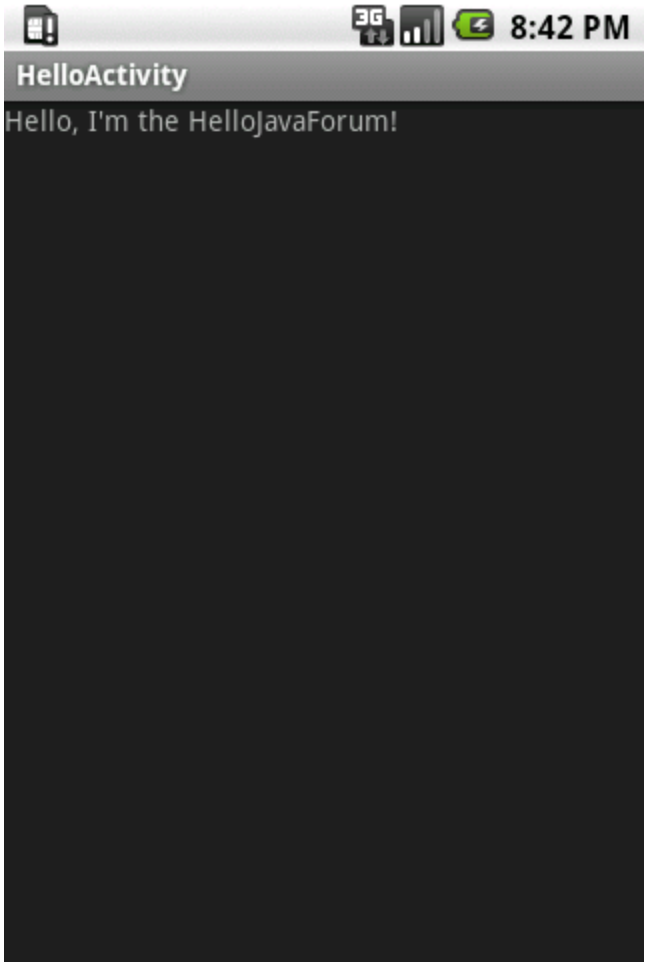
```
package hu.javaforum.android;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int logo=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello=0x7f040001;
    }
}
```

Ebben már szerepel a korábban hozzáadott *logo* nev kép hivatkozása, illetve a most hozzáadott *hello* nev szöveg. Módosítsuk a *main.xml* állományt:

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="@string/hello"/>
</LinearLayout>
```

A kulcsszó itt a **@string/hello** szöveg, amely az *R.java*állomány megfelel sorára mutat, s egyéb erforrásokat is ilyen egyszerséggel tudunk meghivatkozni. Futtassuk újra a programot, s nézzük meg az eredményt:



Felmerülhet a kérdés, hogy az XML fájlban leírt komponenseket miképp tudjuk elérni a programból, hiszen az R.java forrásában nem szerepelnek. Ennek oka, hogy csak azok az erforrások kerülnek bele az R.java leíróba, amelyeknek adunk azonosítót, vagyis kitöltjük az *android:id* attribútum értékét:

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"
    android:id="@+main/layout">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:id="@+main/textView"/>
</LinearLayout>
```

Az *id* mezre annyi megkötés van, hogy a **@+** után kell következzen két név **'/'** jellel elválasztva. A **'/'** jel két oldalán lév szövegbl megszokott Java változók lesznek, amelyet meg is tekinthetünk az újra legenerált *R.java* állományban:

R.java

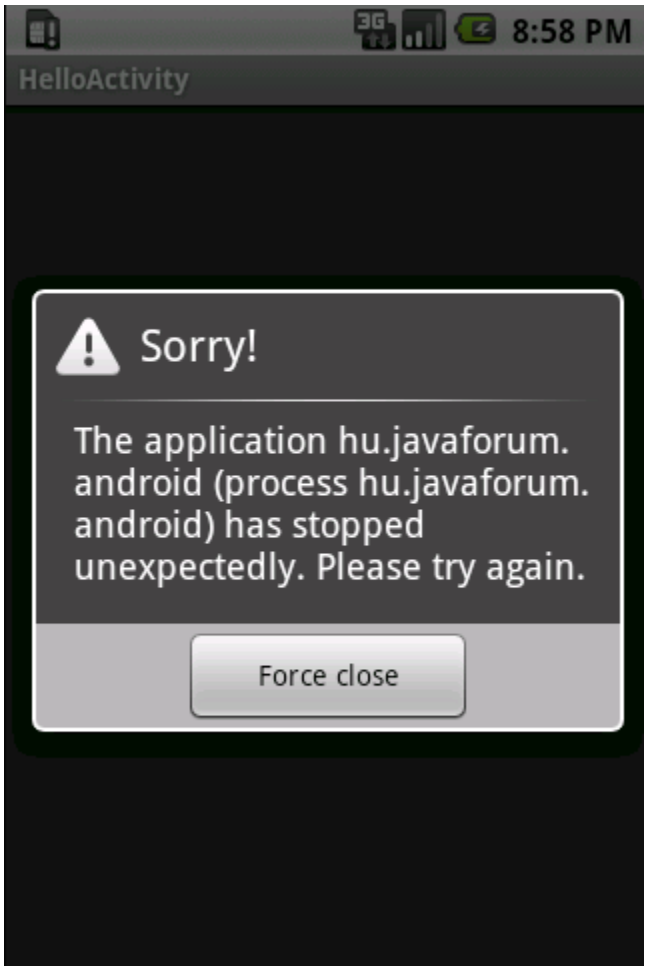
```
package hu.javaforum.android;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int logo=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class main {
        public static final int layout=0x7f050000;
        public static final int textView=0x7f050001;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello=0x7f040001;
    }
}
```

A változás egy *main* nev osztály, amelynek lett kett új mezje. Fontos tudnivaló, hogy az azonosítónak egyedinek kell lennie az alkalmazáson belül, mivel az *R.main.textView* néven férhetünk hozzá a *TextView* példányhoz, függetlenül attól, hogy melyik XML állományban szerepel az adott komponens. Az Activity példányban a *findViewById* metódussal tudjuk lekérdezni az adott erforrás azonosító alapján a komponenst:

Java

```
super.onCreate(savedInstanceState);

TextView view = (TextView)findViewById(R.main.textView);
view.setText("This is it!");

setContentView(R.layout.main);
```

A lekérdezésnél ügyelnünk kell arra, hogy a típuskonverzió helyesen történjen meg, ezek után már úgy tudjuk használni az adott erforrást, mintha mi hoztuk volna létre. Nézzük meg, mit is látunk:



Ez bizony nem szép látvány, a platform leállította a program futását. Ennek oka többnyire egy általunk le nem kezelt kivétel, amely így eljut az Android platformig, amely információ hiányában egy ilyen általános hibaüzenettel szórakoztatja a felhasználót. Nos, itt az ideje, hogy elkapjuk a felmerült kivételeket és valamilyen módon kiírjuk ket:

Java

```
super.onCreate(savedInstanceState);

try
{
  TextView view = (TextView) findViewById(R.main.textView);
  view.setText("This is it!");
  setContentView(R.layout.main);
} catch (Exception except)
{
  TextView textView = new TextView(this);
  final Writer result = new StringWriter();
  final PrintWriter printWriter = new PrintWriter(result);
  except.printStackTrace(printWriter);
  textView.setText(result.toString());
  setContentView(textView);
}
```

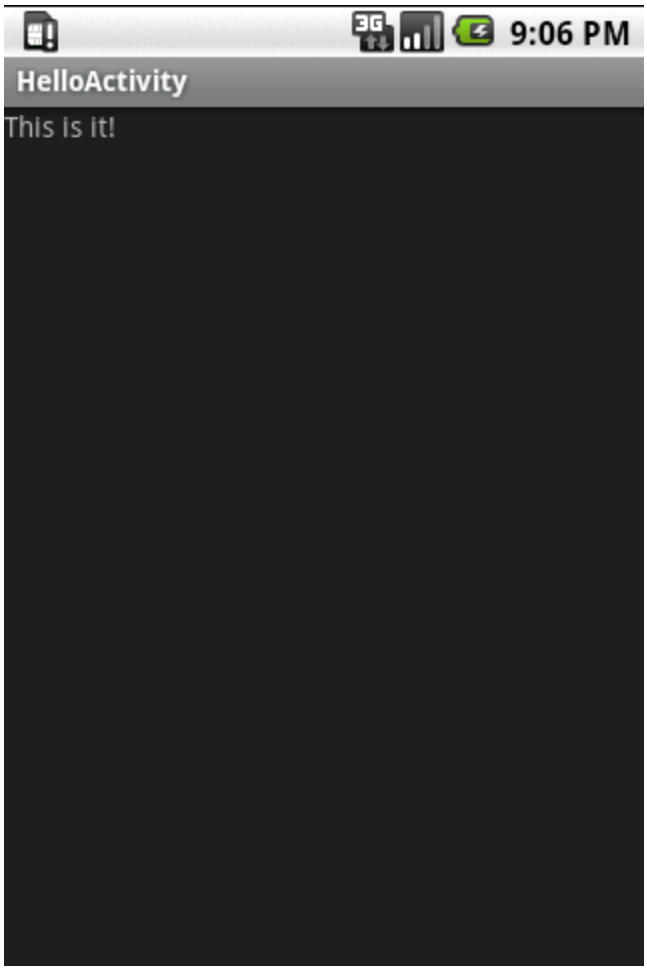
Egy megszokott *try-catch* blokk lehet a megoldás, amely hiba esetén létrehoz egy *TextView* komponenst, amelybe beleírja a kivételben hordozott *stacktrace* listát:

No igen, ez egy csúnya *NullPointerException*, amelyet az okozott, hogy elbb kérdeztük le az erforrások között létrehozott komponenst, mieltt az létrejött volna, ugyanis ezek a példányok a *setContentView* metódus hívásakor jönnek létre, javítsuk ki a programot, vagyis vegyük elre a *setContentView* hívást:

Java

```
super.onCreate(savedInstanceState);

try
{
  setContentView(R.layout.main);

  TextView view = (TextView) findViewById(R.main.textView);
  view.setText("This is it!");
} catch (Exception except)
{
  TextView textView = new TextView(this);
  final Writer result = new StringWriter();
  final PrintWriter printWriter = new PrintWriter(result);
  except.printStackTrace(printWriter);
  textView.setText(result.toString());
  setContentView(textView);
}
```

Nézzük most az eredményt:



Mint láthattuk, az XML leíró elnye, hogy tömörebb lesz tle a kód, és a fába szervezett komponensek sokkal jobban leírhatók XML-ben, hátránya pedig a fent látható aljas hibához hasonló problémák esélye, hiszen egy fejleszt környezet messzirl kiszúr egy *NullPointerException* gyanús helyzetet, ugyanez a *findViewById* esetén nem mködik.