5. Választómezk

Egy érintképernys eszköz felhasználója egy id után kellen utálja, ha virtuális billentyzetet kell használnia, ezért az Android platformra való fejlesztés során a lehetségeinkhez mérten inkább listát, legördül menüt illetve egyéb választási lehetséget tegyünk a képernyre, mint egy *EditText* komponenst, amelybe a felhasználó begépeli a szükséges adatot.

5.1. Az adapterek

Az Android platform nyelvezetében a listák és egyéb választómezk adatainak karbantartásáról az adapterek gondoskodnak, ahogy AWT/Swing esetén a modell gondoskodik errl a területrl. Az adapterek gondoskodnak arról, hogy a listák milyen tartalommal jelenjenek meg, gondoljunk csak egy konfigurációs listára, amelyben egy *CheckBox* komponens is van, így az adott elemet ki-be tudjuk kapcsolni.

5.1.1. ArrayAdapter

A legegyszerbb adapter az *ArrayAdapter*, amely egy tömböt fogad és tesz a platform számára használhatóvá. A választómezket használó *Activity* leglényegesebb része az *ArrayAdapter* létrehozása:

Java forrás

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1,
        newsTitles);
```

Az adapter három paramétere az *Activity* példányra való hivatkozás (**this**); a hozzá kapcsolódó *layout* típus, amely jelen esetben a **android.R.layout.simple_list_item_1**; illetve maga a tömb, amely az értékeket tartalmazza (**newsTitles**). A három paraméter közül magyarázatot a *layout* követel, amely az adapter számára azt határozza meg, hogy a tömb minden elemére hívja meg annak *toString* metódusát.

5.1.2. CursorAdapter

Ha a platform adta adatbázisból kérdezünk le, akkor használhatunk *CursorAdapter*-t is, amely kényelmesen lekezeli az eredmények görgetését.

5.1.3. SimpleAdapter

A *SimpleAdapter* XML adatokból építi fel a listát, így egy szerverrel kommunikálva egyszerbb lehet a listák felépítése.

5.2. A választómezk

A választómezk a platform grafikus felületének leglényegesebb komponensei, mivel ezek adják a felhasználóval való kapcsolat gyakran használt részeit. Szinte mindegyik választómez listát jelenít meg, így legtöbb esetben egy *ArrayAdapter* példányból származnak a megjelenített adatok.

5.2.1. A ListView egyszeren

A lista megjelenítéséhez le kellene rombolnunk mindazt, amit eddig építettünk, így inkább hozzunk létre egy új projektet, a neve legyen *TextList*, a csomagnév a példában *hu.javaforum.android.textlist* lesz, az *Activity* neve pedig *MainActivity*. A *layout* alatti *main.xml* állományba együk az alábbi tartalmat:

main.xml

```
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
             android:orientation="vertical"
             android:layout_height="fill_parent"
             android:layout_width="fill_parent">
  <TextView android:id="@+main/selection"
            android:background="#000044"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"/>
  <ListView android:id="@android:id/list"
            android:drawSelectorOnTop="false"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent"/>
</LinearLayout>
```

A képernyterv alapján annyit kell látnunk, hogy az *Activity* tetején lesz egy sötétkék hátter szövegmez, amely egyelre üres, illetve lesz egy *ListView*, amely a listát tartalmazza. Ezek után a *MainActivity.java*fájlban az alábbi programot kell megírnunk:

MainActivity.java

```
public class MainActivity extends ListActivity
{

  String[] newsTitles =
  {
    "Glassfish távlati tervek",
    "JUM - tizennegyedik alkalom",
    "Tomcat 6.0.24 memória lukak elleni védelemmel",
    "A kenai.com elesett",
    "JBoss HornetQ 2.0.0.GA",
    "EclipseLink 2.0.0",
    "JUM 13. 2010. január 20-án",
    "Java 7 újdonságok",
    "Tomcat 7?",
    "EclipseLink 1.2.0",
    "JRuby 1.4.0",
    "Google Collections",
    "Java 5 - béke poraira?"
  };

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    ArrayAdapter adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1,
            newsTitles);

    setListAdapter(adapter);
  }

  @Override
  public void onListItemClick(ListView parent, View v, int position, long id)
  {
    TextView view = (TextView) findViewById(R.main.selection);
    view.setText(newsTitles[position]);

    if (v instanceof TextView)
    {
      TextView item = (TextView)v;
      item.setBackgroundColor(Color.YELLOW);
      item.setTextColor(Color.BLUE);
    }
  }
}
```

Figyeljük meg, hogy az eddigi *Activity* helyett a *ListActivity* osztályból származtatjuk a képerny programját, és az eddig is használt *onCreate* alatt megjelenik egy *onListItemClick* metódus is, amelyet a platform akkor hív meg, ha kiválasztunk egy elemet a listából. Ne szaladjunk ennyire elre, nézzük inkább a program részeit. A *newsTitles* tömb tartalmazza a javaforum.hu portálon lév hírek címét, ezt szeretnénk a listában megjeleníteni, ezt nem részletezném tovább. Az adapter beállítása triviális lépés, ám könny elfeledkezni róla:

Java forrás

```
setListAdapter(adapter);
```

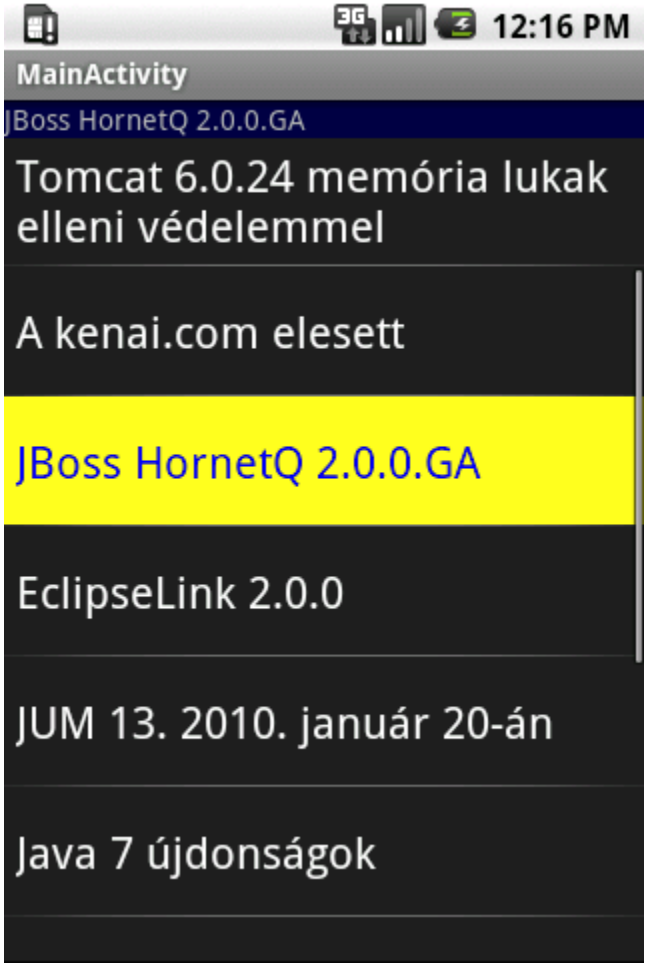
További figyelmet igényel az *onListItemClick* metódus, amelyet a platform hív meg, a lista egy elemének kiválasztásakor:

Java forrás

```
@Override
public void onListItemClick(ListView parent, View v, int position, long id)
{
  TextView view = (TextView) findViewById(R.main.selection);
  view.setText(newsTitles[position]);
  if (v instanceof TextView)
  {
    TextView item = (TextView)v;
    item.setBackgroundColor(Color.YELLOW);
    item.setTextColor(Color.BLUE);
  }
}
```

Ránézésre látszik, hogy semmi egyebet nem teszünk, mint a képerny tetején lév *TextView* komponensbe írjuk a kiválasztott elem szövegét, illetve egy *View* hátterét sárgára állítjuk. Kitalálható, hogy a *parent* az a *ListView*, amelyikhez az esemény érkezett, hiszen több listánk is lehet egy *Activity* felületen. A *v* nev *View* az a komponens, amelyet a listában kiválasztottunk, ennek a hátterét állítjuk sárgára, a szöveg színét pedig kékre. Ez utóbbinak sok értelme ugyan nincs, mivel a platform újrahasznosítja a komponenseket - tehát nem hoz létre egy ezer elem lista esetén ezer *TextView* komponenst, csak annyit, amennyi a képernyn egyszerre látszik. A *position* a kiválasztott elem sorszáma nullától kezdden, az *id* pedig a kiválasztott sor azonosítója.

Nézzük mködés közben az elkészült, lefordított és elindított alkalmazást:



5.2.2. A ListView cifrán

Az egyszer lista nem mindig felel meg az igényekhez, ugyanis az elemei egyszer *TextView* komponensek. Ha ennél több kell - és gyakran több kell, akkor saját igényeink szerint jeleníthetjük meg a lista elemeit, ehhez létre kell hoznunk egy olyan *layout* példány (akár Java, akár XML formátumban), amelyet a lista elemeként szeretnénk viszontlátni, ehhez hozzuk létre a *list_item.xml* nev fájlt a *layout* könyvtárban a következ tartalommal:

list_item.xml

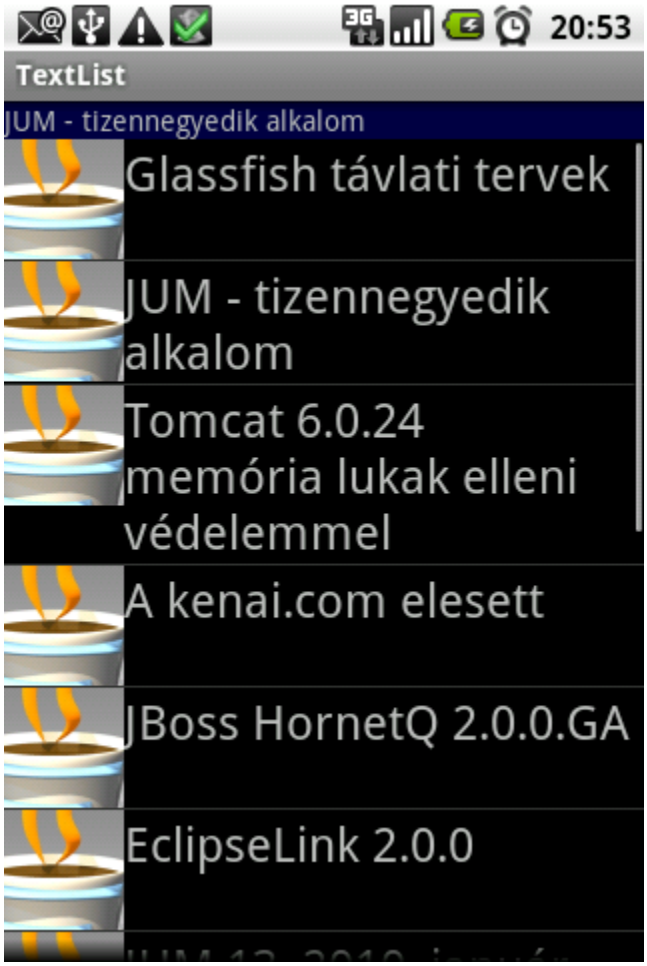
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="horizontal"
              android:layout_height="wrap_content"
              android:layout_width="fill_parent">
  <ImageView android:id="@+list_item/icon"
            android:layout_height="60px"
            android:layout_width="60px"
            android:src="@drawable/icon"/>
  <TextView android:id="@+list_item/label"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:textSize="24px"/>
</LinearLayout>
```

Ez a *layout* tartalmaz egy 60px x 60px méret képet (amelyet el kell helyeznünk a *resources* alatti *drawable* könyvtárban *icon.png* néven), illetve tartalmaz egy címkét, amely a szöveget hordozza. A programunkon minimális mértékben kell változtatni, ezért az egész *onCreate*metódust mutatom meg, két sor változott meg mindössze:

Java forrás

```
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);

  setContentView(R.layout.main);

  ArrayAdapter adapter = new ArrayAdapter<String>(this,
          R.layout.list_item,
          R.list_item.label,
          newsTitles);

  setListAdapter(adapter);
}
```

Az *ArrayAdapter* példány létrehozásánál megadtuk az elbbiekben létrehozott XML *layout*erforrás nevét, illetve a változtatandó címke erforrásának a nevét, a többi maradt a régi:



5.2.3. A ListView változó ikonokkal

Természetesen - az értelmetlen dekorációt kivéve - nincs sok haszna annak, ha az ikon nem változik, ezért gyrjük fel az ingünk ujját, kössük fel a gatyánkat és kezdésképp tegyünk a *drawable* könyvtárba három újabb ikont... :)

Nem kell megijedni, a probléma kulcsa az *Adapter* osztály körül található, ugyanis ez az osztály hozza létre és kezeli a listában megjelen komponenseket. Jó eséllyel ikon melletti szöveg jelleg listára máskor is szükségünk lehet, így hozzunk létre egy osztályt erre, amelynek adjuk az IconListArrayAdapter nevet:

IconListArrayAdapter

```
public class IconListArrayAdapter<T> extends ArrayAdapter<T>
{

  private Activity context;
  private int rowResource;
  private int labelResource;
  private int iconResource;
  private int[] icons;
  private T[] items;

  public IconListArrayAdapter(Activity context,
          int rowResource, int iconResource, int labelResource,
          int[] icons, T[] items)
  {
    super(context, rowResource, items);
    this.context = context;
    this.rowResource = rowResource;
    this.labelResource = labelResource;
    this.iconResource = iconResource;
    this.icons = icons;
    this.items = items;
  }

  @Override
  public View getView(int position, View reusableView, ViewGroup parent)
  {
    LayoutInflater inflater = context.getLayoutInflater();
    View row = inflater.inflate(this.rowResource, null);
    TextView label = (TextView) row.findViewById(this.labelResource);
    ImageView icon = (ImageView) row.findViewById(this.iconResource);

    label.setText((String) this.items[position]);
    icon.setImageResource(this.icons[position]);

    return row;
  }
}
```

Nem foghatjuk rá, hogy összetett osztály, hiszen a konstruktorában átadunk pár paramétert, majd az egyetlen metódusával kezeljük le a lista elemeinek létrehozását, nézzük meg ezen metódus els két sorát részletesebben:

Java forrás

```
LayoutInflater inflater = context.getLayoutInflater();
View row = inflater.inflate(this.rowResource, null);
```

A *LayoutInflater* osztály végzi el azt a munkát, amely során az XML leíróból egy *View* példány lesz, jelen esetben egy egész *View* hierarchia. Ha hibás az XML leíró, akkor kivételt kapunk, amelyet illik lekezelni, de itt most még nem tesszük meg ezt. A *row* nev változóba létrejött az elz fejezetbl örökölt *list_item.xml* fájlban lév *LinearLayout*, illetve a benne lév *ImageView* és *TextView* példány, ezeket erforrás azonosító alapján el tudjuk kérni:

Java forrás

```
TextView label = (TextView) row.findViewById(this.labelResource);
ImageView icon = (ImageView) row.findViewById(this.iconResource);
```

A *findViewById* metódus használható bármelyik *View* metódusaként, s ekkor a platform csak az adott *View* fájában keresi az adott azonosítójú erforrást. Ahogy a fenti két sorból olvasható: elkérjük a címke és az ikon példányát, majd a következ két sorban beállítjuk azok értékét:

Java forrás

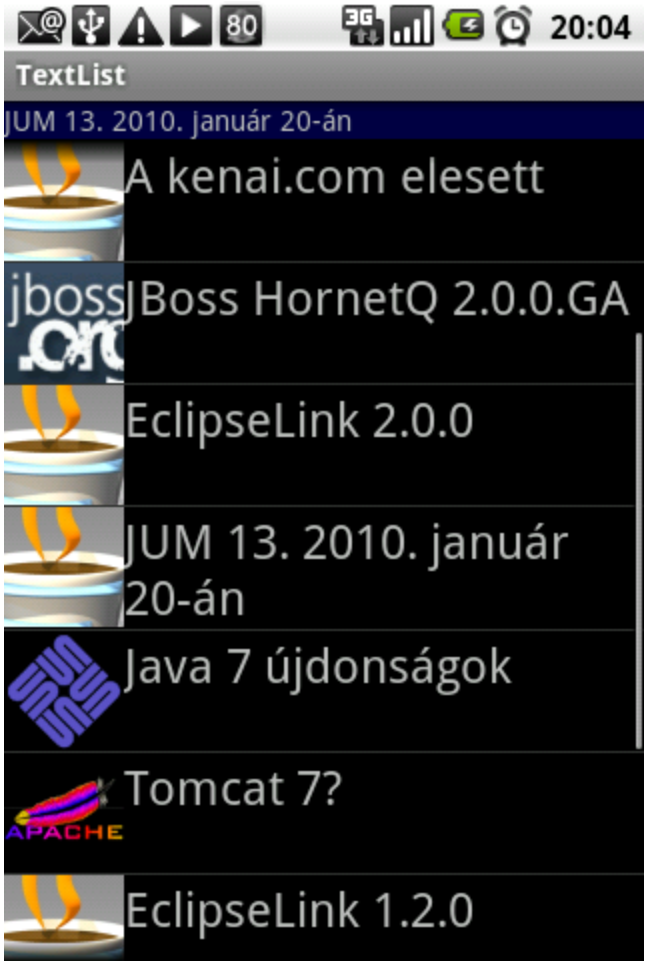
```
label.setText((String) this.items[position]);
icon.setImageResource(this.icons[position]);
```

A *getView* metódus *position* paramétere határozza meg, hogy a listában melyik elemrl van éppen szó, így az osztálynak átadott ikon és szöveg listát felhasználva beállítjuk az értékeket, majd visszaadjuk a létrehozott sor példányát. Az elz fejezetbl örökölt *MainActivity* programban létre kell hoznunk egy ikon listát a címek listája mellé, majd az *ArrayAdapter*létrehozását kell kicserélni, íme a teljes osztály forrása egyben:

MainActivity.java

```
public class MainActivity extends ListActivity
{

  String[] newsTitles =
  {
    "Glassfish távlati tervek",
    "JUM - tizennegyedik alkalom",
    "Tomcat 6.0.24 memória lukak elleni védelemmel",
    "A kenai.com elesett",
    "JBoss HornetQ 2.0.0.GA",
    "EclipseLink 2.0.0",
    "JUM 13. 2010. január 20-án",
    "Java 7 újdonságok",
    "Tomcat 7?",
    "EclipseLink 1.2.0",
    "JRuby 1.4.0",
    "Google Collections",
    "Java 5 - béke poraira?"
  };
  int[] icons =
  {
    R.drawable.icon,
    R.drawable.icon,
    R.drawable.apache,
    R.drawable.icon,
    R.drawable.jboss,
    R.drawable.icon,
    R.drawable.icon,
    R.drawable.sun,
    R.drawable.apache,
    R.drawable.icon,
    R.drawable.icon,
    R.drawable.icon,
    R.drawable.sun,
  };

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    ArrayAdapter adapter = new IconListArrayAdapter<String>(this,
            R.layout.list_item,
            R.list_item.icon,
            R.list_item.label,
            this.icons,
            this.newsTitles);
    setListAdapter(adapter);
  }

  @Override
  public void onListItemClick(ListView parent, View v, int position, long id)
  {
    TextView view = (TextView) findViewById(R.main.selection);
    view.setText(newsTitles[position]);

    if (v instanceof TextView)
    {
      TextView item = (TextView) v;
      item.setBackgroundColor(Color.YELLOW);
      item.setTextColor(Color.BLUE);
    }
  }
}
```

Az eredmény egy ikonokat tartalmazó lista, amelynél az ikonok utalnak a címekre:



5.2.4. A ListView változó ikonokkal - *okosan*

Ha jobban megnézzük, akkor az elz program minden egyes *getView* metódus hívással egy idben létrehoz egy új struktúrát, amely egy hosszú és sokat görgetett lista esetén pazarlás. Ha jópár alkalommal fel-le görgetjük a listát, s közben figyeljük az eszköz *logját*, akkor a rendelkezésre álló memória függvényében elbb-utóbb elkezd szemetet gyjteni:

logcat

```
D/dalvikvm(25792): GC freed 880 objects / 66352 bytes in 76ms
D/dalvikvm(25792): GC freed 351 objects / 19056 bytes in 70ms
D/dalvikvm(25792): GC freed 179 objects / 10096 bytes in 68ms
D/dalvikvm(25792): GC freed 123 objects / 8040 bytes in 71ms
```

Jegyezzük meg, hogy a GC szó a naplóban az **ördögtl való**, a lehet legtöbb esetben használjuk újra a létrehozott példányokat, próbáljuk minimalizálni a példányosításokat, a platform ugyanis sok esetben korlátozott erforrással bír, egy folyamatosan GC-t igényl - de háttérbe tett alkalmazás - hamar kiszívja az éltet energiát az akkumulátorokból. Gondolkodjunk el a következ metódus mködésén és a kapott paraméterek számán:

Java forrás

```
@Override
public View getView(int position, View reusableView, ViewGroup parent)
{
  LayoutInflater inflater = context.getLayoutInflater();
  View row = inflater.inflate(this.rowResource, null);
  TextView label = (TextView) row.findViewById(this.labelResource);
  ImageView icon = (ImageView) row.findViewById(this.iconResource);

  label.setText((String) this.items[position]);
  icon.setImageResource(this.icons[position]);

  return row;
}
```

Egyedül a *position* nev paramétert használtuk fel, a gyanúsan *reusableView* nev paramétert nem (a *parent* paraméterrl majd késbb még szót ejtek :). Rövid gondolkodás után rá kell jönnünk, hogy a platform az általunk visszaadott *View* típust újra felhasználja görgetésnél, csak a látható listaelemekhez hoz létre *View* példányt, hiszen a többi nem látható, a görgetés során pedig az "utolsó pár elre fuss" elv mentén megkapjuk az adott - már létez - listaelemhez tartozó *View* példányt. A trükk mindössze annyi, hogy ellenriznünk kell a kapott *reusableView* tartalmát, ha ez *null*, akkor a *LayoutInflater* létrehoz egy újat, ha nem *null*, akkor csak módosítani kell a tartalmát:

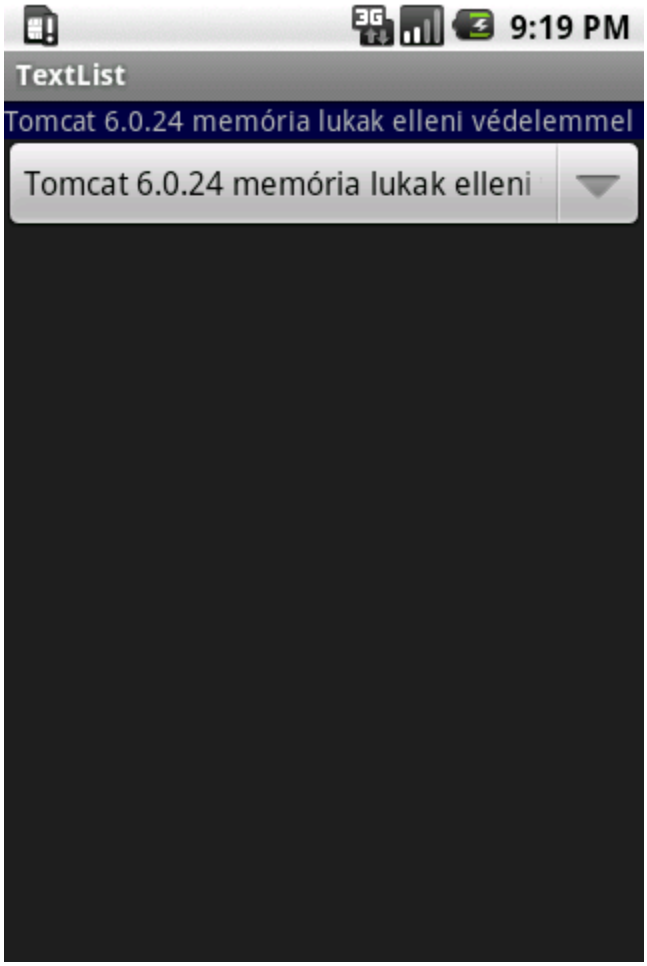
Java forrás

```
@Override
public View getView(int position, View reusableView, ViewGroup parent)
{
  if (reusableView == null)
  {
    LayoutInflater inflater = context.getLayoutInflater();
    reusableView = inflater.inflate(this.rowResource, null);
  }

  TextView label = (TextView) reusableView.findViewById(this.labelResource);
  ImageView icon = (ImageView) reusableView.findViewById(this.iconResource);

  label.setText((String) this.items[position]);
  icon.setImageResource(this.icons[position]);

  return reusableView;
}
```

Ezek után már nem lesz tele szeméttel a memória... :)

5.2.5. A Spinner

A *Spinner* hasonít az elz fejezetekben említett listára, ám egyszer komponensként el tudjuk helyezni egy átlagos *Activity* felületén, mivel a választás lebonyolításához egy felbukkanó ablakot fogunk látni - ahogy a JComboBox teszi ezt a Swing világában:



A képernytervnek megfelel XML tartalom:

main.xml

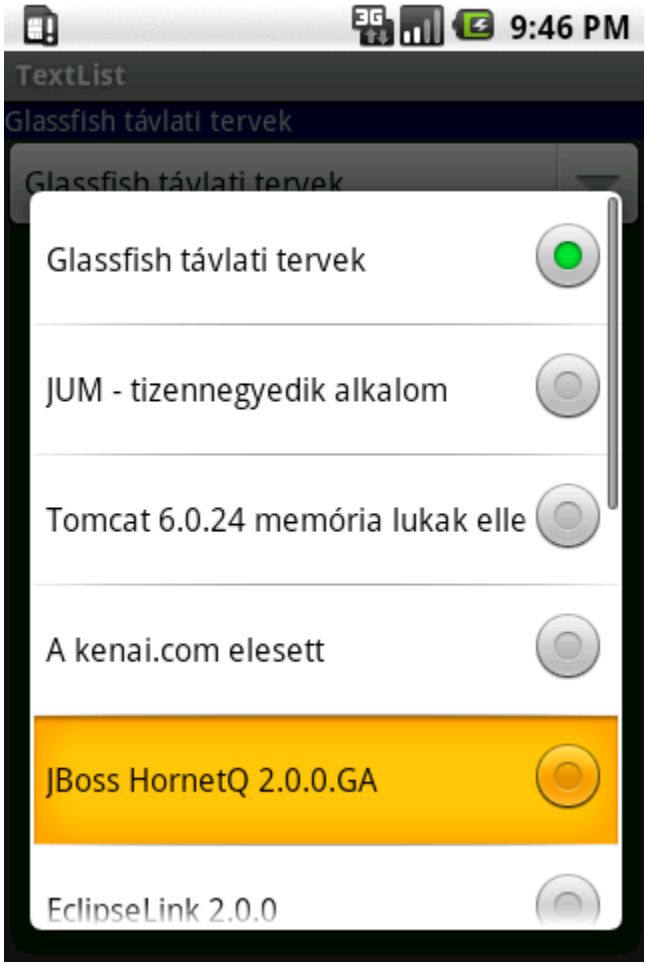
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_height="fill_parent"
              android:layout_width="fill_parent">
  <TextView android:id="@+main/selection"
            android:background="#000044"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"/>
  <Spinner android:id="@+main/spinner"
            android:drawSelectorOnTop="true"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"/>
</LinearLayout>
```

Mint látszik, a *ListView* helyett *Spinner* a komponens neve, s nem kell neki a speciális *android* névtér az *id* attribútum értékeként. Az elrendezéshez tartozó program is kissé más lett:

MainActivity.java

```
public class MainActivity extends Activity implements AdapterView.OnItemSelectedListener
{

  String[] newsTitles =
  {
    "Glassfish távlati tervek",
    "JUM - tizennegyedik alkalom",
    "Tomcat 6.0.24 memória lukak elleni védelemmel",
    "A kenai.com elesett",
    "JBoss HornetQ 2.0.0.GA",
    "EclipseLink 2.0.0",
    "JUM 13. 2010. január 20-án",
    "Java 7 újdonságok",
    "Tomcat 7?",
    "EclipseLink 1.2.0",
    "JRuby 1.4.0",
    "Google Collections",
    "Java 5 - béke poraira?"
  };

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    ArrayAdapter adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_spinner_item,
            newsTitles);

    adapter.setDropDownViewResource(
            android.R.layout.simple_spinner_dropdown_item);

    Spinner spinner = (Spinner) findViewById(R.main.spinner);
    spinner.setOnItemSelectedListener(this);
    spinner.setAdapter(adapter);
  }

  @Override
  public void onItemSelected(AdapterView<?> parent, View v, int position, long id)
  {
    TextView view = (TextView) findViewById(R.main.selection);
    view.setText(newsTitles[position]);
  }

  @Override
  public void onNothingSelected(AdapterView<?> parent)
  {
    TextView view = (TextView) findViewById(R.main.selection);
    view.setText("empty");
  }
}
```

Ezen program leglényegesebb különbsége az elzekhez képest, hogy a közönséges *Activity* osztályból származik, s implementálja az *OnItemSelectedListener* interfész két metódusát, illetve az adapternél megadott *layout* típus is a *Spinner* komponenshez illik. Az *onItemSelected* metódusban egyszeren beállítjuk a szöveges mez értékét a kiválasztott pozíció szövegére, illetve az *onNothingSelected* metódusban "üresre" állítjuk ezt az értéket. Nézzük az eredményt:



A szövegek mellett látható köröcskék közül mindig az van kiválasztva, amelyik az aktuális állapotot hordozza.

5.2.6. A táblázat

A táblázatos elrendezés segíthet, ha sok kis dolog közül kell választani, amelyek akár egymás mellett is elférnek több oszlopban, tipikusan képek kiválasztása ilyen, de rövid szövegekre is hasznos lehet. Hosszabb szöveget nem érdemes táblázatban tartani, mivel a platform nem képes megfelelen tördelni a szavakat, illetve azok egymásba is csúszhatnak. Programozás szempontjából a táblázat alig tér el a *Spinner* megvalósításától, a *main.xml* például a következképp néz ki:

main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_height="fill_parent"
              android:layout_width="fill_parent">
  <TextView android:id="@+main/selection"
            android:background="#000044"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"/>
  <GridView android:id="@+main/grid"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:numColumns="auto_fit"
            android:verticalSpacing="50dp"
            android:horizontalSpacing="10dp"
            android:columnWidth="150dp"
            android:stretchMode="columnWidth"
            android:gravity="center"/>
</LinearLayout>
```

Az egyetlen ismeretlen komponens a *GridView*nevet viseli, ebben írhatjuk le a táblázat kinézetét. Három lényeges paraméter értékét kell jól beállítanunk ahhoz, hogy a készülék kijelzjétl független táblázatunk legyen:

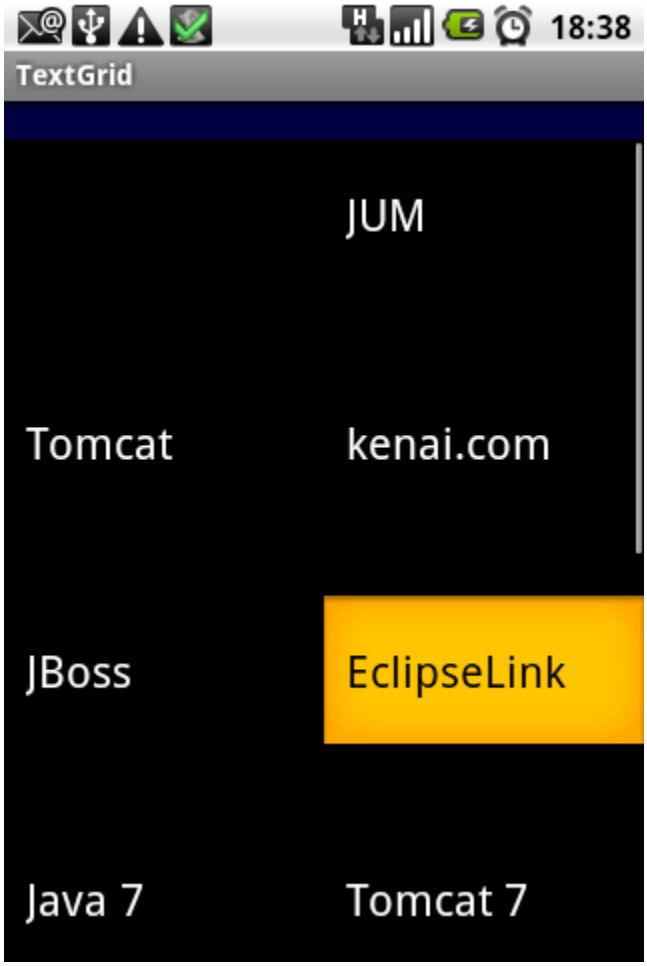
- **columnWidth**, amely az oszlopok szélességét határozza meg
- **horizontalSpacing**, amely az oszlopok közötti helyet határozza meg
- **verticalSpacing**, amely a sorok közötti helyet határozza meg

A többi paraméter az automatikus elhelyezést szolgálja ki, a *gravity* pedig a táblázaton belüli elemek pozícióját. Nézzük a programot:

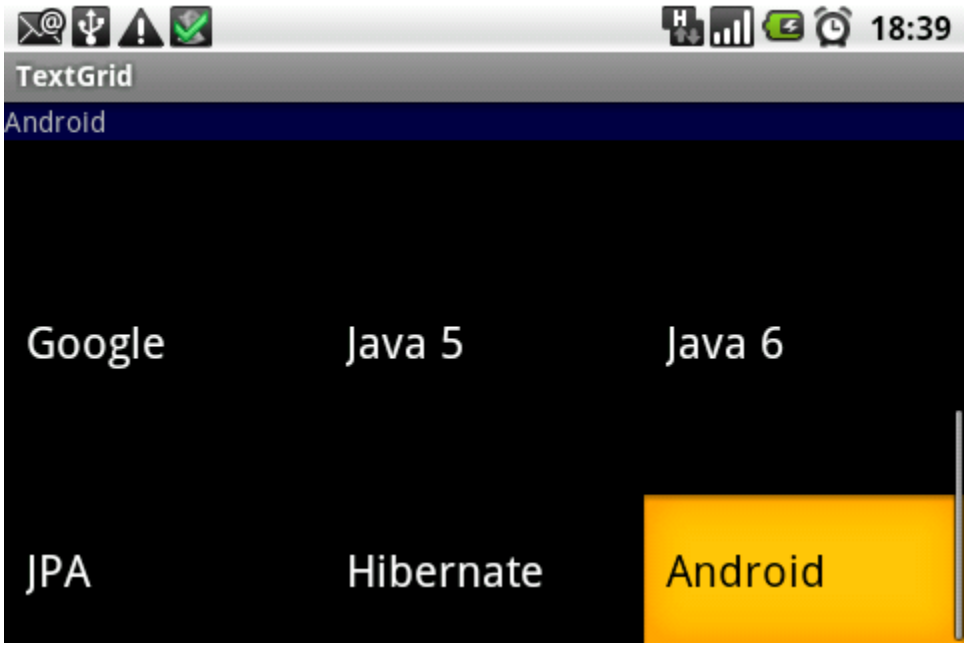
MainActivity.java

```
public class MainActivity extends Activity implements AdapterView.OnItemClickListener
{

  String[] newsTitles =
  {
    "Glassfish", "JUM", "Tomcat",
    "kenai.com", "JBoss", "EclipseLink",
    "Java 7", "Tomcat 7", "JRuby",
    "Google", "Java 5", "Java 6",
    "JPA", "Hibernate", "Android",
  };

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    ArrayAdapter adapter = new ArrayAdapter(this,
            android.R.layout.simple_list_item_1,
            this.newsTitles);

    GridView gridView = (GridView) findViewById(R.main.grid);
    gridView.setAdapter(adapter);
    gridView.setOnItemClickListener(this);
  }

  @Override
  public void onItemClick(AdapterView<?> parent, View v, int position, long id)
  {
    TextView view = (TextView) findViewById(R.main.selection);
    view.setText(newsTitles[position]);
  }
}
```

Azt kell mondjam, hogy nincs mit hozzáfzni, annyira egyértelm kell legyen... nézzük inkább a képernyképeket:



Az elrendezés lényege, hogy a telefont elfordítva három oszlop lesz, hiszen a szélesebb kijelzn több oszlop fér el:



5.2.7. Szabad szöveges választómez

Az egyszer beviteli mez és a legördül menü keresztezésébl született az *autocomplete* beviteli mez, más néven a szabad szöveges választómez. Nézzük ennek a *main.xml* állományát:

main.xml

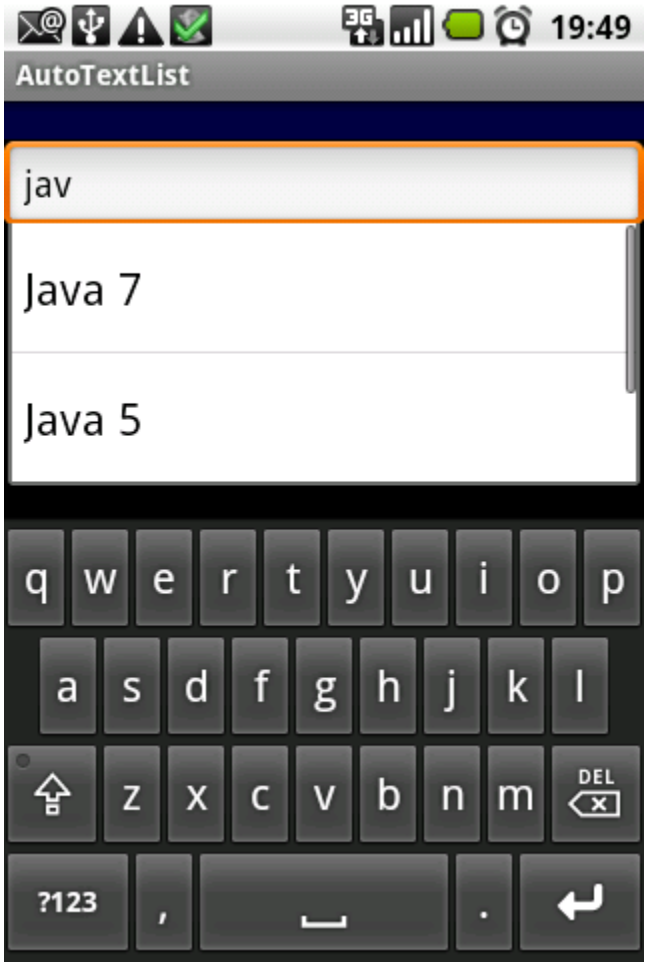
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_height="fill_parent"
              android:layout_width="fill_parent">
  <TextView android:id="@+main/selection"
            android:background="#000044"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"/>
  <AutoCompleteTextView android:id="@+main/edit"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:completionThreshold="3"/>
</LinearLayout>
```

Az *AutoCompleteTextView* pont úgy viselkedik, mint egy normális *TextView*, leszámítva a *completionThreshold* paraméterét, amely azt mondja meg, hogy hány karakter után adjon listát az opciókról. Az XML elrendezéshez tartozó program se mondható hosszúnak:

MainActivity.java

```
public class MainActivity extends Activity
{

  String[] newsTitles =
  {
    "Glassfish", "JUM", "Tomcat",
    "kenai.com", "JBoss", "EclipseLink",
    "Java 7", "Tomcat 7", "JRuby",
    "Google", "Java 5", "Java 6",
    "JPA", "Hibernate", "Android",
  };

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    ArrayAdapter adapter = new ArrayAdapter(this,
            android.R.layout.simple_dropdown_item_1line,
            this.newsTitles);

    AutoCompleteTextView editView = (AutoCompleteTextView) findViewById(R.main.edit);
    editView.setAdapter(adapter);
  }
}
```

Az eredmény az alábbi képen látható:



Fontos észrevenni, hogy az szrt lista elemei abban a sorrendben jelennek meg, ahogy a listában szerepelnek, így fordulhat el, hogy a Java 7 után a Java 5 van, s végül a Java 6 következik.

5.2.8. Dátum és id kiválasztása

Ha a felhasználónak dátumot vagy idt kell megadnia, akkor célszer egy *DatePickerDialog* vagy egy *TimePickerDialog* példányt létrehoznunk, mivel így nem tud beírni 2010 cinóber hónap harmincötödikét, vagy huszonöt óra hatvannyolc percet, amely bevitelek nehéz pillanatok elé állítanák a programunkat. A minimalista példa programunk két darab *TextView* példányt tartalmaz:

main.xml

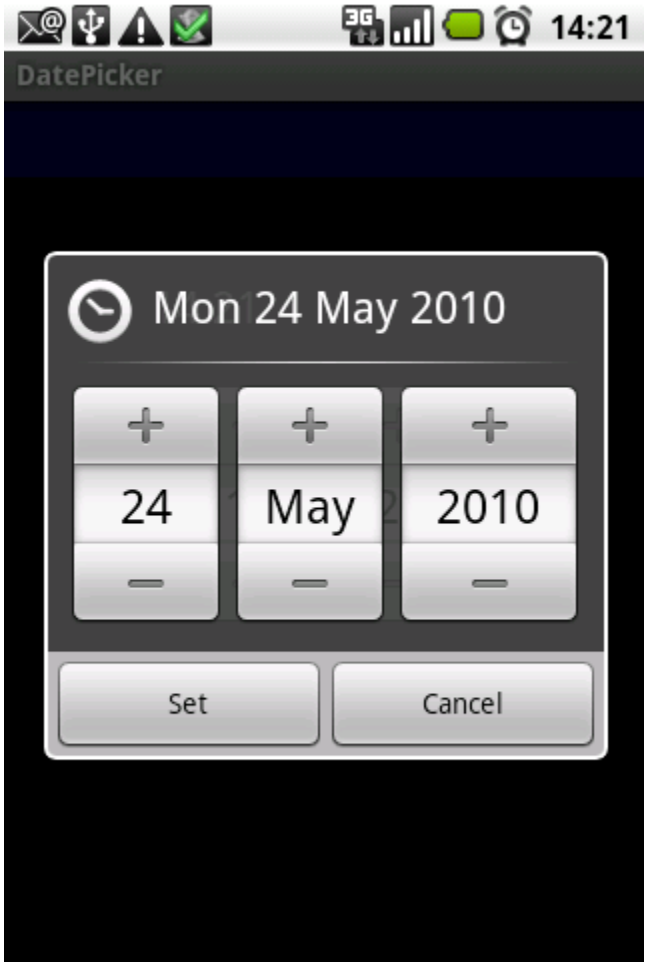
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_height="fill_parent"
              android:layout_width="fill_parent">
  <TextView android:id="@+main/pickedDate"
          android:background="#000044"
          android:layout_height="wrap_content"
          android:layout_width="fill_parent"
          android:label="Pick a date..."/>
  <TextView android:id="@+main/pickedTime"
          android:background="#000044"
          android:layout_height="wrap_content"
          android:layout_width="fill_parent"
          android:label="Pick a time..."/>
</LinearLayout>
```

Az egyik példányba írjuk majd a kiválasztott dátumot, a másikba a kiválasztott idt. Lássuk a programot:

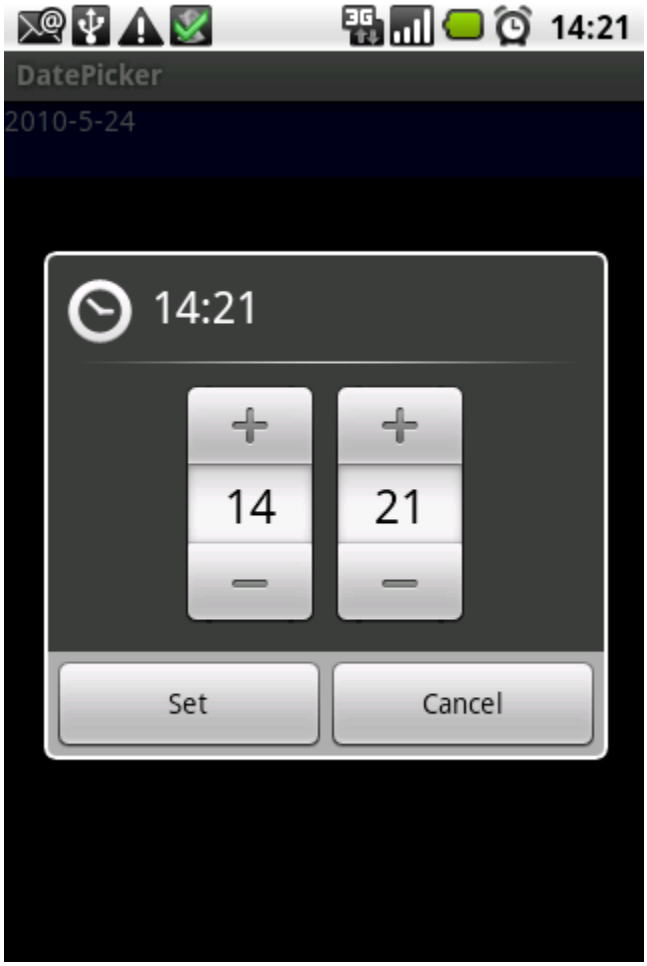
MainActivity.java

```
 public class MainActivity extends Activity implements
        DatePickerDialog.OnDateSetListener,
        TimePickerDialog.OnTimeSetListener
{

  private DatePickerDialog datePickerDialog;
  private TimePickerDialog timePickerDialog;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    Calendar cal = Calendar.getInstance();
    datePickerDialog = new DatePickerDialog(this,
            this,
            cal.get(Calendar.YEAR),
            cal.get(Calendar.MONTH),
            cal.get(Calendar.DAY_OF_MONTH));
    timePickerDialog = new TimePickerDialog(this,
            this,
            cal.get(Calendar.HOUR_OF_DAY),
            cal.get(Calendar.MINUTE),
            true);
  }

  @Override
  public void onResume()
  {
    super.onResume();

    timePickerDialog.show();
    datePickerDialog.show();
  }

  @Override
  public void onDateSet(DatePicker view, int year, int month, int day)
  {
    TextView textView = (TextView) findViewById(R.main.pickedDate);
    textView.setText(year + "-" + (month + 1) + "-" + day);
  }

  @Override
  public void onTimeSet(TimePicker vuew, int hour, int min)
  {
    TextView textView = (TextView) findViewById(R.main.pickedTime);
    textView.setText(hour + ":" + min);
  }
}
```

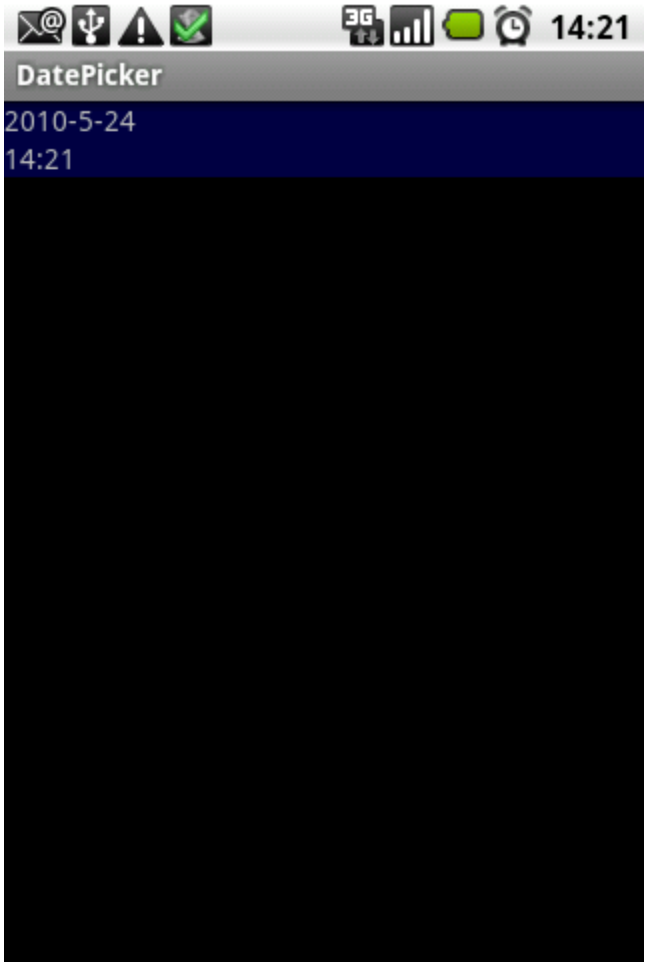
Két lényeges dolgot kell látnunk, mégpedig a két interfész implementációját, az *onDateSet* és az *onTimeSet* metódusokat, ezekben megkeressük a címkéinket, majd beleírjuk a kapott értékeket. Fontos, hogy a hónap - jó Java szokás szerint - nullával kezddik, vagyis érdemes hozzáadnunk egyet. A dátum kiválasztása felbukkanó ablakban történik:



A *Set* gombra kattintva megjelenik az id kiválasztása, amely mindvégig a dátum dialógus ablak alatt volt, ezzel egy idben a dátum a megfelel címke szövege lesz:



Az id elmentése után már csak az eredmény látszik:


14:21
DatePicker
2010-5-24
14:21