

Android alapú szoftverfejlesztés

4. Labor



Alkalmazás komponensek közti kommunikáció Android platformon

Tartalom

1	Felkészülés a laborra.....	2
2	Laborfeladatok	2
2.1	AndroidLabor4 projekt létrehozása.....	2
2.2	Visszatérés egy Todo objektummal	3
2.2.1	Parcelable interfész implementálása	5
2.2.2	Todo példány átadása.....	6
2.3	Implicit intentek	11
2.3.1	Alkalmazás vázának generálása.....	11
2.3.2	Felhasználói felület	11
2.3.3	Működés megvalósítása	12
2.4	Broadcast receiver	13
2.4.1	Az alkalmazás váza.....	13
2.4.2	Intent Filter beállítása	13
2.4.3	Broadcast kezelése	14
2.5	Bónusz feladat: saját Home Screen alkalmazás	16
2.5.1	Alkalmazás beállítása, mint alternatív Home Screen	16
2.5.2	Az alkalmazás váza.....	17
2.5.3	Launcher-ben megjelenítendő alkalmazások lekérése.....	17
2.5.4	Adatkötés	18
2.5.5	Eseménykezelő kattintásra.....	18
2.5.6	Befejezés	19

1 Felkészülés a laborra

A labor célja az alkalmazás komponensek közti kommunikáció megvalósításának és kezelésének megtanulása Android platformon. A mérés során az előző laboron megvalósított *Todo* alkalmazást fejlesztjük tovább, valamint új alkalmazásokat hozunk létre a komponensek bemutatásához. A kódrészletek a tárgy honlapon lévő AndroidLabor 4 patch fájlból másolhatók.

A mérés az alábbi témákat érinti:

- Parcelable interfész használata saját osztályban
- Intentek, Intent filterek használata
- BroadcastReceiver működése
- Saját Home Screen alkalmazás készítése

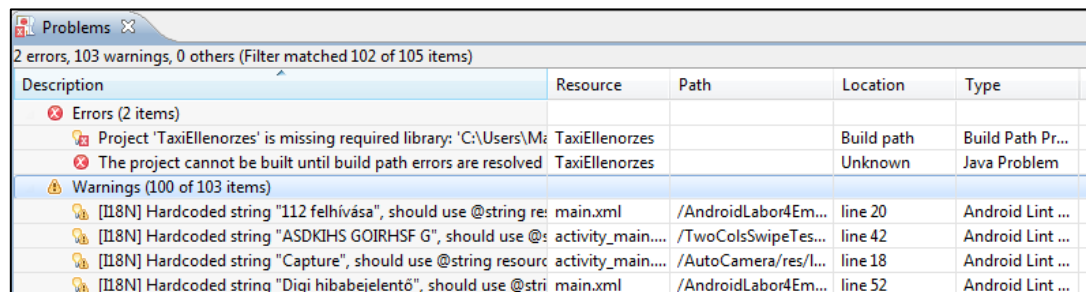
2 Laborfeladatok

2.1 AndroidLabor4 projekt létrehozása

Első lépésként hozzunk létre egy új Android projektet **AndroidLabor4** néven, és másoljuk át a múlt heti labor kódját valamint erőforrás fájljait. A szükséges fájlok letölthetők a következő címről:

https://www.aut.bme.hu/Upload/Course/android/gyakorlat_anyagok/ToDoKiindulas.zip

Figyelem! Az átmásolt fájlokban mindenhol írjuk át a csomagnevet arra, amit a jelenlegi projektnél használunk! Az *AndroidManifest.xml*-ben is! Futtassuk meg az alkalmazást, és győződjünk meg arról, hogy működik a múlt heti funkcionalitás! Ha valami nem stimmel, akkor az Eclipse *“Problems”* nézetében keressük meg a hibát és javítsuk!



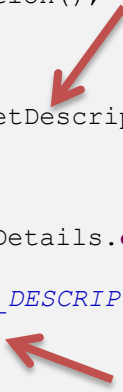
Description	Resource	Path	Location	Type
Errors (2 items)				
Project 'TaxiEllenorzes' is missing required library: 'C:\Users\M... TaxiEllenorzes	TaxiEllenorzes		Build path	Build Path Pr...
The project cannot be built until build path errors are resolved	TaxiEllenorzes		Unknown	Java Problem
Warnings (100 of 103 items)				
[I18N] Hardcoded string "112 felhívása", should use @string res: main.xml		/AndroidLabor4Em...	line 20	Android Lint ...
[I18N] Hardcoded string "ASDKIHS GOIRHSF G", should use @s activity_main....		/TwoColsSwipeTes...	line 42	Android Lint ...
[I18N] Hardcoded string "Capture", should use @string resourc activity_main....		/AutoCamera/res/l...	line 18	Android Lint ...
[I18N] Hardcoded string "Digi hibabejelentő", should use @stri main.xml		/AndroidLabor4Em...	line 52	Android Lint ...

2.2 Visszatérés egy *Todo* objektummal

Az előző labor alkalmával többször is szükség volt arra, hogy egy *Todo* objektumot leíró adatokat komponensek között adjunk át. Amennyiben saját osztályt kell definiálnunk az entitásaink tárolására (üzleti alkalmazások esetén gyakorlatilag mindig), és szeretnénk ezek adatait komponensek között mozgatni futásidőben (ez szintén igen gyakori), akkor a múlt héten alkalmazott módszer a fejlesztő számára kényelmetlen, nehezen karbantartható és bővíthető új tagváltozókkal. Legutóbb azt csináltuk, hogy egy *Todo* megnyitásakor annak csak egy mezőjét, a *leírást* adtuk át a *TodoDetailsFragment*-nek, a *név*, *prioritás* és *dátum* mezőket pedig dummy adatokkal töltöttük ki.

ActivityMain:

```
public void onTodoSelected(Todo selectedTodo) {  
    if (fragmentContainer != null) {  
        FragmentManager fm = getSupportFragmentManager();  
  
        FragmentTransaction ft = fm.beginTransaction();  
        ft.replace(  
            R.id.FragmentContainer,  
  
            TodoDetailsFragment.newInstance(selectedTodo.getDescription())  
        );  
        ft.commit();  
    } else {  
        Intent i = new Intent(this, ActivityTodoDetails.class);  
        i.putExtra(  
            TodoDetailsFragment.KEY_TODO_DESCRIPTION,  
            selectedTodo.getDescription()  
        );  
        startActivity(i);  
    }  
}
```



TodoDetailsFragment:

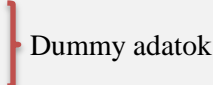


```
public static TodoDetailsFragment newInstance(String todoDesc) {
    TodoDetailsFragment result = new TodoDetailsFragment();

    Bundle args = new Bundle();
    args.putString(KEY_TODO_DESCRIPTION, todoDesc);
    result.setArguments(args);

    return result;
}

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

    if (savedInstanceState == null) {
        if (getArguments() != null) {
            selectedTodo = new Todo(
                "cim",
                Priority.LOW,
                "1987.23.12",
                getArguments().getString(KEY_TODO_DESCRIPTION)
            );
        }
    }
}
```



Amennyiben a *Todo* objektum részleteit megnyitó *Fragmentsben* a leírásnál több adatot szeretnénk megjeleníteni, akkor ez a megoldás bővítésre szorul, az *Intent Extrába* és az argumentumokat tároló *Bundle*-be kézzel fel kell venni a többi tagváltozót (cím, dátum, stb...).

Bizonyára érezzük hogy nem ez a legjobb megoldás, az lenne az igazi, ha az egész *Todo* objektumot át tudnánk adni. Ekkor nem kellene bővíteni semmivel a fenti kódokat még akkor sem, ha a *Todo* osztályba utólag új tagváltozókat vezetünk be. Szerencsére a feladat nem lehetetlen, az Android módot ad arra hogy saját osztályainkat képessé tegyük a *Bundle*-be csomagolhatóságra, ami mindkét esetre (*Intent Extrába* és *args*-ba csomagolás) megoldást jelent. Ez a viselkedés két módon érhető el:

- az osztályunk implementálja a *Serializable* interfészt, vagy
- az osztályunk implementálja a *Parcelable* interfészt

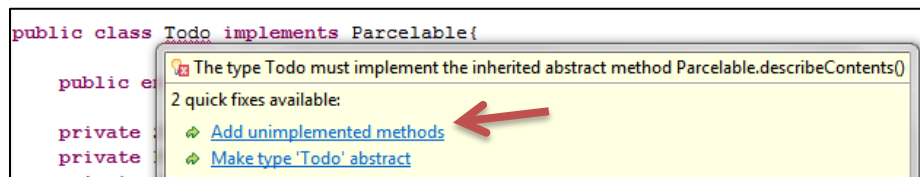
Mindkét interfész arra szolgál, hogy a saját (entitás)osztályainkat egy köztes, csomagolásra és szállításra alkalmas formátumba „exportáljuk”, amit majd a platform-szinen nagyon gyakran használt, általános *Bundle* objektumba fogunk tudni belerakni és így átadni akár *Intent Extra*-ként, akár *Fragments* argumentumként. A laboron a *Parcelable* interfész használatával ismerkedünk meg.

2.2.1 Parcelable interfész implementálása

A *Todo* osztály fejlécében jelezzük, hogy megvalósítja a *Parcelable* interfészt, majd importáljuk a megfelelő osztályt (Ctrl+Shift+O).

```
import android.os.Parcel;  
import android.os.Parcelable;  
  
public class Todo implements Parcelable {  
    ...  
}
```

A fejlesztő környezet ekkor hibát jelez a *Todo* osztályra, mert a *Parcelable* interfész implementálása kötelez minket néhány metódus felüldefiniálására. Vigyük az egeret az aláhúzott *Todo* osztálynév fölé, majd a megjelenő menüben válasszuk az „Add unimplemented methods” lehetőséget.



Ennek hatására az Eclipse két metódus vázát generálja le.

- A *describeContents()*-nek egy bitmaszkot kell visszaadnia, amit csak akkor kell használnunk, ha *FileDescriptor* típusú tagváltozónk van az osztályban. Ekkor a *describeContents()*-nek pontosan a *Parcelable.CONTENTS_FILE_DESCRIPTOR* konstanst kell visszaadnia, minden egyéb esetben 0 a megfelelő visszatérés. Jelen esetben nincs *FileDescriptor* tagváltozónk, így maradhat a generált „return 0”.

```
public int describeContents() {  
    return 0;  
}
```

- A *writeToParcel()* metódusban kell megírunk a lényegét, itt rakjuk bele a kimenő *Parcel*-be az osztályunk azon tagváltozóit, melyeket szeretnénk továbbítani. A sorrend fontos, ugyanis ebben a sorrendben kell majd visszaolvasni a berakott változókat a *Parcel*-ből táplálkozó konstruktorban (még nincs megírva). Az *enum* típusú változót a nevével helyettesítjük.

```
public void writeToParcel(Parcel dest, int flags) {  
    dest.writeString(title);  
    dest.writeString(priority == null ? null : priority.name());  
    dest.writeString(dueDate);  
    dest.writeString(description);  
}
```

Ahhoz, hogy az Android képes legyen *Parcel*-ként átadni az osztálpéldányunkat, a fenti két metódus implementálásán kívül definiálnunk kell egy statikus ***Parcelable.Creator*** típusú objektumot, melynek neve kötelezően ***CREATOR***. A szükséges kódrészlet:

```
public static final Parcelable.Creator<Todo> CREATOR = new
Parcelable.Creator<Todo>() {
    public Todo createFromParcel(Parcel in) {
        return new Todo(in);
    }

    public Todo[] newArray(int size) {
        return new Todo[size];
    }
};
```

Mint látható, a ***CREATOR*** objektum olyan *Todo* konstruktort hív, ami egy *Parcel*-ből képes létrehozni a példányt, így ezt is meg kell írunk. Fontos hogy olyan sorrendben olvassuk ki az attribútumokat ahogy azokat a ***writeToParcel()***-ben beleírtuk, illetve a nevével helyettesített *enum* is visszanyerje eredeti, *Priority* típusú értékét.

```
public Todo(Parcel in){
    this.title = in.readString();

    String priorityName = in.readString();
    if(priorityName != null)
        this.priority= Priority.valueOf(priorityName);

    this.dueDate = in.readString();
    this.description= in.readString();
}
```

Ezzel felkészítettük a *Todo* osztályunkat arra, hogy *Parcel*-be csomagolható legyen, így átadhassunk *Intent Extraként*, vagy *Fragments* argumentumként

2.2.2 Todo példány átadása

Zárjuk be az Eclipse-ben az összes megnyitott fájlt, és nyissuk meg az *ActivityMain* valamint a *TodoDetailsFragment* osztályokat tartalmazó fájlokat.

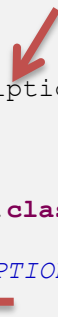
A célunk tehát az, hogy egy *Todo* megnyitásakor ne csak annak leírása adódjon át, hanem az egész objektum. A *Todo* osztályt már előkészítettük, most csak annyi a dolgunk hogy használjuk is az új lehetőséget.

Az *ActivityMain* osztályba két helyen kell belenyúlnunk ehhez:

```
public void onTodoSelected(Todo selectedTodo) {
    if (fragmentContainer != null) {
        FragmentManager fm = getSupportFragmentManager();

        FragmentTransaction ft = fm.beginTransaction();
        ft.replace(
            R.id.FragmentContainer,

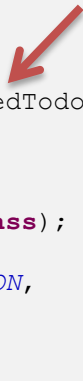
            TodoDetailsFragment.newInstance(selectedTodo.getDescription())
        );
        ft.commit();
    } else {
        Intent i = new Intent(this, ActivityTodoDetails.class);
        i.putExtra(
            TodoDetailsFragment.KEY_TODO_DESCRIPTION,
            selectedTodo.getDescription()
        );
        startActivity(i);
    }
}
```



A leírás helyett mindkét jelzett helyen adjuk át az egész Todo objektumot.

```
public void onTodoSelected(Todo selectedTodo) {
    if (fragmentContainer != null) {
        FragmentManager fm = getSupportFragmentManager();

        FragmentTransaction ft = fm.beginTransaction();
        ft.replace(
            R.id.FragmentContainer,
            TodoDetailsFragment.newInstance(selectedTodo)
        );
        ft.commit();
    } else {
        Intent i = new Intent(this, ActivityTodoDetails.class);
        i.putExtra(
            TodoDetailsFragment.KEY_TODO_DESCRIPTION,
            selectedTodo
        );
        startActivity(i);
    }
}
```



Az Eclipse ennek hatására két dolgot csinál:

- Hibát jelez a *TodoDetailsFragment.newInstance()* metódusnál, mivel ez a függvény arra számít, hogy egy *String*-et kap. Semmi gond, mindjárt javítjuk.
- Nem jelez hibát (!) az *Intent.putExtra()* metódusánál, pedig már nem *String*-et adunk értékként. Ennek oka, hogy a *Todo* osztályunk implementálja a *Parcelable*

interfészt, amit az *Intent Extra* alaptól be tud fogadni (pont ezért valósítottunk meg a *Parcelable*-t)

A *TodoDetailsFragment.newInstance()* módosítása előtt javítsunk ki egy formai hibát. Az *Intent Extra*-ba pakolt *Todo* kulcsának nevét változtassuk *KEY_TODO_DESCRIPTION*-ról *KEY_TODO*-ra, értékét pedig „*todoDesc*”-ról „*todo*”-ra, mivel már nem csak a *Todo* leírását, hanem az egész objektumot fogjuk átadni. Ehhez érdemes használni az Eclipse beépített átnevező eszközét, ami egy változó nevét képes egyszerre átírni az összes előfordulásánál. A kurzorral álljunk rá a *TodoDetailsFragment*-ben lévő névre, és nyomjuk meg az *Alt+Shift+R* billentyűkombinációt, majd írjuk át a nevet. Ne felejtsük el átírni az értéket is „*todo*”-ra.

<pre>public class TodoDetailsFragment extends Fragment { public static final String TAG = "TodoDetailsFragment"; public static final String KEY_TODO_DESCRIPTION = "todoDesc"; private TextView todoDesc; private static Todo selectedTodo; public static TodoDetailsFragment newInstance(String todoDesc) { TodoDetailsFragment result = new TodoDetailsFragment(); Bundle args = new Bundle(); args.putString(KEY_TODO_DESCRIPTION, todoDesc); result.setArguments(args); return result; } }</pre>	<pre>public class TodoDetailsFragment extends Fragment { public static final String TAG = "TodoDetailsFragment"; public static final String KEY_TODO = "todoDesc"; private TextView todoDesc; private static Todo selectedTodo; public static TodoDetailsFragment newInstance(String todoDesc) { TodoDetailsFragment result = new TodoDetailsFragment(); Bundle args = new Bundle(); args.putString(KEY_TODO, todoDesc); result.setArguments(args); return result; } }</pre>
---	---

A fejlesztő környezet ekkor feldob egy Warning dialógusablakot, mivel egy másik érintett fájlban (*ActivityMain.java*) éppen fordítási hiba van (a *newInstance()* paraméterezése miatt), ez most nem gond úgyis mindjárt javítjuk.

Készítsük fel a *TodoDetailsFragment* gyártó (*factory*) metódusát arra, hogy egész *Todo* objektumot kapva is képes legyen legyártani a *Fragments*-t. Ehhez egyszerűen írjuk át a *newInstance()* metódus paraméterét „*String todoDesc*”-ról „*Todo todo*”-ra. Itt szintén érdemes használni a beépített átnevezőt, hiszen a „*todoDesc*” változónév a metódus vázában is előfordul. Az „*args*” bundle feltöltésekor már nem *String*-et, hanem a *Parcelable* interfészt megvalósító *Todo* objektumot használunk, így a *putString()* helyett használjuk a *putParcelable()* metódust. A teljes, módosított metódus kódja a következő:

```
public static TodoDetailsFragment newInstance(Todo todo) {  
    TodoDetailsFragment result = new TodoDetailsFragment();  
  
    Bundle args = new Bundle();  
    args.putParcelable(KEY_TODO, todo);  
    result.setArguments(args);  
  
    return result;  
}
```

Figyeljük meg, hogy az *ActivityMain* osztályban eltűnt a hibajelzés, mivel a *newInstance()* metódus már képes *Todo* objektumból táplálkozni.

A `TodoDetailsFragment` osztályban már csak egy helyen kell módosítanunk a kódot, az `onCreate()` életciklus callback metódusban kell betöltenünk az argumentumban kapott `Todo` objektumot a jelenlegi megoldás helyett.

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

    if (savedInstanceState == null) {
        if (getArguments() != null) {
            selectedTodo =
                getArguments().getParcelable(KEY_TODO);
        }
    }
}
```

Itt semmilyen konstruktor hívásra vagy kasztolásra nincs szükség, mivel:


- Az argumentumok tömbbe egy egész `Todo` objektumot tettünk (nem kell konstruktor)
- A `getArguments().getParcelable()` egy `Parcelable`-t ad vissza, a `selectedTodo` pedig implementálja ezt az interfészt (nem kell kasztolni)

Vegyük észre hogy ez a megoldás nem csak sokkal elegánsabb mint a leírás átadása, de nem igényel bővítést, ha egy `Todo` megnyitásakor nem csak annak leírását akarjuk megjeleníteni. Például ha a címét is ki akarjuk írni, akkor mindössze ennyit kell tennünk:

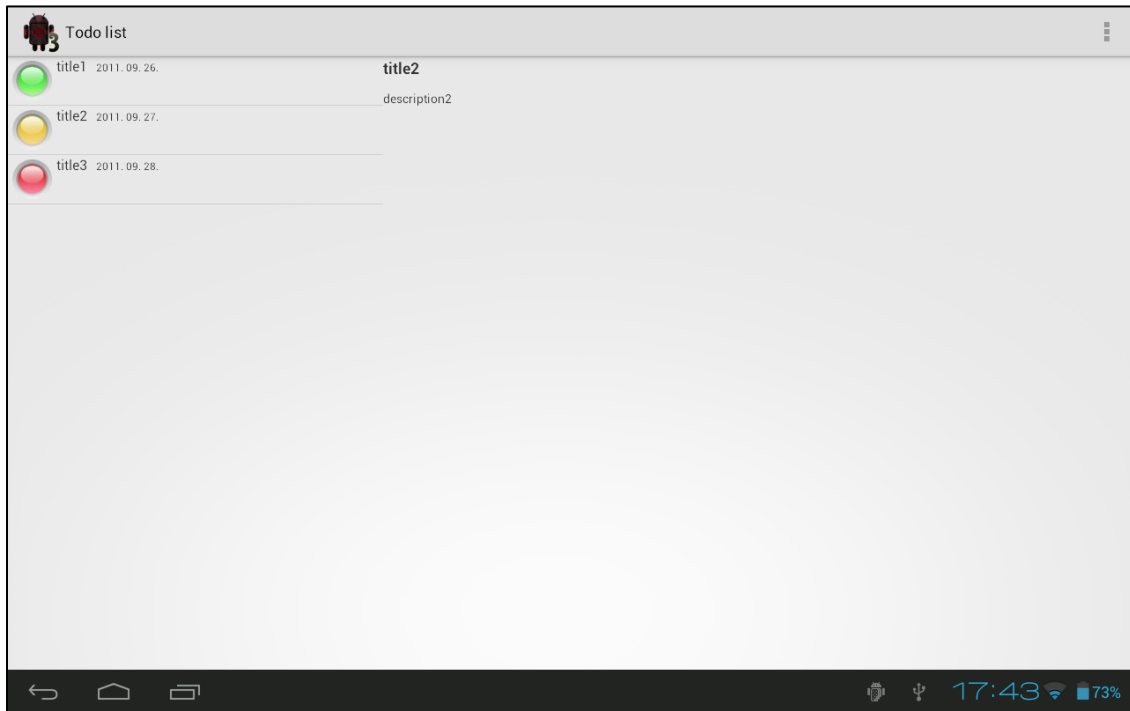
```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container,
    Bundle savedInstanceState) {
    View root = inflater.inflate(
        R.layout.fragment_todo_details,
        container,
        false);

    todoDescription = (TextView)
root.findViewById(R.id.todoDescription);
    todoDescription.setText(Html.fromHtml(
        "<h3>" + selectedTodo.getTitle() + "</h3>"
        + selectedTodo.getDescription()
    ));

    return root;
}
```



Az eredmény:



2.3 *Implicit intentek*

Készítsünk egy új alkalmazást, ami vészhelyzet esetén megkönnyíti a rendőrség vagy az internetes hibabejelentő felhívását.

2.3.1 Alkalmazás vázának generálása

Hozzunk létre egy új Android alkalmazást **AndroidLabor4Emergency** néven. Generáltassunk egy Activity-t a fejlesztő környezettel MainActivity néven.

2.3.2 Felhasználói felület

Az alkalmazás egyetlen képernyőből áll, melynek felületén gombok jelennek meg a 112, illetve a három legnagyobb internetszolgáltató hibabejelentő telefonszámainak felhívásához



A felhasználói felületet definiáló XML a következő linken érhető el:

<https://gist.github.com/3812610> (*main.xml* szekció)

Töltse le a szükséges ikonokat az alábbi címről, majd adja hozzá a projekthez!

https://www.aut.bme.hu/Upload/Course/android/gyakorlat_anyagok/AndroidLabor5Emergency_images.zip

Azt szeretnénk ha az alkalmazást csak álló módban lehetne használni, most nem foglalkozunk a külön fekvő layout készítésével. Ennek elérésére az *AndroidManifest.xml*-ben a megfelelő *<activity>* tagben állítsuk be a megfelelő attribútumot:

```
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name" >
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name"
        android:screenOrientation="portrait" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
```



2.3.3 Működés megvalósítása

A *MainActivity*-ben írjuk meg az alkalmazás működéséhez szükséges kódot. A gombok eseménykezelője legyen közös, a kattintott *View* objektum id-ja alapján állítsa be a felhívandó telefonszámot, majd indítsa a hívást. Az eseménykezelő váza, illetve a 112 hívásának megvalósítása a következő kódrészlettel érhető el:

```
OnClickListener callEmergency = new OnClickListener() {

    public void onClick(View v) {
        String phoneNumber = "tel: ";
        /* *
         * T-Home hibabejelentő: 1412
         * Invitel hibabejelentő: 1445
         * Digi hibabejelentő: 1272
         * */
        switch (v.getId()) {

            case R.id.call112Btn:
                phoneNumber += "112";
                break;

            // TODO: a többi gomb lenyomásának kezelése

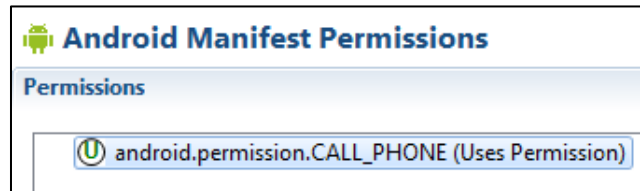
            default:
                break;

        }

        // szam felhivasa
        Intent i = new Intent(
            Intent.ACTION_CALL,
            Uri.parse(phoneNumber)
        );
        startActivity(i);
    }

};
```

Ahhoz hogy telefonhívást indíthassunk az alkalmazásunkból (az implicit *Intent* által indított tárcsázó *Activity* is a mi processzünkben lesz), el kell kérni a *CALL_PHONE* nevű engedélyt. Ehhez az *AndroidManifest Permissions* fülén adjunk hozzá egy új „Uses Permission” típusú elemet, melynek neve: *android.permission.CALL_PHONE*.



Önálló feladat: Valósítsa meg a többi gomb megnyomásának eseménykezelését!

2.4 Broadcast receiver

A következő fejezet bemutatja hogyan lehet *Broadcast Intent*-eket kezelni, ezzel felkészítve az alkalmazásunkat rendszerszintű eseményekre. Célunk, hogy értesüljünk a bejövő SMS üzenekről, illetve *Toast*-ban jelenítsük meg a feladót és az SMS szövegét.

2.4.1 Az alkalmazás váza

Hozzunk létre egy új alkalmazást, és vegyünk fel egy új osztályt, ami a *BroadcastReceiver*-ből származik. Ez a származtatás kötelezően előírja az *onReceive()* metódus felüldefiniálását, aminek vázát megkapjuk az „Add unimplemented methods” kiválasztásával. Ahogy a függvény fejlécéből látszik, megkapjuk az *Intent* objektumot, amire feliratkoztunk a megfelelő *Intent Filter* beállításával.

2.4.2 Intent Filter beállítása

Nyissuk meg az *AndroidManifest.xml* szerkesztő felületét, és az *Application* fülön vegyük fel a *BroadcastReceiver* osztályunkat (*Application Nodes* szekció → *Add* → *Receiver* → jobb oldalt *Name* mellett *Browse* → saját *BR* kiválasztása → mentés). Ezután az XML szerkesztő felületen állítsunk be egy intent filtert a bejövő SMS-ek kezelésére:

```
<receiver android:name="[BR osztály neve]" >
  <intent-filter>
    <action
      android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
  </receiver>
```

Mivel személyes adathoz szeretnénk hozzáférni, permission-t kell kérni a felhasználótól. A manifestben kérjük el az SMS fogadásához szükséges *RECEIVE_SMS*, olvasásához pedig a *READ_SMS* engedélyeket.

2.4.3 Broadcast kezelése

Az *onReceive()* metódusban kezeljük le a Broadcast üzenetet. SMS olvasásnál ez a következő módon történik (csak SMS kiolvasás esetén ilyen bonyolult az adat kinyerése):

```
if(intent.getAction().
    equalsIgnoreCase("android.provider.Telephony.SMS_RECEIVED")){
    // 'pdus' nevu extraban egy Object tombot kapunk, amibol
    kinyerhető az sms
    Object[] pdus = (Object[]) intent.getExtras().get("pdus");
    if(pdus == null){
        Log.e("RECEIVER", "pdus are null");
    }
    else{
        Log.v("RECEIVER",
            "received " + pdus.length + " messages");
        SmsMessage msg = null;
        // Object tombot kaptunk, vegigmegyunk rajta
        for (Object pdu : pdus) {
            // a konkret SMS kinyerese
            msg = SmsMessage.createFromPdu((byte[])pdu);
            if(msg != null){
                showToast(
                    context,
                    "Message from "
                    +msg.getOriginatingAddress()+"": "
                    +msg.getDisplayMessageBody());
            }
            else{
                Log.e("RECEIVER", "Sms is null");
            }
        }
    }
}
```

A *showToast(String)* egy segédfüggvény, ami a *Toast.makeText()* metódust hívja megfelelően paraméterezve.

Írja meg a *showToast()* függvényt, majd tesztelje az alkalmazást! (Az emulatorra a DDMS felületről tud SMS-t küldeni)

Készítse fel a *BroadcastReceiver*-t két további, szabadon választott esemény kezelésére, és tesztelje a kibővített alkalmazást!

2.5 Bónusz feladat: saját Home Screen alkalmazás

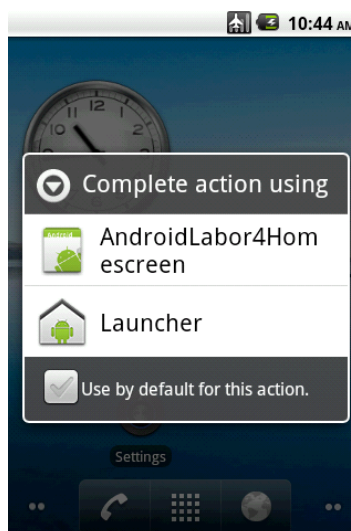
Android platformon minden alkalmazás ugyanolyan jogosultsági szinten működik, a gyári, előre telepített appok nem élveznek elsőbbséget az általunk fejlesztettekkel szemben. Ez gyakorlatilag azt jelenti, hogy bármelyik alkalmazást lecserélhetjük saját készítésűre, így a főképernyőt is. Ahogy előadáson is elhangzott, ez nagyon egyszerűen kivitelezhető a megfelelő *Intent Filter* beállításával. Ebben a fejezetben egy saját indító képernyőt megjelenítő alkalmazást írunk.

2.5.1 Alkalmazás beállítása, mint alternatív Home Screen

Készítsünk egy új Android alkalmazást, és generáltassunk hozzá *Activity*-t *MainActivity* néven. Ahhoz, hogy az operációs rendszer *Home Screen* alternatívaként lássa, az *AndroidManifest*-ben a következő *Intent Filter*t kell beállítanunk a megfelelő `<activity>` tag-ben:

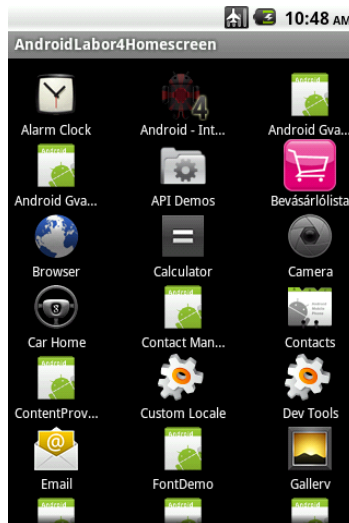
```
<activity android:name=".MainActivity"
          android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.HOME" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

Vegyük észre, hogy az Eclipse által alapból beleírt *LAUNCHER* kategóriát töröltük, ugyanis az alkalmazás listában nem feltétlenül kell megjelennie egy főképernyő jellegű alkalmazásnak. Helyette a *HOME* és a *DEFAULT* bejegyzésekkel értesítjük az Androidot arról, hogy az *Activity*-nk képes alapértelmezett főképernyőként viselkedni. Telepítsük az alkalmazást, majd nyomjuk meg a *Home* gombot. Ekkor a rendszer választást kínál, ahol már szerepel az appunk:



2.5.2 Az alkalmazás váza

Hozzunk létre felhasználói felületet az alkalmazásunkhoz. A célunk, hogy rács elrendezésben látszódjanak a telepített appok, így a main.xml-ben állítsunk be egy teljes képernyőt kitöltő *GridView*-t.



A beállítandó XML:

<https://gist.github.com/3812803> (main.xml szekció)

Hozzunk létre továbbá egy segédosztályt *ApplicationInfo* néven, melyben a megjelenítendő alkalmazások adatait tároljuk:

<https://gist.github.com/3812803> (ApplicationInfo.java szekció)

2.5.3 Launcher-ben megjelenítendő alkalmazások lekérése

Ahhoz hogy a főképernyőn megjeleníthessünk minden szükséges alkalmazást, a *PackageManager* osztály *queryIntentActivities()* metódusát hívjuk segítségül. Ez a kapott *Intent*-nek megfelelő összes *Activity*-t képes visszaadni egy listában, nekünk pedig éppen erre van szükségünk (*Intent* feloldást végez a háttérben). Az így visszkapott *Activity*-k adatait olvassuk be egy *ApplicationInfo* objektumokból álló tömbbe, melyet a *MainActivity* osztályban definiáltunk.

```
public class MainActivity extends Activity {  
  
    private static ArrayList<ApplicationInfo> mApplications;  
  
    ...  
}
```


A háttérben történő Intent feloldáshoz és az `ApplicationInfo` tömb feltöltéséhez csináljunk egy új módszert a `MainActivity` osztályban `loadApplications()` néven. A szükséges kód:

<https://gist.github.com/3812803> (“`MainActivity.java` [`loadApplications()` módszer]” szekció)

2.5.4 Adatkötés

Megvan tehát az megjelenítendő alkalmazások listája, illetve a felhasználói felületen lévő `GridView`. Az adatkötés megvalósításához írunk kell egy `Adapter`-t, ami feltölti a gridet az alkalmazásokkal. Ez szintén mehet belső osztályként a `MainActivity`-be.

<https://gist.github.com/3812803> (“`MainActivity.java` [saját grid adapter]” szekció)

Példányosítsuk az adaptert, és állítsuk is be (ez a módszer is a `MainActivity` osztályba kerüljön)!

```
/**
 * ApplicationsAdapter létrehozása és feltöltése,
 * kattintas esemenykezelő bekötése
 */
private void bindApplications() {
    if (mGrid == null) {
        mGrid = (GridView) findViewById(R.id.all_apps);
    }
    mGrid.setAdapter(new ApplicationsAdapter(this, mApplications));
    mGrid.setSelection(0);
    mGrid.setOnItemClickListener(new ApplicationLauncher());
}
```

A módszer hivatkozik az `mGrid` nevű változóra, melyet deklaráljunk a `MainActivity` osztály tagváltozójaként.

2.5.5 Eseménykezelő kattintásra

A felhasználói felület már végleges kinézetű, azonban alkalmazásra kattintva még nem történik semmi. Az eseménykezelőt már bekötöttük a `bindApplications()` módszerban, most írjuk is meg a `View.OnClickListener` interfészt megvalósító segédosztályt.

```
/**
 * Segedosztaly ami indítja a kattintott alkalmazást
 */
private class ApplicationLauncher implements
AdapterView.OnItemClickListener {
    public void onItemClick(AdapterView parent, View v, int position,
long id) {
        ApplicationInfo app = (ApplicationInfo)
parent.getItemAtPosition(position);
        startActivity(app.intent);
    }
}
```

2.5.6 Befejezés

Az alkalmazásunk minden komponense rendelkezésre áll, a működéshez már csak meg kell hívunk a megírt metódusokat a *MainActivity onCreate()* függvényében:

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

    loadApplications();
    bindApplications();
}
```

Telepítsük az alkalmazást, majd próbáljuk ki a működését!

A forráskód az Android hivatalos minta alkalmazások között megtalálható Home appból származik, ami több funkciót is megvalósít egy teljesértékű HomeScreen-hez. További ötletek és példakód meríthető innen:

<http://developer.android.com/resources/samples/Home/index.html>