

Hello Android! @ SoftDevCon 2011

Konrad Malawski
konrad.malawski@llp.pl
Lunar Logic Polska

17 listopada 2011

Hello!

lunar logic polska



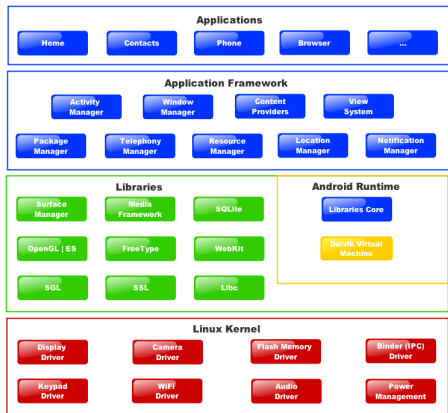
[sckrk]



Konrad Malawski
github: ktoso
twitter: ktosopl

You?

Architektura



Rysunek: Architektura systemu Android

Słowa na dziś:

- ▶ Application
- ▶ Activity
- ▶ View
- ▶ Service
- ▶ Intent
- ▶ _____ Manager

The „Glue”: AndroidManifest.xml



AndroidManifest.xml

```
<manifest xmlns:android="http://..."
    package="pl.project13.android.github.notify"
    android:versionCode="1"
    android:versionName="1.0">

    <application android:name=".guice.GitHubNotifyApplicat
        android:label="@string/app_name"
        android:icon="@drawable/icon"
        android:debuggable="true"
        android:hardwareAccelerated="true">

        <activity android:name=".activity.WatchPreferencesA
            android:icon="@drawable/icon"
            android:label="GitHub_Notify">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"
                <category android:name="android.intent.category.
            </intent-filter>
        </activity>
```

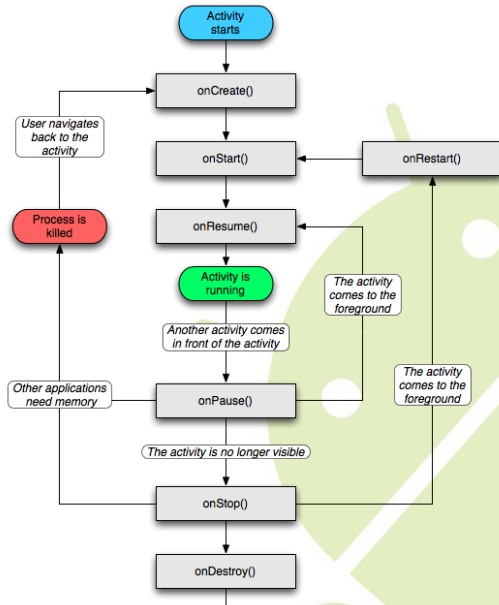

AndroidManifest.xml

```
<service android:name=".service.NewCommitNotifierService" />
</application>

<!-- PERMISSIONS -->
<uses-sdk android:minSdkVersion="6"
          android:targetSdkVersion="11"/>

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

Activity lifecycle



Activity

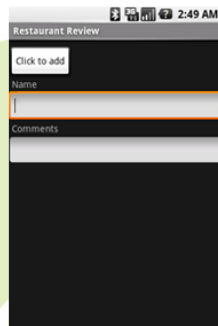
Wołany gdy Activity jest tworzona po raz pierwszy.

```
public class MainActivity extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        /**/  
    }  
}
```

LinearLayout + EditText z wrap_content

```
<LinearLayout
    android:id="@+id/root_layout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <!-- ... -->

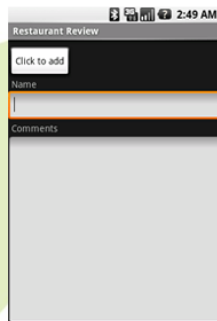
    <TextView android:id="..." ... />
    <EditText android:id="..."
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
</LinearLayout>
```



LinearLayout + EditText z fill_parent

```
<LinearLayout
    android:id="@+id/root_layout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <!-- ... -->

    <TextView android:id="..." ... />
    <EditText android:id="..."
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />
</LinearLayout>
```



Panowanie nad Layoutami

- ▶ Istnieje jeszcze **match_parent**, pojawił się w nowszych wersjach Android, jest analogiczny do **fill_parent**.
- ▶ W przypadku gdy chcemy ustalać "jaką część widoku ma zajmować pewien element", korzystamy z **android:layout_weight**, przyjmuje on liczby (*default = 1*).

Ważne ViewGroups:

- ▶ **FrameLayout** - Layout dla tylko jednego elementu, najprostszy
- ▶ **LinearLayout** - Wystarczający dla prostych widoków, kombinowanie kilku LinearLayout daje przyjemne efekty
- ▶ **ListView** - Sam dba o scrollowanie
- ▶ **TabHost** - Może mieć zakładki

Ważne ViewGroups:

- ▶ **FrameLayout** - Layout dla tylko jednego elementu, najprostszy
- ▶ **LinearLayout** - Wystarczający dla prostych widoków, kombinowanie kilku LinearLayout daje przyjemne efekty
- ▶ **ListView** - Sam dba o scrollowanie
- ▶ **TabHost** - Może mieć zakładki
- ▶ **Spinner, Gallery, GridView, RelativeLayout, ScrollView, SurfaceView, TableLayout, ViewFlipper, ViewSwitcher...**

R, as in Resource

R

R, as in Resource

```
package pl.project13;  
  
public final class R {  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class id {  
        public static final int login=0x7f050000;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    public static final class string {  
        public static final int app_name=0x7f040000;  
    }  
}
```

R, as in Resource

Dodanie elementu id **wewnątrz widoku**, poprzez **+@id/**

```
<EditText android:id="@+id/login" ... />
```

R, as in Resource

Dodanie elementu id **wewnątrz widoku**, poprzez **+@id/**

```
<EditText android:id="@+id/login" ... />
```

Spowoduje wygenerowanie pola w klasie **R**:

```
public final class R {  
    public static final class id {  
        public static final int login = 0x7f050000;  
    }  
}
```

R, as in Resource

Dodanie elementu id **wewnątrz widoku**, poprzez **+@id/**

```
<EditText android:id="@+id/login" ... />
```

Spowoduje wygenerowanie pola w klasie **R**:

```
public final class R {  
    public static final class id {  
        public static final int login = 0x7f050000;  
    }  
}
```

A skorzystamy z niego w np. **Activity**:

```
EditText mLogin = findViewById(pl.project13.R.id.login);
```

Strings as Resources

```
<TextView android:layout_height="fill_parent"  
          android:layout_width="fill_parent"  
          android:text="Witaj świecie!"  
/>
```

Mieszanie treści z layoutem jest złym pomysłem.

Strings as Resources

```
<TextView  android:layout_height="fill_parent"
           android:layout_width="fill_parent"
           android:text="@string/hello_world"
/>
```

```
<resources>
  <string name="hello_world">Witaj swiecie!</string>
</resources>
```

(Tip: W intellij piszemy *@string/hello_world* a następnie ALT+ENTER, aby utworzyć zasób w odpowiednim miejscu.)

Proste I18N (i nie tylko) dzięki klasyfikatorom

res/values-pl/strings.xml

```
<resources>  
  <string name="hello_world">Witaj świecie!</string>  
</resources>
```

res/values-en/strings.xml

```
<resources>  
  <string name="hello_world">Hello world!</string>  
</resources>
```


R, tips and tricks

- ▶ nazwy wykorzystywane dla np. ID **nie muszą** być unikalne, @+id/login raz może oznaczać ten EditText a raz TextView. Rozwiązane jest do wedle 'na czym wołany jest findViewById'.

R, tips and tricks

- ▶ nazwy wykorzystywane dla np. ID **nie muszą** być unikalne, @+id/login raz może oznaczać ten EditText a raz TextView. Rozwiązane jest do wedle 'na czym wołany jest findById'.
- ▶ Korzystamy raczej z 'notacji_z_podkresleniami_tutaj'

R, tips and tricks

- ▶ nazwy wykorzystywane dla np. ID **nie muszą** być unikalne, @+id/login raz może oznaczać ten EditText a raz TextView. Rozwiązane jest do wedle 'na czym wołany jest findViewById'.
- ▶ Korzystamy raczej z 'notacji_z_podkreśleniami_tutaj'
- ▶ W kontekście gdzie nie masz **findViewById**, skorzystaj z **android.content.res.Resources.getSystem().get_____()**

Logger

Android Logger

Logujemy przy pomocy **android.util.Log**.

Korzystamy z niego następująco:

```
class MyClass {  
    private final String TAG = getClass()  
                                .getSimpleName();  
  
    void doStuff() {  
        Log.d(TAG, "doing stuff ..."); // debug  
    }  
}
```

Android Log levels

Dostępne poziomy logowania to:

- ▶ `Log.v()` - VERBOSE
- ▶ `Log.d()` - DEBUG



Android Log levels

Dostępne poziomy logowania to:

- ▶ `Log.v()` - VERBOSE
- ▶ `Log.d()` - DEBUG
- ▶ `Log.i()` - INFO



Android Log levels

Dostępne poziomy logowania to:

- ▶ `Log.v()` - VERBOSE
- ▶ `Log.d()` - DEBUG
- ▶ `Log.i()` - INFO
- ▶ `Log.w()` - WARN



Android Log levels

Dostępne poziomy logowania to:

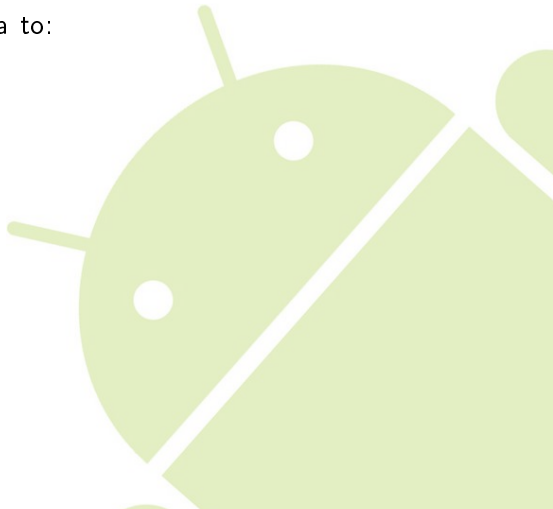
- ▶ `Log.v()` - VERBOSE
- ▶ `Log.d()` - DEBUG
- ▶ `Log.i()` - INFO
- ▶ `Log.w()` - WARN
- ▶ `Log.e()` - ERROR



Android Log levels

Dostępne poziomy logowania to:

- ▶ `Log.v()` - VERBOSE
- ▶ `Log.d()` - DEBUG
- ▶ `Log.i()` - INFO
- ▶ `Log.w()` - WARN
- ▶ `Log.e()` - ERROR
- ▶ `Log.wtf()`



Android Log levels

Dostępne poziomy logowania to:

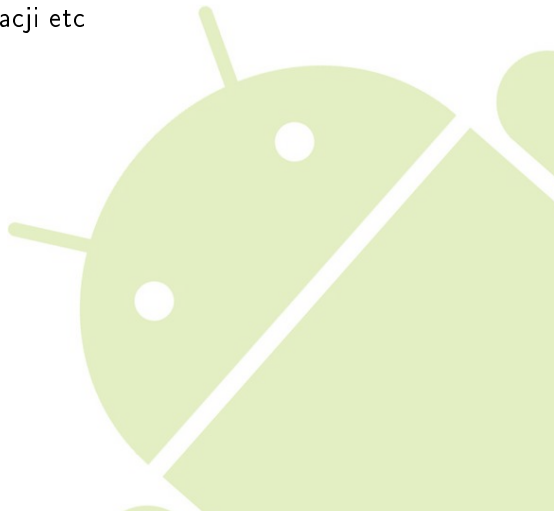
- ▶ `Log.v()` - VERBOSE
- ▶ `Log.d()` - DEBUG
- ▶ `Log.i()` - INFO
- ▶ `Log.w()` - WARN
- ▶ `Log.e()` - ERROR
- ▶ `Log.wtf()` - **W**hat a **T**errible **F**ailure

Data Storage



Sposoby przechowywania danych

- ▶ **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc



Sposoby przechowywania danych

- ▶ **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- ▶ **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnętrznej pamięci** urządzenia (dobre dla cache obrazków etc)

Sposoby przechowywania danych

- ▶ **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- ▶ **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnętrznej pamięci** urządzenia (dobre dla cache obrazków etc)
- ▶ **External Storage** - Zapisywanie plików w swoim formacie na np. **karcie SD** (dobre dla cache obrazków etc)

Sposoby przechowywania danych

- ▶ **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- ▶ **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnętrznej pamięci** urządzenia (dobre dla cache obrazków etc)
- ▶ **External Storage** - Zapisywanie plików w swoim formacie na np. **karcie SD** (dobre dla cache obrazków etc)
- ▶ **SQLite Database** - Zwyczajna instancja bazy danych SQLite - m.in. tym sposobem dostajemy informacje o kontaktach

Sposoby przechowywania danych

- ▶ **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- ▶ **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnętrznej pamięci** urządzenia (dobre dla cache obrazków etc)
- ▶ **External Storage** - Zapisywanie plików w swoim formacie na np. **karcie SD** (dobre dla cache obrazków etc)
- ▶ **SQLite Database** - Zwyczajna instancja bazy danych SQLite - m.in. tym sposobem dostajemy informacje o kontaktach
- ▶ **Cloud Storage** - Nie trzymamy danych lokalnie, pchamy i pobieramy wszystko z chmurki

Shared Preferences



SharedPreferences - API

Uzyskanie instancji:

```
// zawsze zadziala :  
Context ctx = getApplicationContext();  
  
// w Activity lub Service jest latwiej :  
// ctx = this;  
  
SharedPreferences prefs = PreferenceManager  
    .getDefaultSharedPreferences( ctx );
```

SharedPreferences - Read API

Przykład **odczytania** zmiennej:

```
// oto jak korzystać z @strings/ w Activity
String key = getString(R.string.key_sound_notif);

String s = preferences.getString(key, "undefined");

// analogicznie dla Integer/Long/Double/StringSet
```

SharedPreferences - Write API

Przykład **zapisania** zmiennej:

```
SharedPreferences.Editor editor = preferences.edit();  
editor.putString(key, "new_value");  
// analogicznie dla Integer/StringSet/Double ...  
editor.commit();
```

Shared Preferences

Shared w sensie „wewnątrz **naszej** aplikacji”, nie między wieloma.
SharedPreferences zapisywane są w:

```
/data/data/pl.project13.myapp/shared_prefs
```

Shared Preferences

Shared w sensie „wewnątrz **naszej** aplikacji”, nie między wieloma.
SharedPreferences zapisywane są w:

```
/data/data/pl.project13.myapp/shared_prefs
```

Można się tam dostać gdy się jest **root**:

```
$ abd shell
> cat /data/data/pl.project13.myapp/shared_prefs

<map>
  <string name="pass">zomg_its_plain_text</string>
  <!-- ... -->
</map>
```

Security a SharedPreferences

Wniosek jest prosty:
Szyfrujemy ważne rzeczy trzymane gdziekolwiek na komórce.

RoboGuice

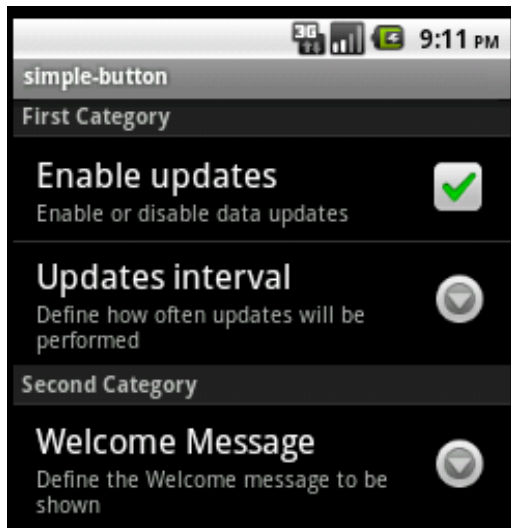


PreferenceActivity



PreferenceActivity

Ręczne edytowanie SharedPreferences szybko robi się nudne...
Wtedy właśnie korzystamy z PreferenceActivity:



/res/xml/preferences.xml

```
<PreferenceScreen xmlns:android="http://...">
  <PreferenceCategory
    android:title="Ustawienia Usera"
    android:key="user_preferences">

    <CheckBoxPreference
      android:key="@string/pref_key_notify_user"
      android:summary="Powiadamiasz użytkownika"
      android:title="Włącz powiadomienia"
      android:defaultValue="true"
    />

  </PreferenceCategory>
<!-- ... -->
```

/res/xml/preferences.xml

```
<!-- ... -->
<PreferenceCategory
    android:title="Ustawienia_pozostale"
    android:key="other_preferences">

    <EditTextPreference
        android:key="@string/pref_key_welcome_msg"
        android:title="Wiadomosc_powitalna"
        android:summary="Wiadomosc_jaka_zostanie
        powitany_uzytkownik"
        android:dialogTitle="Wiadomosc_powitalna"
        android:dialogMessage="Wpisz_wiadomosc:"
        android:defaultValue="Jak_sie_masz," />
    </PreferenceCategory>
</PreferenceScreen>
```

PreferenceActivity - implementacja

W przeciwieństwie do powyższych 2 slajdów xml, tutaj kodu jest malutko:

```
public class SettingsActivity
    extends PreferenceActivity {

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

        addPreferencesFromResource(R.xml.preferences);
    }
}
```

Jak uruchomić inną Activity?

I w tym miejscu czas poznać: Incencje

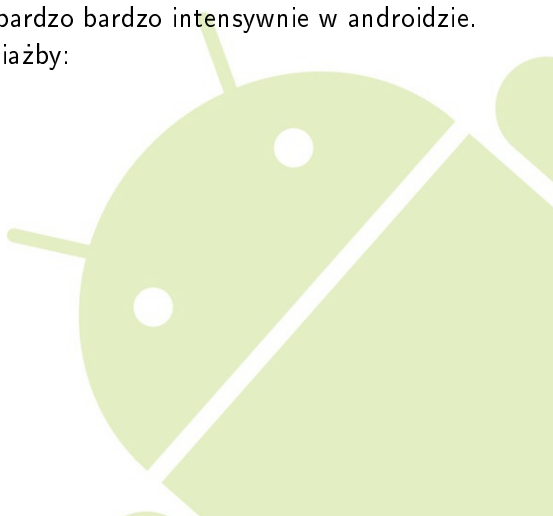
Intent



Intent - czyli „Intencja wykonania X”

Intent'y wykorzystywane są bardzo bardzo intensywnie w androidzie.
Poprzez intent działają chociażby:

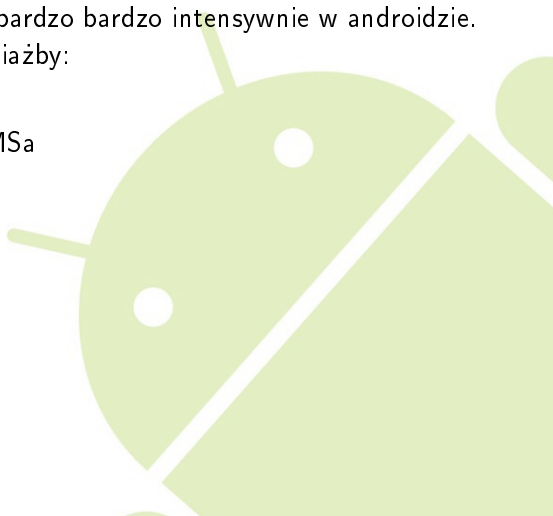
- ▶ Otworzenie linka



Intent - czyli „Intencja wykonania X”

Intent'y wykorzystywane są bardzo bardzo intensywnie w androidzie.
Poprzez intent działają chociażby:

- ▶ Otworzenie linka
- ▶ Wysłanie/odebranie SMSa



Intent - czyli „Intencja wykonania X”

Intent'y wykorzystywane są bardzo bardzo intensywnie w androidzie.
Poprzez intent działają chociażby:

- ▶ Otworzenie linka
- ▶ Wysłanie/odebranie SMSa (Intent albo przez SMSManager)

Intent - czyli „Intencja wykonania X”

Intent'y wykorzystywane są bardzo bardzo intensywnie w androidzie.
Poprzez intent działają chociażby:

- ▶ Otworzenie linka
- ▶ Wysłanie/odebranie SMSa (Intent albo przez SMSManager)
- ▶ Uruchomienie usługi (Service)

Intent - czyli „Intencja wykonania X”

Intent'y wykorzystywane są bardzo bardzo intensywnie w androidzie.
Poprzez intent działają chociażby:

- ▶ Otworzenie linka
- ▶ Wysłanie/odebranie SMSa (Intent albo przez SMSManager)
- ▶ Uruchomienie usługi (Service)
- ▶ Uruchomienie Activity

Intent - czyli „Intencja wykonania X”

Intent'y wykorzystywane są bardzo bardzo intensywnie w androidzie.
Poprzez intent działają chociażby:

- ▶ Otworzenie linka
- ▶ Wysłanie/odebranie SMSa (Intent albo przez SMSManager)
- ▶ Uruchomienie usługi (Service)
- ▶ Uruchomienie Activity
- ▶ Przekazanie danych innej części aplikacji

Intent - czyli „Intencja wykonania X”

Intent'y wykorzystywane są bardzo bardzo intensywnie w androidzie.
Poprzez intent działają chociażby:

- ▶ Otworzenie linka
- ▶ Wysłanie/odebranie SMSa (Intent albo przez SMSManager)
- ▶ Uruchomienie usługi (Service)
- ▶ Uruchomienie Activity
- ▶ Przekazanie danych innej części aplikacji
- ▶ nasłuchiwanie na „**system-wide**” zdarzenia

Intent - czyli „Intencja wykonania X”

Intent = „Chciałbym zrobić X”.

Intent - czyli „Intencja wykonania X”

Intent = „Chciałbym zrobić X”.
Chciałbym otworzyć przeglądarkę www:

```
String action = Intent.ACTION_VIEW;  
Uri uri = Uri.parse("http://www.geecon.org")  
  
Intent viewIntent = new Intent(action, uri);  
startActivity(viewIntent); // uruchom
```

Intent - przykłady cd.

Chcę wysłać SMSa, przy pomocy **jakiejs aplikacji**,
która potrafi się tym zająć.

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);  
sendIntent.putExtra("sms_body", "Halo_Szczecin!");  
sendIntent.setType("vnd.android-dir/mms-sms");  
startActivity(sendIntent);
```

Intent - przykłady cd.

Chcę wysłać SMSa, przy pomocy **jakiejs aplikacji**,
która potrafi się tym zająć.

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);  
  
sendIntent.putExtra("sms_body", "Halo_Szczecin!");  
sendIntent.setType("vnd.android-dir/mms-sms");  
  
startActivity(sendIntent);
```

SMSy również można wysłać przy pomocy **SMSManager**.

Intent-Filter - „Słuchacze”

Aby „słuchać” na globalne Intent trzeba dodać w **AndroidManifest.xml**:

```
<application ...>
  <receiver android:name=".SmsReceiver">
    <intent-filter>
      <action android:name=
        "android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
  </receiver>
</application>
```

Intent-Filter - „Słuchacze”

Aby „słuchać” na globalne Intent trzeba dodać w **AndroidManifest.xml**:

```
<application ...>
  <receiver android:name=".SmsReceiver">
    <intent-filter>
      <action android:name=
        "android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
  </receiver>
</application>
```

MAIN oraz **LAUNCHER** dla Activity, definiujące główne Activity naszej aplikacji również rejestrujemy przez Intent-Filtry!

IntentReceiver - przykład dla SMS_RECEIVED

```
public class SmsReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras();
        if(bundle == null) return;

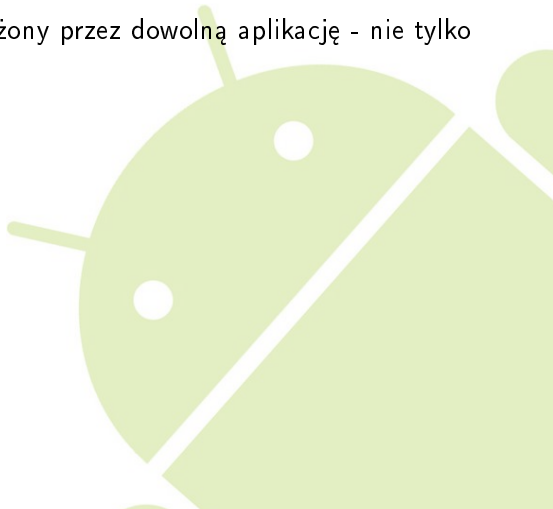
        Object[] pdus = (Object[]) bundle.get("pdus");

        for (Object aPdu : pdus) {
            SmsMessage msg;
            msg = SmsMessage.createFromPdu((byte[]) aPdu);

            String from = msg.getOriginatingAddress();
            String body = msg.getMessageBody().toString();
            Log.i(TAG, format("%s: %s", from, body));
        }
    }
}
```

Intent - fun facts

- ▶ **Intent** może być obsługiwany przez dowolną aplikację - nie tylko „nasze”



Intent - fun facts

- ▶ **Intent** może być obsługiwany przez dowolną aplikację - nie tylko „nasze”
- ▶ W przypadku gdy **Intent** natrafi 2+ „Recievery”, pyta użytkownika którego ma użyć. Przykład:

Intent - fun facts

- ▶ **Intent** może być obsługiwany przez dowolną aplikację - nie tylko „nasze”
- ▶ W przypadku gdy **Intent** natrafi 2+ „Recievery”, pyta użytkownika którego ma użyć. Przykład:
I: „Otwórz ten link.”

Intent - fun facts

- ▶ **Intent** może być obsługiwany przez dowolną aplikację - nie tylko „nasze”
- ▶ W przypadku gdy **Intent** natrafi 2+ „Recievery”, pyta użytkownika którego ma użyć. Przykład:
I: „Otwórz ten link.”
A: „W Operze czy w Firefoxie?”

Intent - fun facts

- ▶ **Intent** może być obsługiwany przez dowolną aplikację - nie tylko „nasze”
- ▶ W przypadku gdy **Intent** natrafi 2+ „Recievery”, pyta użytkownika którego ma użyć. Przykład:
I: „Otwórz ten link.”
A: „W Operze czy w Firefoxie?”
- ▶ **Intent** może nieść ze sobą masę dodatkowych informacji oraz flag. Vide metody klasy Intent.

Moar* Fun with Views

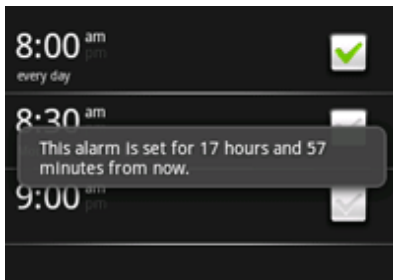
* sic

Giving Feedback

(Toasts and Dialogs)



Pyszne tosty z masłem (android.widget.Toast)



Przykład użycia:

```
Toast.makeText(getApplicationContext(), // note: explain  
                "Halo Warsaw!",  
                Toast.LENGTH_LONG)  
        .show();
```

Co więcej potrafi Toast?

```
Toast t = Toast.makeText(MyActivity.this, txt, LENGTH_S
```

Co więcej potrafi Toast?

```
Toast t = Toast.makeText(MyActivity.this, txt, LENGTH_S
```

Można mu zmienić pozycję:

```
t.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);
```


Co więcej potrafi Toast?

```
Toast t = Toast.makeText(MyActivity.this, txt, LENGTH_S
```

Można mu zmienić pozycję:

```
t.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);
```

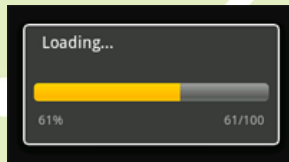
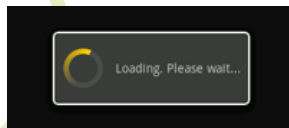
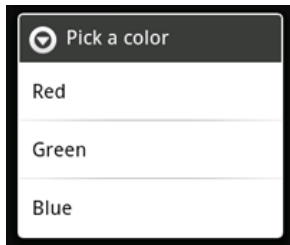
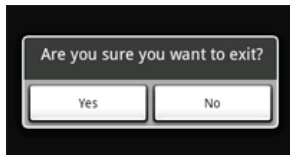
lub podmienić widok:

```
View customView = findViewById(R.id.custom_view);  
/**/  
t.setView(customView)
```

Dialog

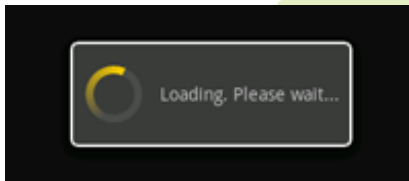


Dialog - wyskakuje 'nad' Activity



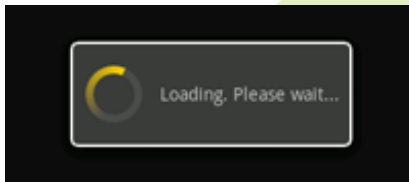
Progress Dialog (Spinning)

```
ProgressDialog dialog = ProgressDialog
    .show(MyActivity.this,
        "",
        "Loading. Please wait...",
        true);
```



Progress Dialog (Spinning)

```
ProgressDialog dialog = ProgressDialog  
                        .show(MyActivity.this ,  
                             "" ,  
                             "Loading. Please wait..."  
                             true);
```



```
dialog.hide();
```

Sloooooooooow stuff

Dotychczas siedzieliśmy na tzw. „Main Thread”.

- ▶ Zajmuje się on m.in. rysowaniem komponentów
- ▶ Jest współdzielony między Activity oraz Service!
- ▶ „Zajęcie” głównego wątku na zbyt długo spowoduje **UBICIE** naszej aplikacji!

Sloooooooooow stuff

Introducing: **LazyWorker.java**:

```
public class LazyWorker {  
    List<String> getData() {  
        sleep(10000);  
  
        return data;  
    }  
  
    void sleep(int howLong) { /**/ }  
}
```

Będzie on udawał pobieranie danych z sieci.

Fun Fact: **NetworkOnMainThreadException**

Od wersji 3.0, Android **wymusza** korzystanie z wątków celem robienia czegokolwiek związanego z siecią.

W przypadku zawołania np. `GET(''http://google.com'')`; na będąc na `MainThread`, zostanie rzucony wyjątek:

`android.os.NetworkOnMainThreadException`

GET - fikcyjna implementacja pobierająca content z sieci

Więc... `new Thread()`?

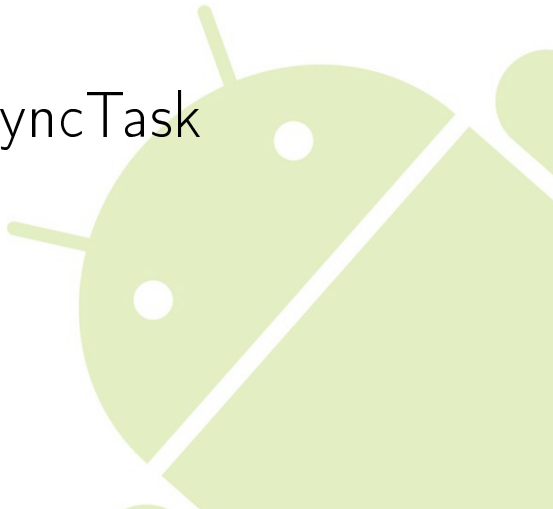
„My się wątków nie boimy!”
oświadczył dzielny rycerz.

Więc... `new Thread()`?

„My się wątków nie boimy!”
oświadczył dzielny rycerz.

Szybko jednak zmienił zdanie, znajduwszy się w paszczy
Mutexowego smoka.

AsyncTask



AsyncTask

<Params, Progress, Result>



AsyncTask<Params, Progress, Result>

Jedna z śliczniejszych abstrakcji na zadania asynchroniczne w API Androida.

Podstawowa implementacja wygląda tak:

```
new AsyncTask<Void , Void , Void>(){  
    Void doInBackground(Void ... voids){  
        /**/  
    }  
}.execute();
```

AsyncTask<Params, Progress, Result>

Jedna z śliczniejszych abstrakcji na zadania asynchroniczne w API Androida.

Podstawowa implementacja wygląda tak:

```
new AsyncTask<Void, Void, Void>(){  
    Void doInBackground(Void... voids){  
        /**/  
    }  
}.execute();
```

Anyone remember **java.lang.Void**? :-)

AsyncTask<Params, Progress, Result>

Typowe zastosowanie, „pobieracz” danych:

```
List<String> datas =  
    new AsyncTask<Void, Integer, List<String>>() {  
        @Override  
        protected List<String> doInBackground(Void... voids)  
            return /* get stuff from the internet */;  
    }  
}.execute() // not blocking  
.get(); // blocking
```

AsyncTask<Params, Progress, Result>

Typowe zastosowanie, „worker” + „zestaw danych”:

```
String[] data = getData();

new AsyncTask<String, Integer, Void>() {

    protected void onPreExecute(){
        Log.i(TAG, "Warning, will download the internet!")
    }

    protected List<String> doInBackground(Void... voids) {
        return /* get stuff from the internet */;
    }

    protected void onPostExecute(Void result) { // result
        Log.i(TAG, "Wow, we've downloaded the web!")
    }
}.execute(data); // varargs!
//.execute("a", "b", "c", "d"); // przypomnienie
```


AsyncTask + ProgressDialog

```
final ProgressDialog d
    = new ProgressDialog(MyAct.this);
d.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
d.setMessage("Loading details ...");
d.setCancelable(false);
```

AsyncTask + ProgressDialog

```
Handler handler = new Handler();

final List<String> finalData = data;
progressDialog.setMax(data.size());

new AsyncTask<String, Integer, Void>() {
    @Override
    protected void onPreExecute() { /**/ }

    @Override
    protected void onPostExecute(Void aVoid) { /**/ }

    @Override
    protected Void doInBackground(String... strings) { /**/

    @Override
    protected void onProgressUpdate(Integer... values) { /
}.execute(data);
```

AsyncTask + ProgressDialog

Simple version:

```
@Override  
protected void onPreExecute() {  
    progressDialog.show();  
}
```

progressDialog musi być zadekladowany final powyżej.

AsyncTask + ProgressDialog

Przy pomocy statycznego pomocnika:

```
ProgressDialog dialog;  
  
@Override  
protected void onPreExecute() {  
    this.dialog = ProgressDialog  
        .show( TasksActivity.this ,  
            "Loading" ,  
            "Loading▯details ..." ,  
            true ,  
            false );  
}
```

Unikamy zaśmiecania scope powyżej finalną zmienną.
Dałoby się również tutaj tradycyjnym **new** zrobić to samo.

AsyncTask + ProgressDialog (**Overkill**, do not use)

Sposób z handlerem, na wypadek źle zbindowanego ProgressDialog z którym musimy sobie jakoś poradzić.

Nie koniecznie w tej sytuacji dobre wyjście, ale to tylko przykład:

```
@Override
protected void onPreExecute() {
    handler.post(new Runnable() {
        @Override
        public void run() {
            dialog.show();
        }
    });
}
```

Jest to o tyle ciekawe że **handlerowi** możemy wysyłać na przykład tylko proste wiadomości zamiast Runnable etc.

AsyncTask + ProgressDialog

Jedyna z omawianych metod wołana w tle (nie na „UIThread”).

```
@Override
protected Void doInBackground(String... strings) {
    int i = 1;
    for (final String data : finalDatas) {
        Details details = lazyWorker.getDetails(data);
        // ...

        publishProgress(i++);
    }

    return null;
}
```

AsyncTask + ProgressDialog

Tutaj przydaje się handler, jeśli byśmy chcieli notyfikować co właśnie „opracowuje” `doInBackground`.

```
@Override
protected Void doInBackground(String... strings) {
    int i = 1;
    for (final String data : finalDatas) {
        final Details details = lazyWorker.getDetails(data);

        handler.post(new Runnable() {
            public void run() {
                Toast.makeText(TasksActivity.this,
                    "processed:_" + details,
                    Toast.LENGTH_SHORT)
                    .show();
            }
        });
        publishProgress(i++);
    }
}
```

AsyncTask + ProgressDialog

„Publikowanie postępów”:

```
doInBackground ...    doInBackground ...  
  |                  /  |  
  | \ publishProgress /  |  
  V                        V  
onProgressUpdate    onProgressUpdate
```


AsyncTask + ProgressDialog

Odbieranie informacji o postępach, ponownie wracamy na **UIThread** tutaj.

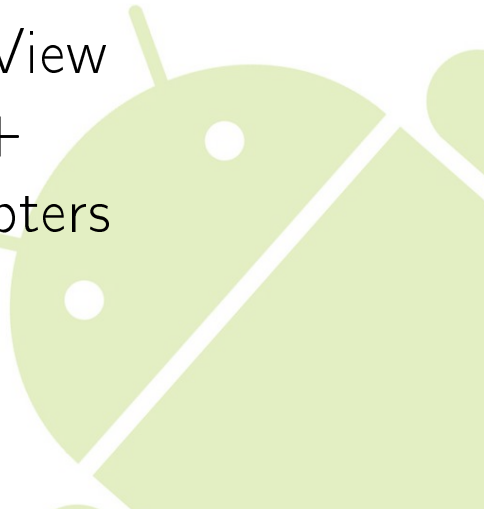
```
@Override  
protected void onProgressUpdate(Integer... values) {  
    progressDialog.setProgress(values[0]);  
}
```

AsyncTask + ProgressDialog

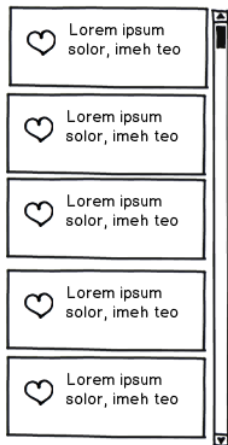
```
@Override  
protected void onPostExecute(Void aVoid) {  
    progressDialog.dismiss();  
}
```

Istnieje również **ProgressDialog#hide()**, jednak w tym przypadku chcemy zwolnić również zasoby po tym dialogu.

ListView + Adapters



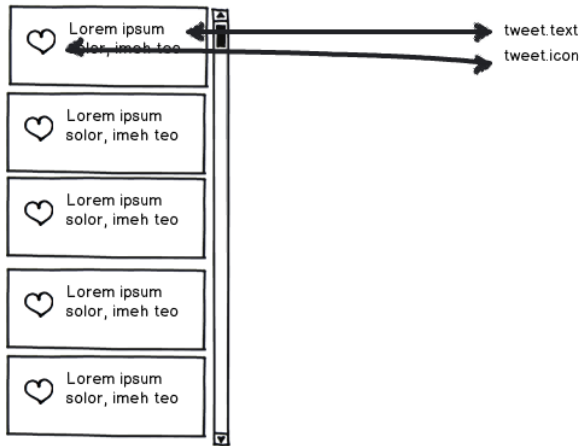
ListView Adapter



tweet.text
tweet.icon
tweet.text
tweet.icon
tweet.text
tweet.icon
tweet.text
tweet.icon

created with Balsamiq Mockups - www.balsamiq.com

ListView Adapter



created with Balsamiq Mockups - www.balsamiq.com

ListActivity

Dla ułatwienia nam sprawy, skorzystamy jeszcze z **ListActivity**.

ListActivity

Dla ułatwienia nam sprawy, skorzystamy jeszcze z **ListActivity**.
... a nawet **RoboListActivity**.

Jest wiele takich ____Activity, np. MapActivity

Przypomnienie: AndroidManifest.xml

Krótkie przypomnienie - aby **Activity** było „widziane” przez Androida, musimy je dodać do Manifestu.

W IntelliJ po prostu robimy ALT-INSERT > Android Component, XML boilerplate zostanie dodany za nas.

ListActivity

```
class MyActivity extends RoboListActivity {  
    @Inject  
    CountriesResource countriesRes;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(bundle);  
        // setContentView() // not needed!  
  
        ListView lv = getListView(); // magic?  
        // ...  
    }  
}
```

ArrayAdapter

```
ArrayAdapter<String> adapter =  
    new ArrayAdapter<String>(this ,  
                             R.layout.list_row ,  
                             R.id.text1 ,  
                             /*(String[])*/ getCountries());  
setListAdapter(adapter);  
lv.setTextFilterEnabled(true);
```

OnItemClickListener

Odrobinę inaczej niż zazwyczaj, gdyż chcemy dostać widok który kliknięto:

```
lv.setOnItemClickListener(  
    new AdapterView.OnItemClickListener() {  
        public void onItemClick(AdapterView<?> parent ,  
                                View view ,  
                                int position ,  
                                long id) {  
            ListView lv = (ListView) view;  
            TextView tv = (TextView) lv.findViewById(R.id.text1)  
  
            // our friend , the toast  
            Toast.makeText(getApplicationContext() ,  
                           tv.getText() ,  
                           Toast.LENGTH_SHORT)  
                .show();  
        }  
    });
```

Done,
działający ListView :-)

Custom Adapter

Da się oczywiście implementować własne adaptery.
Zobaczmy to na przykładzie Task'a:

TaskAdapter

```
TaskAdapter ta = new TaskAdapter(BoardActivity.this,
                                   R.layout.list_item_task,
                                   tasks);
tasksListView.setAdapter(ta);
```

TaskAdapter

```
public class TaskAdapter extends ArrayAdapter<Task> {  
    public View getView(int position ,  
                        View convertView ,  
                        ViewGroup parent) {  
        View v = convertView;  
  
        if (v == null) {  
            LayoutInflater vi = getLayoutInflater();  
            v = vi.inflate(R.layout.list_item_task ,  
                          null);  
        }  
  
        Task task = tasks.get(position);  
  
        if (task != null) populateTaskView(v, task);  
  
        return v;  
    }  
}
```

populateTaskView()

```
private void populateTaskView(final View v, final Task t) {  
    TextView topText = (TextView) v.findViewById(R.id.top_  
    TextView bottomText = (TextView) v.findViewById(R.id.b_  
  
    topText.setText(task.getTitle());  
    bottomText.setText("Description:_" + task.getDescription()  
}
```


Inne rodzaje adapterów

Są różne rodzaje adapterów, najważniejsze to jednak:

- ▶ ArrayAdapter - proste listy
- ▶ CursorAdapter - kursor (z zapytania do SQLite)

Ciekawostka: „SimpleExpandableTreeAdapter”

```
new SimpleExpandableListAdapter(  
    WorkspacesAndProjectsActivity.this ,  
  
    workspaces(workspaces) ,  
    R.layout.workspaces_workspace ,  
    new String [] { "name" } ,  
    new int [] { R.id.workspace_name } ,  
  
    projectsInWorkspace(workspaces) ,  
    R.layout.workspaces_project ,  
    new String [] { "name" } ,  
    new int [] { R.id.project_name }  
);
```

Testowanie a sprawa Androida

Jest pewien problem z 'zwyczajnym testowaniem' aplikacji androidowych... Wcześniej czy później wpadnie się na:

```
java.lang.RuntimeException: Stub!
```



<http://pivotal.github.com/robolectric/index.html>

Robolectric zastępuje implementacje rzucające te wyjątki domyślnymi (return 0 / null).

Plain Old Testing

Jakby ktoś faktycznie chciał rzeźbić bardziej niskopoziomowo swoje testy, oto jak to zrobić:

`http://developer.android.com/resources/
tutorials/testing/
activity_test.html`