

Moar* Fun with Views

* sic

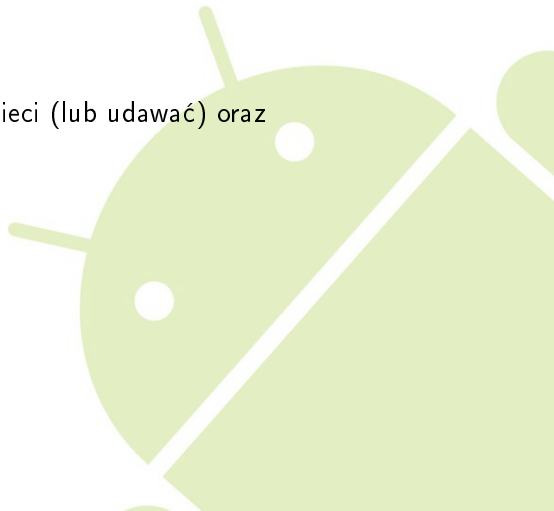
Zmierzamy w kierunku kolejnego Activity

- Dodamy nowe Activity



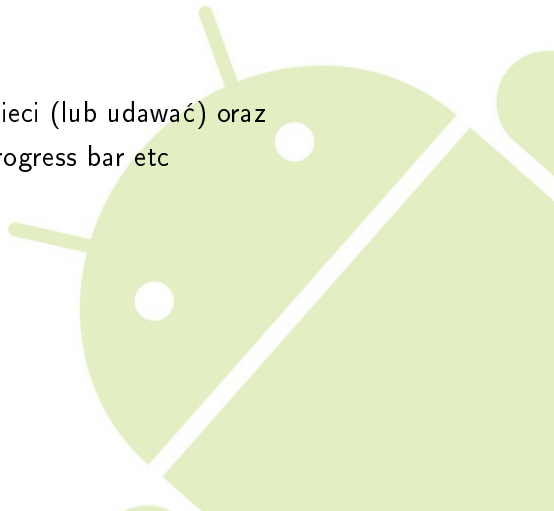
Zmierzamy w kierunku kolejnego Activity

- ▶ Dodamy nowe Activity
- ▶ Będzie pobierać coś z sieci (lub udawać) oraz



Zmierzamy w kierunku kolejnego Activity

- ▶ Dodamy nowe Activity
- ▶ Będzie pobierać coś z sieci (lub udawać) oraz
- ▶ hint: przyda się jakiś progress bar etc



Zmierzamy w kierunku kolejnego Activity

- ▶ Dodamy nowe Activity
- ▶ Będzie pobierać coś z sieci (lub udawać) oraz
- ▶ hint: przyda się jakiś progress bar etc
- ▶ utworzy z tych danych ListView

Zmierzamy w kierunku kolejnego Activity

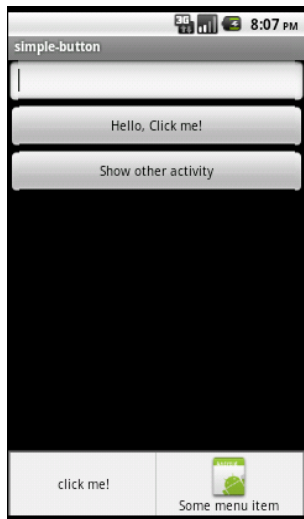
- ▶ Dodamy nowe Activity
- ▶ Będzie pobierać coś z sieci (lub udawać) oraz
- ▶ hint: przyda się jakiś progress bar etc
- ▶ utworzy z tych danych ListView
- ▶ przejdziemy do niego przez **menu** w obecnym Activity

Menu



Menu (vide przycisk menu)

Cel:




```
<menu xmlns:android="http://...">

    <item android:id="@+id/click_me_menu_item"
          android:title="click_me!"
          />

    <item android:id="@+id/some_menu_item"
          android:title="Some_menu_item"
          android:icon="@drawable/icon"
          />

</menu>
```

SomeActivity#onCreateOptionsMenu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();

    inflater.inflate(R.menu.sample_menu, menu);
    return true;
}
```

SomeActivity#onCreateOptionsMenu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();

    inflater.inflate(R.menu.sample_menu, menu);
    return true; // true == ma zostac pokazane
}
```

SomeActivity#onOptionsItemSelected

```
@Override
public boolean onOptionsItemSelected(Menulitem item) {
    int itemId = item.getItemId();

    switch (itemId){
        case R.id.click_me_menu_item:
            doSomething();
            break;
        default:
            Log.i(TAG, "Some weird action was requested");
    }

    return true;
}
```

SomeActivity#onOptionsItemSelected

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int itemId = item.getItemId();

    switch (itemId){
        case R.id.click_me_menu_item:
            doSomething();
            break;
        default:
            Log.i(TAG, "Some weird action was requested");
            return false;
    }

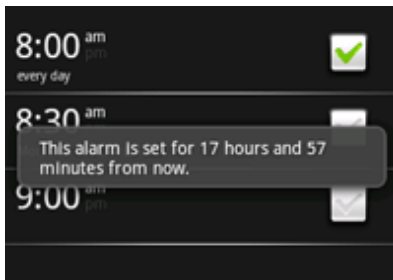
    return true; // true == obsluzylismy event
                //           == no need to bubble it
}
```

Giving Feedback

(Toasts and Dialogs)



Pyszne tosty z masłem (android.widget.Toast)



Przykład użycia:

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

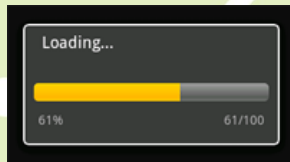
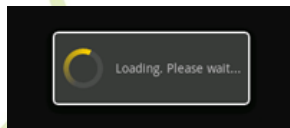
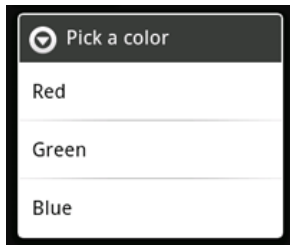
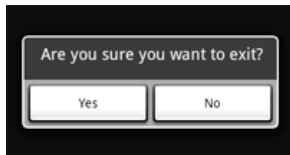
lub podmienić widok:

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

Dialog

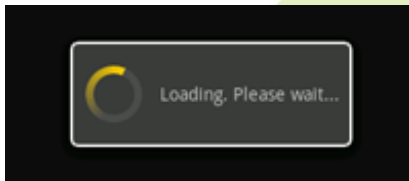


Dialog - wyskakuje 'nad' Activity



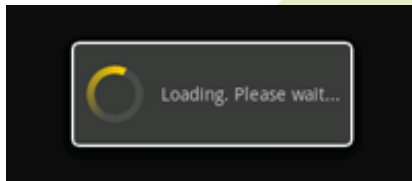
Progress Dialog (Spinning)

```
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.

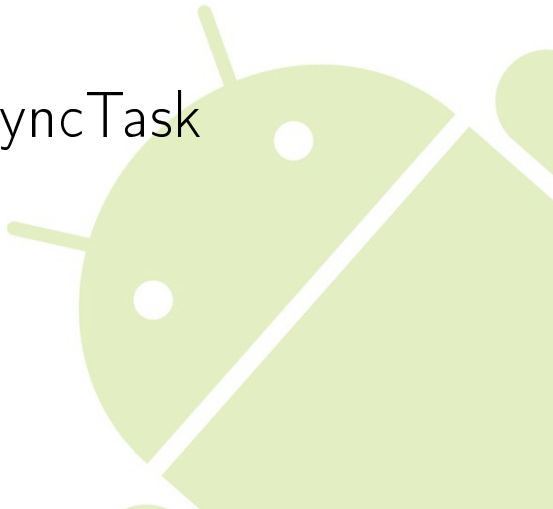


Więc... `new Thread()`?

„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 zadeklarowany 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”:

```
graph TD; subgraph AsyncTask; direction TB; A["doInBackground ..."] -- "publishProgress" --> B["onProgressUpdate"]; end; A -- "V" --> B; A -- "doInBackground ..." --> C["onProgressUpdate"]; C -- "// async" --> B;
```

The diagram illustrates the flow of progress updates within an AsyncTask. It shows two parallel paths from the `doInBackground ...` method to the `onProgressUpdate` method. The left path is a direct call, indicated by a vertical line and a 'V' symbol. The right path involves calling `publishProgress`, which then triggers `onProgressUpdate` asynchronously, indicated by a vertical line, a 'V' symbol, and the text `// async`.

AsyncTask + ProgressDialog

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

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

AsyncTask + ProgressDialog

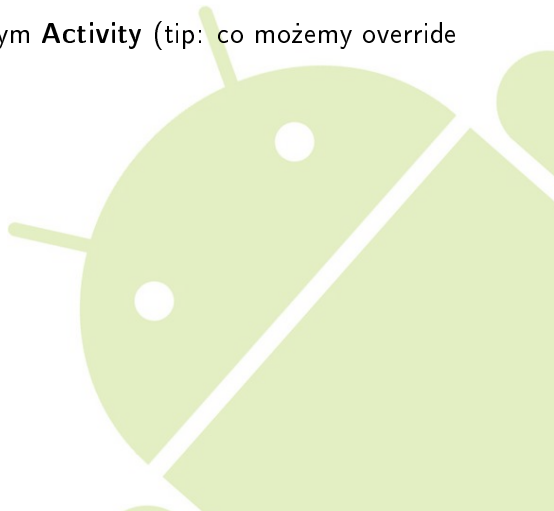
```
@Override  
protected void onPostExecute(Void aVoid) {  
    progressDialog.dismiss();  
}
```

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

Zadanko - Menu oraz nowe Activity

Za zadanie jest:

- ▶ Robimy menu w obecnym **Activity** (tip: co możemy override w Activity?)



Zadanko - Menu oraz nowe Activity

Za zadanie jest:

- ▶ Robimy menu w obecnym **Activity** (tip: co możemy override w Activity?)
- ▶ kliknięcie w jeden z przycisków ma przekierować do nowego, **ListActivity** (tip: **startActivity()**)

Zadanko - Menu oraz nowe Activity

Za zadanie jest:

- ▶ Robimy menu w obecnym **Activity** (tip: co możemy override w Activity?)
- ▶ kliknięcie w jeden z przycisków ma przekierować do nowego, **ListActivity** (tip: **startActivity()**)
- ▶ pobieramy w **onCreate()** tego Activity listę Państw (**@Inject CountriesResource**) w **AsyncTask**'u

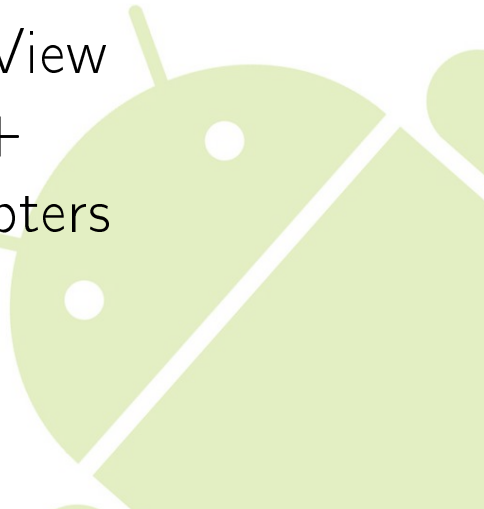
Zadanko - Menu oraz nowe Activity

Za zadanie jest:

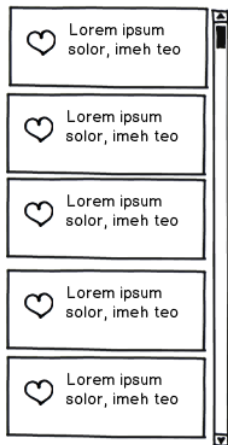
- ▶ Robimy menu w obecnym **Activity** (tip: co możemy override w Activity?)
- ▶ kliknięcie w jeden z przycisków ma przekierować do nowego, **ListActivity** (tip: **startActivity()**)
- ▶ pobieramy w **onCreate()** tego Activity listę Państw (**@Inject CountriesResource**) w **AsyncTask**'u
- ▶ Feedback przy pomocy **ProgressDialog** oraz Toastów mile widziany.

Na razie nie martwimy się o wyświetlenie tych danych jakoś.

ListView + Adapters



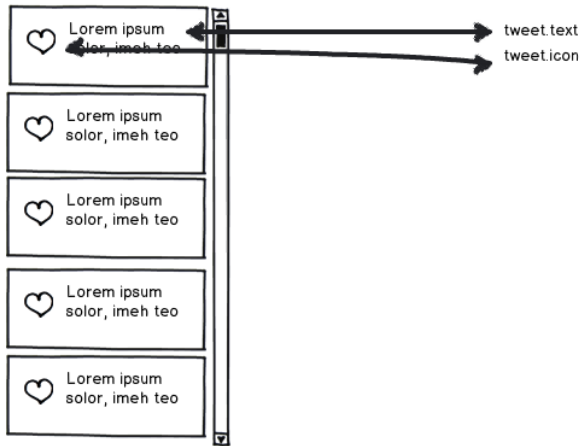
ListView Adapter



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

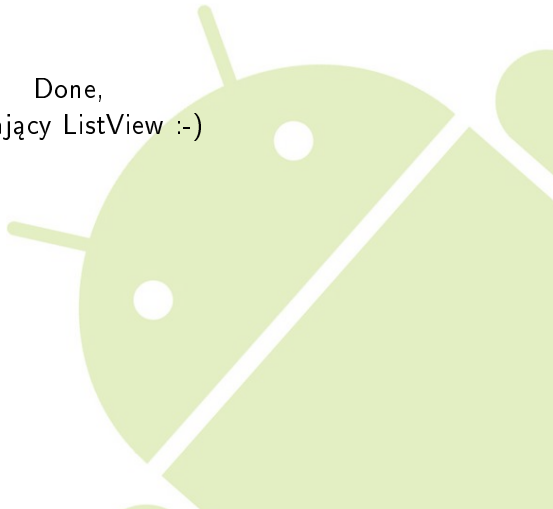
```
ArrayAdapter<String> adapter =  
    new ArrayAdapter<String>(this ,  
                             R.layout.list_row ,  
                             R.id.text1 ,  
                             /*(String[])*/ getCountries());  
setListAdapter(adapter);  
lv.setTextFilterEnabled(true);
```

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 adaptory.
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 }  
);
```

Zadanie - ListActivity

- Zmieniamy nasze Activity na ListActivity

Zadanie - ListActivity

- ▶ Zmieniamy nasze Activity na ListActivity
- ▶ Piszemy własne widoki dla niego, aby lista miała co najmniej 2 pola w jednym wierszu.

Zadanie - ListActivity

- ▶ Zmieniamy nasze Activity na ListActivity
- ▶ Piszemy własne widoki dla niego, aby lista miała co najmniej 2 pola w jednym wierszu.
- ▶ Musimy napisać własny adapter, na wzór TaskAdapttera.