Spis treści

Licencja na wykorzystanie materiałów

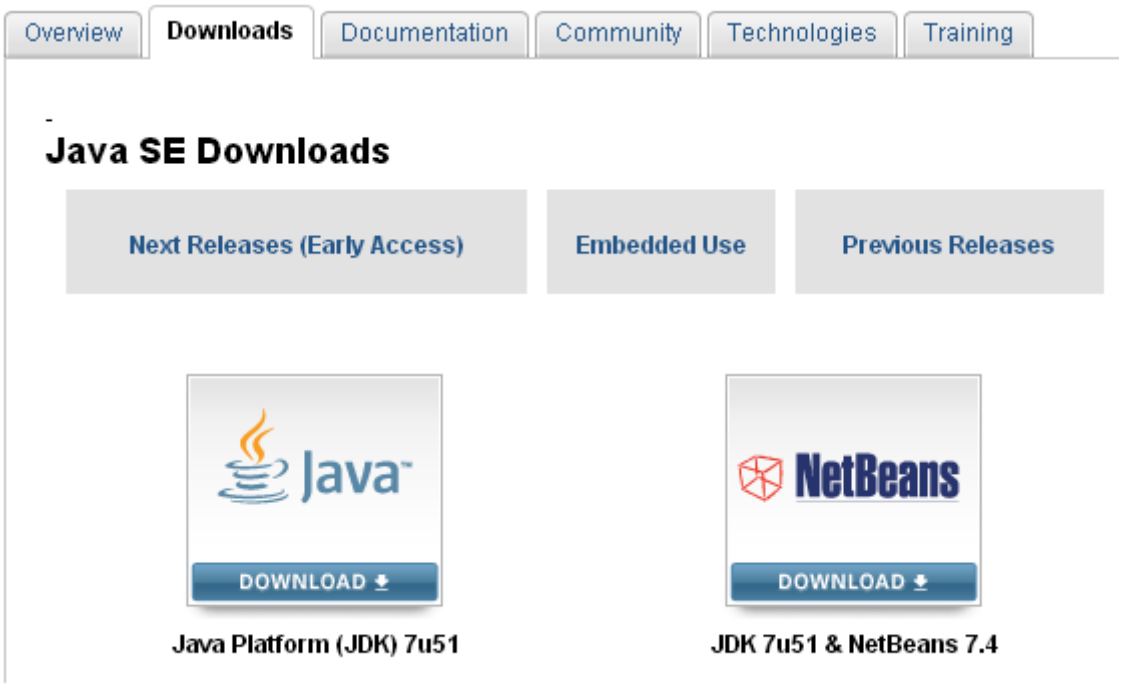
Instalacja i konfiguracja

Instalacja

Do rozpoczęcia przygody z programowaniem dla platformy Android będą nam potrzebne dwie rzeczy: Java Developer Kit, oraz zestaw Android SDK. JDK to po prostu Java z narzędziami dla programistów (np. kompilator Javy). Android SDK zawiera kompilator programów na platformę Android, narzędzia do tworzenia i zarządzania wirtualnymi urządzeniami (na których możemy testować nasze programy), a także środowisko programistyczne Eclipse. Eclipse jako taki jest bardzo popularnym IDE wśród programistów Javy, tutaj dostajemy jego nieco zmodyfikowaną pod programowanie na Androida wersję. Nic nie stoi na przeszkodzie abyśmy używali innych środowisk, ale wykorzystanie takiej spreparowanej, gotowej wersji Eclipse będzie najprostszym rozwiązaniem.

W pierwszej kolejności pobieramy i instalujemy JDK. Możemy je pobrać ze strony http://www.oracle.com/technetwork/java/javase/downloads/index.html

Wybieramy Java Platform(JDK), pierwszą od lewej. Druga różni się od pierwszej dodatkowym narzędziem – NetBeans który nie będzie nam tutaj potrzebny.



Po kliknięciu przechodzimy do ekranu wyboru wersji. Wybieramy wersję dla naszego systemu operacyjnego.

Java SE Development Kit 7u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

Dalej pobieramy Android SDK ze strony : http://developer.android.com/sdk/index.html

Get the Android SDK

The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.

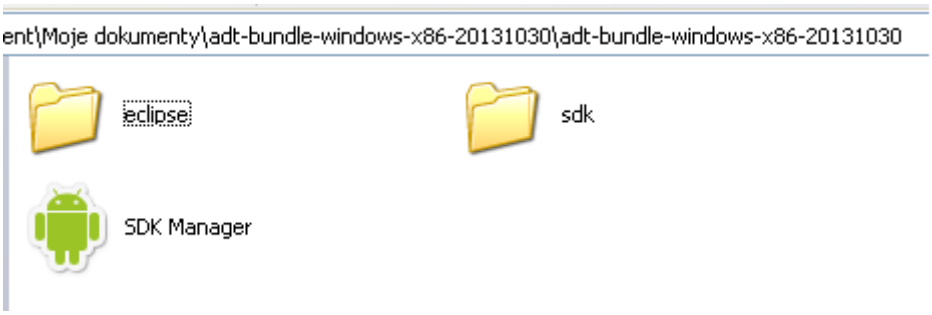
If you're a new Android developer, we recommend you download the ADT Bundle to quickly start developing apps. It includes the essential Android SDK components and a version of the Eclipse IDE with built-in **ADT (Android Developer Tools)** to streamline your Android app development.

With a single download, the ADT Bundle includes everything you need to begin developing apps:

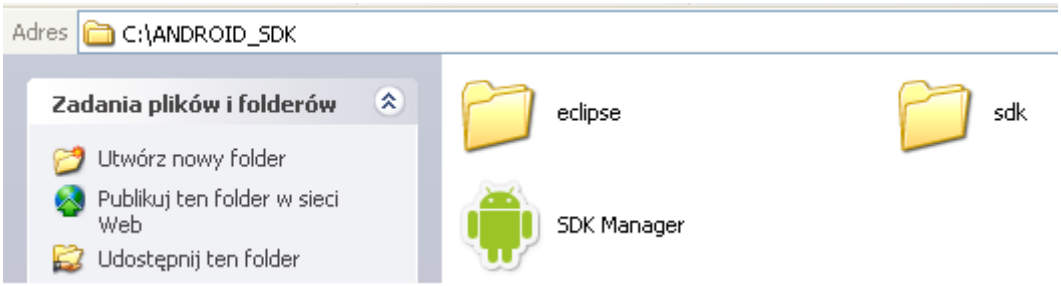
- Eclipse + ADT plugin

Download the SDK
ADT Bundle for Windows

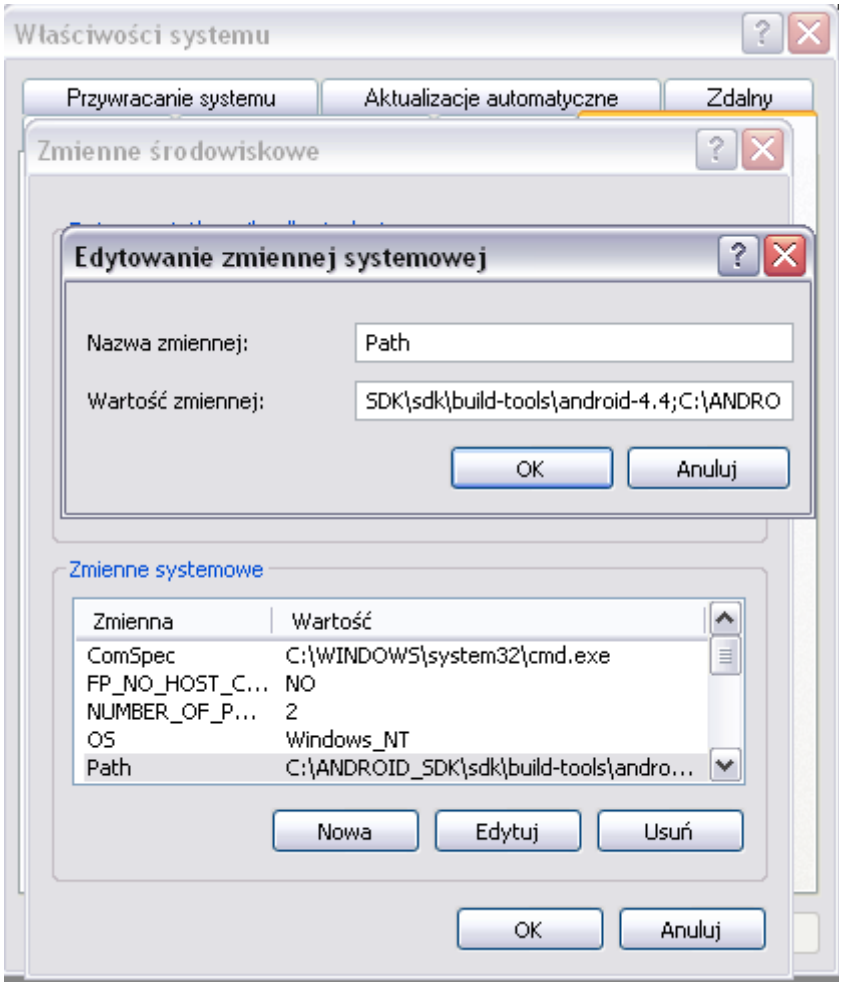
Po pobraniu i rozpakowaniu, w rozpakowanym katalogu powinniśmy zobaczyć katalogi eclipse, sdk, oraz program SDK Manager.



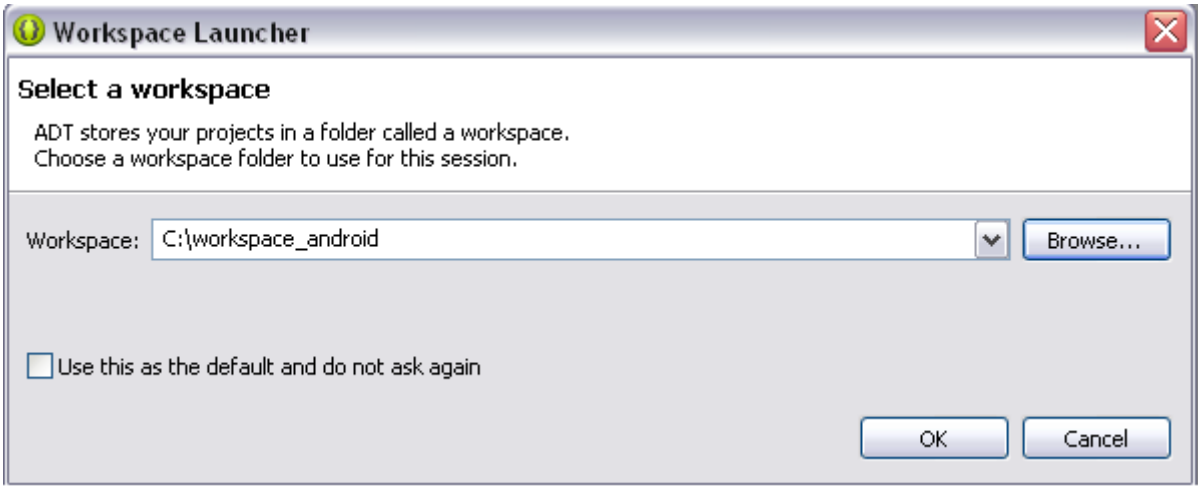
W katalogu eclipse znajduje się IDE z którego będziemy korzystać, sdk to narzędzia takie jak kompilator. Całość warto umieścić w jakimś katalogu którego nie będziemy później nigdzie przenosić, ponieważ będziemy za moment dodawać podkatalogi katalogu sdk do zmiennej środowiskowej PATH. Zmiana położenia tego katalogu wiązałaby się z koniecznością ponownej modyfikacji zmiennej środowiskowej.



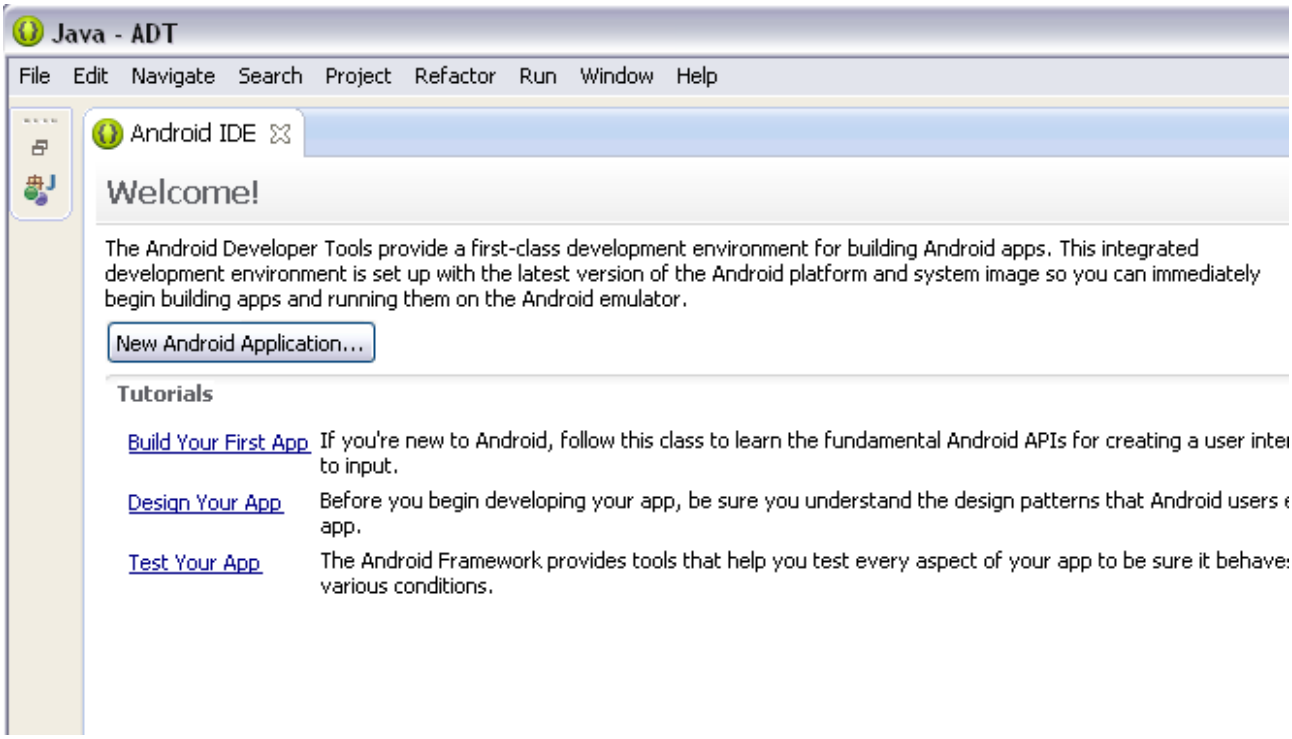
Aby ustawić zmienną środowiskową w systemie Windows, klikamy prawym przyciskiem myszy na ikonę „Mój Komputer”, wybieramy „Właściwości”. Przechodzimy do zakładki „zaawansowane” i klikamy przycisk „Zmienne środowiskowe”. Zaznaczamy zmienną Path spośród zmiennych systemowych i klikamy edytuj. Do zmiennej dopisujemy ścieżki do podkatalogów tools, platform-tools oraz build-tools katalogu jdk rozdzielając ścieżki średnikami. Zatwierdzamy zmiany.



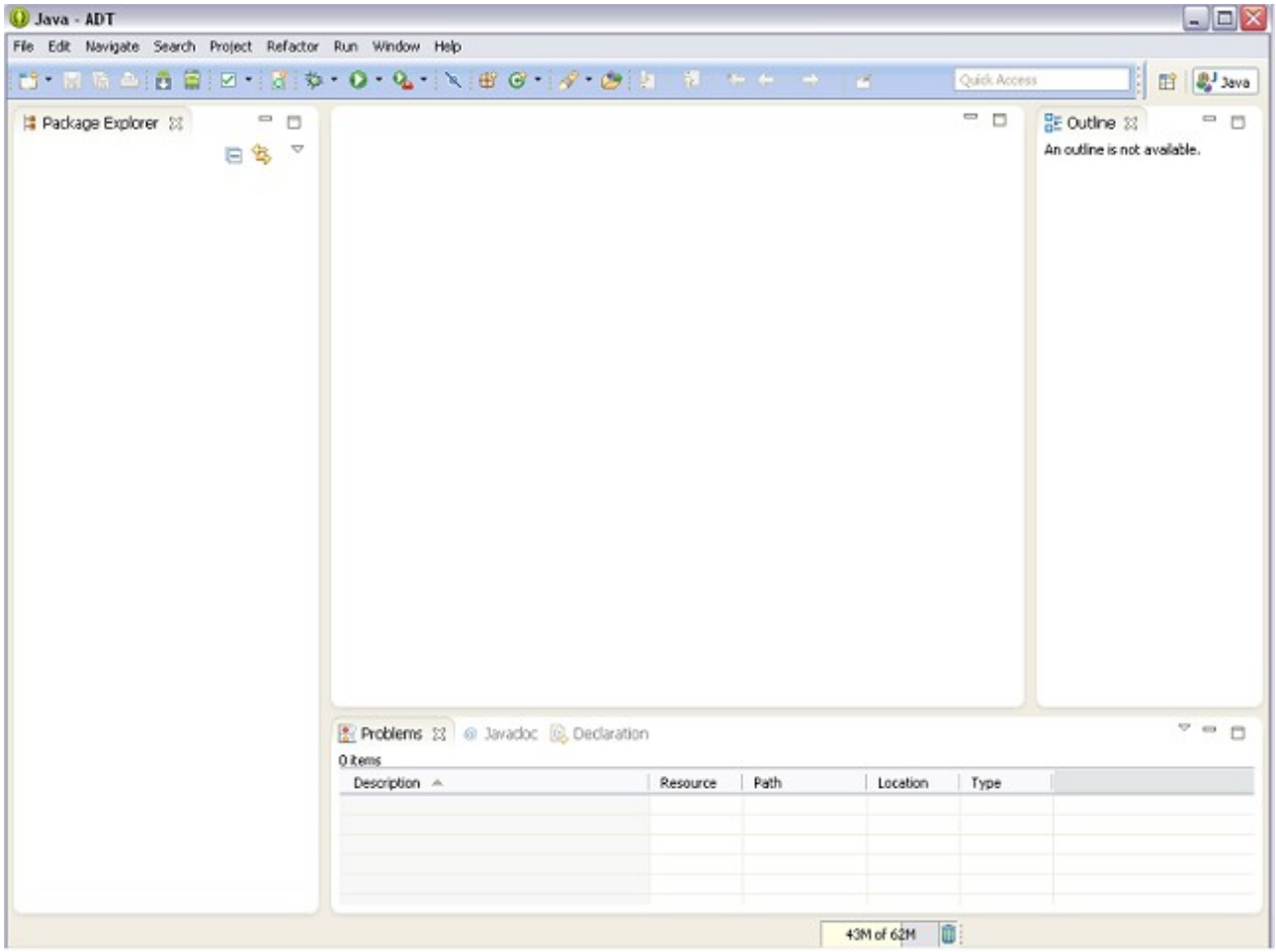
Uruchamiamy środowisko Eclipse z podkatalogu eclipse. Podczas uruchamiania program zapyta nas o położenie naszego workspace. Workspace to przestrzeń w której będzie zapisywany tworzony przez nas kod źródłowy. Oczywiście najlepiej byłoby gdybyśby wskazali jakiś katalog (możemy go utworzyć przy okazji) którego później nie będziemy przenosić.



Po pierwszym uruchomieniu powinniśmy zobaczyć taki ekran powitalny:



Zamykamy go. Powinniśmy teraz zobaczyć mniej więcej taki widok:



Gdy zaczniemy tworzyć nieco bardziej zaawansowane programy, może okazać się że domyślne ustawienia pamięci dla Eclipse są niewystarczające i środowisko zacznie chodzić bardzo wolno, wieszać się. Warto więc już na starcie zadbać o to, by Eclipse miał wystarczającą ilość pamięci. Aby to skonfigurować, edytujemy plik eclipse.ini znajdujący się w katalogu Eclipse. Modifikujemy parametry pamięciowe np. tak:

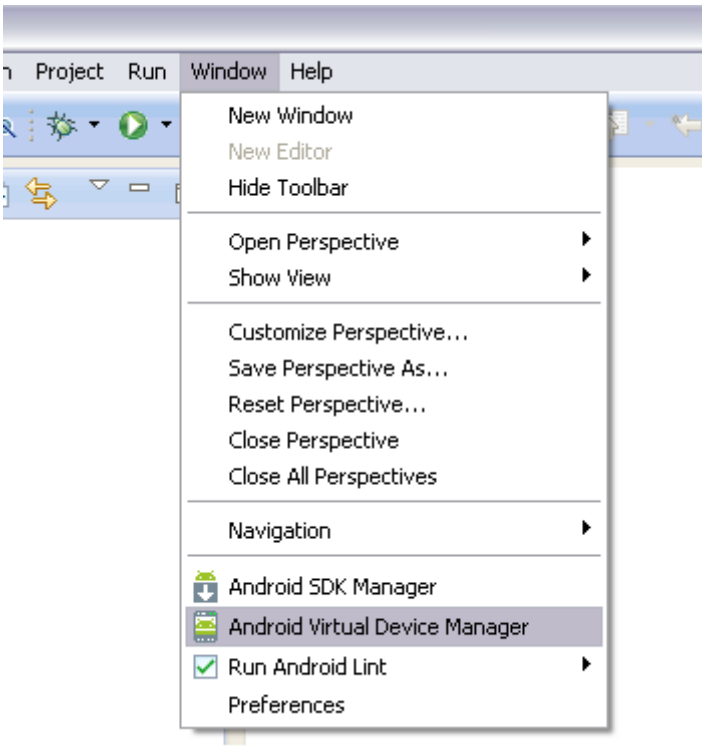
eclipse.ini

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20120522-1813.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.200.v20
-product
com.android.ide.eclipse.adt.package.product
--launcher.XXMaxPermSize
512M
-showsplash
com.android.ide.eclipse.adt.package.product
--launcher.XXMaxPermSize
512m
--launcher.defaultAction
openFile
-vmargs
-Dosgi.requiredJavaVersion=1.6
-Xms512m
-Xmx768m
-Declipse.buildId=v22.3.0-887826
-XX:MaxPermSize=768M
```

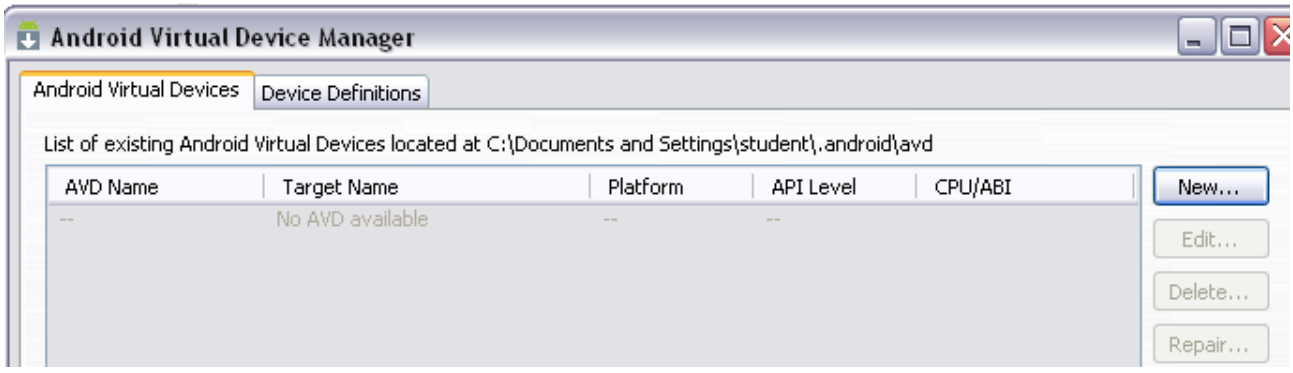
Nowe parametry będą obowiązywać on następnego uruchomienia Eclipse, warto więc go zrestartować. Efekt zmian powinien być zauważalny już teraz, w postaci różnicy czasu uruchamiania środowiska.

Konfiguracja wirtualnych urządzeń

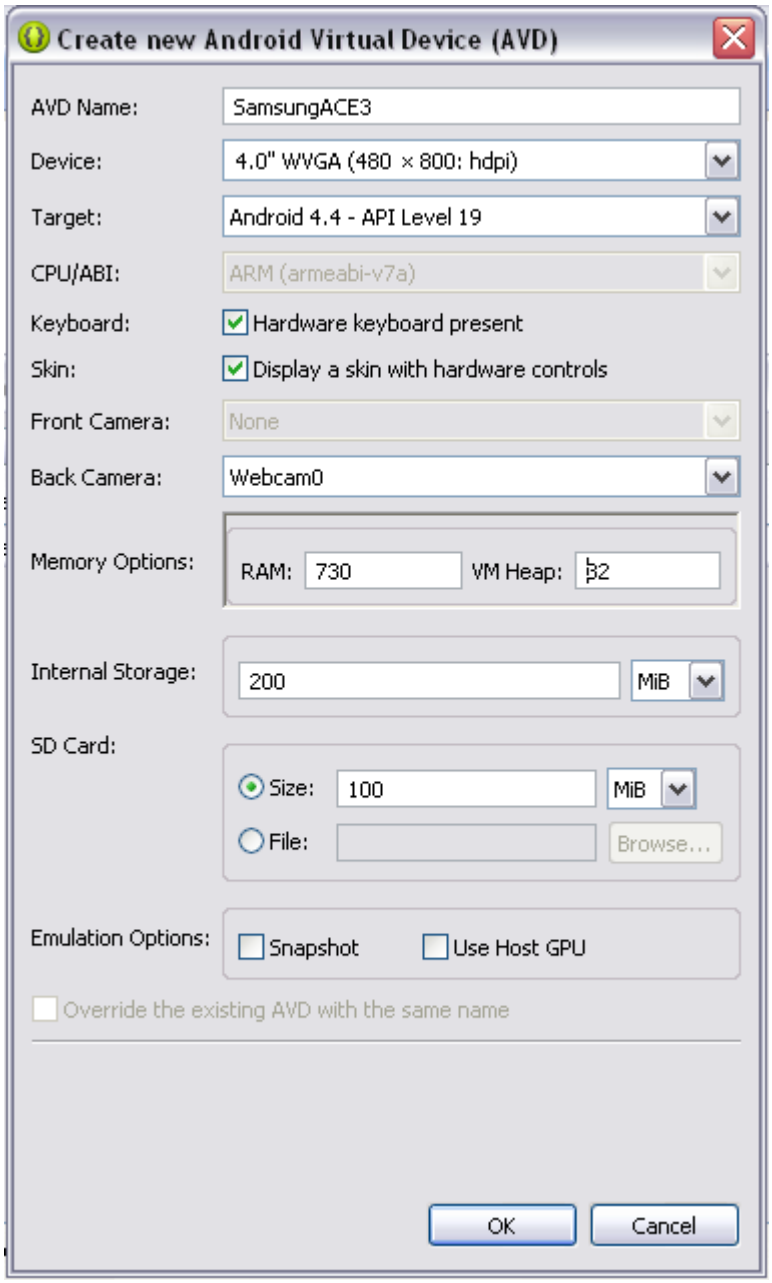
Aby testować nasze programy możemy używać realnych fizycznych urządzeń, albo wykorzystać wirtualne. Dzięki takim wirtualnym urządzeniom będziemy też mogli przetestować działanie naszego programu przy różnych konfiguracjach urządzeń i różnych wielkościach ekranu. Aby stworzyć takie wirtualne urządzenie, wybieramy z menu „window” w Elipsie element „Android Virtual Device Manager”.



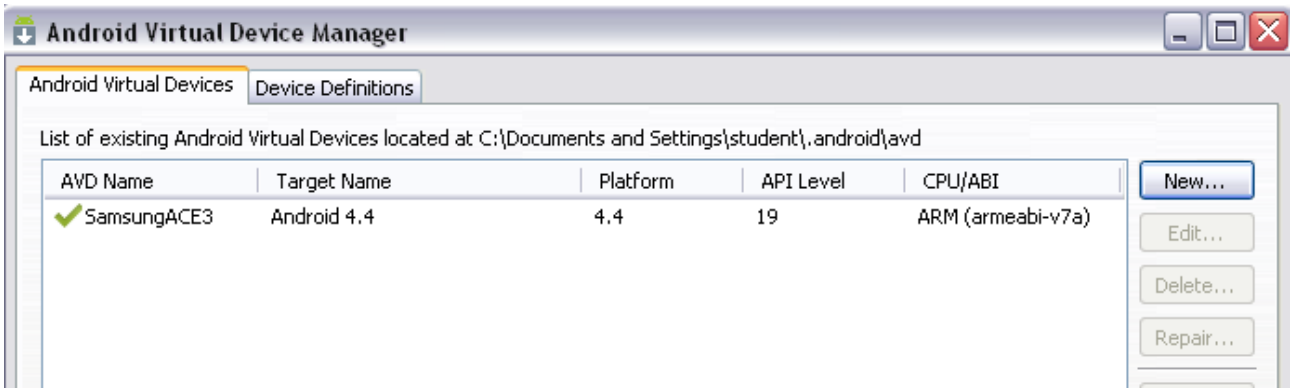
Pojawi nam się lista dostępnych wirtualnych urządzeń, tymczasowo pusta.



Klikamy przycisk „New” i wprowadzamy konfigurację urządzenia. Możesz wzorować się na tym co ja wprowadziłem. To urządzenie ma symulować model telefonu SamsungAce3 przynajmniej jeśli chodzi o konfigurację ekranu. W rzeczywistości ten telefon ma znacznie więcej pamięci operacyjnej, ale na komputerze na którym to konfiguruję jest 2GB ramu, a trzeba jeszcze obsłużyć inne programy.



Po zatwierdzeniu powinieneś zobaczyć taki widok. Takich urządzeń możemy stworzyć dowolną ilość i np. testować program na różnych wielkościach ekranu.



Struktura katalogów w aplikacji

Przyjrzyjmy się zawartości katalogów naszej aplikacji.

Katalog src zawiera klasy które będziemy tworzyć do obsługi działania naszego programu, przykładowo klasy aktywności.

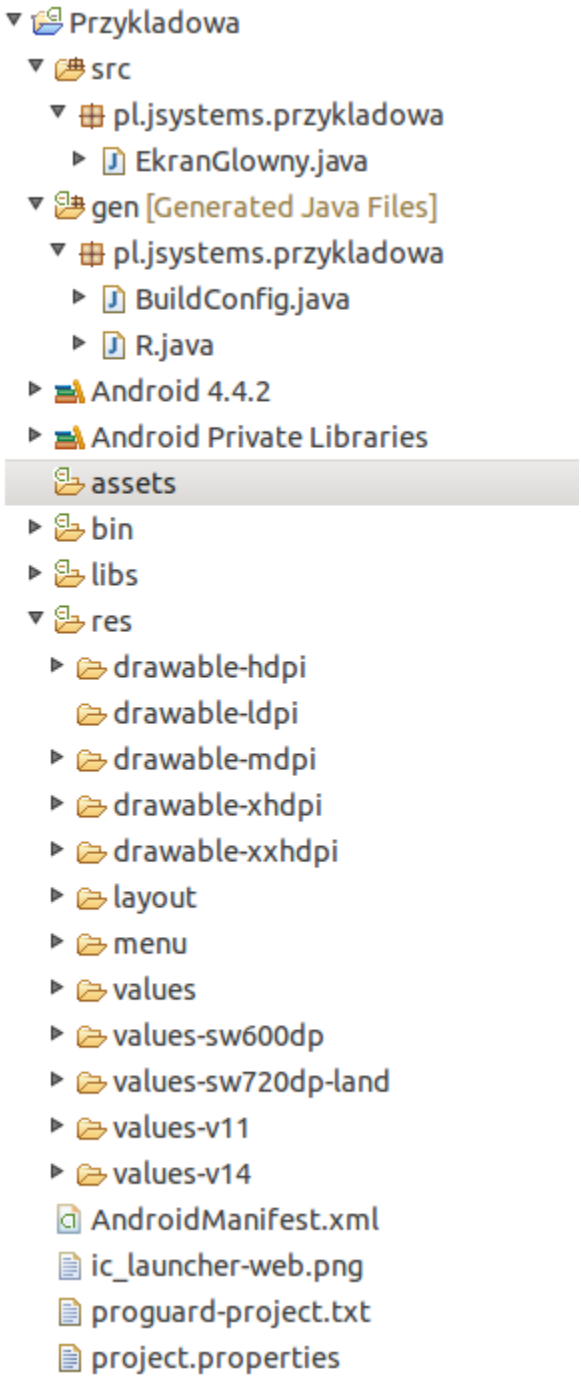
Katalog gen zawiera elementy wynegerowane przez kompilator. Położenie elementów określamy w plikach Xml, zasoby typu obrazki czy dźwięki będziemy wrzucać jako pliki. Program po skompilowaniu nie będzie zawierał tych elementów w postaci plików takich jakie wrzucimy, a pod postacią pliku R.java który jest zrozumiały dla środowiska uruchomieniowego. Generalnie zawartości tego katalogu nie powinniśmy modyfikować.

Katalog bin zawiera skompilowane klasy i naszą aplikację w postaci gotowego archiwum aplikacji – apk. Zawartości tego katalogu też nie ruszamy.

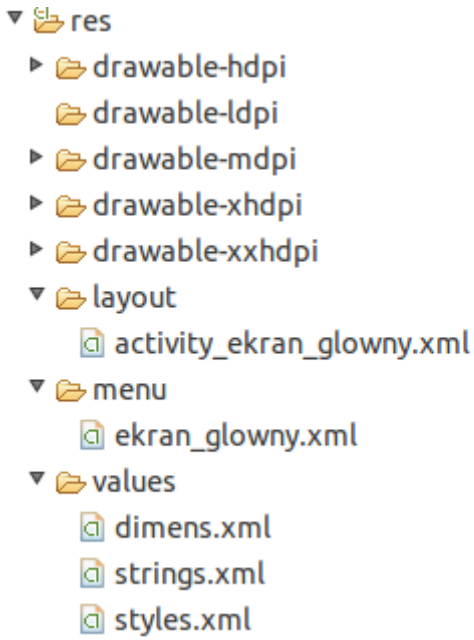
Katalog libs zawiera niezbędne do działania programu biblioteki. Możemy tam wrzucić np. jary google maps jeśli będziemy z nich korzystać.

Katalog res (od resources) wszystkie statyczne zasoby – obrazki, pliki dźwiękowe, video etc.

Widzimy też „wolno leżący” plik AndroidManifest.xml w którym znajdują się ustawienia takie jak określenie która aktywność (ekran) jest głowna, własności całej aplikacji jako takiej (typu ikonka etc), uprawnienia jakie są wymagane przez program (np. do korzystania z internetu czy odbiornika GPS).


Przykladowa
src
pl.jsystems.przykladowa
EkranGlowny.java
gen [Generated Java Files]
pl.jsystems.przykladowa
BuildConfig.java
R.java
Android 4.4.2
Android Private Libraries
assets
bin
libs
res
drawable-hdpi
drawable-ldpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
layout
menu
values
values-sw600dp
values-sw720dp-land
values-v11
values-v14
AndroidManifest.xml
ic_launcher-web.png
proguard-project.txt
project.properties

Przyjrzyjmy się jeszcze zawartości katalogu res. Cokolwiek wrzucisz do tego katalogu, stanie się do zasobem widocznym i dostępnym z dowolnego miejsca aplikacji. To właśnie tutaj powinny trafić wszystkie obrazki, filmiki etc. Katalogi drawable-xxxxx są przeznaczone na obrazki. Jest ich kilka bo możemy mieć kilka wersji tego samego obrazka dla różnych rozdzielczości. Nazwy są dosyć intuicyjne – do katalogu drawable-ldpi wrzucamy obrazki przeznaczone dla małych rozdzielczości - low dot per inch. Do drawable-hdpi wrzucamy obrazki przeznaczone dla dużych rozdzielczości - high dot per inch. W katalogu layout mamy pliki xml określające wygląd poszczególnych aktywności. W katalogu menu mamy pliki określające wygląd menu na różnych ekranach.

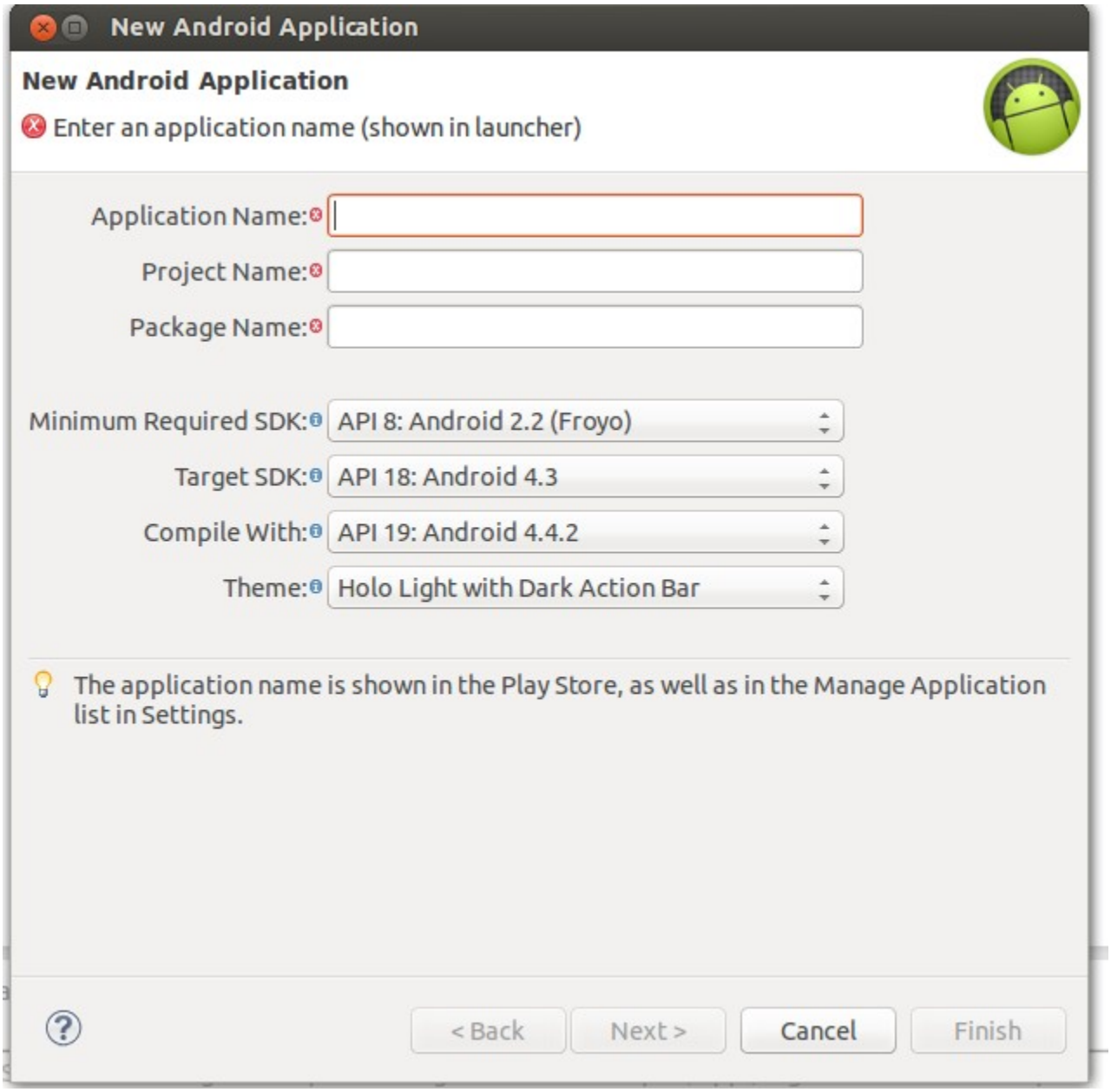


W katalgou values mamy np. plik strings.xml. Tutaj określamy wszelkie napisy które pojawiają się w aplikacji. Napisy takie można oczywiście ustawiać i w kodzie javy, jednak jeśli będziemy je umieszczać w tym pliku, będzie nam łatwiej później zarządzać napisami mając je w jednym miejscu. Przechowywanie napisów w osobnym pliku XML ma szczególne znaczenie dla wygody późniejszej internacjonalizacji programu.

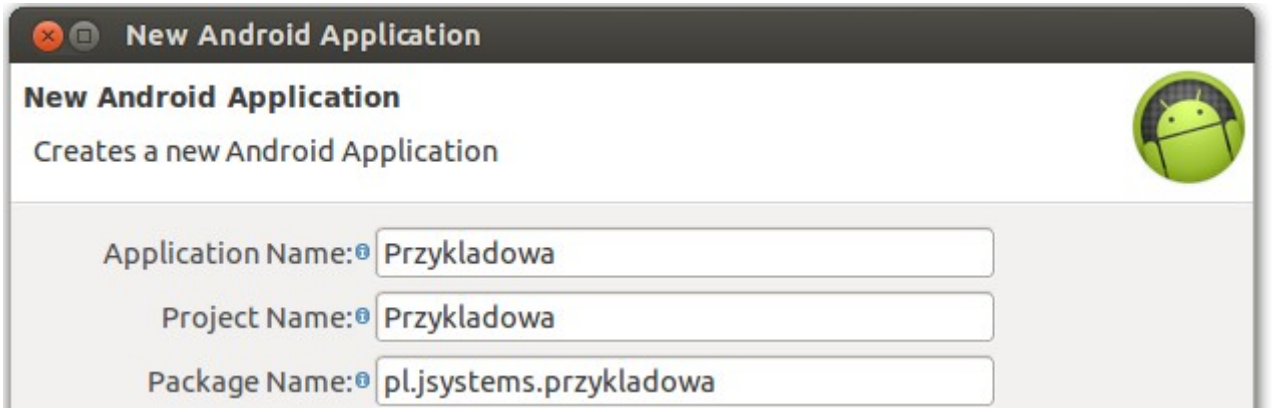
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">Przykladowa</string>
5      <string name="action_settings">Settings</string>
6      <string name="hello_world">Hello world!</string>
7
8  </resources>
9
```

Tworzenie podstawowej aplikacji

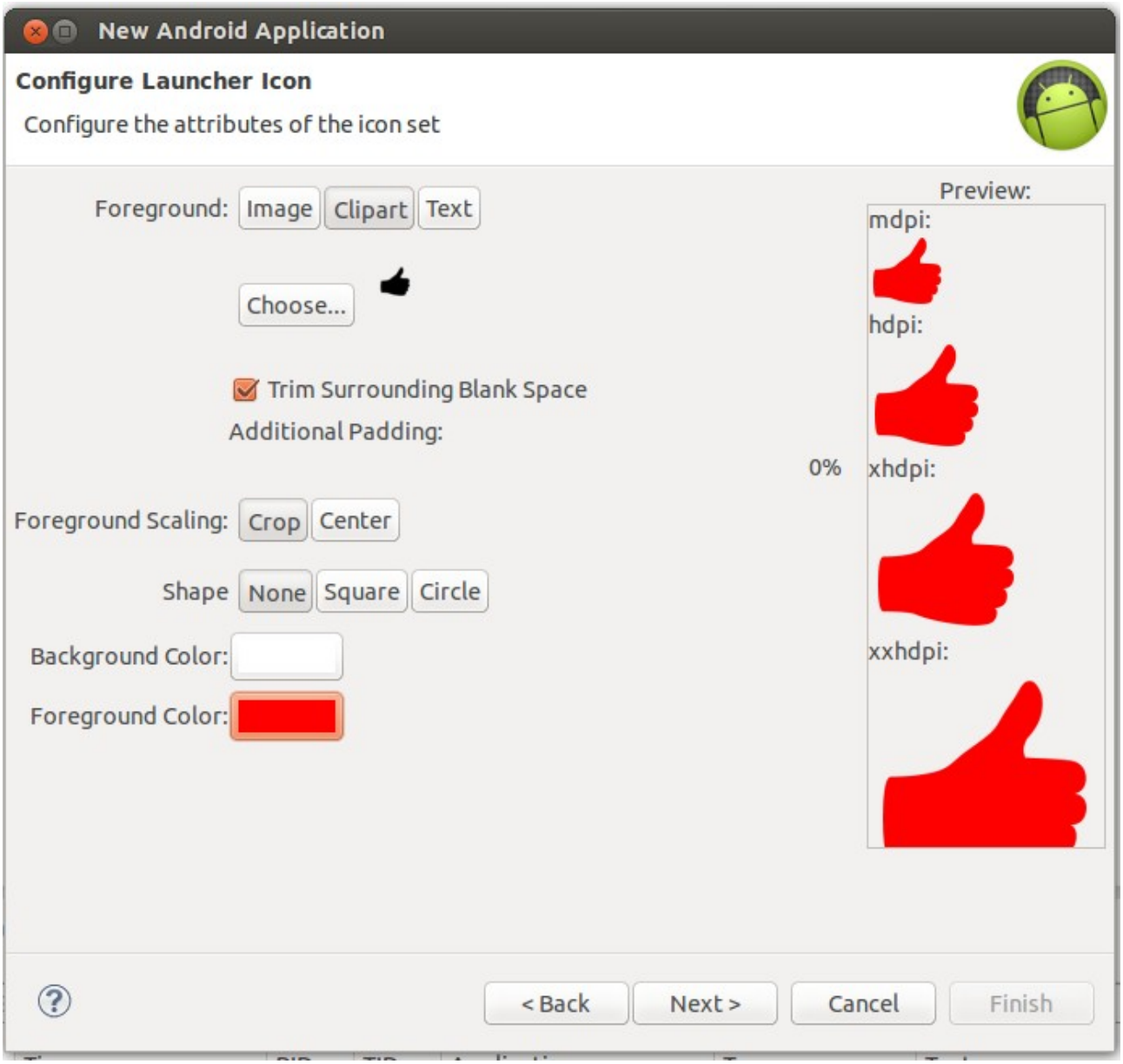
Tworzenie aplikacji rozpoczynamy od kliknięcia prawym przyciskiem myszy na znajdującym się po lewej stronie polu „package explorer” i wybraniu New-->Android Application. Powinien nam się pojawić taki ekran:



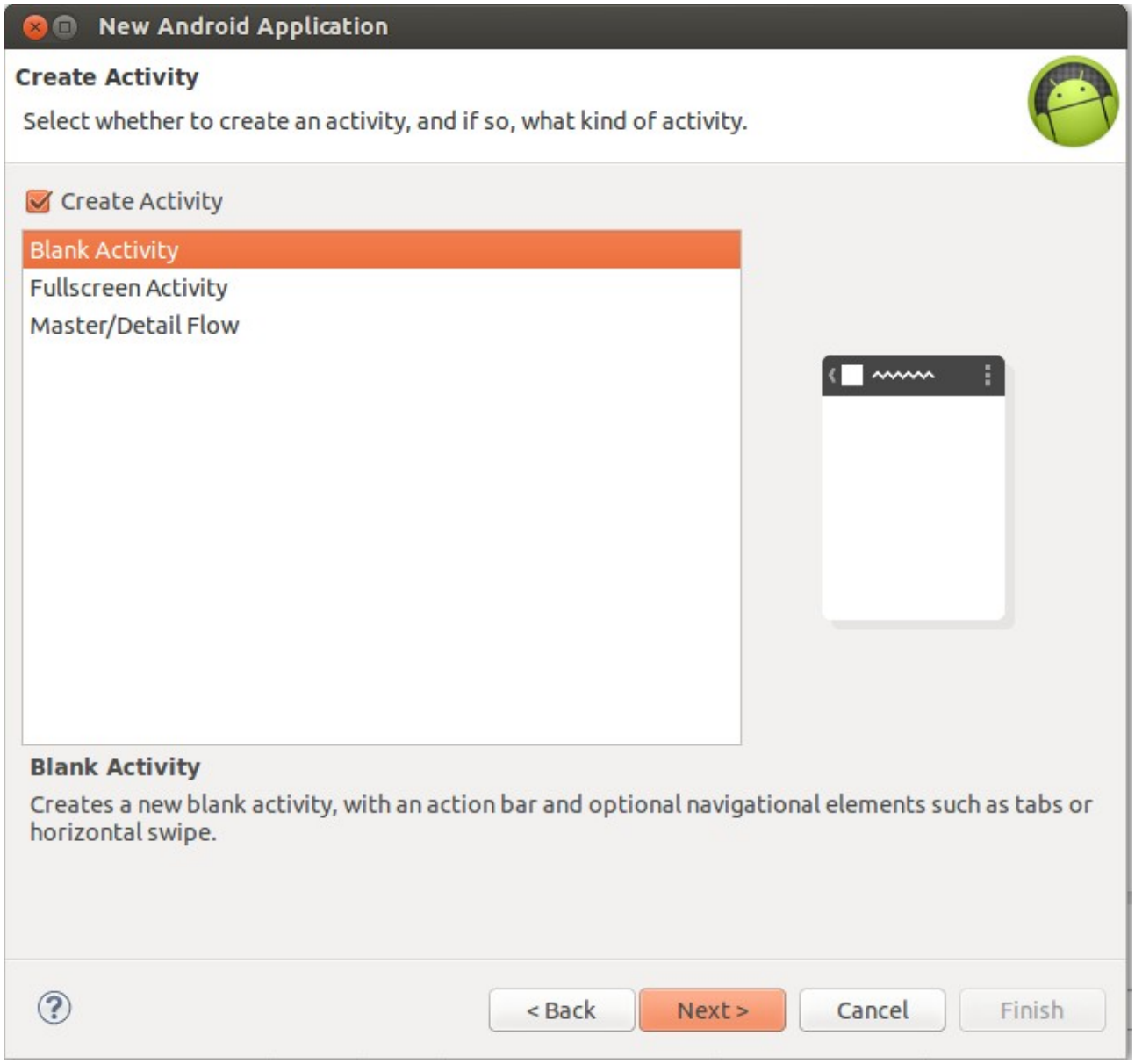
W tym miejscu musimy nadać nazwy aplikacji, projektowi oraz pakietowi w którym będą znajdować się nasze klasy. Nazwa projektu najlepiej by była taka jak nazwa aplikacji (choć nie jest to konieczne). Jeśli posiadasz domenę internetową, to najlepiej będzie odwrócić frazę i na końcu dodać nazwę nowo tworzonej aplikacji. Przykładowo jeśli mam domenę jsystems.pl i tworzę aplikację „KebabFinder” to tworzę pakiet pl.jsystems.kebabfinder. Taka jest powszechna praktyka i zapewnia unikalność nazw. Nie dojdzie do sytuacji że będą dwie klasy KlasaGlowna znajdujące się w pakietach o takiej samej nazwie com.example, choć będą pochodzić od różnych dostawców i robić różne rzeczy.



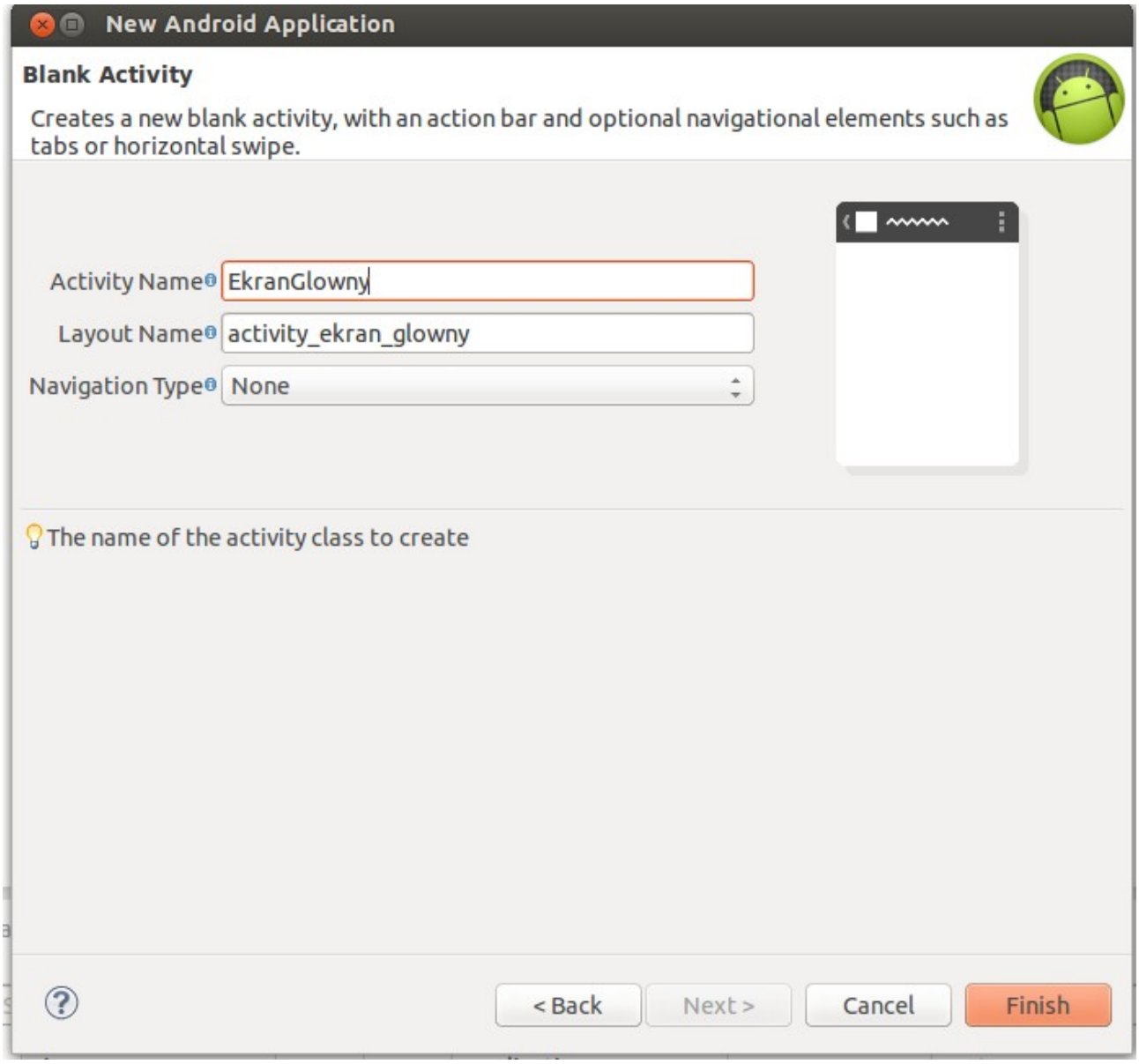
Przechodzimy dalej, aż pojawi nam się ekran wyboru ikony aplikacji. Domyślnie ikoną aplikacji będzie logo Androida. Możemy je zmienić na jakiś własny obraz (wybieramy go wtedy z dysku), jakiś obrazek z Clip Arta, albo logo tekstowe. Ja nacisnąłem „Clipart”, następnie choose i wybrałem znak kciuka. Dalej zmieniłem domyślny błękitny kolor logo na czerwony klikając na „Foreground Color”.



Przechodzimy dalej. Na kolejnym ekranie mamy możliwość wyboru sposobu wyświetlania aplikacji. Czy będzie to zwykła aplikacja z widocznym menu systemowym (Domyślny wybór) czy wyświetlanie pełnoekranowe (np. kiedy piszemy jakąś grę), czy też formularz typu master/detail.



Po przejściu do kolejnego ekranu wybieramy nazwę klasy aktywności głównej, oraz nazwę pliku xml w którym określony będzie wygląd aktywności głównej. Aktywność – można to rozumieć w sporym uproszczeniu jako ekran – dane okno w aplikacji. Aktywności aplikacja może mieć wiele, użytkownik może poruszać się między aktywnościami (między ekranami – uruchom np. swoją aplikację i przyjrzyj się procedurze ustawiania celu a zrozumiesz co mam na myśli). Każda aplikacja musi za to mieć jaką aktywność (ekran) która pojawi się jako pierwsza po uruchomieniu programu. Z taką aktywnością wiąże się klasa ją obsługująca, a nazwę tej klasy określamy w polu „Activity Name". W tej klasie będziemy później określać jak dany ekran ma się zachowywać. Kolejne pole – layout Name określa nazwę pliku XML w którym będzie zapisany wygląd tego ekranu. W tym pliku będzie zapisane rozmieszczenie elementów oraz ich własności.



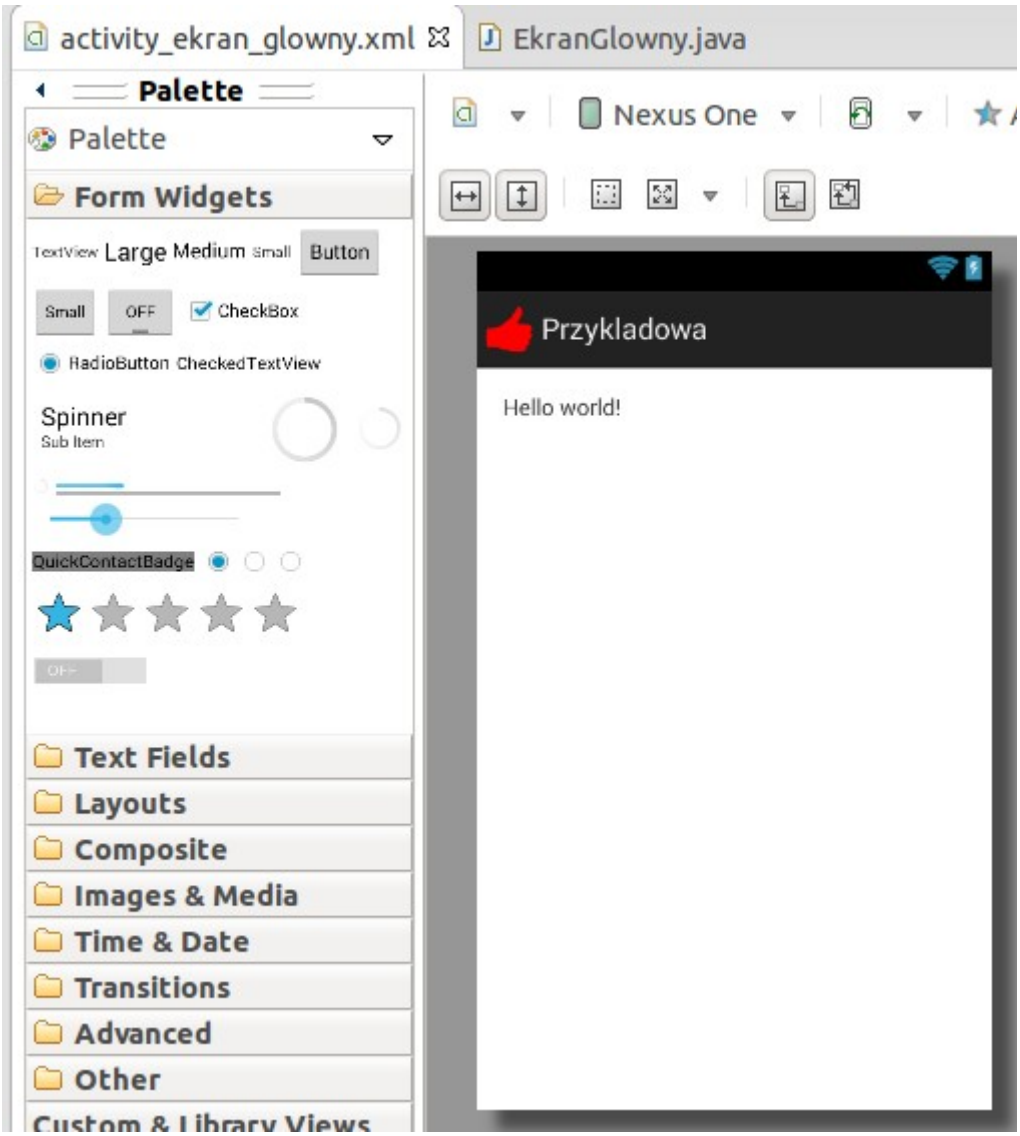
Po zakończeniu powinniśmy zobaczyć klasę naszej aktywności:

activity_ekran_glowny.xml | EkranGlowny.java

```
1  package pl.jsystems.przykladowa;
2
3  import android.os.Bundle;
6
7  public class EkranGlowny extends Activity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_ekran_glowny);
13     }
14
15     @Override
16     public boolean onCreateOptionsMenu(Menu menu) {
17         // Inflate the menu; this adds items to the action bar if it is present.
18         getMenuInflater().inflate(R.menu.ekran_glowny, menu);
19         return true;
20     }
21
22 }
23
```

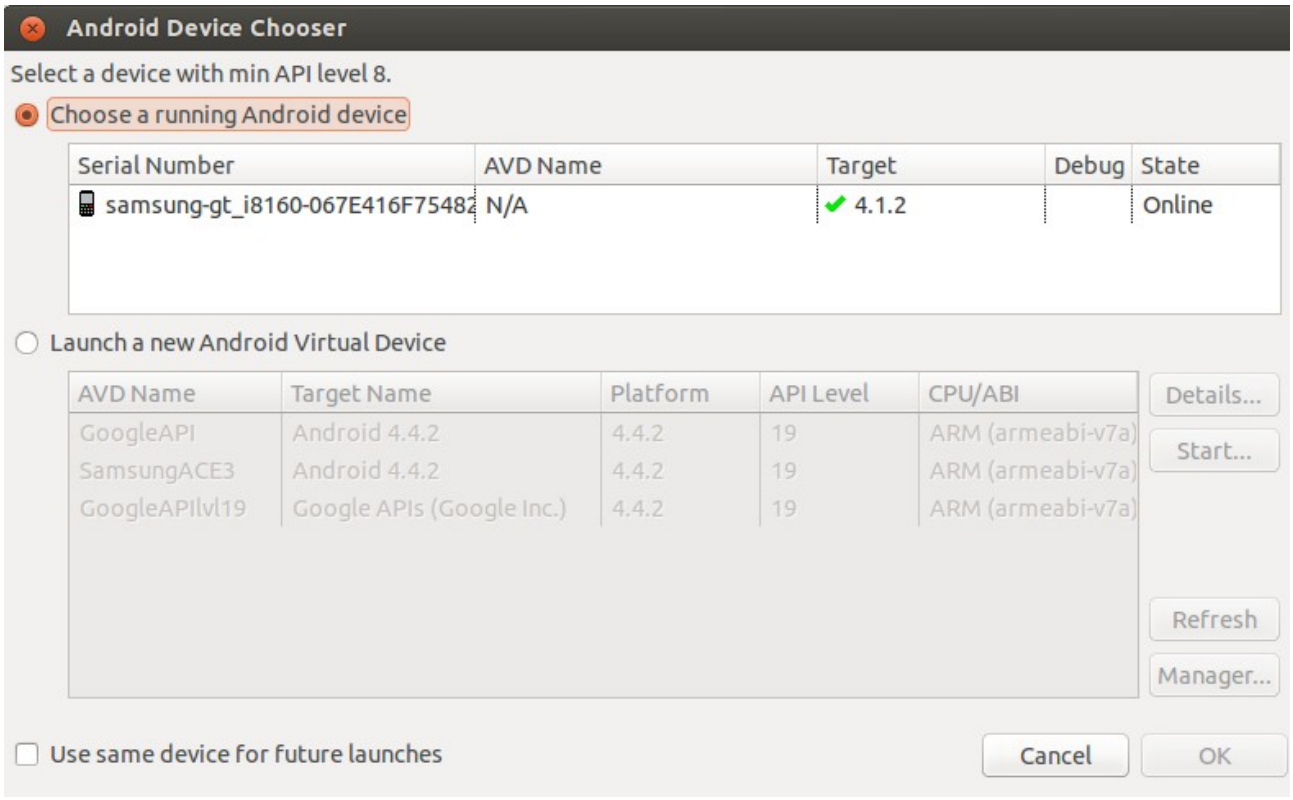
Na razie bez szczegółów – tak tylko informacyjnie: metoda onCreate jest uruchamiana kiedy dany ekran zostaje wyświetlony, metoda onCreateOptionsMenu kiedy użytkownik naciśnie przycisk menu.

Kiedy klikniemy na zakładkę z naszym plikiem layoutu (na poprzedniej ilustracji u mnie to activity_ekran_glowny.xml) zobaczymy projekt naszego głównego ekranu. Po lewej stronie mamy menu z elementami które możemy przyklejać na ekranie.

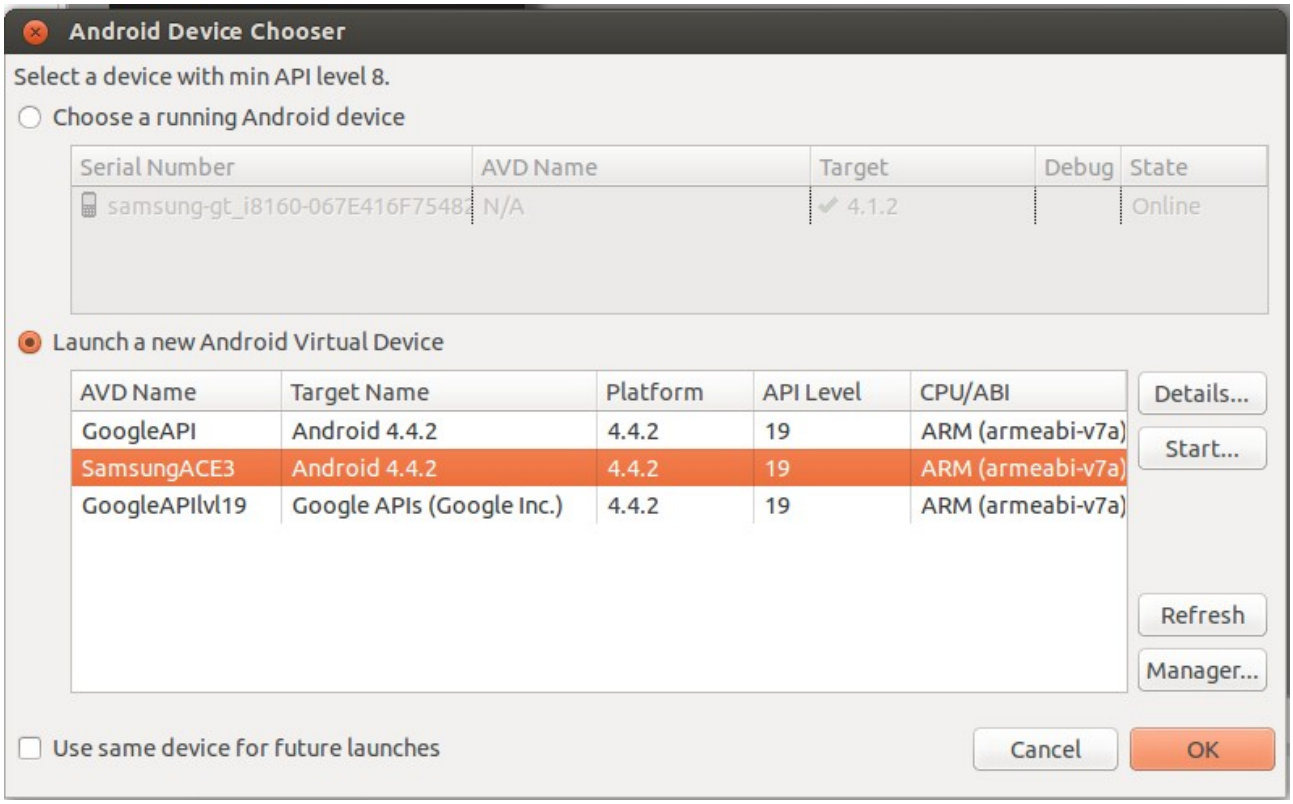


Na ten moment zostawiamy tradycyjne „Hello World” i uruchomimy aplikację na emulatorze i prawdziwym telefonie.

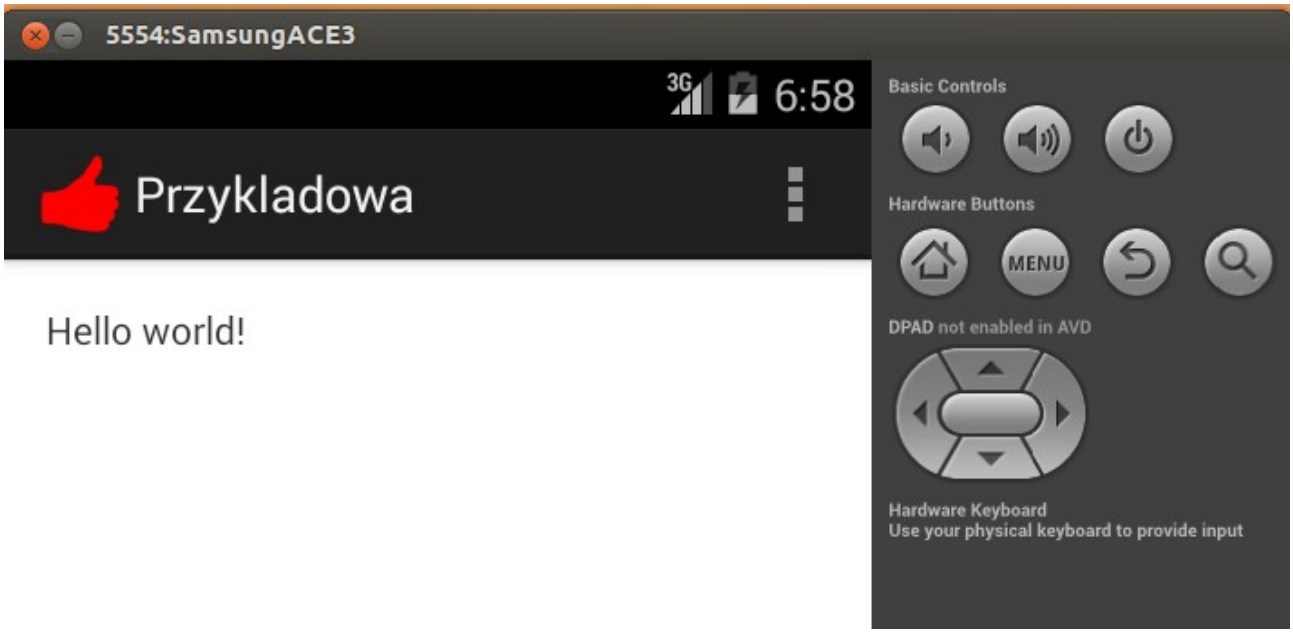
Aby uruchomić aplikację, klikamy prawym przyciskiem myszy na katalogu projektu w polu „Package Explorer”, dalej „Run As”-->”Android Application”. Powinno nam się pojawić takie okno:



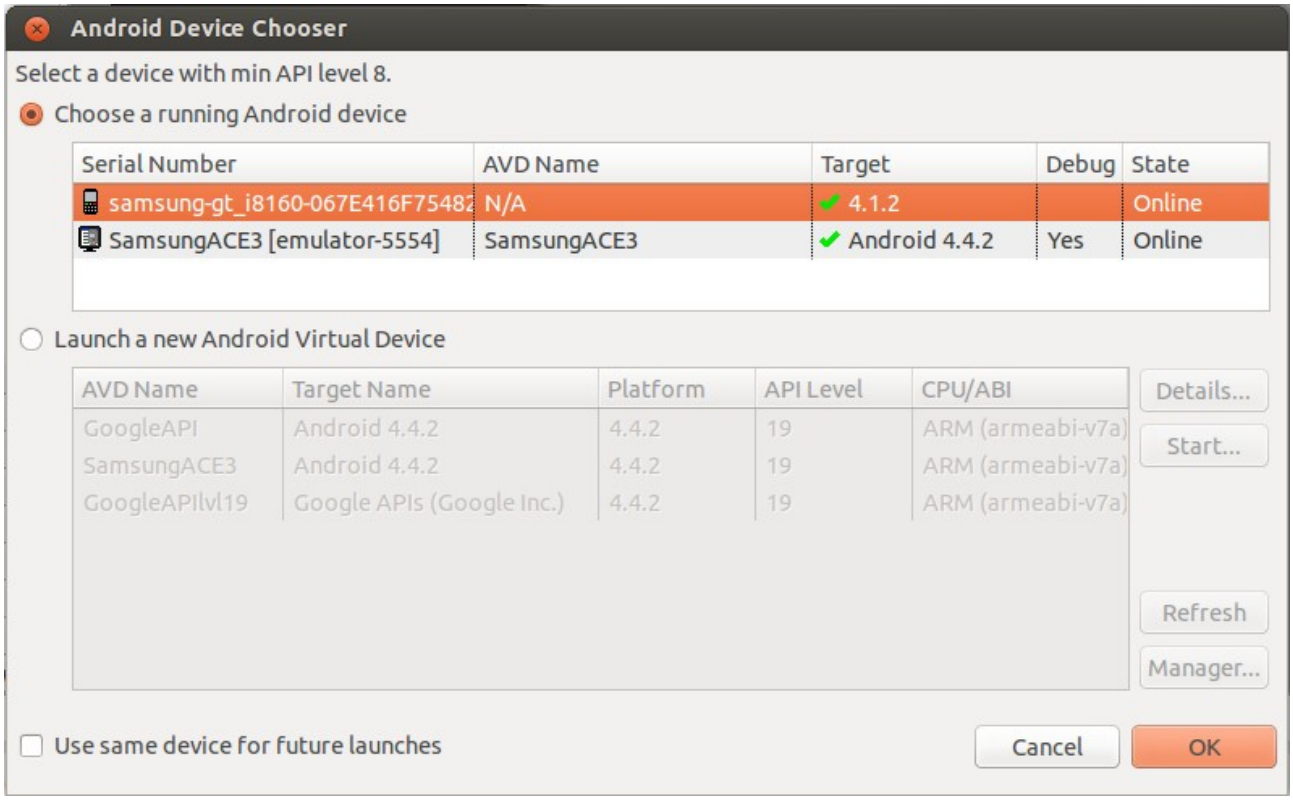
Możemy tutaj wybrać uruchomienie z użyciem wirtualnego lub realnego urządzenia. Na ten moment mam stworzone wirtualne urządzenia, ale żadne nie jest uruchomione. Odpalimy więc aplikację jednocześnie uruchamiając wirtualne urządzenie. Zaznaczam „Launch a new Android Virtual Device” , zaznaczam wybrane urządzenie i wciskam „Ok”.



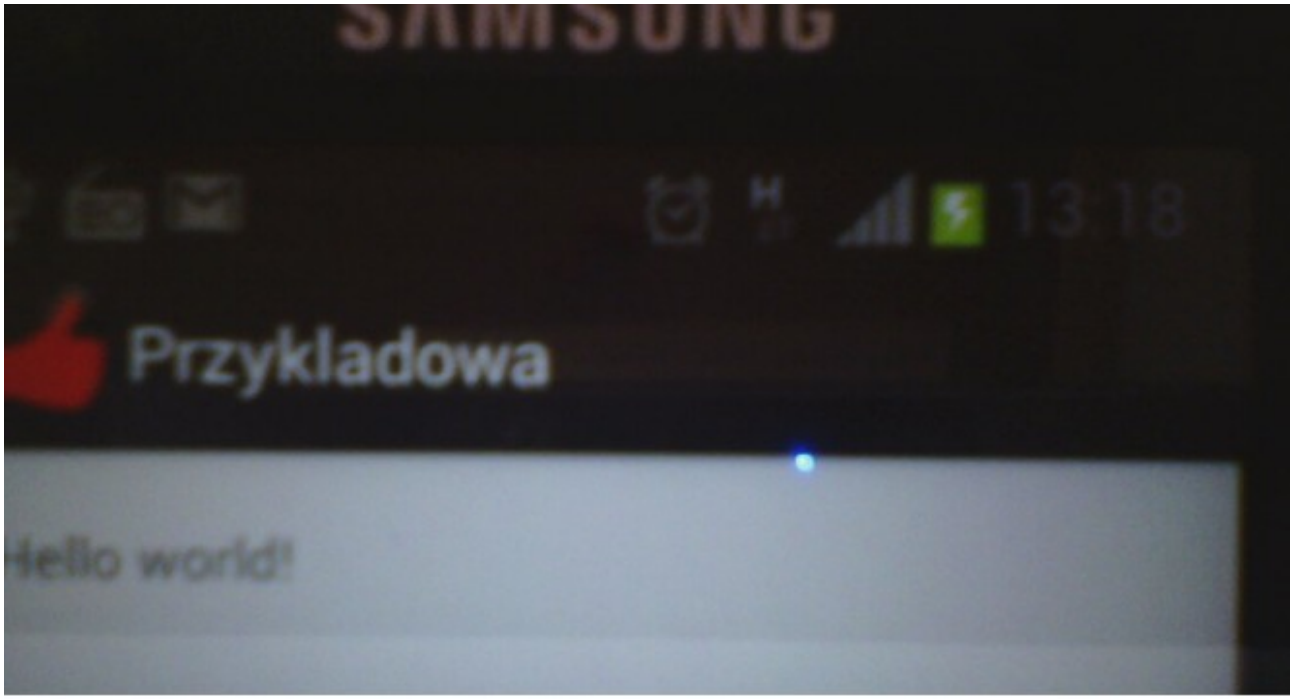
Virtualne urządzenie może uruchamiać się nawet 2-3 minuty, wszystko zależy od sprzętu którym dysponujemy. Po uruchomieniu na urządzeniu wirtualnym:



Tym razem uruchomimy program na realnym urządzeniu. Podobnie jak i wcześniej klikamy PPM na projekcie, wybieramy Run –> Run as Anroid Application. Pojawi się okno jak wcześniej. W sekcji „Choose a running Android device” widzimy dwa urządzenia. Jedno to wcześniej uruchomione urządzenie wirtualne, drugie to realne urządzenie – Samsung ACE3. Aby móc stąd wrzucać aplikacje na takie urządzenie, musimy je wcześniej przygotować włączając na nim opcje programistyczne. Aby to zrobić, wchodzimy w „Ustawienia” , dalej szukamy na dole opcji „opcje programisty” w które wchodzimy i aktywujemy. Bez tego nie będzie możliwości debuggowania aplikacji, czyli nie wdrożymy jej z Eclipsa na dany telefon. Po włączeniu opcji, podłączamy telefon przewodem do komputera. Przy kolejnym uruchomieniu aplikacji nasz telefon powinien być już tutaj widoczny. Zauważyłem już że niektóre urządzenia w ogóle nie chcą rozmawiać z Eclipsem w ten sposób, choćby wszystko było powłączane i połączone jak należy. Miałem tak np. z telefonem Prestigio. Zanim więc zaczniesz grzebać w opcjach i się irytować, po prostu sprawdź z innym urządzeniem.



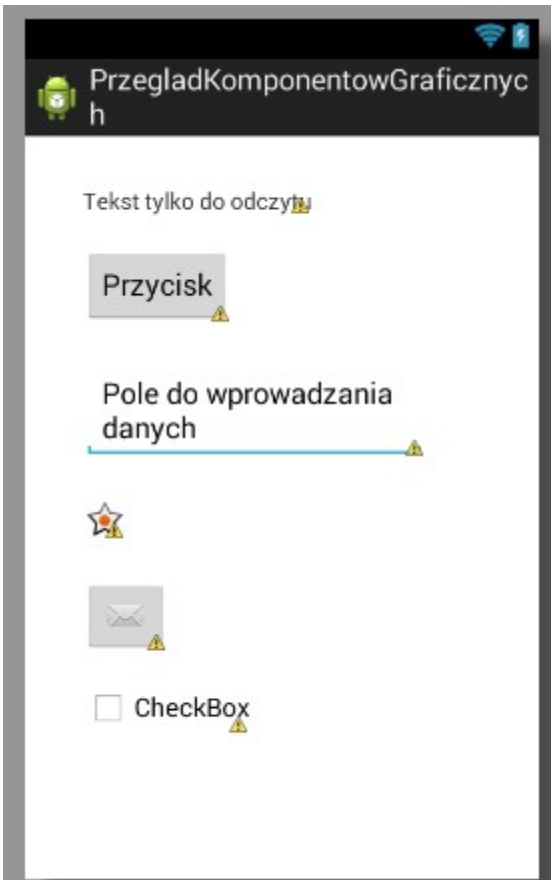
Zaznaczam swój telefon i wybieram „Ok”. Po odpaleniu:



Elementy wizualne

Przegląd podstawowych komponentów graficznych

Tutaj zajmiemy się najczęściej używanymi, podstawowymi komponentami graficznymi. Na ekranie przykleiłem kilka takich elementów i poniżej każdy z nich opisuję.



TextView

Pierwszy komponent od góry. Służy do wyświetlania tekstu. Użytkownik aplikacji nie może zmodyfikować jego zawartości. Na palecie komponentów mamy różne jego wariacje – w tym LargeText, MediumText i SmallText. W rzeczywistości te trzy niby różne elementy to ta sama klasa, ale z różnie ustawionymi parametrami wielkości czcionki. Dostępny jest w palecie w panelu Form Widgets.

Button

Drugi element od góry. Przycisk. Możemy zaprogramować w Javie sposób jego zachowania po przyciśnięciu. Dostępny jest w palecie w panelu Form Widgets.

EditText

Pole edycyjne w które użytkownik aplikacji może wprowadzać tekst. Dostępny jest w palecie w panelu TextFields. Znajdziemy tam również kilka wariacji tego komponentu, umożliwiających np. wprowadzanie wyłącznie liczb, albo tylko dat. Wszystkie one są obiektami klasy EditText, jednak z różnie ustawionymi parametrami wyświetlania i wprowadzania danych.

ImageView

Element trzeci od dołu. Służy do wyświetlania obrazków. Dostępny jest w palecie w panelu Images & Media.

ImageButton

Widoczny jest pod ImageView. Jest czymś pośrodku między Buttonem a ImageView. Możemy na nim wyświetlić obrazek, a jednocześnie oprogramować jego zachowanie po kliknięciu. Dostępny jest w palecie w panelu Images & Media.

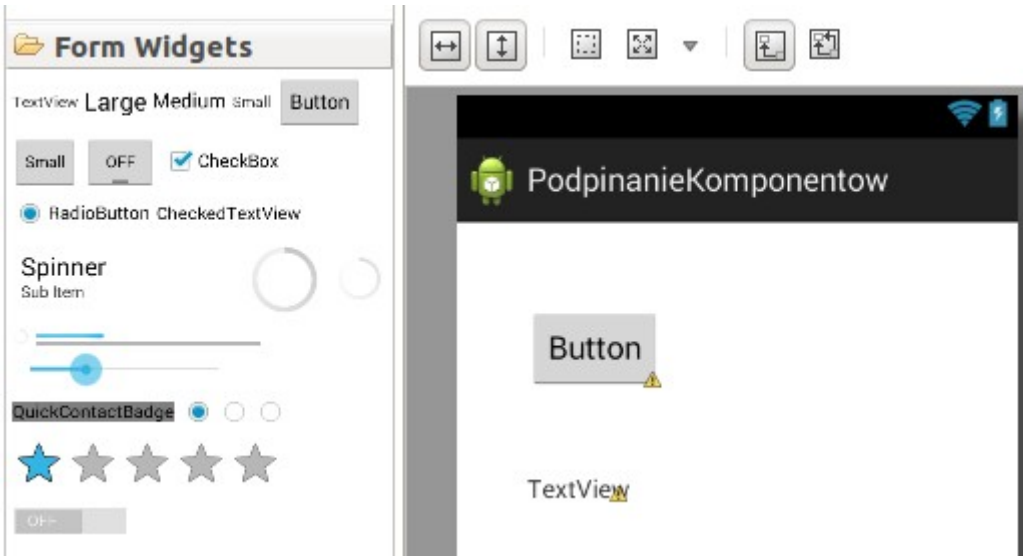
CheckBox

Umożliwia zaznaczenie lub odznaczenie jakiejś opcji (np. akceptacja regulaminu). Dostępny jest w palecie w panelu Form Widgets.

Korzystanie z komponentów wizualnych w kodzie

Umieszczanie komponentów na ekranach aplikacji odbywa się najczęściej poprzez ich przeciągnięcie na ekran (co tak naprawdę generuje kod XML w pliku layoutu) lub ręczne definiowanie ich w pliku layoutu. Poza tym, że figurują jako obiekty XML, nie ma do nich żadnego dowiązania w kodzie aktywności. Jeśli zechcemy w jakiś sposób z tych komponentów korzystać, musimy w klasie aktywności podpiąć do nich referencje.

Zaczynamy od przyklejenia na ekran jakichś przykładowych komponentów:



Gdy zajrzymy do pliku layoutu, zobaczymy że zostało wygenerowane trochę dodatkowego kodu związanego z tymi nowymi komponentami:

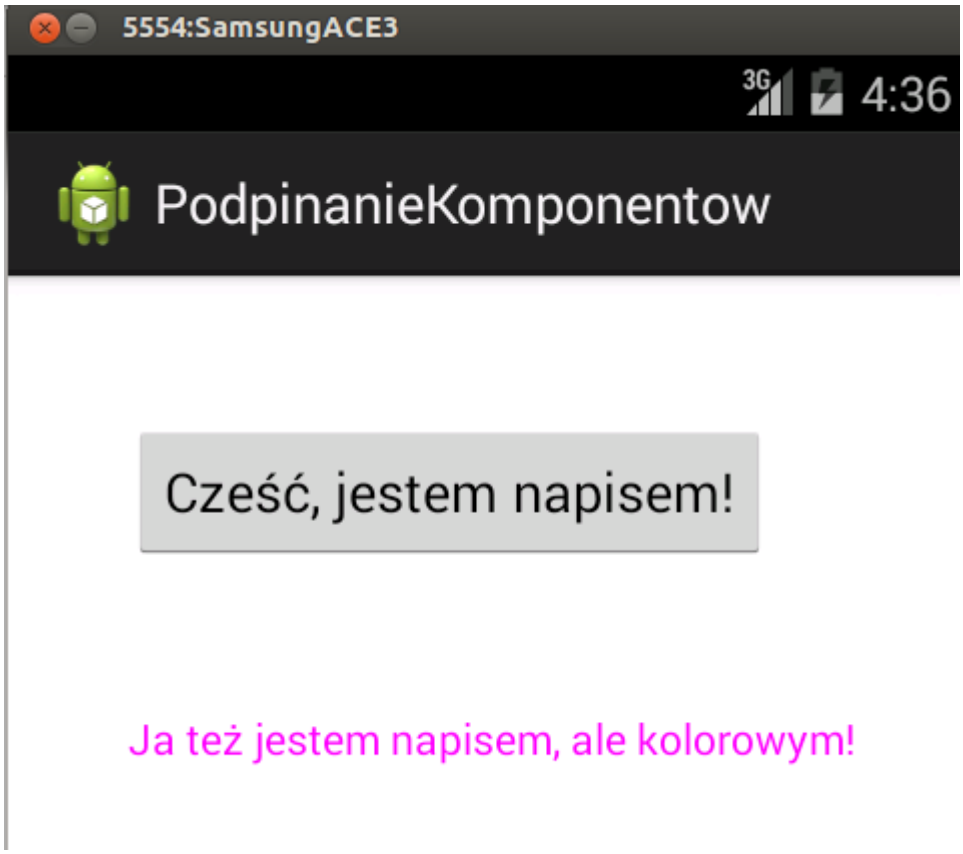
activity_main.xml | MainActivity.java

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="24dp"
        android:layout_marginTop="32dp"
        android:text="Button" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button1"
        android:layout_marginTop="48dp"
        android:text="TextView" />

</RelativeLayout>
```

Przejdźmy teraz do kodu aktywności. Aby korzystać z przyklejonych komponentów, musimy przede wszystkim posiadać obiekty klas których są komponenty. Definiuję je więc w liniach 12 i 13. Mógłbym to zrobić równie dobrze w metodzie onCreate (wywoływanej przy uruchamianiu aktywności), ale taka globalna deklaracja sprawi, że będę miał do tych obiektów dostęp również w innych metodach. W liniach 19 i 20 podpinam do tych obiektów referencję do komponentów. Służy do tego metoda findViewById. Jako jej parametr podaję wskaźnik do zdefiniowanego komponentu. Zauważ że każdy kompontent określony w pliku layoutu ma parametr android:id. To właśnie on jednoznacznie identyfikuje dany komponent. Nasz guzik na ten przykład ma identyfikator „button1”. Chcąc więc podpiąć referencję do tego obiektu piszę w parametrze metody findViewById „R.id.” i po tym podaję identyfikator komponentu o który mi chodzi. Od tego momentu mogę ustawić czy czytać parametry podpiętych już komponentów wizualnych.

```
10 public class MainActivity extends Activity {
11
12     Button b;
13     TextView t;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19         b=(Button) findViewById(R.id.button1);
20         t=(TextView) findViewById(R.id.textView1);
21         b.setText("Cześć, jestem napisem!");
22         t.setTextColor(Color.MAGENTA);
23         t.setText("Ja też jestem napisem, ale kolorowym!");
24     }
25
26
27
28 }
29
```

Po uruchomieniu wygląda to tak:



Obsługa i konfiguracja podstawowych komponentów graficznych.

Zasadniczo każdy z elementów omawianych w tej publikacji można zwyczajnie przeciągnąć z palety na ekran. Omawiając poszczególne elementy będę też wskazywał sposób umieszczenia ich bezpośrednio poprzez plik XML layoutu. Ma to na celu lepsze zrozumienie zasady działania i parametryzacji komponentów. Należy jedynie pamiętać, że jest to tylko jedna z możliwości.

TextView

```
<TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tekst tylko do odczytu" />
```

Tekst jaki ma zostać wyświetlony na komponencie możemy ustawić przy pomocy parametru android:text w pliku layoutu, tak jak to widać na przykładzie powyżej, lub poprzez metodę setText klasy TextView, jak to widać poniżej:

```
TextView tv = (TextView) findViewById(R.id.textView1);
           tv.setText("Napisik tylko do odczytu...");
```

Button

```
<Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Przycisk" />
```

Tekst jaki ma zostać wyświetlony na komponencie możemy ustawić przy pomocy parametru android:text w pliku layoutu, tak jak to widać na przykładzie powyżej, lub poprzez metodę setText klasy Button, jak to widać poniżej:

```
Button b=(Button) findViewById(R.id.button1);
           b.setText("Napis na guziku");
```

Obsługa zdarzenia kliknięcia na komponentach (nie tylko na przycisku) została opisana w osobnej publikacji – to dość spory temat.

EditText

```
<EditText
      android:id="@+id/editText1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Pole do wprowadzania danych" >
```

Tekst domyślny jaki ma zostać wyświetlony na komponencie możemy ustawić przy pomocy parametru android:text w pliku layoutu, tak jak to widać na przykładzie powyżej, lub poprzez metodę setText klasy EditText, jak to widać poniżej:

```
EditText et=(EditText)findViewById(R.id.editText1);
           et.setText("Domyślny tekst");
```

Tekst wprowadzony przez użytkownika możemy odebrać przy pomocy metody getText klasy EditText:

```
EditText et=(EditText)findViewById(R.id.editText1);
           String coWpisano=et.getText()+"";
```

Ponieważ metoda getText zwraca obiekt klasy Editable (a nie jak byśmy się mogli spodziewać - String), dodałem fragment +"" wymuszający niejawną konwersję na String. Bez tego aplikacja się nie kompiluje.

ImageView

```
<ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@android:drawable/star_on" />
```

Ścieżkę obrazka który ma zostać wyświetlony ustawiamy przy użyciu parametru android:src w pliku layoutu. Obrazek który ma zostać wyświetlony wrzucamy do dowolnego z katalogów drawable-xxxx będących podkatalogami katalogu res, a następnie odnosimy się poprzez wskaźnik zbudowany na tej zasadzie: @android:drawable/nazwa_pliku_bez_rozszerzenia

ImageButton

```
<ImageButton
      android:id="@+id/imageButton1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@android:drawable/sym_action_email" />
```

Ścieżkę obrazka który ma zostać wyświetlony ustawiamy przy użyciu parametru android:src w pliku layoutu. Obrazek który ma zostać wyświetlony wrzucamy do dowolnego z katalogów drawable-xxxx będących podkatalogami katalogu res, a następnie odnosimy się poprzez wskaźnik zbudowany na tej zasadzie: @android:drawable/nazwa_pliku_bez_rozszerzenia

CheckBox

```
    <CheckBox
        android:id="@+id/checkBox1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CheckBox" />
```

Tekst który ma zostać wyświetlony obok pola zaznaczania określamy przy użyciu parametru android:text w pliku layoutu lub przy użyciu metody setText klasy CheckBox, tak jak to widać poniżej:

```
CheckBox cb=(CheckBox) findViewById(R.id.checkBox1);
           cb.setText("Przeczytałem i akceptuję regulamin");
```

Odczyt stanu zaznaczenia możemy przeprowadzić przy użyciu metody isChecked klasy CheckBox, tak jak to widać poniżej:

```
CheckBox cb=(CheckBox) findViewById(R.id.checkBox1);

           if(cb.isChecked()){
                 //zaznaczone
           }else{
                 //nie zaznaczone
           }
```

Zasoby tekstowe – wykorzystanie pliku strings.xml

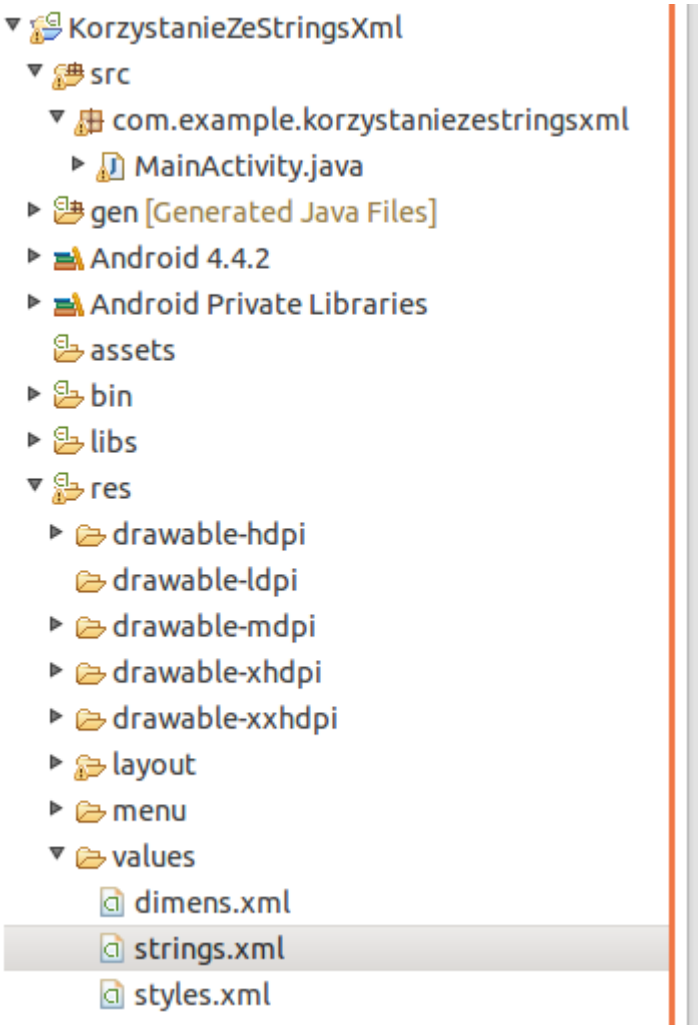
Wyobraźmy sobie taką sytuację. Wyklejam formularz taki jak np. ten na dole. Podobnych ekranów w aplikacji jest np. 30. W każdym pojawiają się przyciski „Zatwierdź” i „Anuluj”. W niektórych pojawia się też fraza „Imię:”, „Nazwisko” etc.

Teraz musimy zrobić niemieckojęzyczną wersję programu, albo choćby zmienić wszystkie nazwy na pisane małą literą....

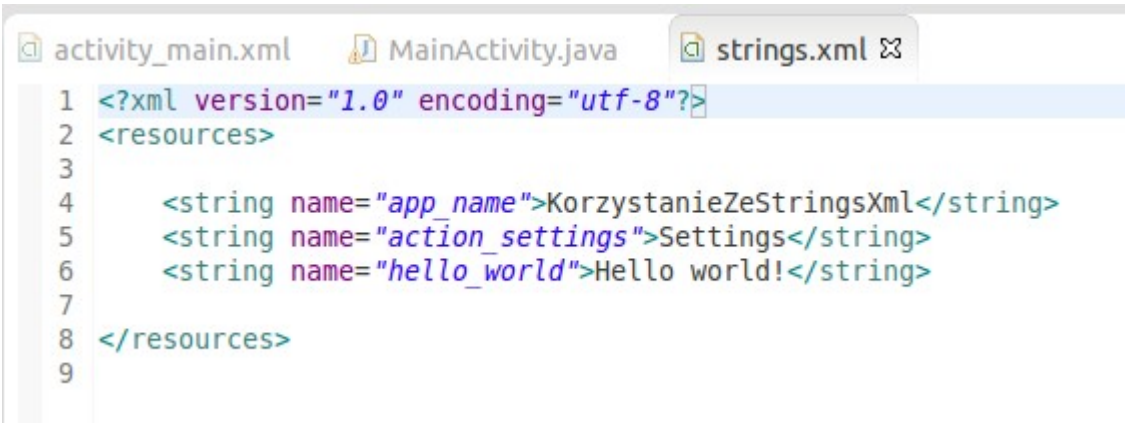
Dopuki teksty te określam „na sztywno” w pliku layoutu, tak jak to widać poniżej (linie 15,25,34), doputy będę musiał przy każdej tego typu zmianie modyfikować plik layoutu każdej aktywności której zmiana ma dotyczyć.

```
10
11      <TextView
12          android:id="@+id/textView1"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:text="Imię:" />
16
17      <TextView
18          android:id="@+id/textView2"
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:layout_alignParentLeft="true"
22          android:layout_below="@+id/textView1"
23          android:layout_marginTop="16dp"
24          android:layout_toLeftOf="@+id/editText3"
25          android:text="Nazwisko:" />
26
27      <TextView
28          android:id="@+id/textView3"
29          android:layout_width="wrap_content"
30          android:layout_height="wrap_content"
31          android:layout_alignLeft="@+id/textView2"
32          android:layout_below="@+id/textView2"
33          android:layout_marginTop="18dp"
34          android:text="Telefon:" />
35
```

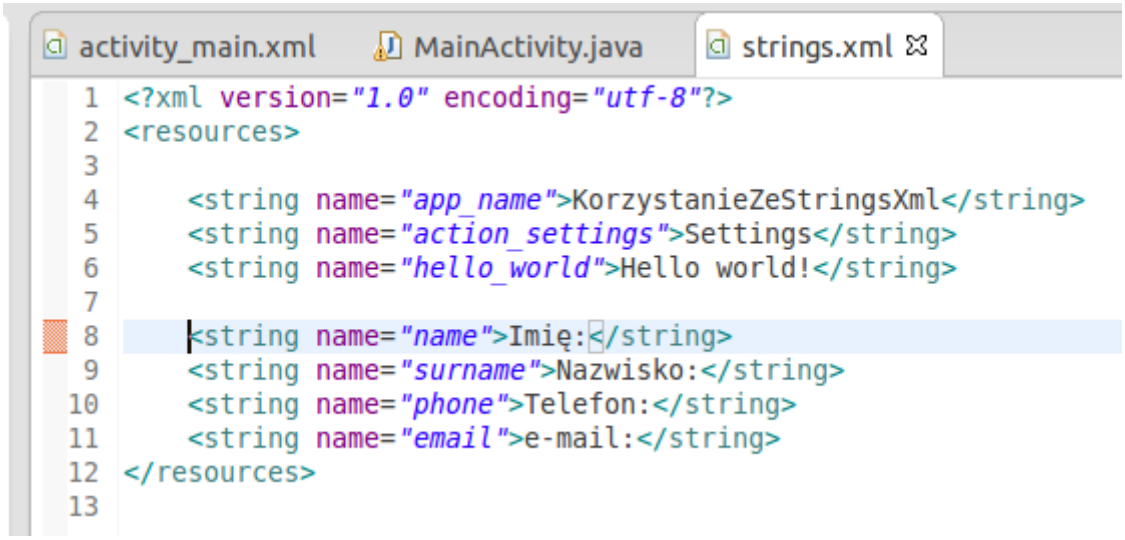
Lepszym rozwiązaniem będzie odnalezienie pliku strings.xml znajdującego się w katalogu values będącego podkatalogiem katalogu res i skorzystanie z możliwości jakie daje.



Kiedy go wyedytujemy, powinniśmy zobaczyć taką zawartość:



Dodajmy tam kilka elementów. Od teraz teksty które pojawiają się nam na formularzu będą definiowane tutaj. Do każdej frazy przypisana jest nazwa np. do frazy „Imię:” przypisana jest nazwa „name”:



Teraz w elemencie „android:text” zamiast pisać na sztywno tekst który ma się pojawić, piszę odwołanie po nazwie do wpisów w pliku strings.xml (linie 15,25,34):

activity_main.xml | MainActivity.java | strings.xml

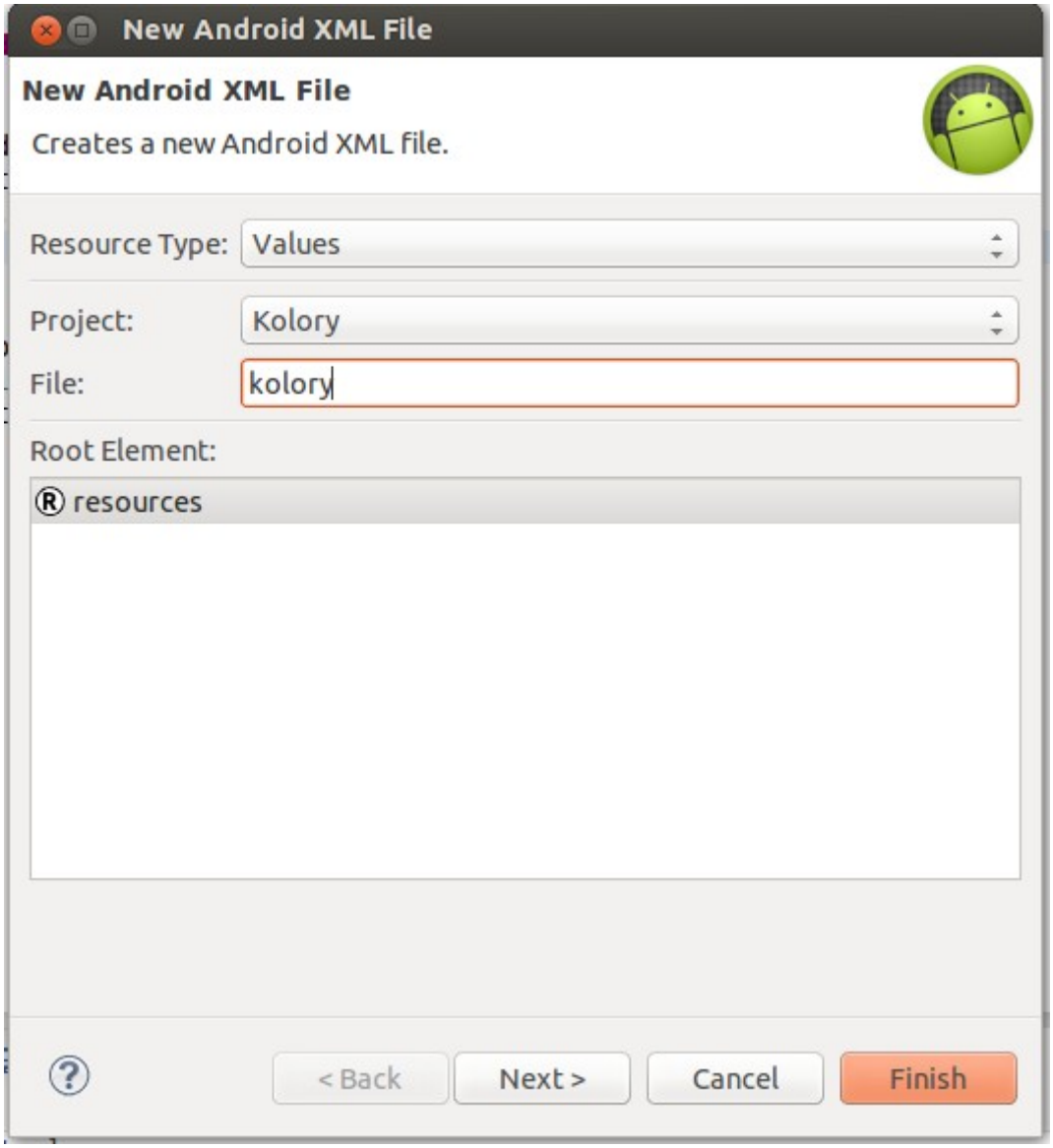
```
5       android:paddingBottom="@dimen/activity_vertical_margin"
6       android:paddingLeft="@dimen/activity_horizontal_margin"
7       android:paddingRight="@dimen/activity_horizontal_margin"
8       android:paddingTop="@dimen/activity_vertical_margin"
9       tools:context=".MainActivity" >
10
11      <TextView
12          android:id="@+id/textView1"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:text="@string/name" />
16
17      <TextView
18          android:id="@+id/textView2"
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:layout_alignParentLeft="true"
22          android:layout_below="@+id/textView1"
23          android:layout_marginTop="16dp"
24          android:layout_toLeftOf="@+id/editText3"
25          android:text="@string/surname" />
26
27      <TextView
28          android:id="@+id/textView3"
29          android:layout_width="wrap_content"
30          android:layout_height="wrap_content"
31          android:layout_alignLeft="@+id/textView2"
32          android:layout_below="@+id/textView2"
33          android:layout_marginTop="18dp"
34          android:text="@string/phone" />
35
```

Możemy takie wskaźniki zrobić we wszystkich plikach layoutów. Dzięki takiemu zabiegowi, zmiana tekstu na wszystkich guzikach służących do zatwierdzania formularza, czy nawet dorobienie alternatywnej wersji językowej aplikacji wymagać będzie zmian w jednym tylko miejscu i bez wielokrotnej zmiany tego samego tekstu w różnych plikach layoutów.

Szablony kolorów

Jeśli zechcemy często używać w naszej aplikacji jakiegoś koloru, lub np. zrobić dwukolorową aplikację, możemy wszędzie podawać kodowo kolor np. #FF0033, albo stworzyć szablon i odwolywac się do koloru na zasadzie „kolor_tla”. Informacja o kodzie koloru dla „kolor_tla” znajduje się w osobnym pliku xml. Dzięki takiemu rozwiązaniu możemy szybko modyfikować kolory w całej aplikacji, np. jeśli zechcemy zrobić skórkę świąteczną dla naszego programu :)

Zaczynamy od utworzenia pliku xml w katalogu values będącym podkatalogiem katalogu res. Klikamy na nim PPM → new → new Android XML file. Wyskakuje nam takie okienko. Podajemy nazwę pliku i zatwierdzamy.

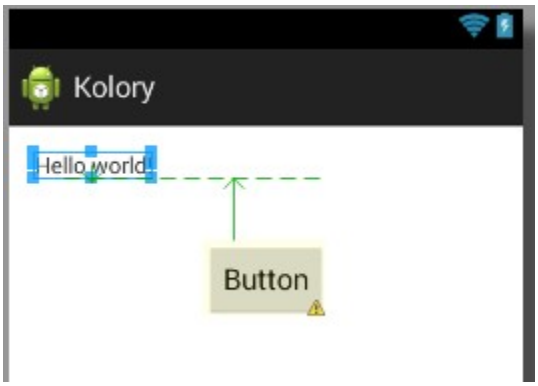


W pliku powinny znaleźć się wpisy podobne do tych:

activity_main.xml | MainActivity.java | kolory.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <color name="barbie">#CC66FF</color>
4     <color name="bananowo">#FFCC00</color>
5 </resources>
6
```

To jest przypisanie koloru do nazwy. Gdziekolwiek w aplikacji odwołam się teraz @color/barbie, wyświetli mi się kolor #CC66FF. Łatwiej też będzie zmieniać kolorystykę aplikacji – wystarczy że zmienię kod koloru w tym pliku XML, a zmiana będzie obowiązywała dla wszystkich elementów gdzie odwołuję się do nazw z etego pliku. Przetestujemy działanie. Do ekranu przyklejam guziczka:



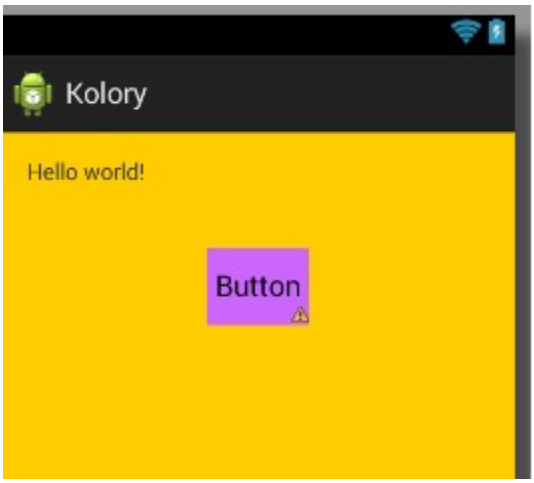
A w pliku layoutu tej aktywności odwołuję się do tych kolorów w taki sposób:
@color/nazwa_koloru :

activity_main.xml | MainActivity.java | kolory.xml

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:background="@color/bananowo"
10     tools:context=".MainActivity" >
11
12     <TextView
13         android:id="@+id/textView1"
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="@string/hello_world" />
17
18     <Button
19         android:id="@+id/button1"
20         android:background="@color/barbie"
21         android:layout_width="wrap_content"
22         android:layout_height="wrap_content"
23         android:layout_below="@+id/textView1"
24         android:layout_centerHorizontal="true"
25         android:layout_marginTop="38dp"
26         android:text="Button" />
27
28 </RelativeLayout>
```

Przyjrzyj się limion 9 i 20. Odwołałem się po nazwach do wcześniej zdefiniowanych w pliku kolory.xml kolorów. W linii nr 9 ustawiłem tło dla ekranu, w linii nr 20 ustawiłem kolor dla przycisku.

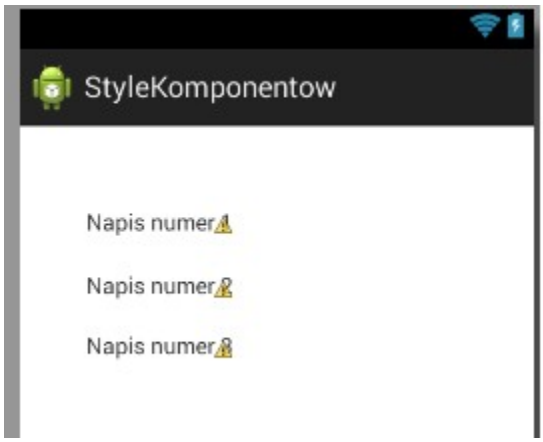
Przełączam się teraz na widok projektowania graficznego i widzę to:



Wyszło paskudnie, chociaż wierzę że gdzieś na świecie są ludzie którzy mają dom urządzony w takich kolorach. Na szczęście w przypadku aplikacji na Androida (w przeciwieństwie do domu) można to łatwo i szybko zmienić.

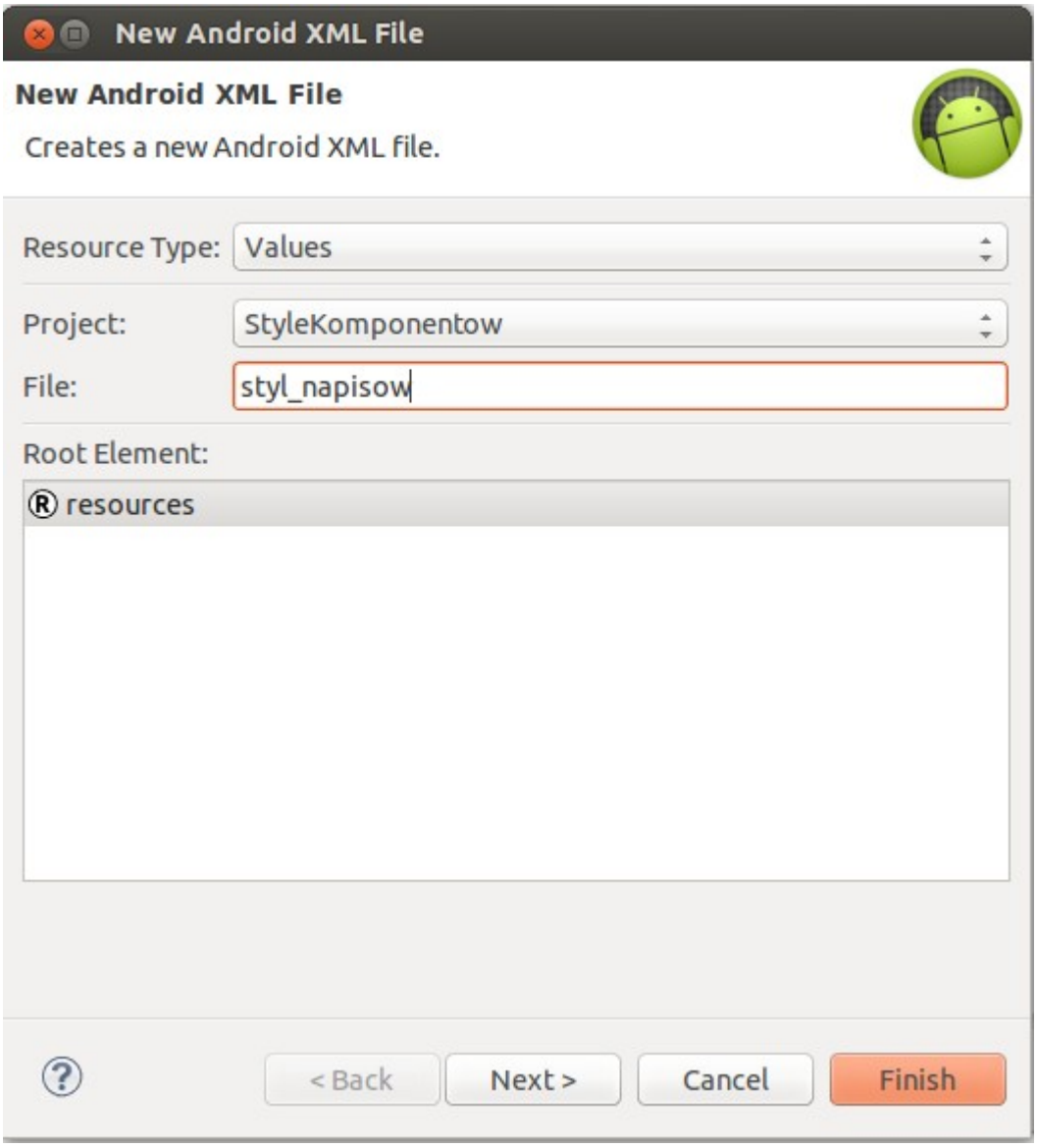
Style komponentów

Przypuśćmy że mam taką sytuację, mam trzy napisy na ekranie i dla wszystkich chcę określić taki sam kolor czcionki, wielkość czcionki, ewentualnie inne parametry.



Mógłbym to robić osobno dla każdego komponentu ustawiając parametry w pliku layoutu danej aktywności. To więcej pracy niż to warte, a ponadto gdybym zechciał coś zmienić, musiałbym zmieniać dla każdego komponentu osobno. Każdy kto napisał choć jedną podstronę w HTML powinien doskonale rozumieć w czym rzecz. Podobnie jak w HTML, tak i tutaj możemy zdefiniować style i z ich użyciem stosować określone własności dla wielu komponentów naraz. Filozofia jest podobna do stosowanych przy tworzeniu stron internetowych plików css.

Zaczynam od stworzenia pliku XML ze stylem. Klikam PPM na katalogu values będącym podkatalogiem katalogu res. Wybieram new-->Android XML File. Pojawia mi się takie okno:



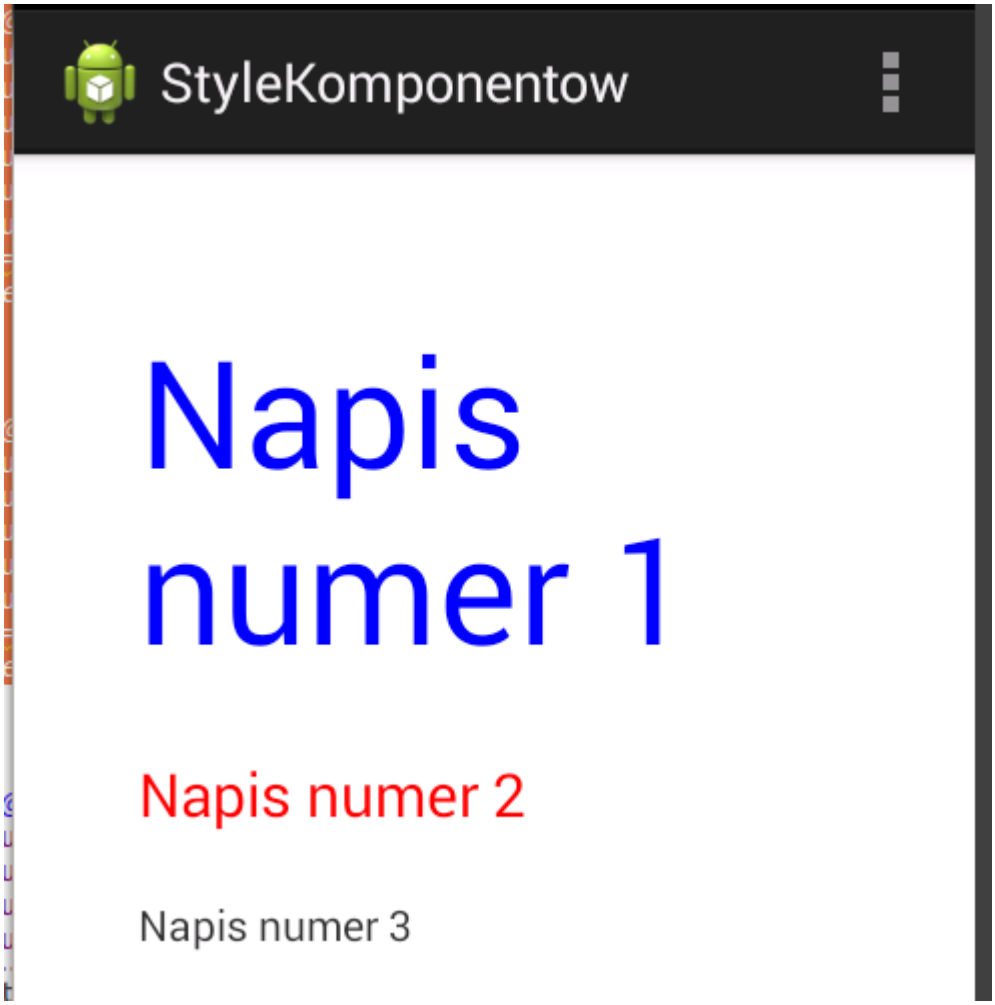
Podaję nazwę pliku w którym opiszę styl komponentów. Nazwałem go styl_napisow. Teraz w pliku wprowadzam kilka zmian. Dodaję dwa różne style napisów. Zawartość mojego pliku ze stylami:

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <style name="wielki_napis">
        <item name="android:textSize">50dp</item>
        <item name="android:textColor">#0000FF</item>
    </style>

    <style name="maly_napis">
        <item name="android:textSize">20dp</item>
        <item name="android:textColor">#FF0000</item>
    </style>

    </resources>
```

Jak widac określam wielkość i kolor czcionek. Są dwa style – dla dużego niebieskiego napisu, oraz mniejszego czerwonego. Można tutaj oczywiście określić znacznie więcej rzeczy – np. szerokość i wysokość komponentu, napis na nim, a nawet metodę która obsługuje kliknięcie komponentu. W pliku layoutu dla danej aktywności dodaję do komponentów nowy parametr: style, oraz wskazuję nazwę stylu sprzed momentem stworzonego pliku:

```
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="25dp"
        android:layout_marginTop="35dp"
        android:text="Napis numer 1"
        style="@style/wielki_napis"
         />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="22dp"
        android:text="Napis numer 2"
        style="@style/maly_napis"
         />
```

Efekt po uruchomieniu:



Style możemy stosować dla wszelakich komponentów, nie tylko dla napisów.

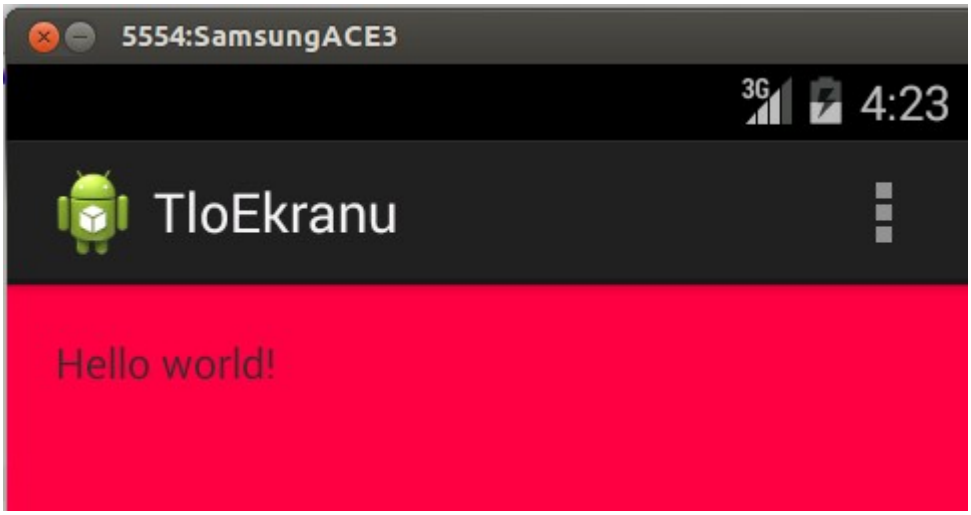
Tło ekranu aplikacji

Zaczniemy od ustawienia koloru tła, to jest zdecydowanie najprostsze. Stworzyłem nowy projekt. Całość sprowadza się do ustawienia parametru android:background (linia 9) w XMLowym pliku layoutu.

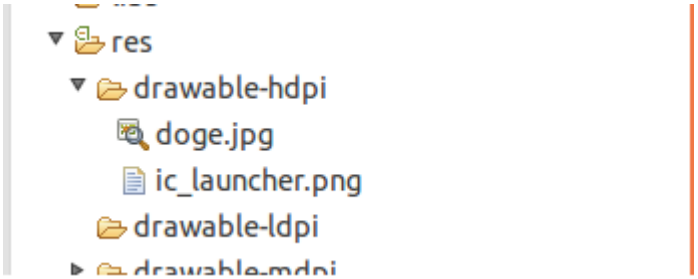
activity_main.xml | MainActivity.java

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:background="#FF0044"
10     tools:context=".MainActivity" >
11
12     <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="@string/hello_world" />
16
17 </RelativeLayout>
18
```

Jeśli chcesz wybrać jakiś kolor, możesz znaleźć zestawienie kolorów dla HTML (np. tutaj: http://www.kurshtml.edu.pl/html/wykaz_kolorow,kolory.html) Ustawiłem przypadkowy kod koloru FF0044 i wyszło mi coś takiego(fuj):



Tłem może być również obrazek. Wrzuciłem obrazeł do podkatalogu drawable-hdpi:

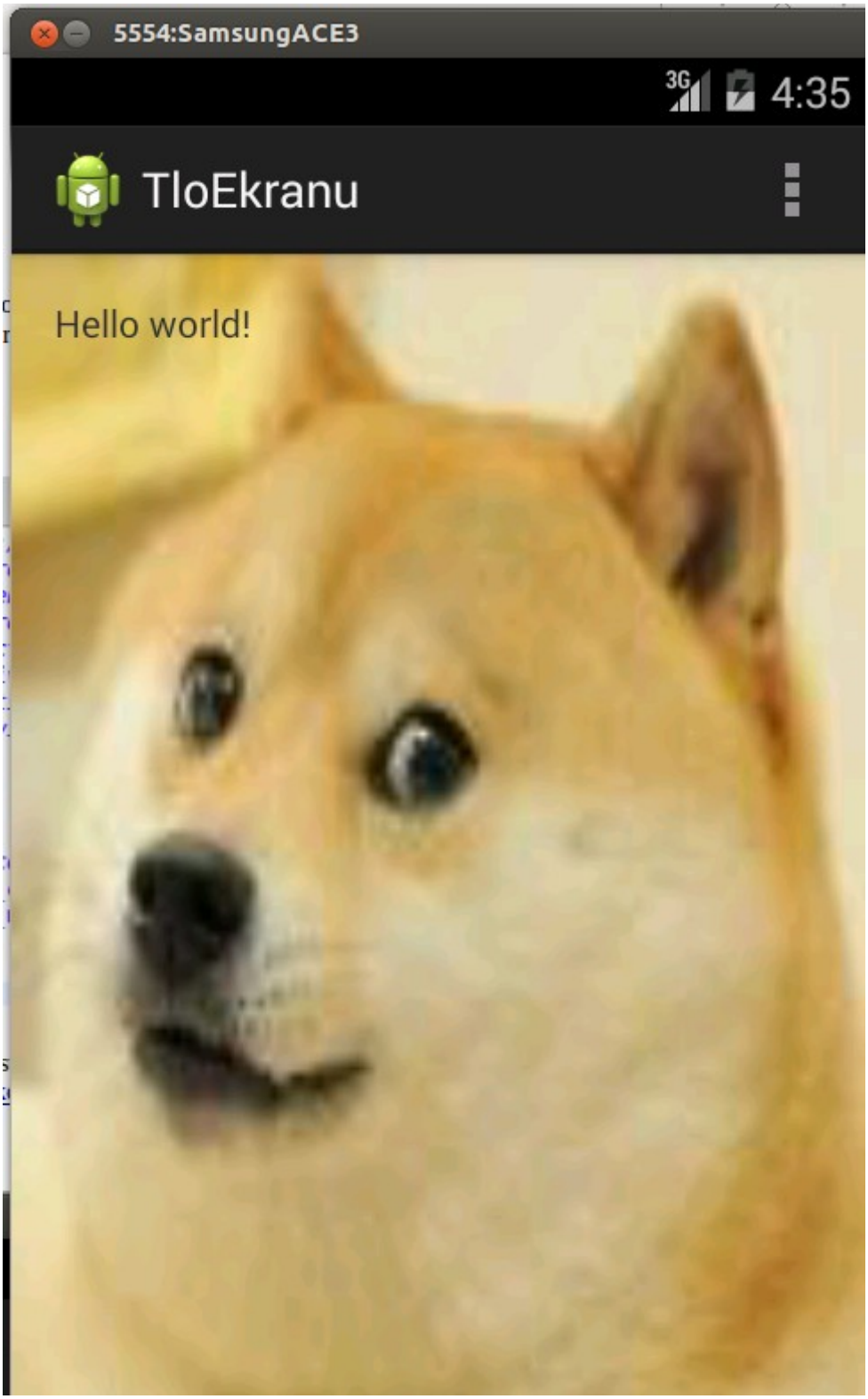


Tym razem zamiast podawać kod koloru, podaję w pliku layoutu w parametrze android:background wskaźnik do obrazka (patrzymy na linię nr 9):

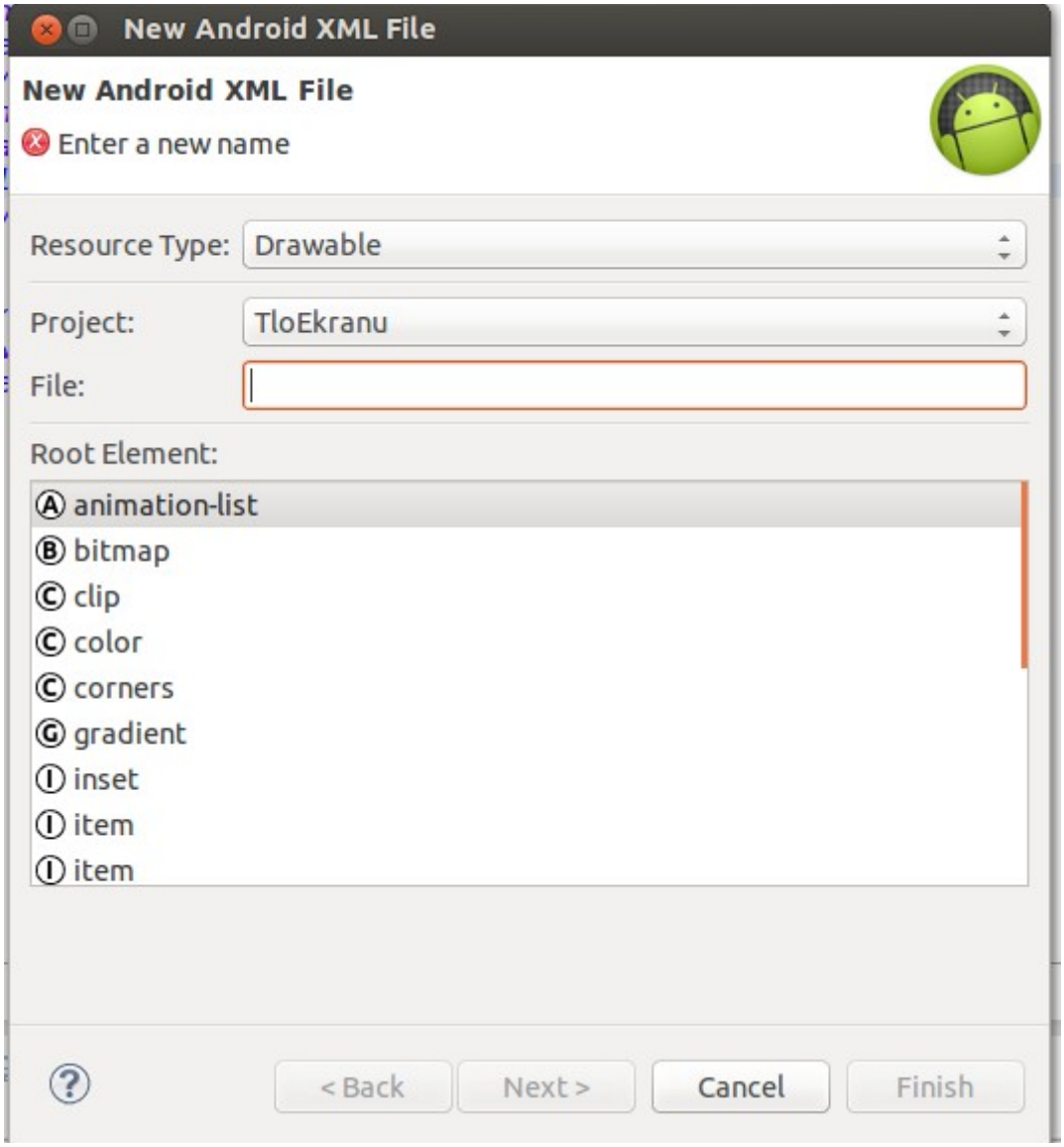
activity_main.xml | MainActivity.java

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:background="@drawable/doge"
10     tools:context=".MainActivity" >
11
12     <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="@string/hello_world" />
16
17 </RelativeLayout>
```

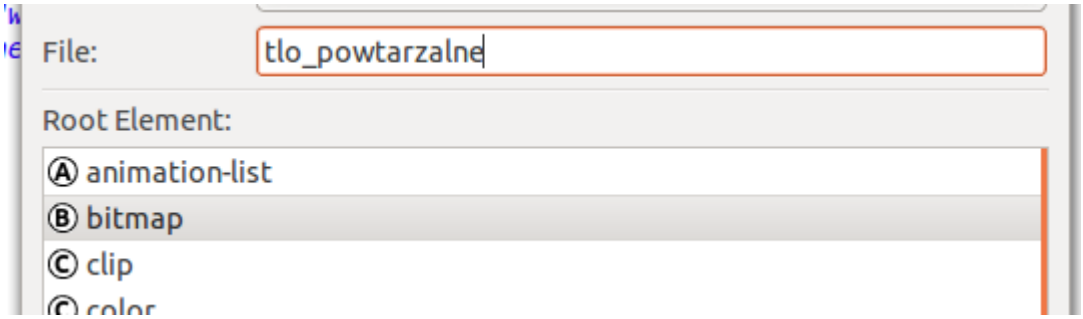
Efekt:



Pieseł nam się trochę rozciągnął. W zależności od rozmiaru i proporcji ekranu będziemy mieć większego albo mniejszego, chudszego albo grubszego pieseła. To nie zawsze musi dobrze wyglądać. Może lepiej byłoby by obrazek tła się powtarzał? Aby uzyskać taki efekt, musimy stworzyć plik XML który będzie opisywał sposób prezentacji obrazka. W tym celu klikamy PPM na katalogu drawable-hdpi i wybieramy New-->Android XML File. Pojawi nam się takie okno:



Zaznaczamy „bitmap” i wpisujemy nazwę pliku xml:



Zatwierdzamy. Zawartość pliku uzupełniamy tak by wyglądał w taki sposób:

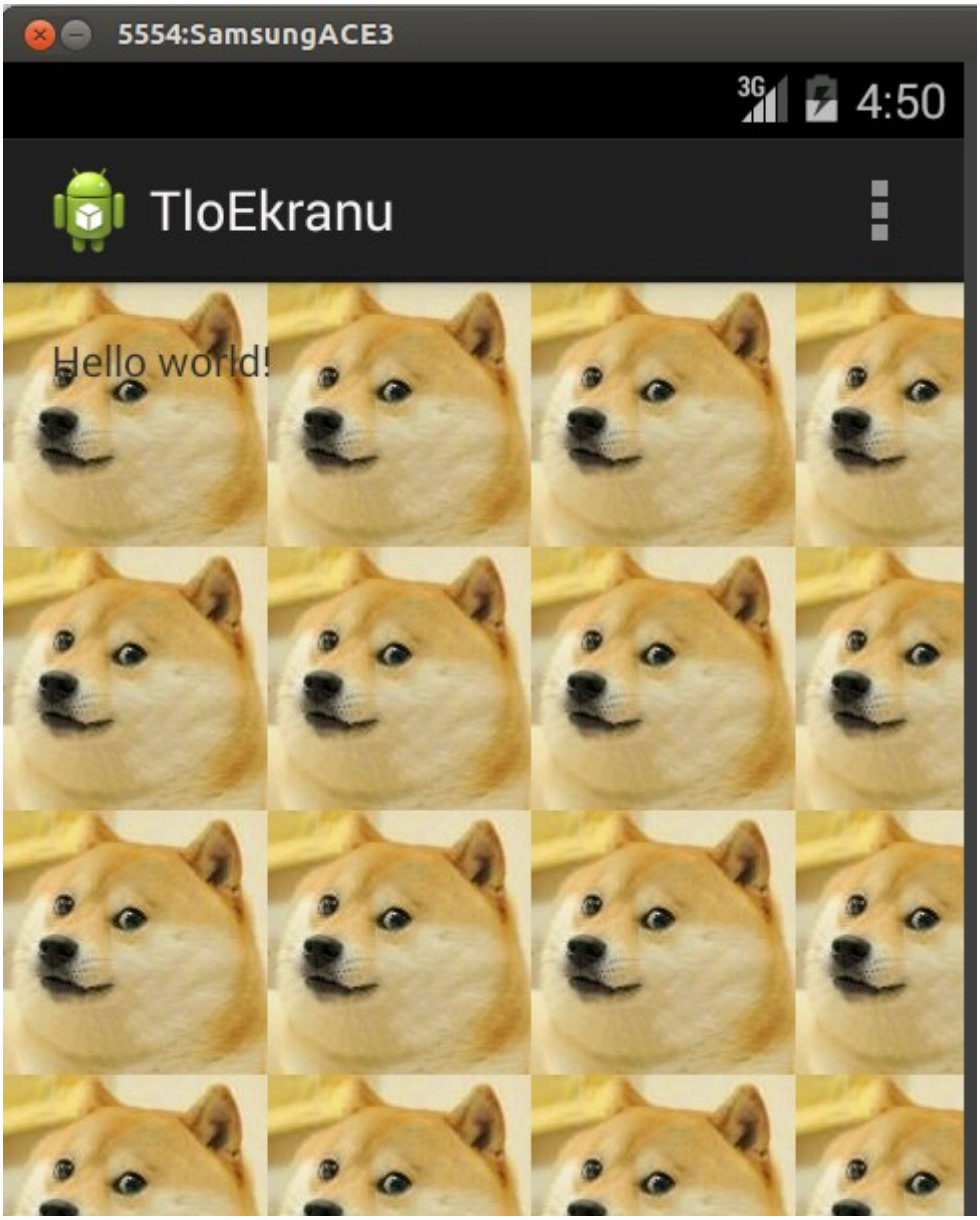
```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/doge"
    android:tileMode="repeat"
    >
</bitmap>
```

Oczywiście w parametrze android:src podajemy wskaźnik do swojego obrazka. Parametr android:background w pliku layoutu modyfikujemy tak, by wskazywał na nasz nowo utworzony plik XML a nie plik obrazka:

activity_main.xml | MainActivity.java | tlo_powtarzalne.xml

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:background="@drawable/tlo_powtarzalne"
10     tools:context=".MainActivity" >
11
12     <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="@string/hello_world" />
16
17 </RelativeLayout>
```

Po tych zabiegach ekran programu po uruchomieniu wygląda tak:

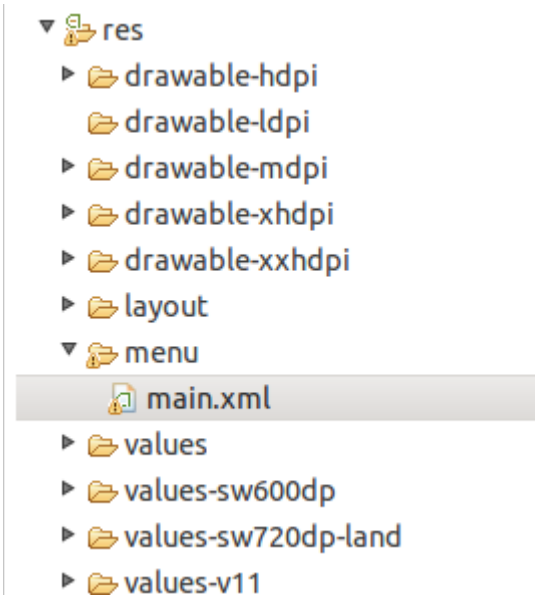


Konfiguracja własnego menu

W większości urządzeń z Androidem (telefony, tablety) mamy dostępny przycisk menu. Warto byłoby oprogramować to co się dzieje po jego wciśnięciu. Możemy tam wstawić np. opcję wejścia w ustawienia programu, informacje o autorze, albo pomoc. Jeśli przyjrzymy się domyślnie automatycznie tworzonej klasie głównej aktywności, zobaczymy że zawiera ona metodę onCreateOptionsMenu:

```
7  public class MainActivity extends Activity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_main);
13     }
14
15     @Override
16     public boolean onCreateOptionsMenu(Menu menu) {
17         // Inflate the menu; this adds items to the action bar if it is present.
18         getMenuInflater().inflate(R.menu.main, menu);
19         return true;
20     }
21
22 }
```

W niej wskazywany jest plik zawierający opis zawartości menu. Powyższy (domyślny) wpis wskazuje na plik main.xml znajdujący się w katalogu menu będącym podkatalogiem katalogu res:



Możemy go przerobić pod własne potrzeby, lub stworzyć nowy i przerobić metodę onCreateOptionsMenu tak by wskazywała nowo stworzony. Domyślnie jego zawartość wygląda tak:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/action_settings"/>

</menu>
```

Mamy jedną opcję w menu, z napisem określonym w pliku strings.xml znajdującym się w katalogu res --> values pod nazwą „action_settings”. Przyglądamy się mu:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Kompas</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>

</resources>
```

Widzimy, że napis na tej opcji w menu to „Settings”. Teraz zmienimy plik main.xml, by wprowadzić własne opcje menu. Nic nie stoi też na przeszkodzie bysmy stworzyli osobny plik, musimy jedynie pamiętać o zmianie ustawień w metodzie onCreateOptionsMenu aktywności. Do pliku strings.xml dodaję jeden wpis:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">KonfiguracjaMenu</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="autor">O autorze</string>

</resources>
```

Dalej modyfikuję zawartość pliku main.xml wg własnych potrzeb:

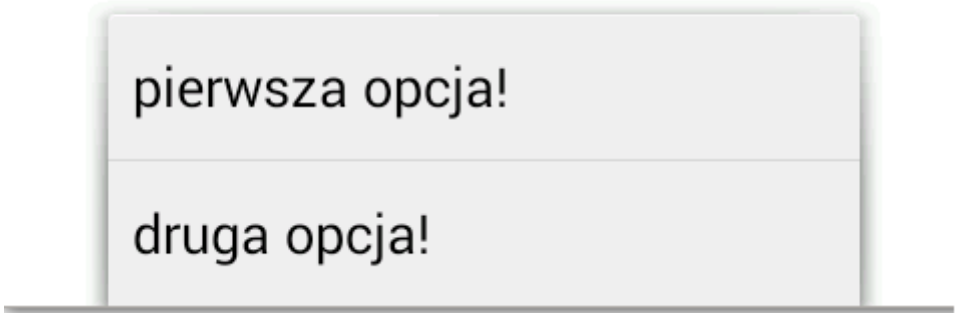
```
1  <menu xmlns:android="http://schemas.android.com/apk/res/android" >
2
3      <item
4          android:id="@+id/opcja1"
5          android:orderInCategory="100"
6          android:title="Opcja numer 1"/>
7       <item
8          android:id="@+id/opcja2"
9          android:orderInCategory="110"
10         android:title="Opcja numer 2"/>
11       <item
12         android:id="@+id/opcja3"
13         android:orderInCategory="120"
14         android:title="Opcja numer 3"/>
15          <item
16         android:id="@+id/opcja4"
17         android:orderInCategory="120"
18         android:title="@string/autor"/>
19
20 </menu>
```

Każdy element <item> to kolejna opcja w menu. Zrobiłem trzy elementy z napisami wpradzonymi bezpośrednio w tym pliku, a jeden z napisem pobieranym z pliku strings.xml Napis na opcji okresla parametr android:title. Parametr android:orderInCategory określa kolejność wyświetlania w menu od góry. Opcja android:id to identyfikator danej opcji w menu. Musimy zadbać o to by były one różne dla każdego elementu menu, inaczej nie będziemy później w stanie odróżnić która opcja została wybrana. Na ten moment menu po uruchomieniu wygląda tak:

Jeśli tak wolisz, nie musisz definiować zawartości menu w pliku XML. Możesz to zrobić bezpośrednio w kodzie Javy:

```
16
17      @Override
18      public boolean onCreateOptionsMenu(Menu menu) {
19          // Inflate the menu; this adds items to the action bar if it is present.
20      //  getMenuInflater().inflate(R.menu.main, menu);
21          menu.add(1, 1, 1, "pierwsza opcja!");
22          menu.add(2, 2, 2, "druga opcja!");
23          return true;
24      }
25
```

Metoda add dla obiektu klasy Menu (czyli naszego menu) wywoływana tutaj ma 4 parametry. Według kolejności: Numer grupy, numer elementu, kolejnośc wyświetlania, napis. Kod w działaniu:



Wracam jednak do starej metody z plikiem XML. Teraz dodamy obsługę kliknięcia. Dorzucam metodę onOptionsItemSelected która jest automatycznie wywoływana za każdym razem gdy użytkownik kliknie w menu. Przez parametr tej metody przekazywany jest obiekt elementu menu który został kliknięty. Mogę pobrać jego identyfikator, jednak nie będzie to identyfikator typu „opcja1” , „opcja2” czyli takie jak zdefiniowałem w pliku main.xml, a identyfikator liczbowy, który dopiero mogę porównać z identyfikatorami liczbowymi moich opcji menu. Tak to niestety trzeba na około robić:

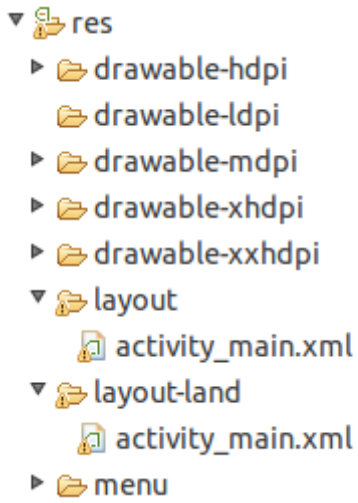
```
25
26      @Override
27      public boolean onOptionsItemSelected(MenuItem item){
28          Log.d("Kliknięto","x="+item.getItemId());
29          int id=item.getItemId();
30          if(id==R.id.opcja1){
31              Log.d("klik!","Opcja 1");
32          }else if(id==R.id.opcja2){
33              Log.d("klik!","Opcja 2");
34          }else if(id==R.id.opcja3){
35              Log.d("klik!","Opcja 3");
36          }else if(id==R.id.opcja4){
37              Log.d("klik!","Opcja 4");
38          }
39          return true;
40      }
```

W zależności od identyfikatora wyświetlam właściwy tekst. Wyświetlam też identyfikator liczbowy. Kliknąłem najpierw pierwszą opcję menu, potem trzecią. Tak wyglądają efekty:

```
ig AbsListView                  Get MotionRecognitionManager
ig Kliknięto                    x=2131230720
ig klik!                        Opcja 1
ig Kliknięto                    x=2131230722
ig klik!                        Opcja 3
```

Różne layouty w różnych orientacjach

Ponieważ ekrany telefonów i tabletów z reguły nie są kwadratowe, a nasze aplikacje mogą działać zarówno w pionie jak i poziomie, warto byłoby uwzględnić takie dwie możliwości. Wbrew ostatnim trendom, w tym przypadku mamy wyłącznie dwie orientacje ;) Układ komponentów na ekranie projektowany dla orientacji pionowej, najprawdopodobniej „rozjedzie się", gdy obrócimy ekran do orientacji pionowiej. Mamy możliwość zrobienia różnych layoutów, które będą używane w zależności od tego jak będziemy trzymać urządzenie. Jeśli pozostaniemy przy domyślnym jednym pliku layoutu, będzie on używany w obu przypadkach. Aby rozróżnić układ komponentów w zależności od sposobu trzymania telefonu, wystarczy dodać w katalogu res katalog o nazwie layout-land i w nim umieścić alternatywny plik layoutu. Wszystko odbywa się automatycznie.



W przypadku położenia pionowego, wczytywany będzie layout z katalogu layout, w przypadku poziomego z layout-land. Pamiętaj tylko, że w metodzie onCreate aktywności jest wskazywana nazwa pliku z layoutem, jeśli więc chcielibysmy aby to działało w założony sposób, musimy zadbać o to by oba pliki layoutu nazywały się tak samo.

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

Wyłączenie obracania ekranu w aplikacji

Podczas obracania ekranu następuje automatyczne wywołanie metod onPause, onSaveInstance, onStop a następnie onRestoreInstance i onResume. Zasadniczo zawartość pól takich jak komponenty klasy EditText nie „czyszczą” się, wszystko wraca do pierwotnej postaci po obrocie ekraniu. Nie dotyczy to pól takich klas jak TextView ani zdeklarowanych przez nas zmiennych. Wszystko jest zerowane do takiej postaci jak przy uruchomieniu aplikacji. Oczywiście możemy zaimplementować metody onSaveInstance i onRestoreInstance, a następnie przekazywać dane które mają zostać odtworzone przez obiekt klasy Bundle, jednak przy większej ilości zmiennych może to być czasochłonne. Innym wyjściem jest zablokowanie możliwości obracania ekranu. Zachowanie jak przy resetowaniu aplikacji w ogóle nie będzie miało wtedy miejsca. Niektóre aplikacje wymagają jakiejś określonej pozycji ekranu bez jej obracania. Np. dziwnie by się oglądało film wyświetlany na pionowo ustawionym ekranie. Całość sprowadza się do dodania wpisu

```
android:screenOrientation="portrait"
```

W elementcie Application w pliku manifestu. Powyższe ustawnienie (portrait) blokuje ekran w pozycji pionowej, można też ustawić na landscape – czyli poziome. Podgląd na element aplpication, czyli jak to jest zmontowane całe:

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.gpsinfo.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>
```

Układy elementów na ekranie – rodzaje layoutów

Rozkład elementów na ekranie opisany jest w xml'owym pliku layoutu. Możemy go edytować ręcznie, lub przyklejać kompotenty na zasadzie „przeciągnij i upuść”. To w jaki sposób określane jest położenie poszczególnych elementów zależy od lodzaju layoutu. Poszczgólne rodzaje layoutów opisuję poniżej. W pierwszej kolejności musimy znaleźć właściwy plik layoutu dla aktywności której układ elementów chcemy zmieniać. Stworzyłem nową aplikację „PrzykladTableLayout”. W jej katalogu res-->layout znajduje się plik xml w którym będzie opisany layout naszej głównej aktywności. Jeżeli tworząc projekt zaakceptowałeś domyślną nazwę aktywności, to Twój plik będzie się nazywał activity_main.xml

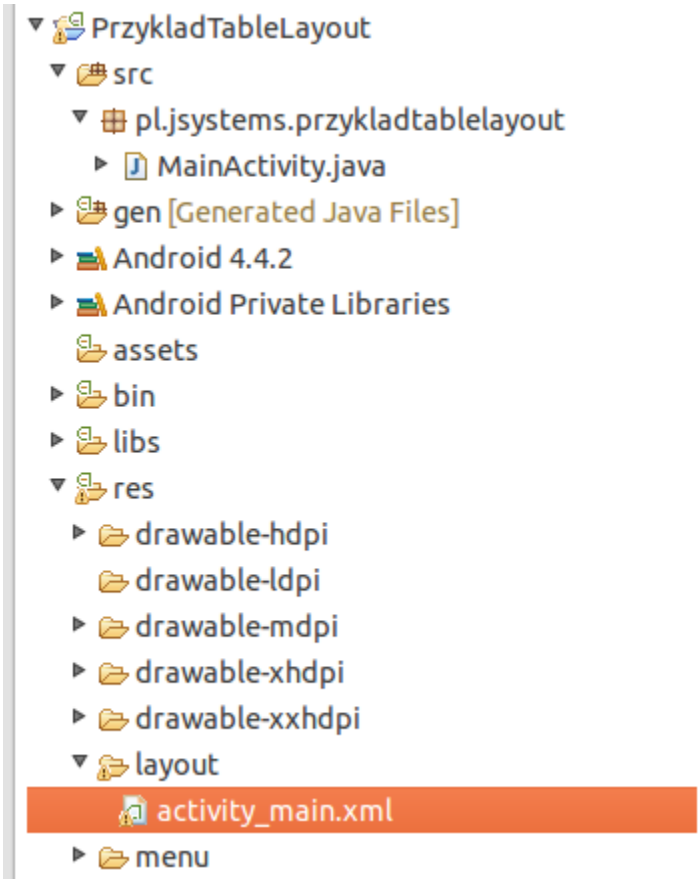


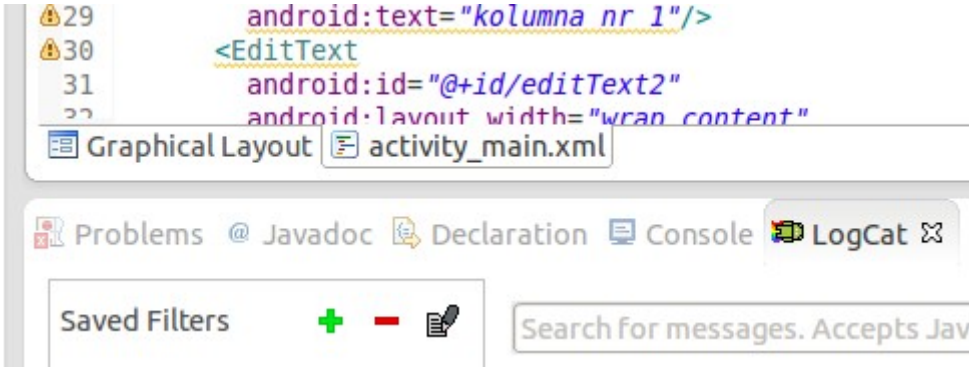
Table layout – układ tabelaryczny

Skoro zabrałeś/łaś się za programowanie na Androida to zakładam, że takie rzeczy jak podstawy HTML są dla Ciebie banalne. Ten rodzaj układu jest bardzo zbliżony do elementu Table znanego z języka HTML. Definiujemy poszczególne wiersze i kolumny. Skasuj całą zawartość swojego pliku layoutu i wklej taki kod:

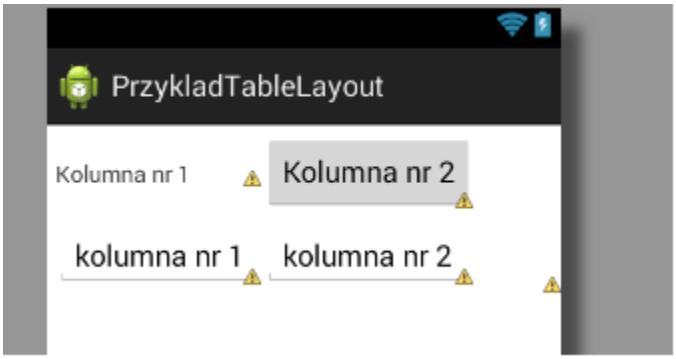
```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tableLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TableRow
        android:id="@+id/tableRow1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="5dip" >
        <TextView
            android:id="@+id/textView1"
            android:text="Kolumna nr 1"/>
        <Button
            android:id="@+id/button1"
            android:text="Kolumna nr 2" />
    </TableRow>

    <TableRow
        android:id="@+id/tableRow1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="5dip" >
      <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="kolumna nr 1"/>
      <EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="kolumna nr 2"/>
    </TableRow>



</TableLayout>
```

Kolejne elementy <TableRow> określają kolejne wiersze w tabeli. Kolumn będzie tyle, ile będzie elementów w wierszu w którym jest najwięcej elementów – dokładnie jak w HTML.

Domyślnie wyświetlana jest formatka do montowania elementów metodą drag & drop. Aby przejść do edycji ręcznej pliku kliknij zakładkę „activity_main.xml” znajdującą się u dołu.



Po wklejeniu kodu możesz wrócić do ekranu drag & drop i powinieneś zobaczyć taki widok:



Możesz przeciągać elementy z menu znajdującego się po lewej stronie, a następnie modyfikować ich położenie ręcznie , lub umieszczać elementy przy użyciu kodu a przesuwać graficznie. Nie ma to znaczenia, ponieważ wszystkie zmiany i tak zachodzą w pliku XML – edytor graficzny jest tylko nakładką :)

Linear layout – układ liniowy, obiekty ułożone pod sobą lub obok siebie

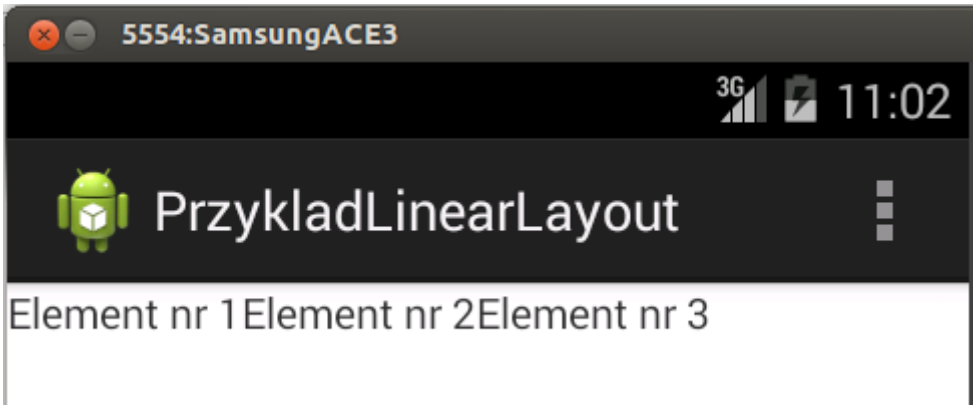
Ten rodzaj layoutu działa w oparciu o zasadę, że elementy znajdują się kolejno pod sobą lub obok siebie. Podobnie jak poprzednio, stwórz nowy projekt i zawartość pliku layoutu zamień na ten kod:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Element nr 1" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Element nr 2" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Element nr 3" />

</LinearLayout>
```

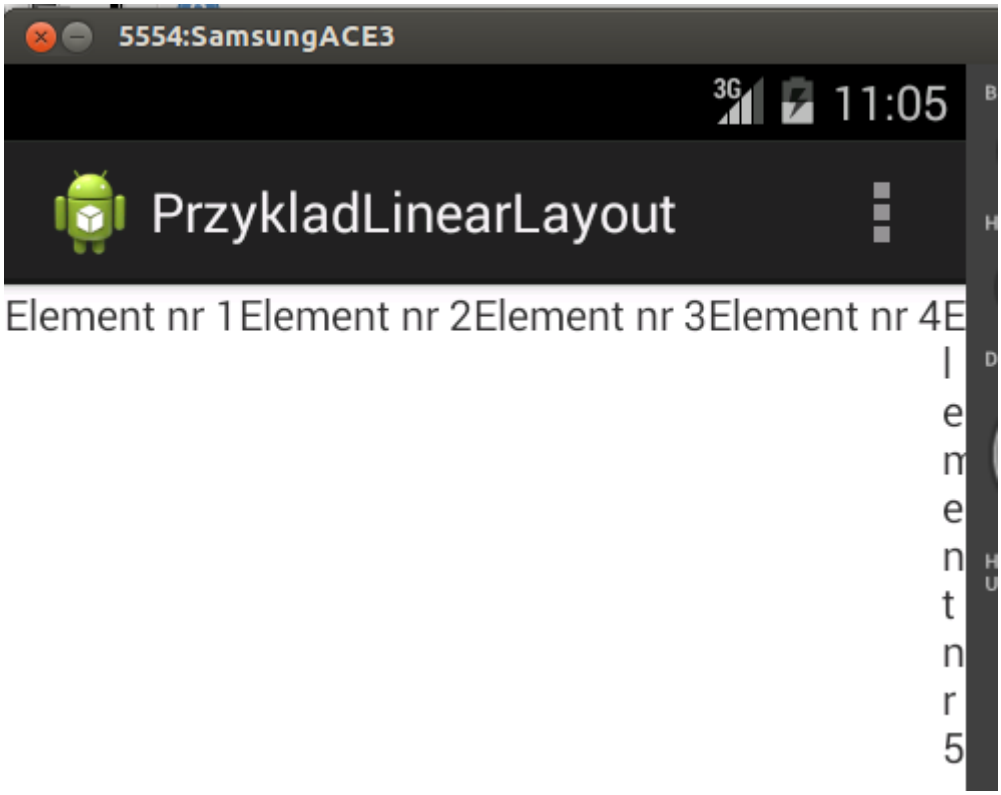
Określiliśmy tutaj 3 pola wyświetlające tekst. Zwróć przy okazji uwagę na element:

```
android:orientation="horizontal"
```

znajdujący się w tagu <LinearLayout... Horizontal określa, że elementy będą się znajdowały obok siebie, a nie pod sobą. Po uruchomieniu aplikacja wygląda tak:



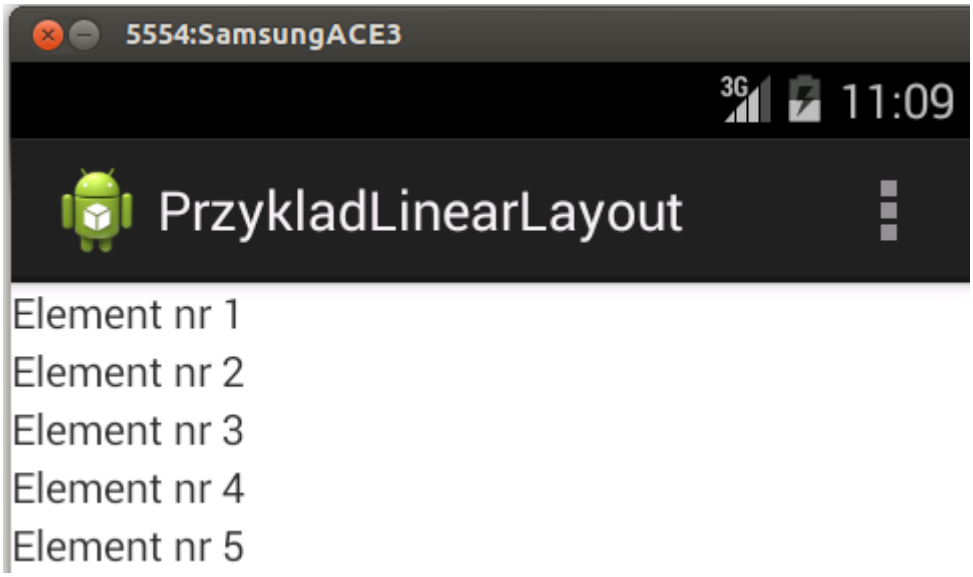
W ramach eksperymentu dodałem przy tych ustawieniach jeszcze dwa elementy TextView. Jak widać, nie wygląda to najlepiej. Do takiego efektu może dojść np. na telefonach o mniejszych ekranach lub rozdzielczości:



Modyfikuję teraz nieco element LinearLayout, konkretnie własność orientation z horizontal na vertical:

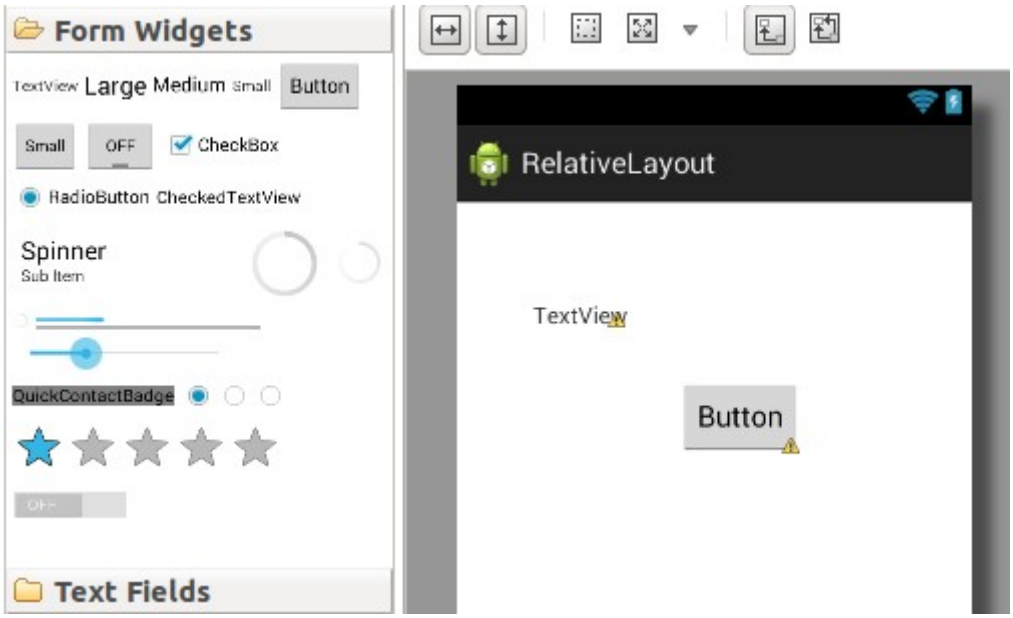
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
```

Ta zmiana sprawi, że elementy nie będą ze sobą sąsiadowały obok siebie, a jeden pod drugim.
Efekt:



RelativeLayout – położenie elementów względem innych elementów

Jeśli wykleisz sobie elementy na aktywności przy użyciu przeciągania komponentów z palety, domyślnie ich położenia zostaną określone jako położenie względem innego elementu. Dla przykładu przykleiłem dwa komponenty:



System domyślnie ustawia RelativeLayout. Tutaj zawartość wygenerowanego pliku layoutu:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="31dp"
        android:layout_marginTop="46dp"
        android:text="TextView" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_marginLeft="37dp"
        android:layout_marginTop="30dp"
        android:layout_toRightOf="@+id/textView1"
        android:text="Button" />

</RelativeLayout>
```

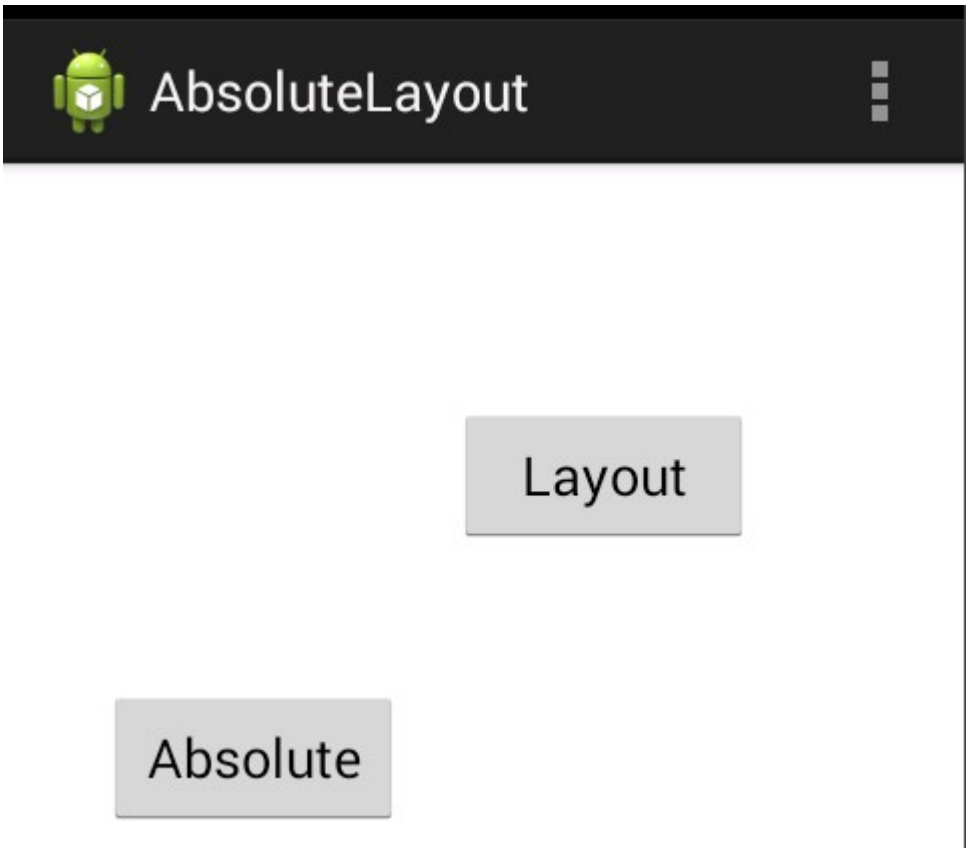
Jak widać, położenie elementu klasy TextView jest wskazane jako odległości od górnego i lewego marginesu, a położenie elementu Button jako odległości względne od elementu TextView. Takie podejście jest zrozumiałe przy przeciąganiu elementów, ale takie rozmieszczanie przy użyciu kodowania ręcznego nie byłoby specjalnie wygodne. Warto znać ten typ layoutu jedynie ze względu na możliwość nanoszenia ręcznych poprawek do automatycznie generowanego kodu. Chodzi o sytuacje gdy przeciągamy graficznie komponenty, ale nie możemy ustawić elementów tak jak byśmy sobie tego życzyli, bo nam się wszystko „rozjeżdża”.

AbsoluteLayout – położenia na sztywno

AbsoluteLayout jest takim rodzajem layoutu, gdzie ustalamy na sztywno x i y górnej lewej krawędzi elementu. Stwórz kolejny projekt i zamiast dotychczasowego kodu w pliku layoutu wstaw ten:

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="Absolute"
      android:layout_x="50px"
      android:layout_y="261px" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="Layout"
      android:layout_x="225px"
      android:layout_y="120px" />

</AbsoluteLayout>
```

Po uruchomieniu aplikacji:



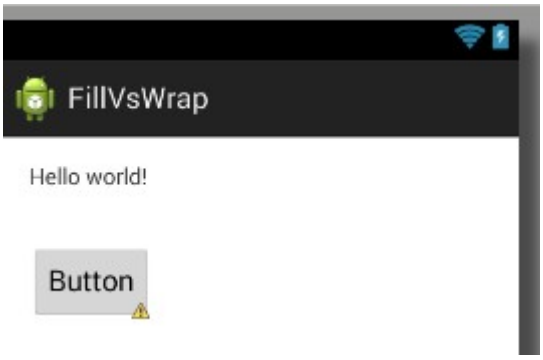
Z jednej strony, rozkład elementów nam się nie „rozjedzie”, ale z drugiej ustawienie współrzędnych elementu obowiązuje zarówno przy pionowym jak i poziomym położeniu ekranu. Jest też niezależne od wielkości i rozdzielczości ekranu. Może się więc okazać że na niektórych telefonach, albo po obróceniu ekranu, część komponentów znalazła się poza widocznym obszarem.

Parametr wrap_content vs fill_parent

Kiedy przyklejamy element taki jak guzik, w pliku layoutu dodawany jest wpis odnoszący się do tego obiektu. Taki jak ten:

```
<Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/textView1"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="32dp"
      android:text="Button" />
```

Parametry typu marginTop czy tekst są jasne, ale mamy turaj layout_height i layout_width określone nie jak się można by było spodziewać w pikselach, a jako „wrap_content”. Takie ustawienie jest domyślne i oznacza ono, że element ma być tak szeroki i wysoki by zmieścić znajdujący się w nim napis (lub obrazek w przypadku elementu imageView, czy inną zawartość w przypadku inych komponentów). Efekt wizualnie wygląda tak:



Alternatywnie zamiast wrap_content możemy wpisać fill_parent:

```
<Button
      android:id="@+id/button1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/textView1"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="32dp"
      android:text="Button" />
```

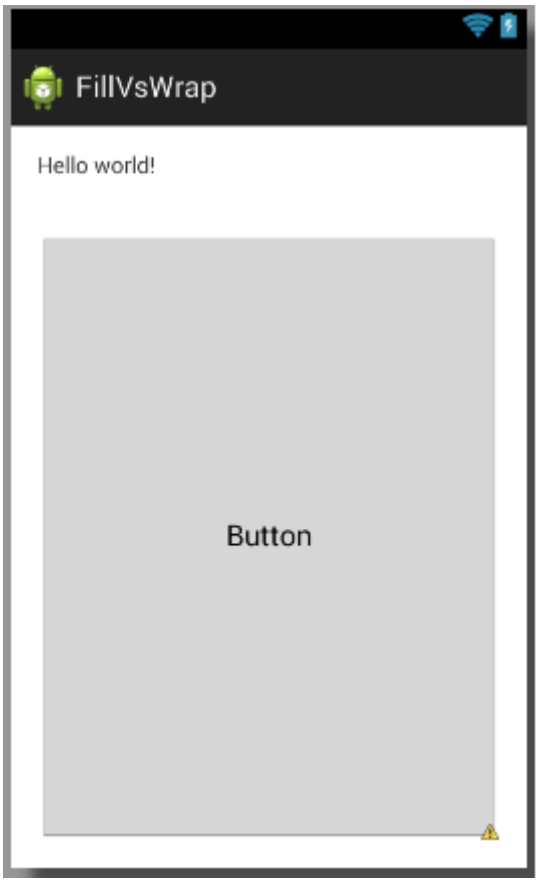
Takie ustawienie oznacza, że komponent ma się dopasować szerokością/wysokością do kontenera w którym się znajduje (w tym przypadku w ogóle na ekranie). Ustawilem parametr fill_parent dla szerokości.

Efekt prezentuje się tak:

Ponownie wprowadzam zmianę w konfiguracji. Tym razem ustawiam fill_parent również dla wysokości komponentu:

```
<Button
        android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="32dp"
        android:text="Button" />
```

Mamy w efekcie duuuży guzik :



Scroll View – ekran przewijany

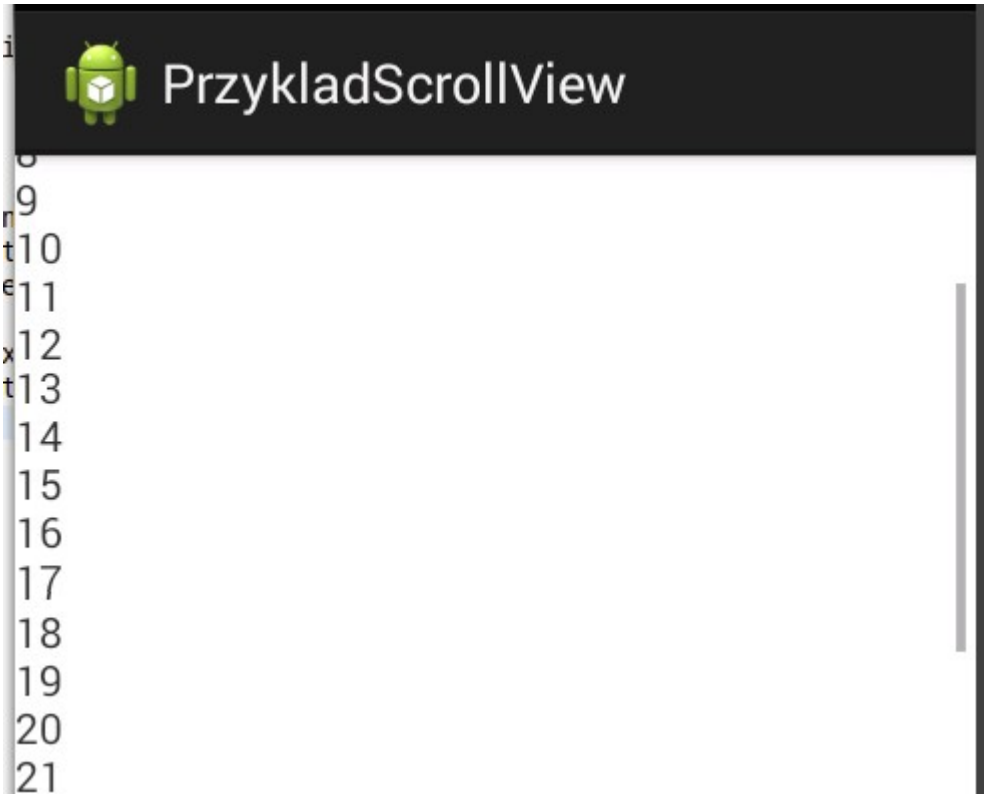
Niekiedy nie wystarczy nam ekranu na wszystkie elementy które chcielibyśmy umieścić. Przykładowo mamy aplikację z dużą ilością guzików w ramach menu wyboru. Guzików jest za dużo w stosunku do długości ekranu by mogły się zmieścić. W takich przypadkach stosuje się scroll view, czyli rodzaj layoutu który pozwala na przewijanie. Wejdź do menu ustawień w Androidzie a zrozumiesz o co chodzi. Aby przygotować taki wygląd, w pierwszej kolejności zmieniam layout w pliku xml layoutu na scroll view:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

        <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=" " />

</ScrollView>
```

Umieściłem też tam element tekstowy do którego wrzucę trochę tekstu, tak by było widać efekt przesuwania ekranu. W klasie aktywności dodaję kawałek kodu który do elementu tekstowego zdefiniowanego przed momentem wstawia kolejne liczby w kolejnych liniach:

```
8  public class MainActivity extends Activity {
9
10     TextView t;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16         t=(TextView) findViewById(R.id.textView1);
17         for(int x=0;x<=100;x++){
18             t.setText(t.getText()+""+x+"\n");
19         }
20     }
21
22
23 }
```

Po uruchomieniu mogę przeciągać ekran suwaczkiem znajdującym się po prawej stronie:



Czas na programowanie!

Wykorzystanie Log.d w debugowaniu

W „normalnej” Javie jeśli zechcielibyśmy sprawdzić co się dzieje w programie i gdzie się „przewraca”, najpewniej użylibyśmy metody typu

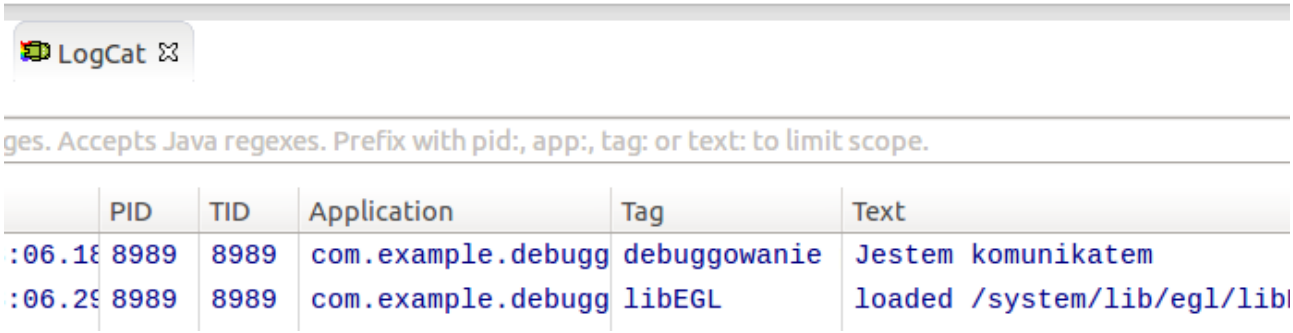
System.out.println(”Trololo, jestem po obliczeniach”);

:)

W Androidzie nie ma konsoli systemowej (javovej), mamy za to możliwość wyrzucania komunikatów na konsolę LogCat w równie prosty sposób. Wystarczy takie wywołanie:

```
Log.d("debuggowanie programu", "Jestem komunikatem");
```

a w konsoli LogCata zobaczymy komunikat :



To co podamy do metody „d” jako pierwszy parametr trafi do kolumng „Tag”, to co podamy jako drugi parametr trafi do kolumny „Text”. Metoda ta działa zarówno przy uruchamianiu programu na wirtualnym urządzeniu, jak i fizycznym sprzęcie podpiętym do komputera.

Cykl życia aplikacji

Aktywność może znajdować się w czterech stanach :

1. Jest na pierwszym planie

2. Jest wyświetlana, ale nie znajduje się na pierwszym planie – np. kiedy wyświetlimy jakieś małe okienko z komunikatem

3. Jest wyłączona. Nie działa, nie ma jej w pamięci operacyjnej.

4. Wstrzymana. Nie jest zamknięta, ale też nie jest widoczna. Działa w tle. Tak się stanie kiedy przejdziemy z jednej aktywności do drugiej. Aktywność pozostająca w tym stanie może w każdej chwili zostać zabita przez system, w razie gdyby ten potrzebował pamięci którą ta aktywność wykorzystuje.

Klasa dziedzicząca po klasie Activity będzie dziedziczyła kilka metod które są uruchamiane przy przechodzeniu aktywności między stanami. Możemy je przesłonić. Wprowadź taki kod jak na poniższej ilustracji, uruchom aplikację a następnie pobaw się urządzeniem tj. poobracaj je, zamknij aplikację, otwórz jeszcze raz, naciśnij przycisk „Home” itp. Popatrz co się dzieje podczas wykonywania tych czynności na konsoli LogCat (znajduje się w dolnej części ekranu Eclipse).

```
 8  public class MainActivity extends Activity {
 9
10      @Override
11      protected void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.activity_main);
14          Log.d("Komunikat!","Create!");
15      }
16
17
18      @Override
19      public void onStop() {
20          Log.d("Komunikat!","Stop!");
21          super.onStop();
22      }
23
24      @Override
25      public void onPause(){
26          Log.d("Komunikat!","Pause!");
27          super.onStop();
28      }
29
30      @Override
31      public void onResume(){
32          Log.d("Komunikat!","Resume!");
33          super.onStop();
34      }
35
36
37      @Override
38      public boolean onCreateOptionsMenu(Menu menu) {
39          // Inflate the menu; this adds items to the actior
40          getMenuInflater().inflate(R.menu.main, menu);
41          return true;
42      }
43
44  }
```

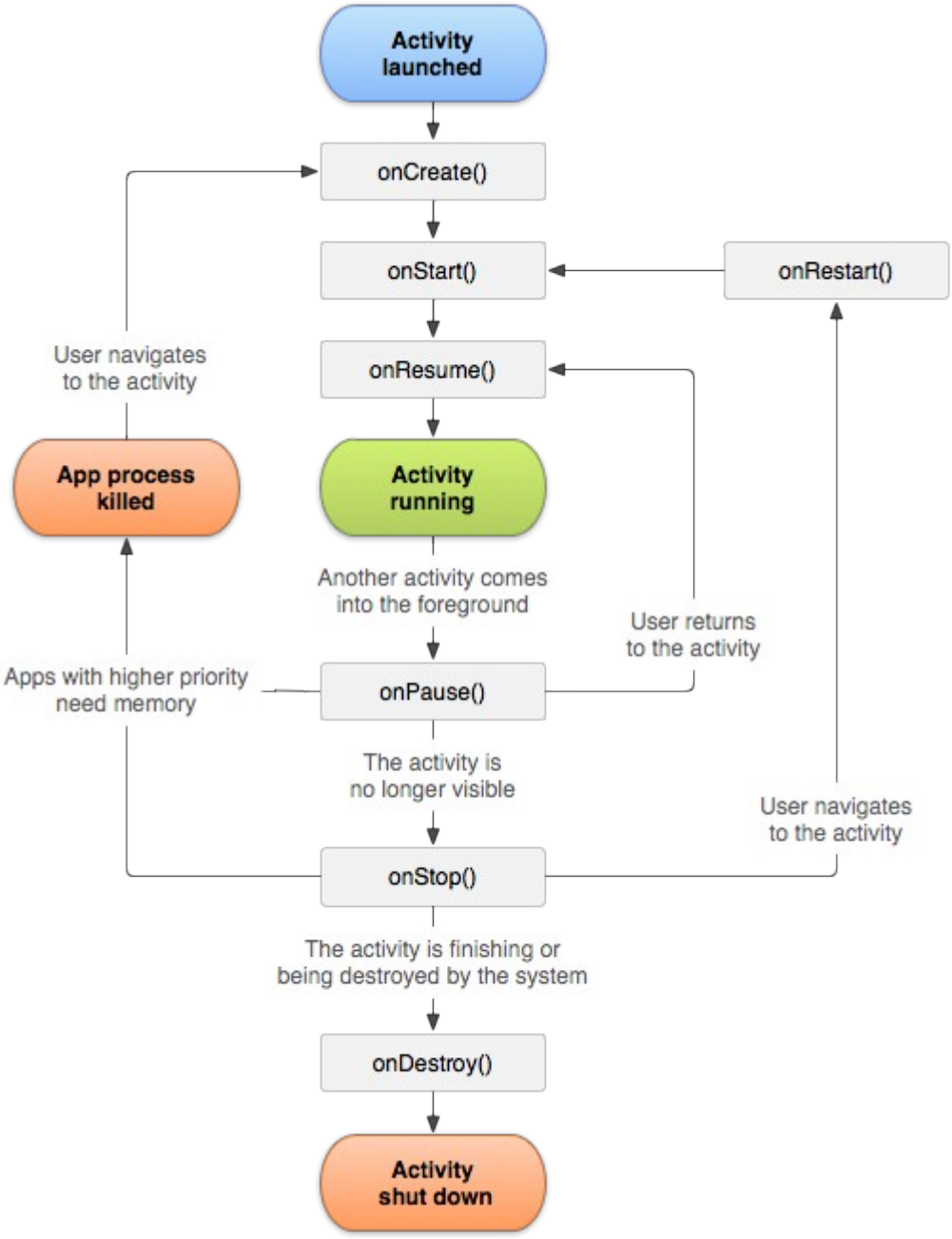
Metoda **onCreate** jest wywoływana wtedy, kiedy aktywność jest uruchamiana po raz pierwszy. Z tego powodu to właśnie ją wykorzystuje się najczęściej i to właśnie w niej ustawiamy wygląd ekranu oraz ustawienia początkowe.

Metoda **onStop** jest wywoływana kiedy aplikacja jest zamykana. Możesz tutaj wprowadzić kod który zapisze stan aplikacji np. do bazy danych.

Metoda **onPause** jest wywoływana np. kiedy obracamy ekran, a także przed metodą onStop w przypadku zamknięcia aplikacji.

Metoda **onResume** jest wywoływana również kiedy obracamy ekran, lub przy powrocie do działania po wcześniejszym zapałzowaniu.

Pełen diagram przechodzenia aplikacji przez różne stany (pochodzi z dokumentacji Androida):



Komunikaty Toast

Komunikaty „Toast” służą do krótkotrwałego wyświetlania mało ważnych komunikatów. Komunikat pojawia się na kilka sekund i znika, użytkownik może nawet tego nie zauważyć, więc nie powinieneś w ten sposób wyświetlać np. komunikatów o błędach. Doskonale się za to nadaje to wyświetlania np. podpowiedzi.

```
package com.example.toastprzyklad;

import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toast.makeText(getApplicationContext(), "Jestem wiadomością...", Toast.LENGTH_SHORT).show();

    }

}
```

Wszystko sprowadza się tak naprawdę do linii:

```
Toast.makeText(getApplicationContext(), "Jestem wiadomością...",
Toast.LENGTH_SHORT).show();
```

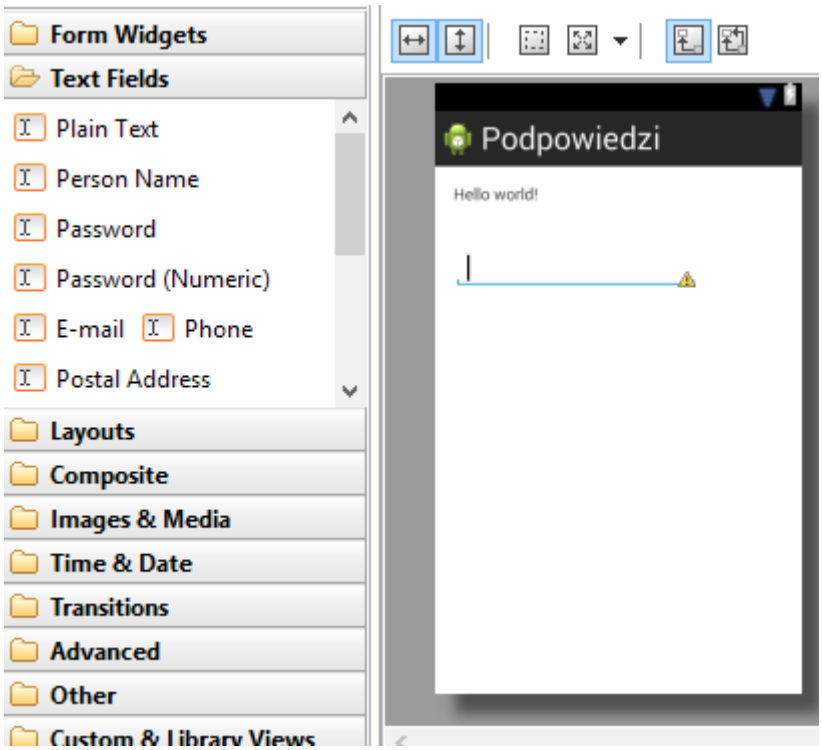
Która to jest odpowiedzialna za cały komunikat. Pozostała część kodu widoczna na obrazie powyżej nie jest dla nas istotna.

Tutaj zrobiłem tak, że komunikat pojawia się zaraz po uruchomieniu programu. Parametr `Toast.LENGTH_SHORT` określa czas wyświetlania komunikatu. Zamiast `Toast.LENGTH_SHORT` możemy też dać `Toast.LENGTH_LONG` ,wtedy komunikat będzie się wyświetlał nieco dłużej. W pierwszej wersji jest to około 2 sekund, w drugiej około 4. Niestety nie ma możliwości prostej ręcznej konfiguracji czasu wyświetlania takiego komunikatu dla pojedynczej aplikacji. Efektem jest wyświetlenie u dołu aplikacji takiego komunikatu:

Podpowiedzi w oknach edycyjnych

Na niektórych urządzeniach z Androidem mamy mały ekran. W takich sytuacjach nie ma miejsca na dodatkowe „dymki” z podpowiedziami lub pola opisowe. Alternatywą mogą być podpowiedzi w samych oknach edycyjnych. Podpowiedź będzie „wyszarzona” i będzie mówiła co w danym oknie należy wprowadzić.

Zaczynam od przyklejenia na ekranie jednego komponentu klasy PlainText :

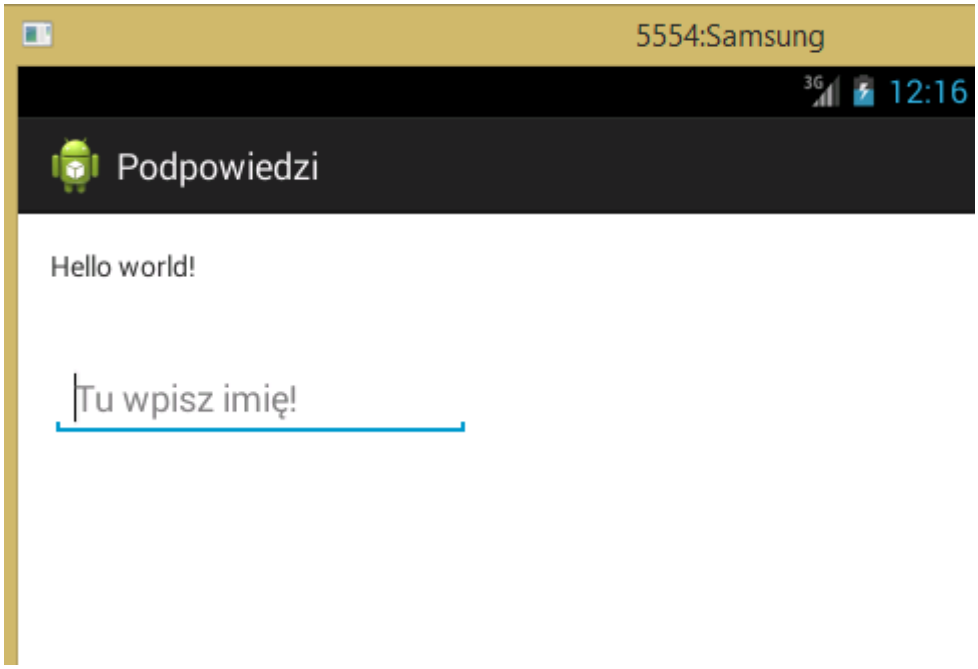


Dalej w klasie aktywności wykorzystuję metodę „setHint” klasy EditText. Zawartość linii nr 14 służy jedynie podpięciu referencji do obiektu (muszę się jakoś do niego odnieść). Samo ustawienie w związku z umieszczeniem go w metodzie „onCreate” będzie działało już od uruchomienia.

activity_main.xml MainActivity.java

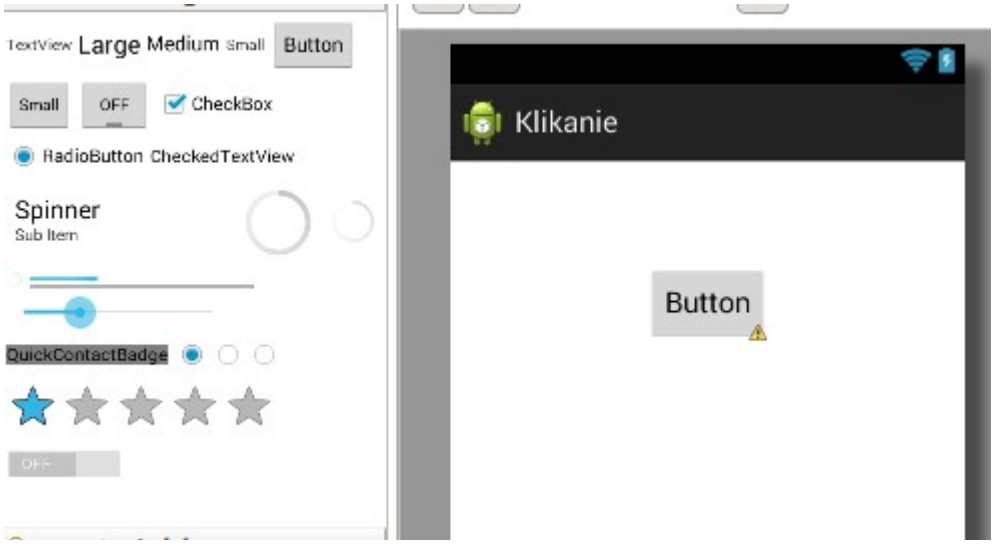
```
1  package com.example.podpowiedzi;
2
3  import android.os.Bundle;
7
8  public class MainActivity extends Activity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14         EditText et =(EditText) findViewById(R.id.editText1);
15         et.setHint("Tu wpisz imię!");
16     }
17
18
19
20 }
21
```

Efekt:



Obsługa zdarzenia kliknięcia na komponent

Obsługiwać zdarzenie kliknięcia komponentu można na kilka sposobów. Przedstawię trzy najpopularniejsze. Do wszystkich trzech sposobów będzie nam potrzebny jakiś komponent na którego kliknięcie będziemy reagować. Z tego powodu przyklejam na ekranie jeden przycisk.

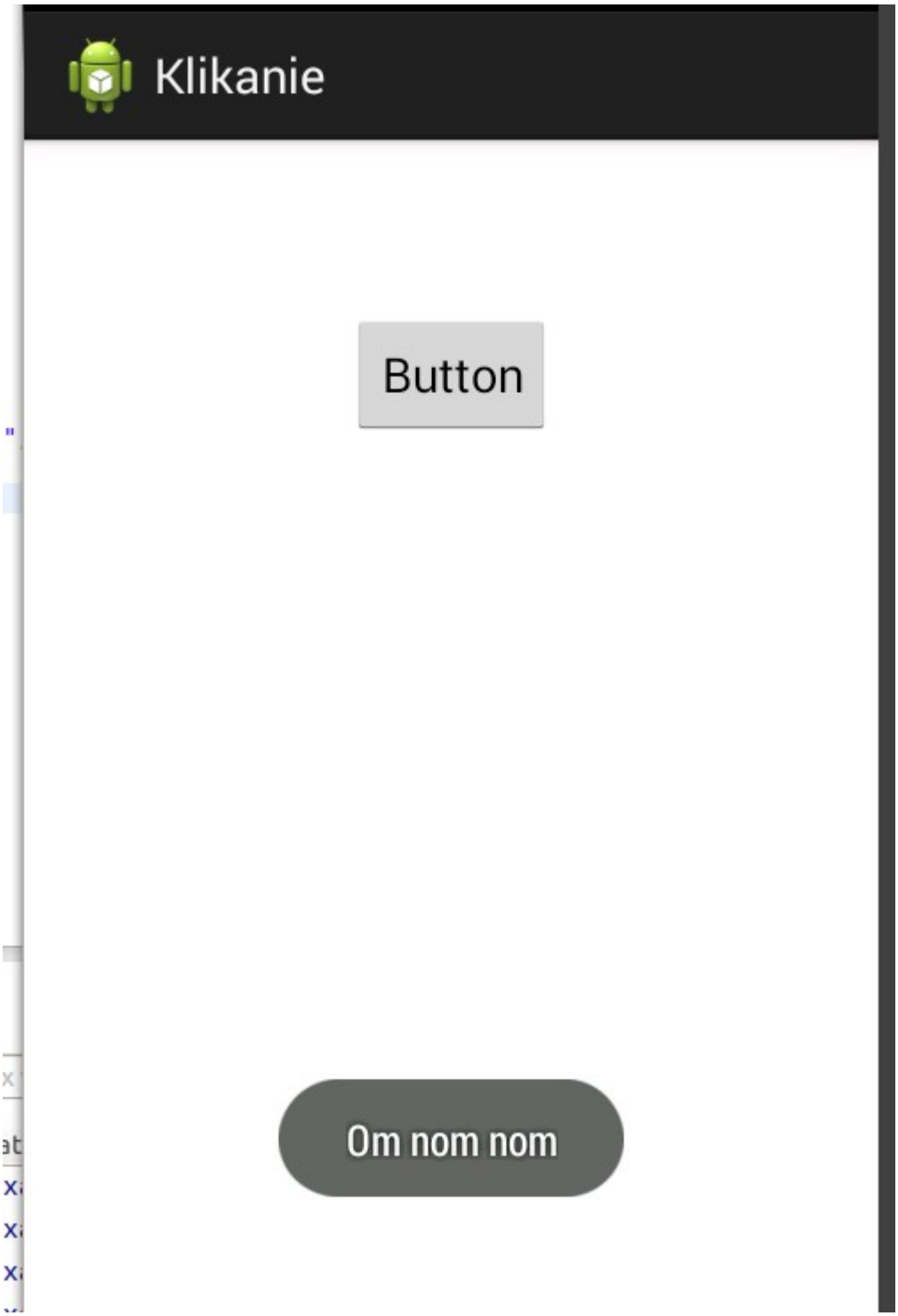


Pierwszy najprostszy w przygotowaniu i chyba najbardziej przejrzysty, ale też najmniej czytelny i wymagający dużo kodu. Tworzymy osobno listener kliknięcia (czyli słuchacza zdarzenia kliknięcia – czeka aż klikniesz i reaguje) i podpinamy go do guzika:

```
10
11 public class MainActivity extends Activity {
12
13     Button b1;
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18         b1=(Button) findViewById(R.id.button1);
19         OnClickListener l = new OnClickListener() {
20
21             @Override
22             public void onClick(View arg0) {
23                 Toast.makeText(getApplicationContext(),"Om nom nom", Toast.LENGTH_SHORT).show();
24
25             }
26         };
27         b1.setOnClickListener(l);
28
29     }
30
31
32 }
```

To co się ma wydarzyć w wyniku kliknięcia jest opisane w metodzie onClickk listenera. W tym przypadku jest to wyświetlenie komunikatu Toast z tekstem „om nom nom” (linia 23). Samo podpięcie stworzonego listenera odbywa się w linii 27.

Po kliknięciu:



Alternatywnie choć na podobnej zasadzie możemy utworzyć obiekt listenera w momencie jego tworzenia – linia 19. Mniej kodu, ale czytelność na tym traci. Efekt działania identyczny jak poprzednio.

```
11 public class MainActivity extends Activity {
12
13     Button b1;
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18         b1=(Button) findViewById(R.id.button1);
19         b1.setOnClickListener(new OnClickListener() {
20
21             @Override
22             public void onClick(View v) {
23                 Toast.makeText(getApplicationContext(),"Om nom nom", Toast.LENGTH_SHORT).show();
24             }
25         });
26     }
27
28
29 }
```

Trzecia alternatywna metoda jest chyba najmniej intuicyjna, ale jak się zrozumie zasadę działania to kod przy większej ilości guzików do obsługi będzie najbardziej przejrzysty i bardzo wygodny w rozwijaniu. Tym razem nie definiuję żadnego listenera! Wszystko co ma się zdarzyć po kliknięciu przycisku jest opisane w metodzie obslugaKlikniecia. Wazne by metoda ta przyjmowala jako parametr obiekt klasy View (po tej klasie dziedziczą wszystkie komponenty takie jak guzik i to właśnie pod postacią tego parametru jest przekazywany obiekt komponentu do metody) i nic nie zwracała. Pozornie metoda jako taka nie jest w żaden sposób związana z obsługą kliknięcia na przycisk:

```
11 public class MainActivity extends Activity {
12
13
14     public void obslugaKlikniecia(View v){
15         Toast.makeText(getApplicationContext(),"Om nom nom", Toast.LENGTH_SHORT).show();
16     }
17
18     Button b1;
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23         b1=(Button) findViewById(R.id.button1);
24
25     }
26
27
28 }
```

Magia zaczyna dziać się dopiero w pliku layoutu związanym z tą aktywnością:

```
10
11      <Button
12          android:id="@+id/button1"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:layout_alignParentTop="true"
16          android:layout_centerHorizontal="true"
17          android:layout_marginTop="48dp"
18          android:text="Button"
19          android:onClick="obslugaKlikniecia"
20           />
```

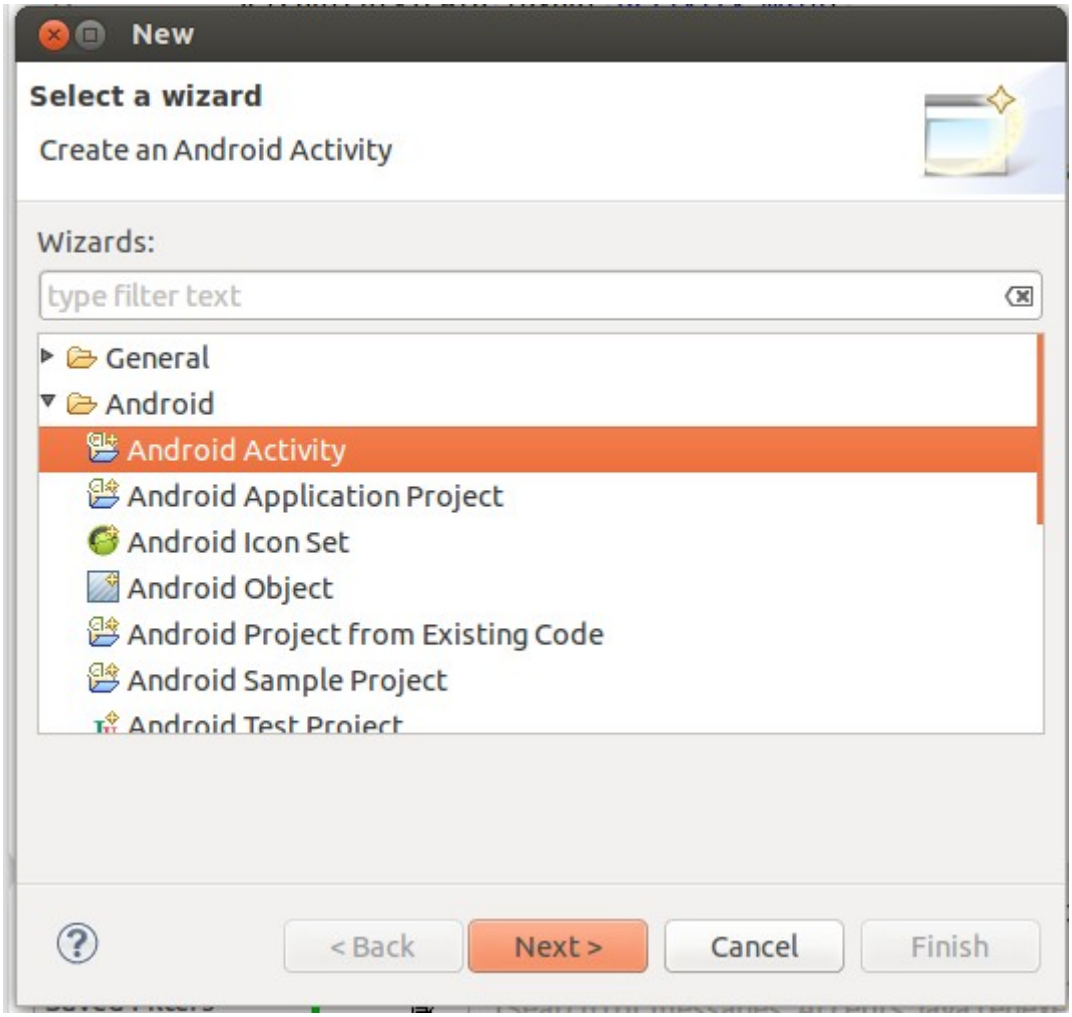
Pojawił nam się tutaj (linia 19) nowy element: android:onClick. Zamiast związywać na poziomie kodu javowego metodę obslugaKlikniecia z specjalnie do tego tworzonym listenerem, wskazujemy nazwę metody przy użyciu parametru przycisku w pliku layoutu. Takie rozwiązanie ma ten plus, że pisze się znacznie mniej kodu, a sam kod przy większej liczbie obsługiwanych przycisków jest czytelniejszy. Z drugiej strony, jeśli nie będziemy pamiętać o takim rozwiązaniu, możemy za jakiś czas zachodzić w głowę jak to zostało zrobione :)

Pamiętajmy że obsługa kliknięcia nie dotyczy tylko przycisków, ale każdego komponentu dziedziczącego po View – w tym napisy i pola edycyjne.

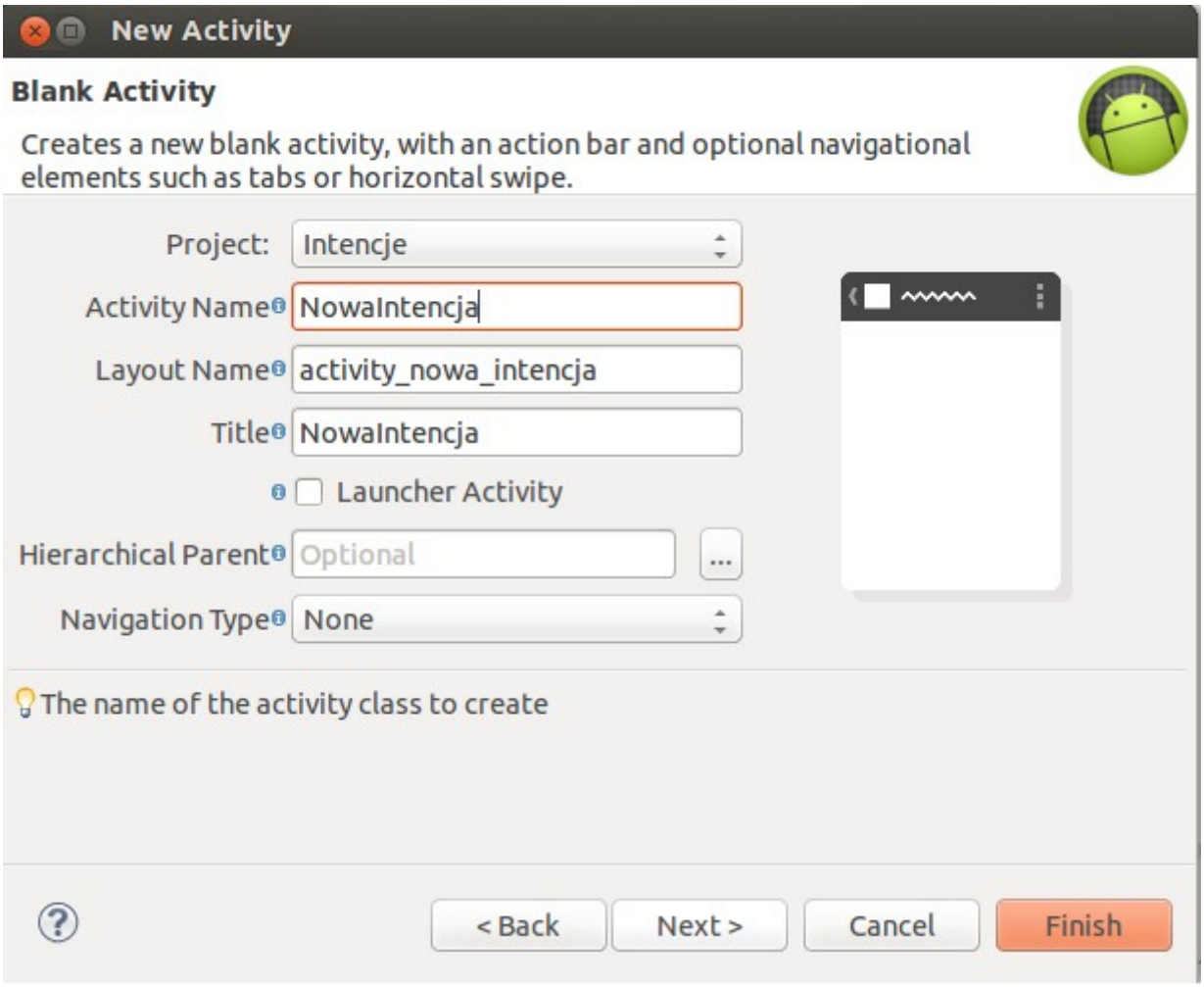
Intencje – wywoływanie ekranów i akcji w systemie

Intencja to zamiar wykonania jakiejś akcji, działanie np. wyślij SMS, wyświetl stronę www, zrób zdjęcie,wyświetl inny ekran. Intencje mogą być różnego typu. W zależności od potrzeb tworzymy intencję danego typu (do zrobienia zdjęcia, uruchomienia przeglądarki, wywołania jakiejkolwiek innej akcji) a następnie uruchamiamy ją. Przy okazji możemy przekazać jakieś parametry przez intencję. Np. z widoku listy wszystkich faktur uruchamiamy ekran z podglądem jednej wybranej. Musimy wtedy przekazać id faktury do wyswietlenia.

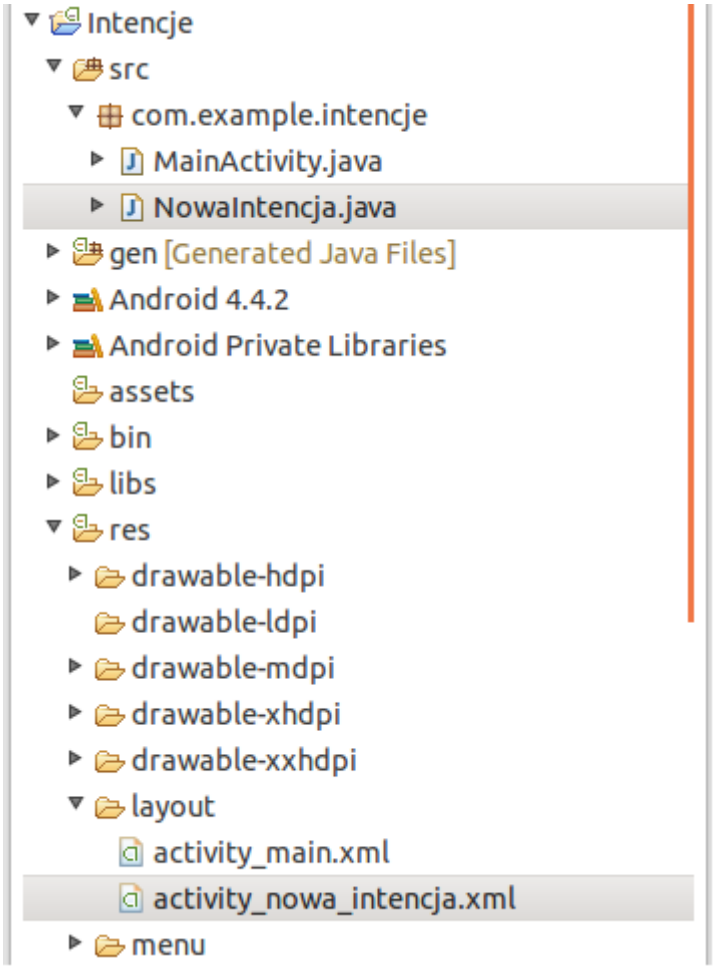
Pierwszy przykład to będzie zwykłe wyswietlenie innego ekranu aplikacji. Zaczynam od stworzenia zwykłego projektu na Androida. Następnie dodaję nową aktywność klikając PPM na pakiecie i wybierając New-->Other i z okna które się pojawi wybieram Android Activity:



Nadaję nazwy klasie tej aktywności oraz plikowi XML w którym będzie opisany Layout.



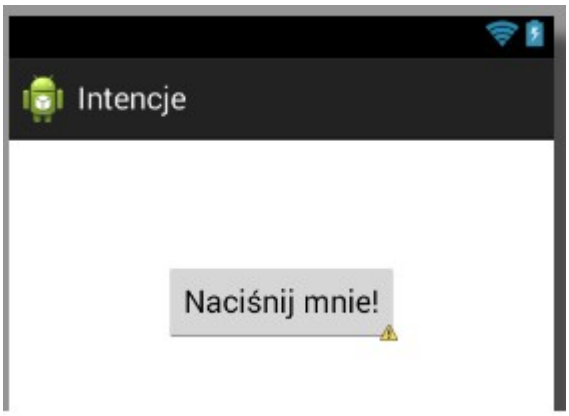
Po zatwierdzeniu w projekcie pojawiają mi się owe dwa pliki:



A w pliku manifestu nowy wpis:

```
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.intencje.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.example.intencje.NowaIntencja"
            android:label="@string/title_activity_nowa_intencja" >
        </activity>
    </application>

</manifest>
```

Na ekranie aktywności głównej przyklejam jeden przycisk:

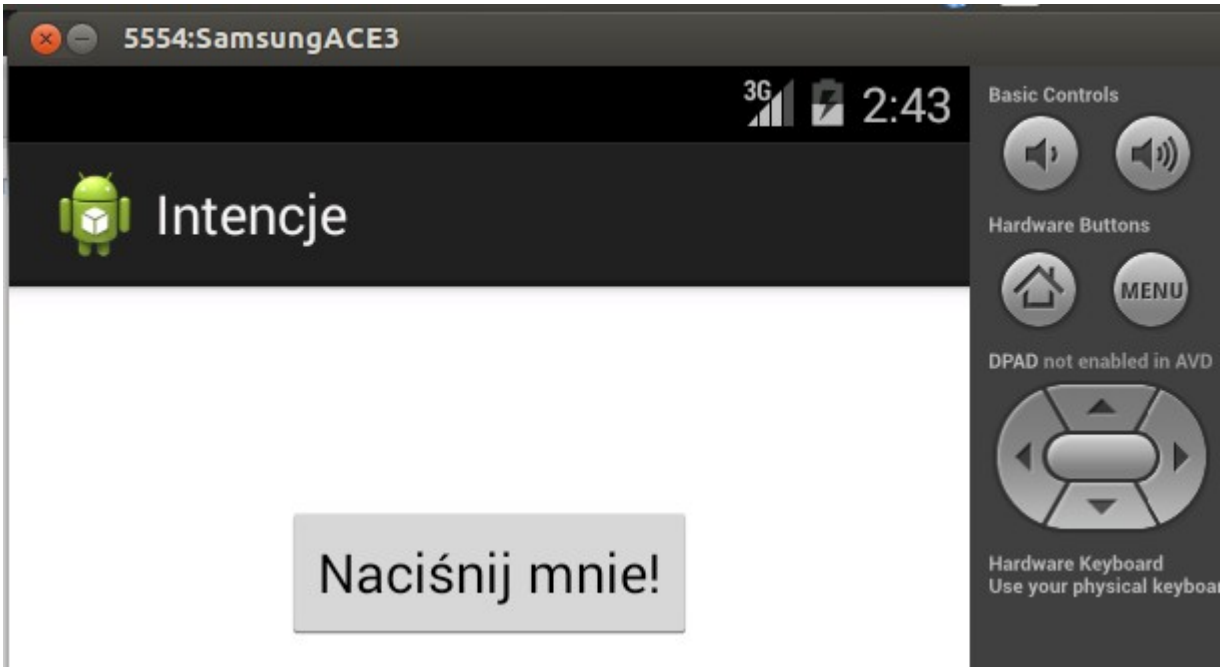


Przejdźmy teraz do kodu:

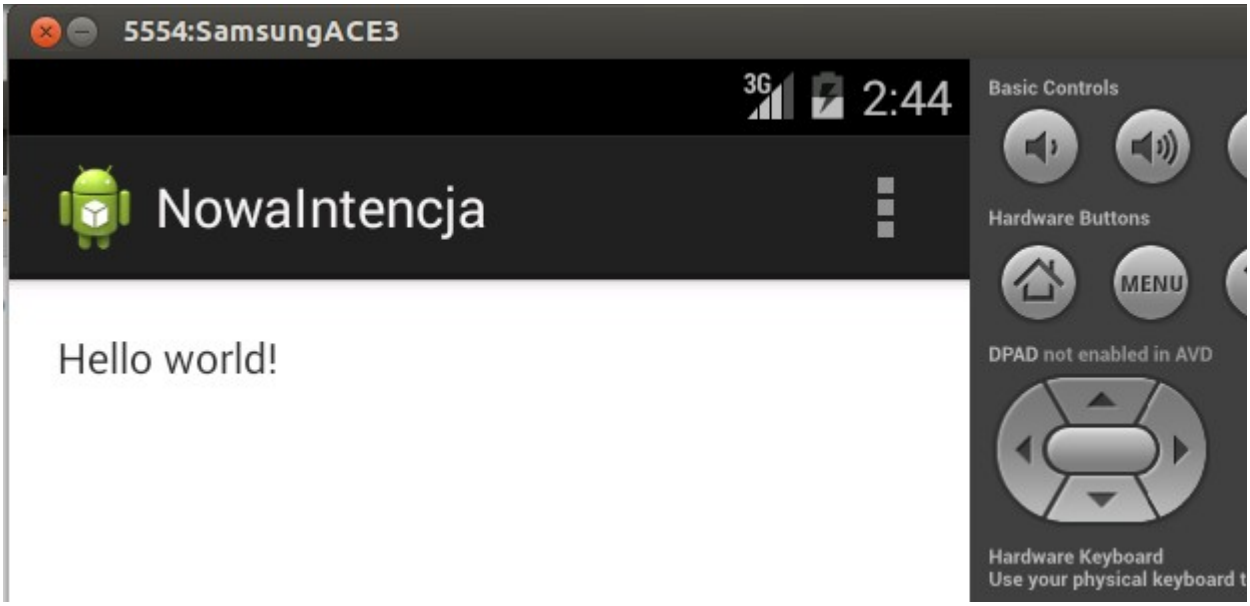
```
 3 import android.os.Bundle;
10
11 public class MainActivity extends Activity {
12
13     public void reakcja(){
14         Intent i = new Intent(this,NowaIntencja.class);
15         startActivity(i);
16     }
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_main);
22
23
24         Button b=(Button)findViewById(R.id.button1);
25         OnClickListener l = new OnClickListener() {
26             @Override
27             public void onClick(View v) {
28                 reakcja();
29             }
30         };
31         b.setOnClickListener(l);
32
33
34     }
35
36
37
38 }
39
```

Linie 24-31 to podpięcie wywołania metody reakcja() pod naciśnięcie guzika. W samej metodzie reakcja jest napisane co się ma w takiej sytuacji zadziać. Jak widać tworzę nowy obiekt klasy Intent (linia 14) i w kolejnej linii uruchamiam intencje metodą startActivity (dziedziczoną po klasie Activity). Ponieważ intencją którą wywołuję jest inna aktywność, muszę przekazać jej kontekst, oraz wskazać klasę aktywności która ma zostać wywołana. Nie zapominamy oczywiście by po stworzeniu intencji ją wywołać :) Druga aktywność – ta którą wywołuję to zwykły najprostszy ekran z napisem „Hello World!”.

Zobaczmy działanie aplikacji. Aplikacja tuż po uruchomieniu wczytuje moją główną aktywność:



Po kliknięciu powyższego przycisku mamy taki widok:



Została wywołana inna aktywność.

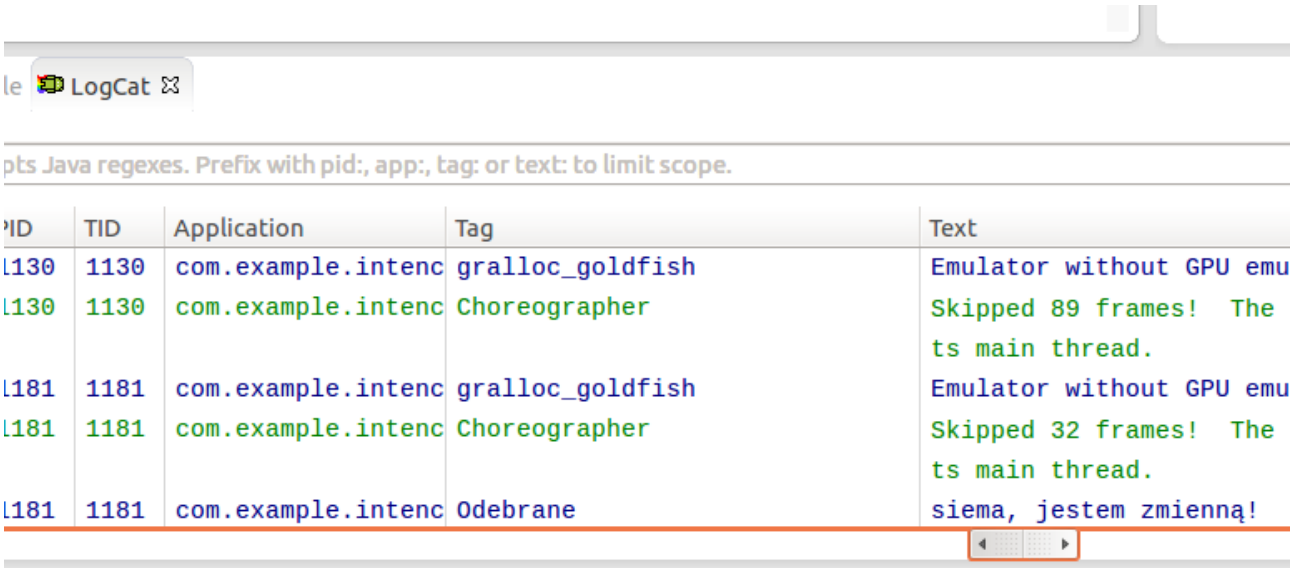
Rozbudujmy nieco naszą aplikację. Do wywołania innego ekranu dodam teraz przekazanie zmiennej. Wykorzystuję do tego metodę intencji putExtra. Musimy pamiętać by wywoływać ją przed startActivity(Intent i). W tym przypadku przekazuję tekst : „siema jestem zmienną!” pod postacią zmiennej o nazwie „zmienna”. Tekst ten zostanie przekazany do właśnie wywoływanej aktywności i wyświetlony na konsoli (o wyswietlenie tego tekstu zadbam już na poziomie tej nowej aktywności):

```
10
11 public class MainActivity extends Activity {
12
13     public void reakcja(){
14         Intent i = new Intent(this,NowaIntencja.class);
15         i.putExtra("zmienna", "siema, jestem zmienną!");
16         startActivity(i);
17     }
18
```

Aby odebrać przekazany obiekt, tworzę po stronie wywoływanej aktywności obiekt klasy Bundle z którego pobieram zbiór przekazanych obiektów przy użyciu metody getExtras(). Z wiązki tej muszę teraz wybrać obiekt który mnie interesuje (linia 16). Odebrany tej tekst wyświetlam teraz na konsoli LogCata:

```
 8 public class NowaIntencja extends Activity {
 9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_nowa_intencja);
14
15         Bundle b = getIntent().getExtras();
16         String odbior=b.getString("zmienna");
17         Log.d("Odebrane",odbior);
18     }
19
```

Efekt po uruchomieniu i kliknięciu przycisku:



Przyjrzyjmy się teraz innemu przykładowi aktywności. Tym razem będzie to wywołanie strony internetowej. Podaję do systemu intencję będącą żądaniem otwarcia strony internetowej:

```
11
12 public class MainActivity extends Activity {
13
14     public void reakcja(){
15         /*Intent i = new Intent(this,NowaIntencja.class);
16         i.putExtra("zmienna", "siema, jestem zmienną!");
17         startActivity(i);*/
18         Uri u =Uri.parse("http://jsystems.pl");
19         Intent i = new Intent(Intent.ACTION_VIEW, u);
20         startActivity(i);
21     }
22
```

Platforma Android sprawdza jaki program odpowiada za obsługę takich intencji i przekazuje mu ją. Ponieważ chcemy nawiązać połączenie z internetem, musimy zadbać o dodanie stosownych uprawnień do pliku manifestu:

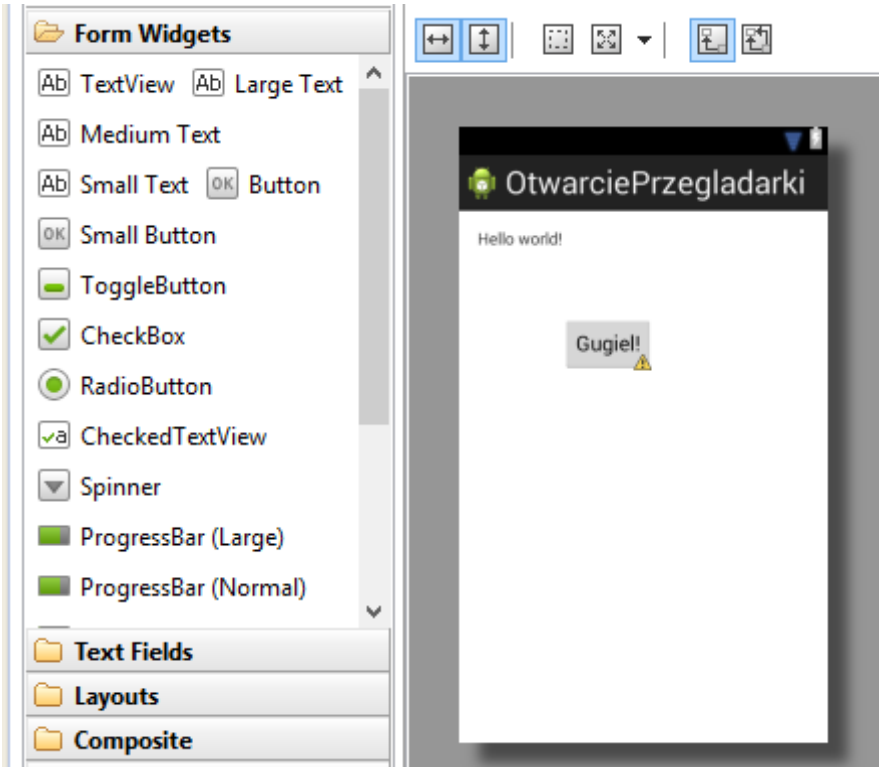
```
25         <activity
26             android:name="com.example.intencje.NowaIntencja"
27             android:label="@string/title_activity_nowa_intencja" >
28         </activity>
29     </application>
30
31 <uses-permission android:name="android.permission.INTERNET"/>
32
33 </manifest>
34
```

Efekt:


5554:SamsungACE3
jsystems.pl
JSYSTEMS
Strona Główna
Szkolenia
Usługi
Referencje
Blog
Kontakt
Dlaczego my?
Wybierz dziedzinę szkoleń
Masz pytania?
Bazy danych
Programowanie
Inne

Otwieranie przeglądarki WWW z aplikacji

W tym przykładzie pokażę w jaki sposób możemy otworzyć zewnętrzną (domyślną) przeglądarkę z poziomu aplikacji. Zaczynam jak zawsze od najprostszego przykładu. Stworzyłem nowy projekt i przykleiłem guzik z napisem „Gugiel!”. Po naciśnięciu przycisku powinna zostać uruchomiona przeglądarka z uruchomionym adresem http://google.pl

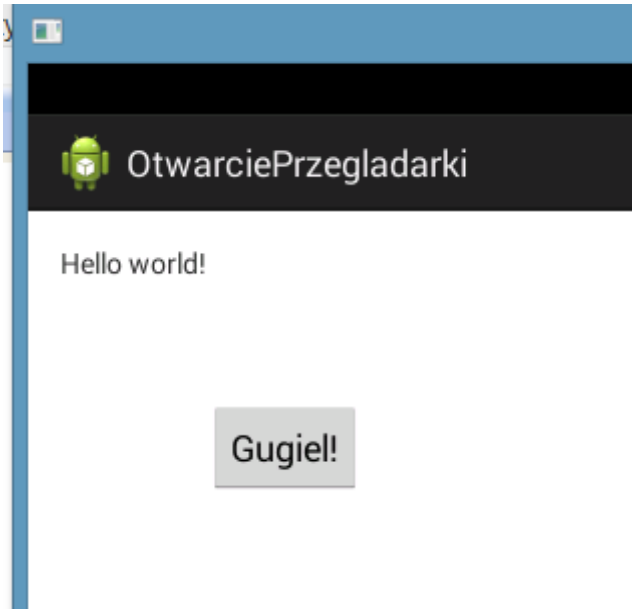


Przechodzimy teraz do kodu aktywności. Linia 18 to podpięcie referencji do guzika. Muszę to zrobić by móc sięgo niego odwoływać. Linie 19-27 to podpięcie listenera pod kliknięcie guzika Właściwa akcja dzieje się w linijkach 22-24 i w zasadzie tylko do nich się ogranicza. Adres który ma zostać otwarty będę musiał przekazać jako obiekt klasy Uri, tworzę więc go w oparciu o adres http://google.pl. Tu może być np. strona Twojej firmy, albo link do wyszukiwarki z przekazanymi przez pasek parametrami. Aby uruchomić przeglądarkę musimy stworzyć podając adres uri i wywołać nową intencję – co zawiera się w liniach 23,24.

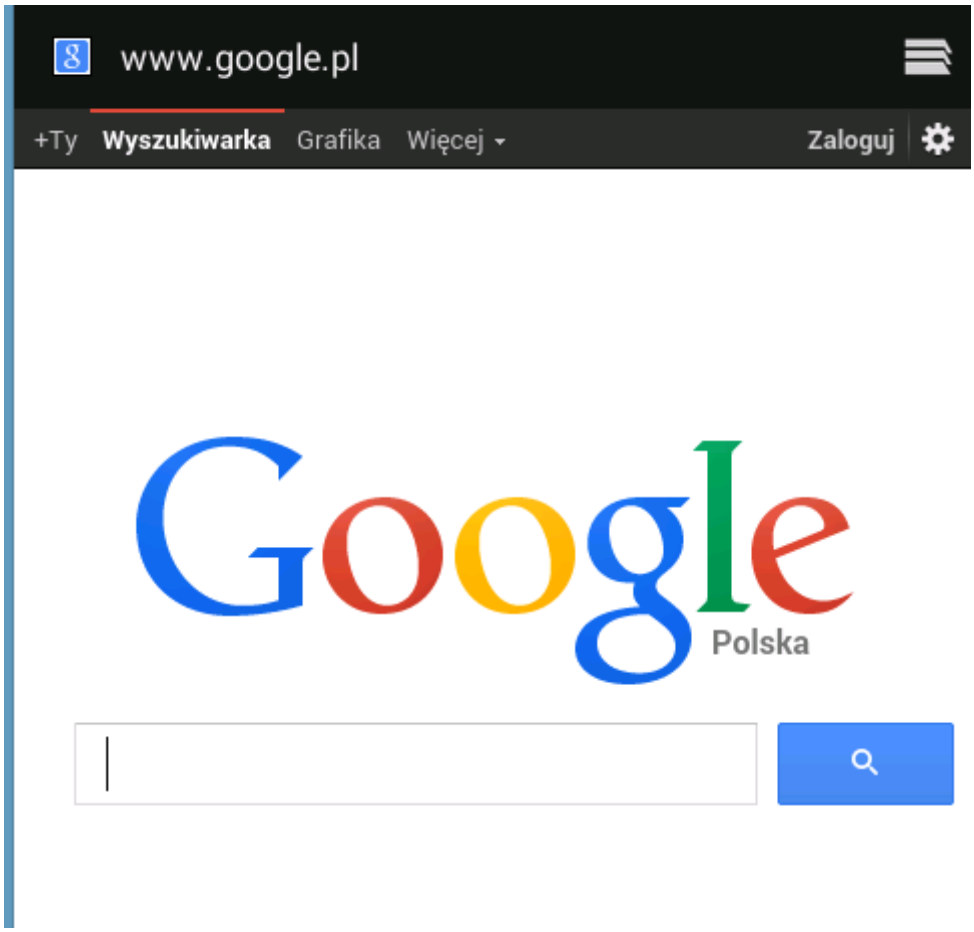
activity_main.xml MainActivity.java

```
1  package com.example.otwarcieprzegladarki;
2
3  import android.net.Uri;
4  import android.os.Bundle;
5  import android.app.Activity;
6  import android.content.Intent;
7  import android.view.Menu;
8  import android.view.View;
9  import android.view.View.OnClickListener;
10 import android.widget.Button;
11
12 public class MainActivity extends Activity {
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18         Button b = (Button) findViewById(R.id.button1);
19         OnClickListener l = new OnClickListener() {
20             @Override
21             public void onClick(View v) {
22                 Uri uri = Uri.parse("http://google.pl");
23                 Intent i = new Intent(Intent.ACTION_VIEW,uri);
24                 startActivity(i);
25             }
26         };
27         b.setOnClickListener(l);
28     }
29
30
31 }
```

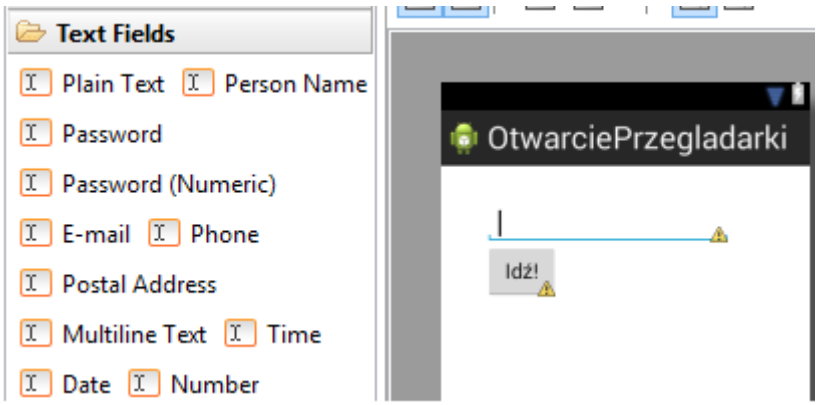
Ekran po uruchomieniu:



Po naciśnięciu przycisku uruchomiła mi się domyślna przeglądarka z uruchomionym wskazanym adresem:



Troszkę przerobimy teraz naszą aplikację. Użytkownik sam wpisze adres strony która ma zostać wywołana. Dodaję do widoku element EditText i zmieniam tekst na guziku:

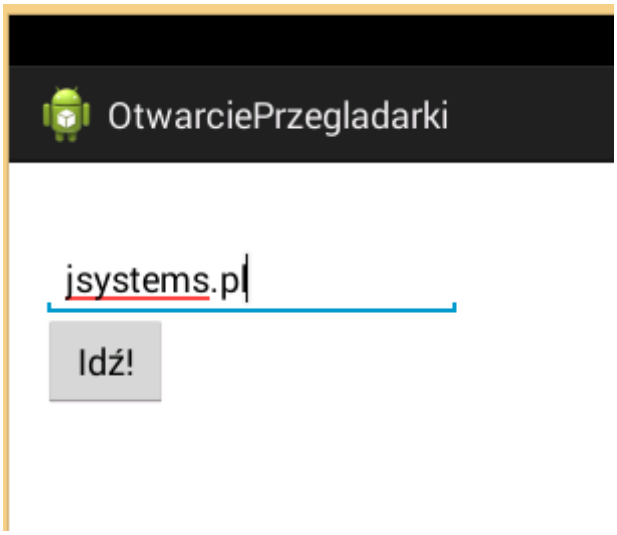


Musi za tym iść mała przeróbka w kodzie. Interesują nas linie 23 i 24. W linii 23 dodałem uchwyt do okienka edycyjnego. W linii 24 konkatenuję adres url na podstawie początku http:// (bo o tym pewnie każdy użytkownik zapomni, a bez tego nie zadziała) oraz tekstu wpisanego w oknie edycyjnym. Pozostała część programu pozostaje bez zmian.

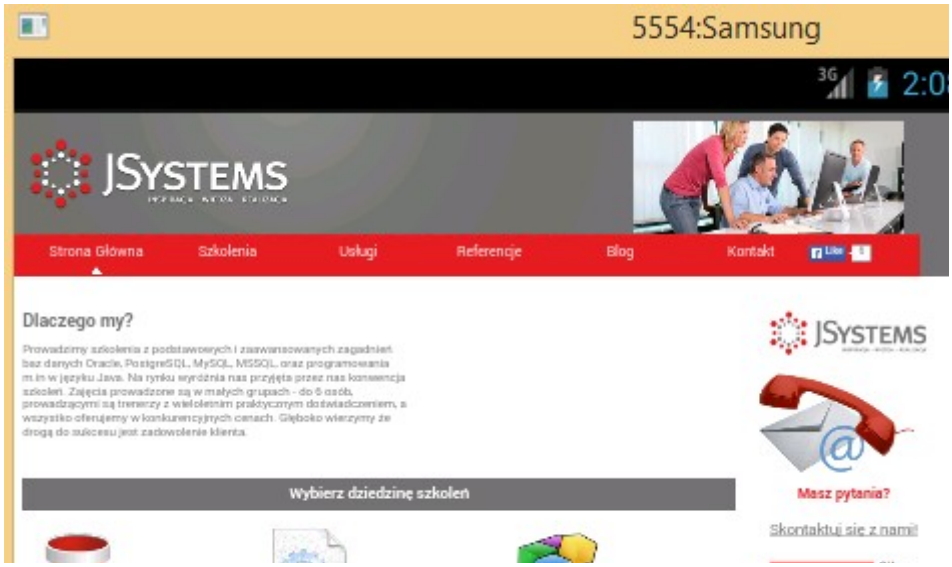
activity_main.xml | MainActivity.java

```
1  package com.example.otwarcieprzegladarki;
2
3  import android.net.Uri;
12
13 public class MainActivity extends Activity {
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19         Button b = (Button) findViewById(R.id.button1);
20         OnClickListener l = new OnClickListener() {
21             @Override
22             public void onClick(View v) {
23                 EditText t = (EditText) findViewById(R.id.editText1);
24                 Uri uri = Uri.parse("http://"+t.getText());
25                 Intent i = new Intent(Intent.ACTION_VIEW,uri);
26                 startActivity(i);
27             }
28         };
29         b.setOnClickListener(l);
30     }
31
32
33 }
```

Uruchomiłem program i wpisałem adres strony internetowej:



Po uruchomieniu:



Jeszcze jedna ciekawostka, aczkolwiek bardziej w charakterze „bajeru”. Jeśli odnajdziemy w pliku określającym layout element odpowiedzialny za nasz EditText i dopiszemy android:inputType=”textUri”, to po kliknięciu na okienko do wprowadzania tekstu wyskoczy nam klawiatura ze specjalnymi guzikami do wprowadzania adresów URL np. „www” :)

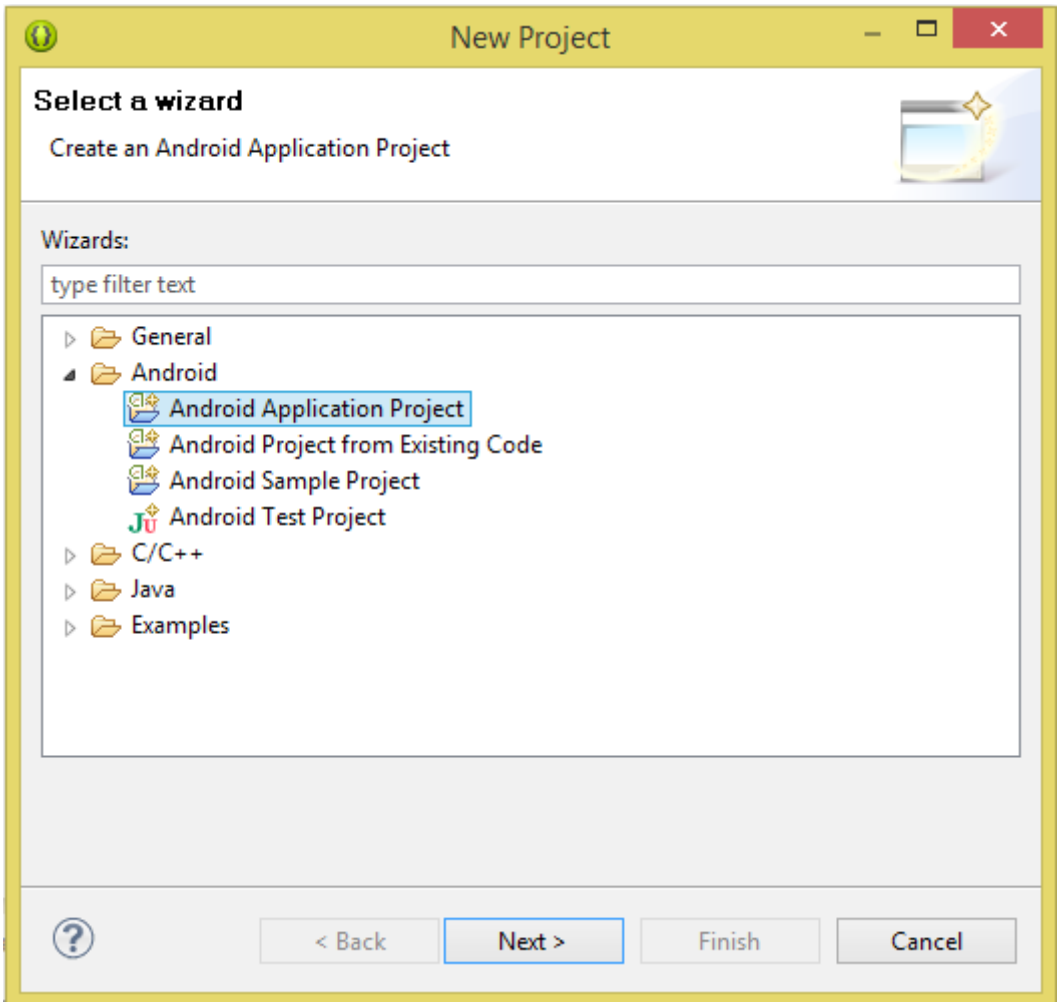
```
17
18      <EditText
19          android:id="@+id/editText1"
20          android:layout_width="wrap_content"
21          android:layout_height="wrap_content"
22          android:layout_alignLeft="@+id/button1"
23          android:layout_alignParentTop="true"
24          android:layout_marginTop="22dp"
25          android:ems="10"
26          android:inputType="textUri"
```

Tych inputType'ów jest znacznie więcej rodzajów, między innymi do wprowadzania adresów email, liczb etc .

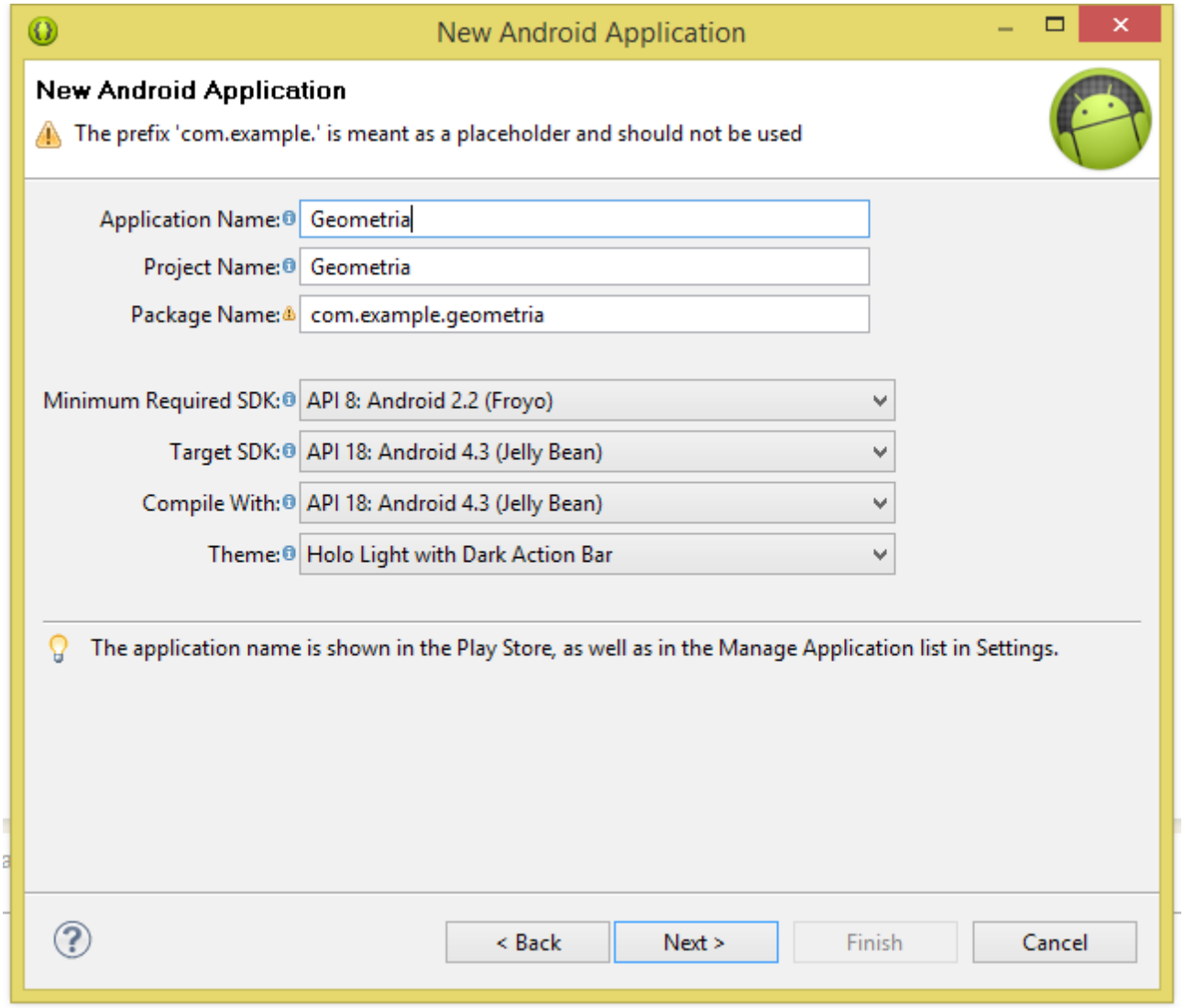
Połączmy to wszystko! Podsumowanie podstaw.

W tej lekcji zajmiemy się wywoływaniem nowych aktywności (czyli przechodzeniem pomiędzy ekranami), a także elementami Button, EditText oraz TextView. Stworzymy przy okazji prosty program do obliczania pól figur geometrycznych.

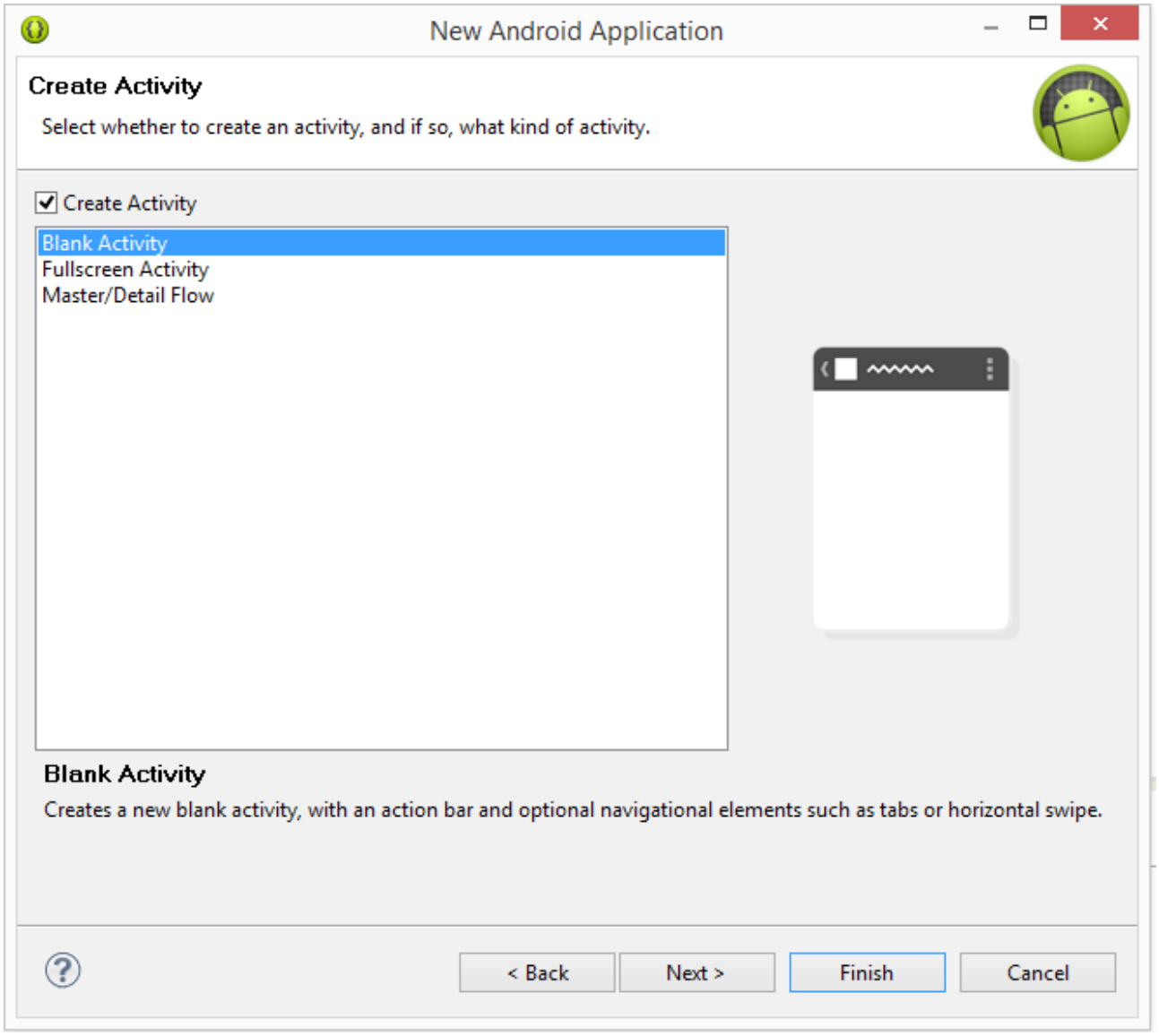
Zaczynamy od stworzenia projektu. Z menu wybieramy „Android Application Project”:



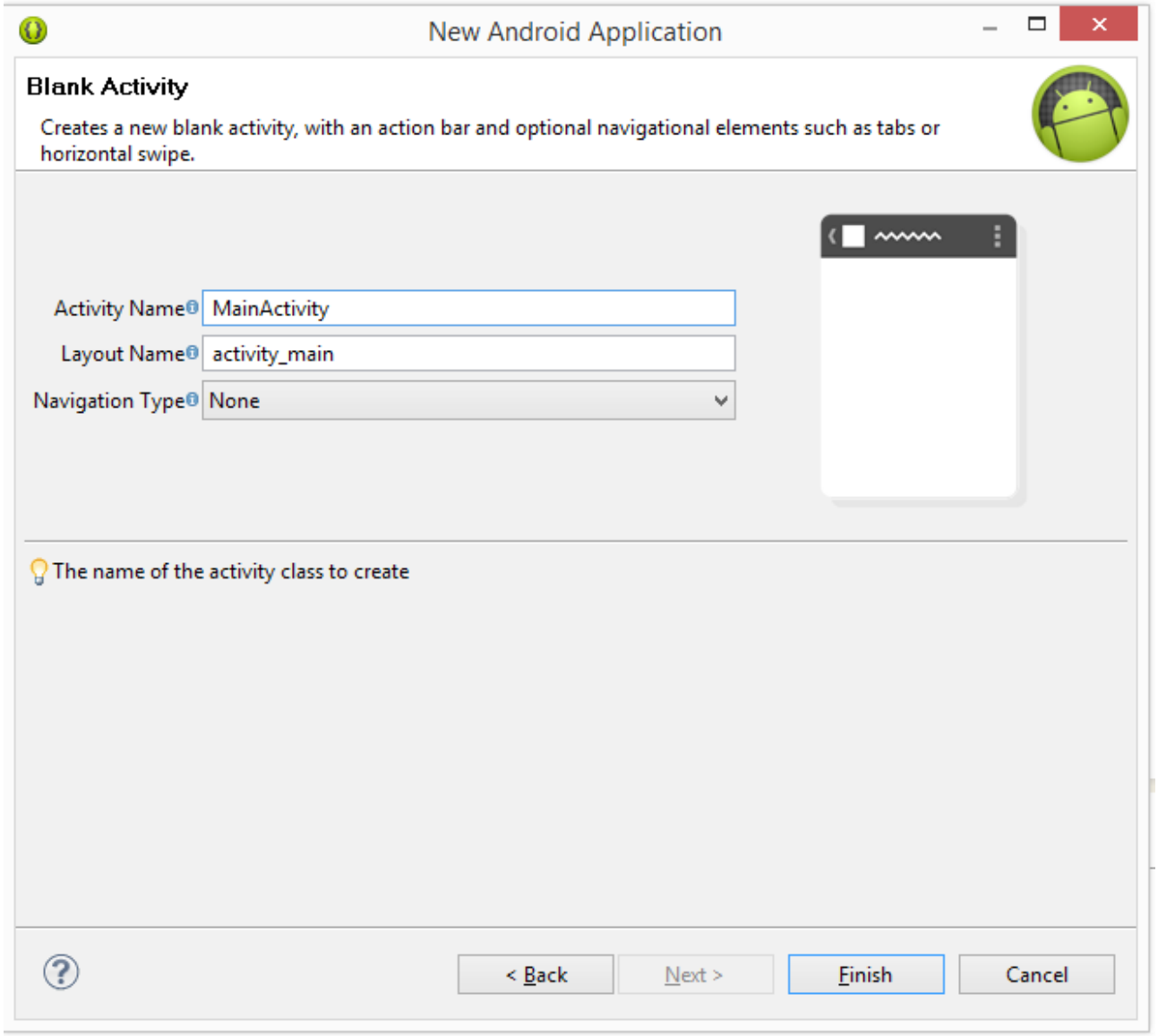
Dalej nadajemy naszej aplikacji nazwę:



Przy tworzeniu projektu, tworzymy również od razu pierwszą aktywność. Będzie to po prostu pierwszy ekran jaki się pojawi po uruchomieniu aplikacji. Decydujemy o jego pokroju - zostawiamy domyślne „Blank Activity”:



Podajemy nazwę klasy będącej pierwszą aktywnością (ekranem) w polu „Activity Name”. Do naszej klasy będzie też potrzebny plik XML w którym to określimy jak mają być rozmieszczone elementy w danej aktywności (guziki, okienka etc). Nazwę tego pliku określamy w polu „Layout Name”:



Gdy projekt zostanie utworzony, zostanie otwarta klasa głównej aktywności. Będzie zawierać dwie metody. Pierwsza „onCreate” to metoda która jest uruchamiana w momencie startu aktywności. Druga „onCreateOptionsMenu” to metoda uruchamiana kiedy ktoś wybierze przycisk menu. Tę drugą usuwamy, nie jest nam na razie potrzebna.

activity_main.xml | MainActivity.java

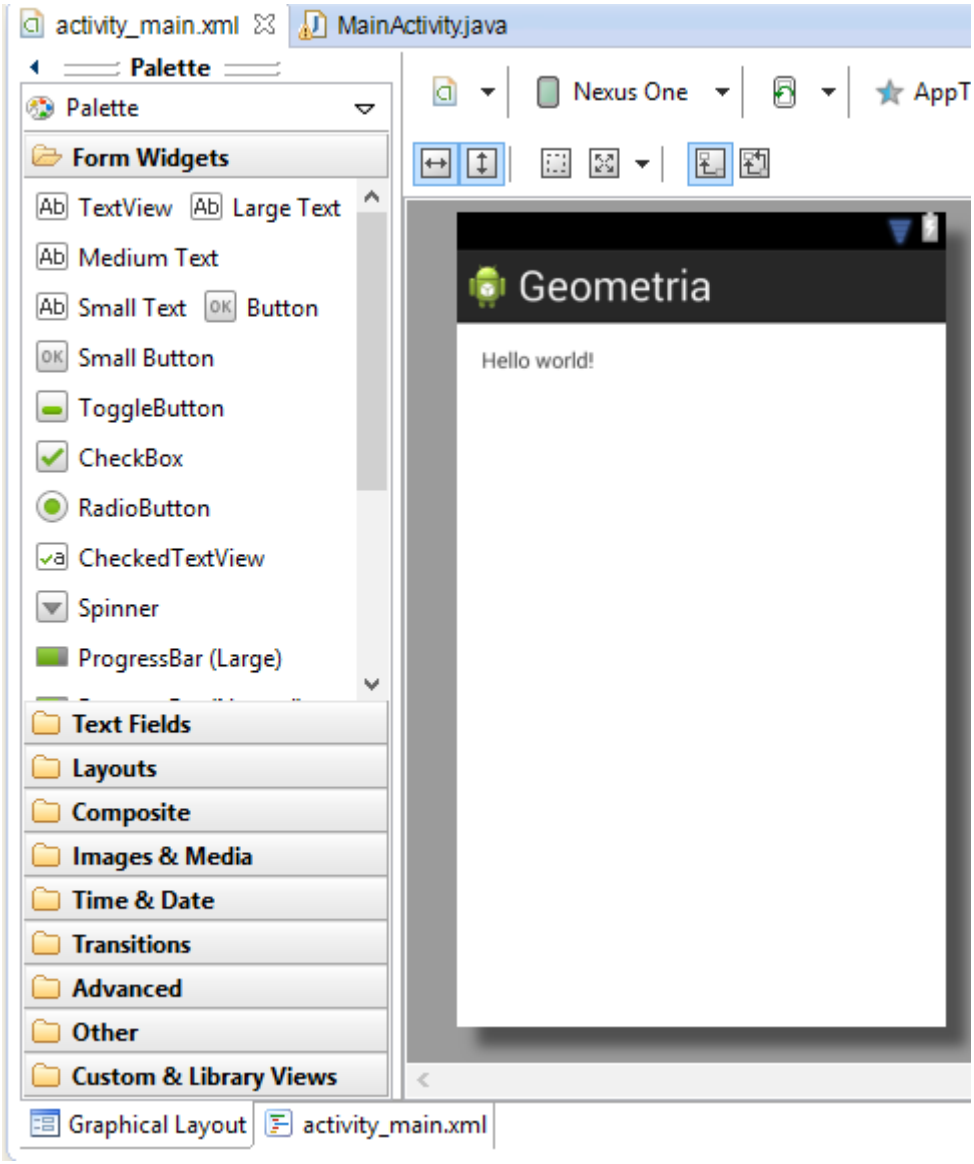
```
package com.example.geometria;

import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Nasza klasa powinna wyglądać teraz mniej więcej tak:

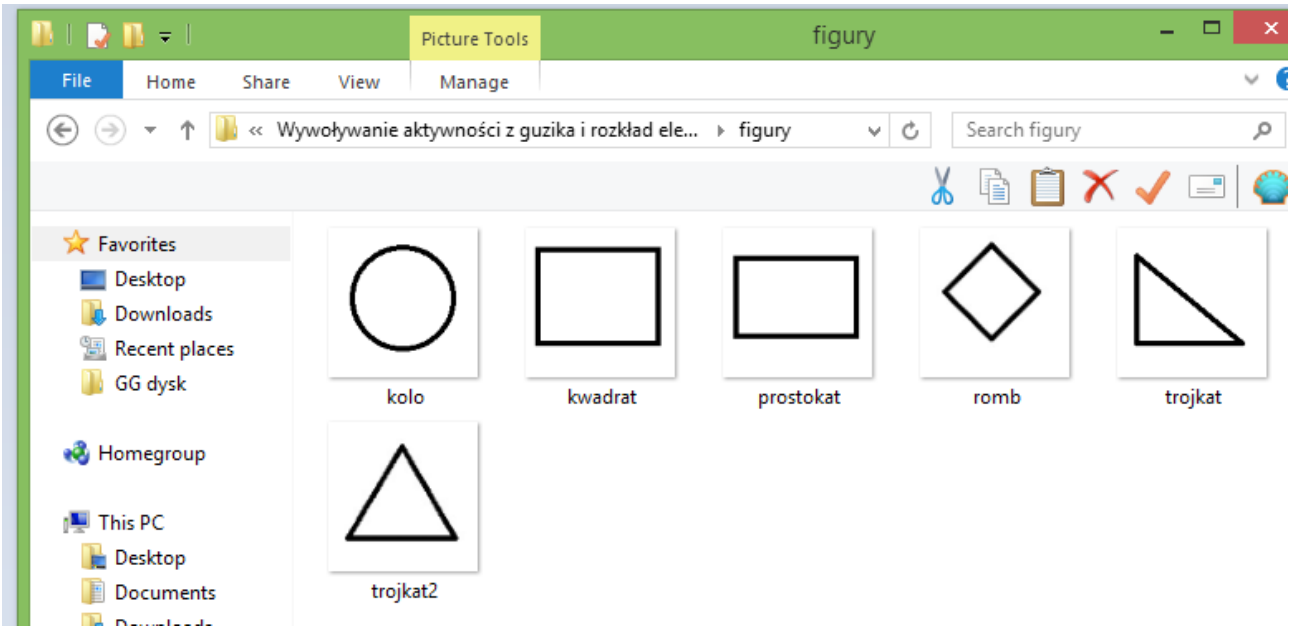
activity_main.xml | MainActivity.java

```
package com.example.geometria;

import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

}
```

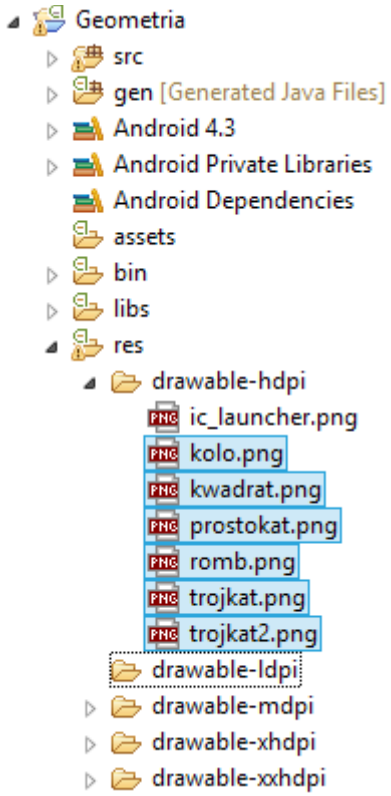
Zadbamy teraz o wygląd pierwszego ekranu. Przechodzimy do pliku layoutu. Powinniśmy zobaczyć taki ekran. Po prawej mamy naszą aktywność, po lewej paletę komponentów. Gdyby zamiast aktywności widoczny był kod XML, wybierz z zakładek na dole „Graphical Layout”:



Zawczasu przygotowałem sobie kilka obrazków które będą mi potrzebne w aplikacji (narysowałem w Paincie :p). Będziemy ich używać w naszej aplikacji. Zrobimy sobie listę figur pojawiającą się na „Dzień dobry” wraz z obrazkami. Po kliknięciu na nazwę figury program przejdzie do formularza służącego obliczaniu pola danej figury.

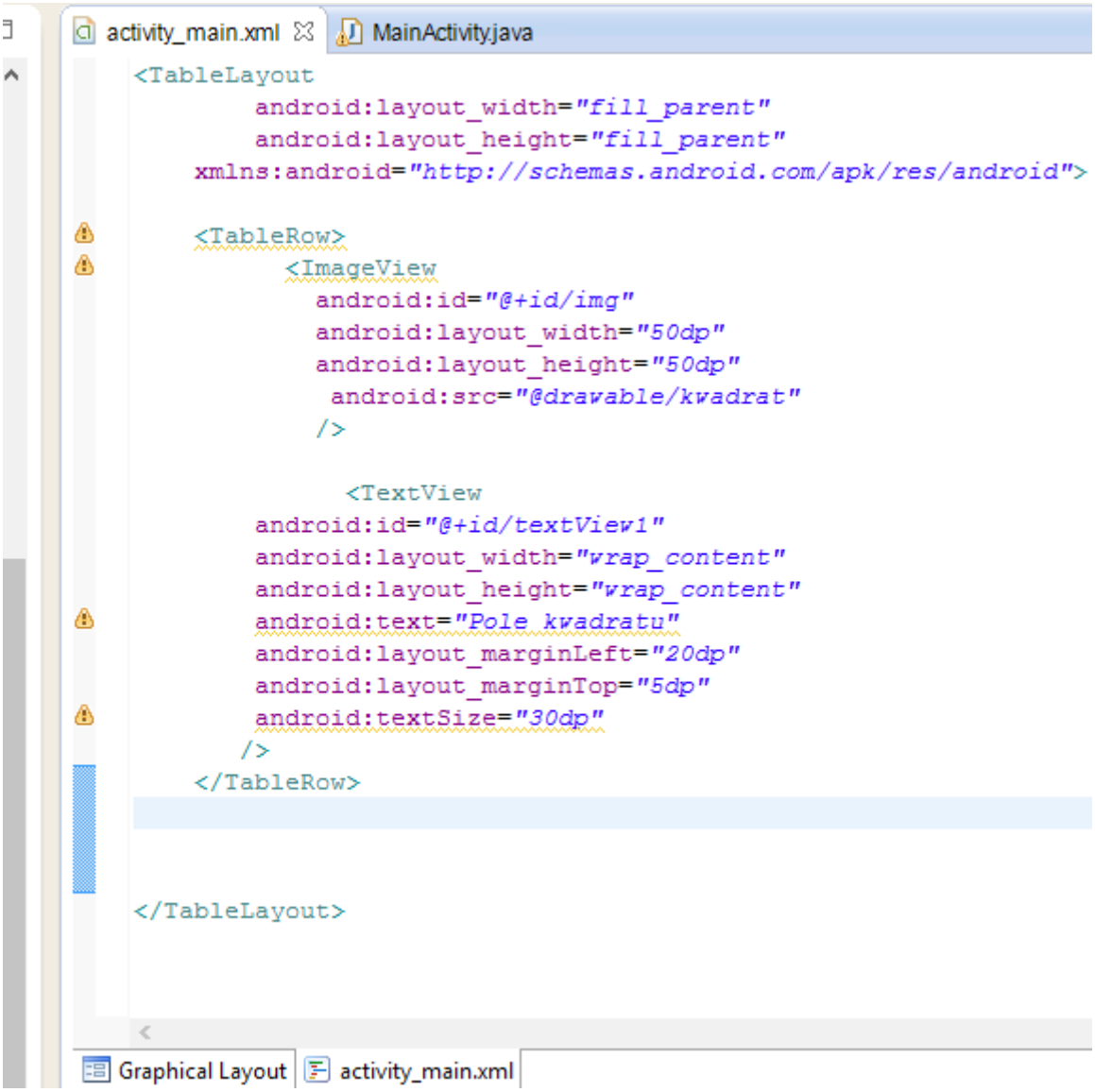


Muszę je teraz umieścić w projecie. Zaznaczam je w katalogu, robię CTRL+C, w projekcie wchodzę w podkatalog res, następnie drawable-hdpi (lub inny drawable-....) i robię CRTL+V:



Przechodzę teraz do edycji pliku XML określającego layout naszej głównej aktywności. Będziemy edytować ręcznie, więc tym razem nie wybieramy „Graphical Layout” a „activity_main.xml” (lub jak tam nazwałeś swój plik :)).

Będzie tam już trochę tekstu, ale nie jest nam potrzebny więc go w całości kasujemy. Teraz doprowadzamy zawartość pliku do takiej mniej więcej formy:

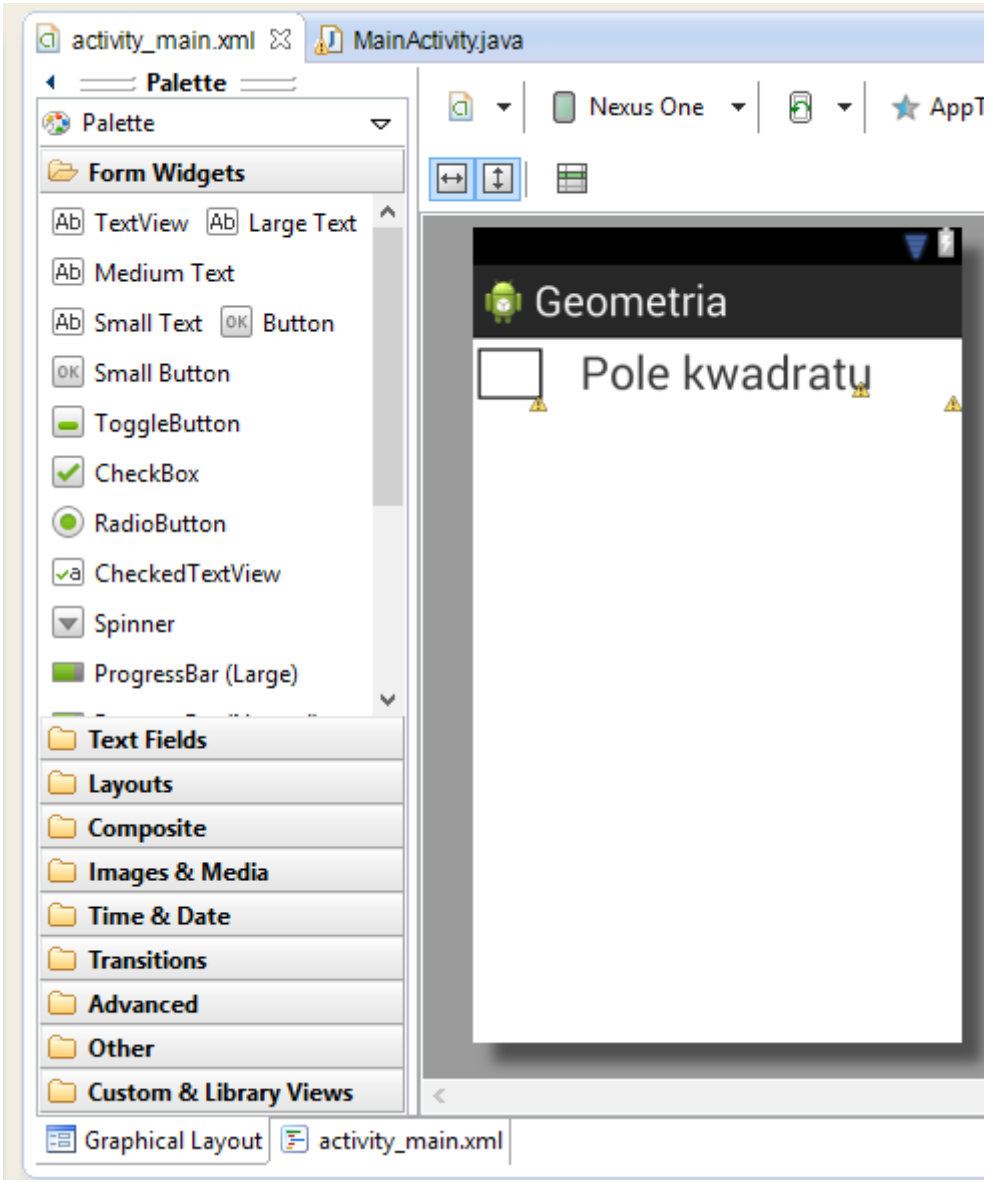


„Table Layout” to tabelaryczny układ elementów na ekranie aktywności , coś jak w HTML. Kolejne „TableRow” to po prostu kolejne wiersze w tabeli. Elementy zawarte w elemencie „TableRow” będą ze sobą sąsiadować w ramach danego poziomu w tabeli. Mamy tutaj dwa elementy: ImageView, oraz TextView. Pierwszy służy do wyświetlania obrazków, drugi do wyświetlania (ale nie wprowadzania) tekstu. W obu mamy parametr **android:id**. Nadajemy nim unikalną nazwę elementowi w ramach danej aktywności. Wartość tego parametru musi się zaczynać od „@+id/” po których następuje nasz identyfikator. Ten identyfikator jest niezbędny do rozróżniania elementów i jednoznacznego wskazywania jednego z nich. W obu też występują parametry **layout_width** i **layout_height**. Określają one szerokość i wysokość elementu. Jednostka DP to piksele niezależne od rozdzielczości ekranu.

W elemencie ImageView jest jeszcze parametr **android:src**. Poprzez niego podajemy jaki obrazek ma zostać wyświetlony. Jeśli chodzi o obrazek zawarty w projecie (a nie znajdujący się np. na zdalnym serwerze czy karcie pamięci) to zaczynamy wartość od „@drawable/” po którym następuje nazwa obrazka wrzuconego w jednym z poprzednich kroków do katalogu drawable-hdpi. Nie podajemy rozszerzenia (tutaj .png) w nazwie pliku.

W elemencie TextView mamy dodatkowo element **android:text** przy użyciu którego ustawiamy tekst jaki ma być wyświetlany na elemencie. Związany z nim jest jeszcze element **android:textSize**, który określa wielkość czcionki. Użyłem tutaj też parametrów **android:layout_marginLeft**, oraz **android:layout_marginTop**. Określają one margines lewy , oraz górny.

Wracamy do widoku projektowania graficznego. Nasza aplikacja powinna w tej chwili wyglądać mniej więcej tak:

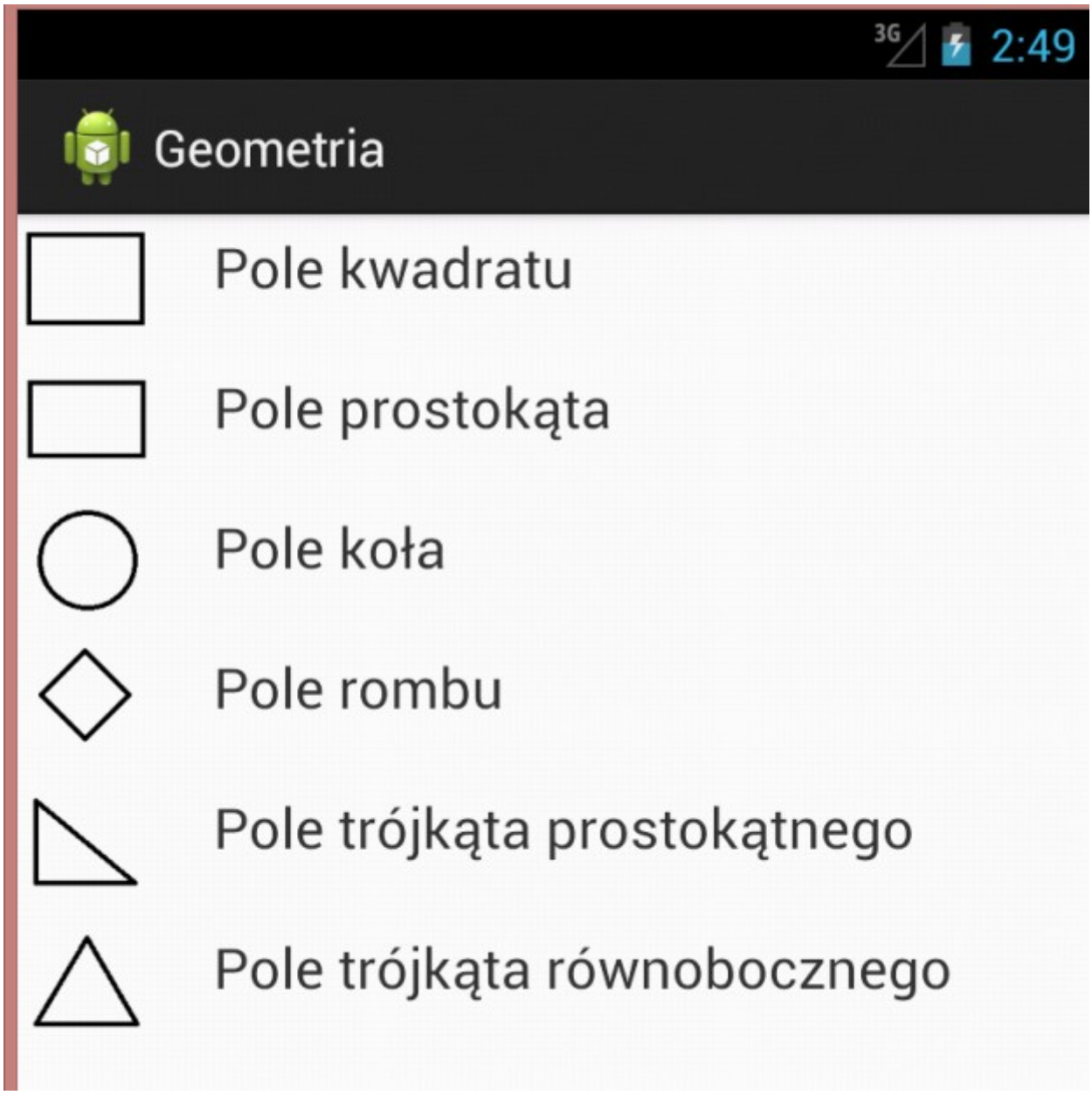


Dodałem analogicznie kolejne wiersze tabeli z kolejnymi elementami. Zmieniłem też wielkość czcionki na 20dp ponieważ niektóre opisy nie mieściły się w jednej linii:

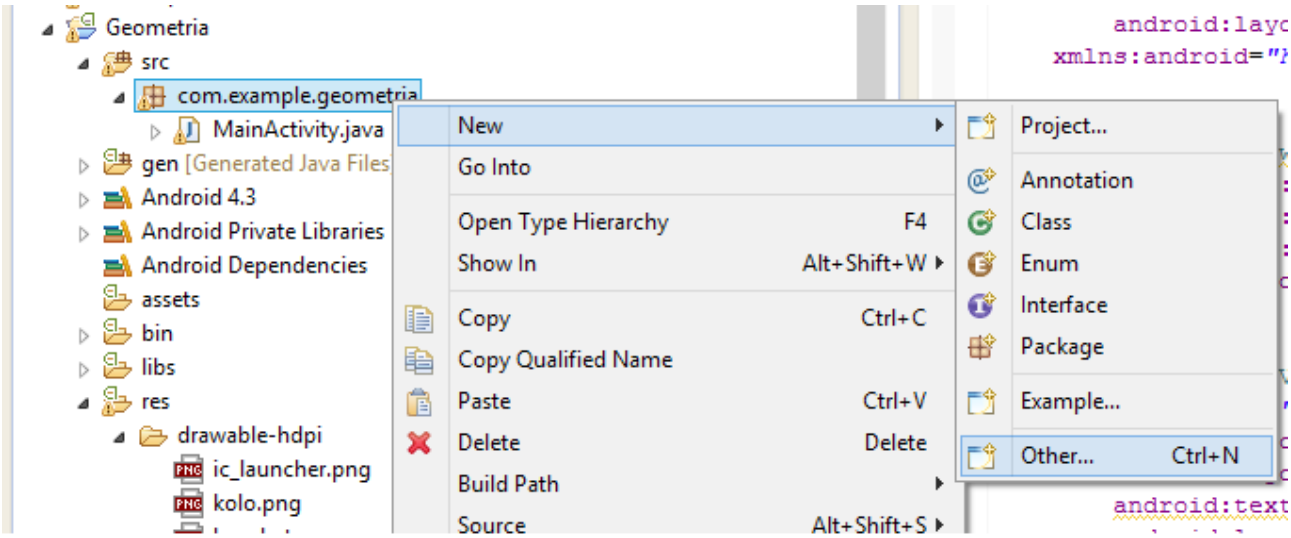
activity_main.xml | MainActivity.java

```
<TableLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <TableRow>
          <ImageView
            android:id="@+id/img"
            android:layout_width="50dp"
            android:layout_height="50dp"
             android:src="@drawable/kwadrat"
            />

              <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Pole kwadratu"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="5dp"
        android:textSize="20dp"
       />
    </TableRow>

     <TableRow>
          <ImageView
            android:id="@+id/img2"
            android:layout_width="50dp"
            android:layout_height="50dp"
             android:src="@drawable/prostokat"
            />

              <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Pole prostokata"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="5dp"
        android:textSize="20dp"
       />
    </TableRow>
```

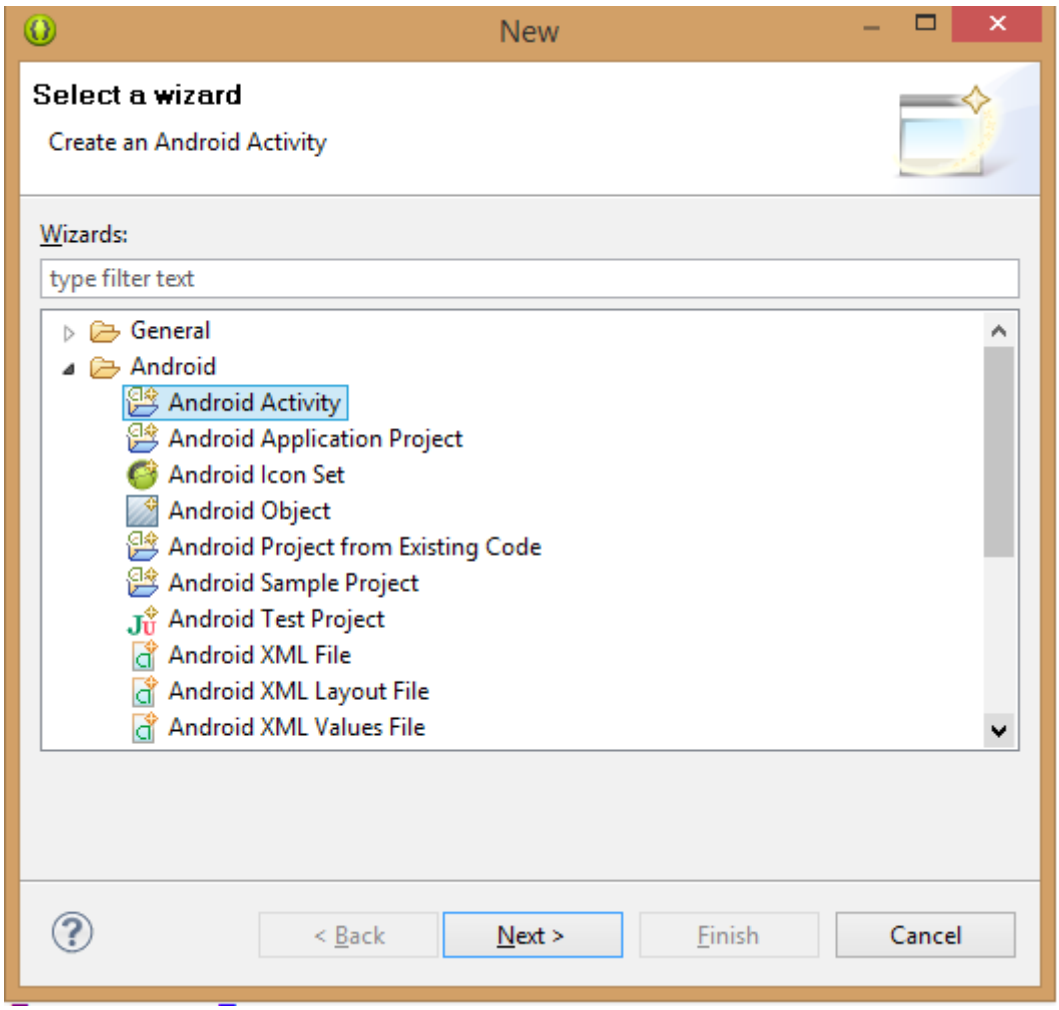
Po tym zabiegu program wygląda tak:



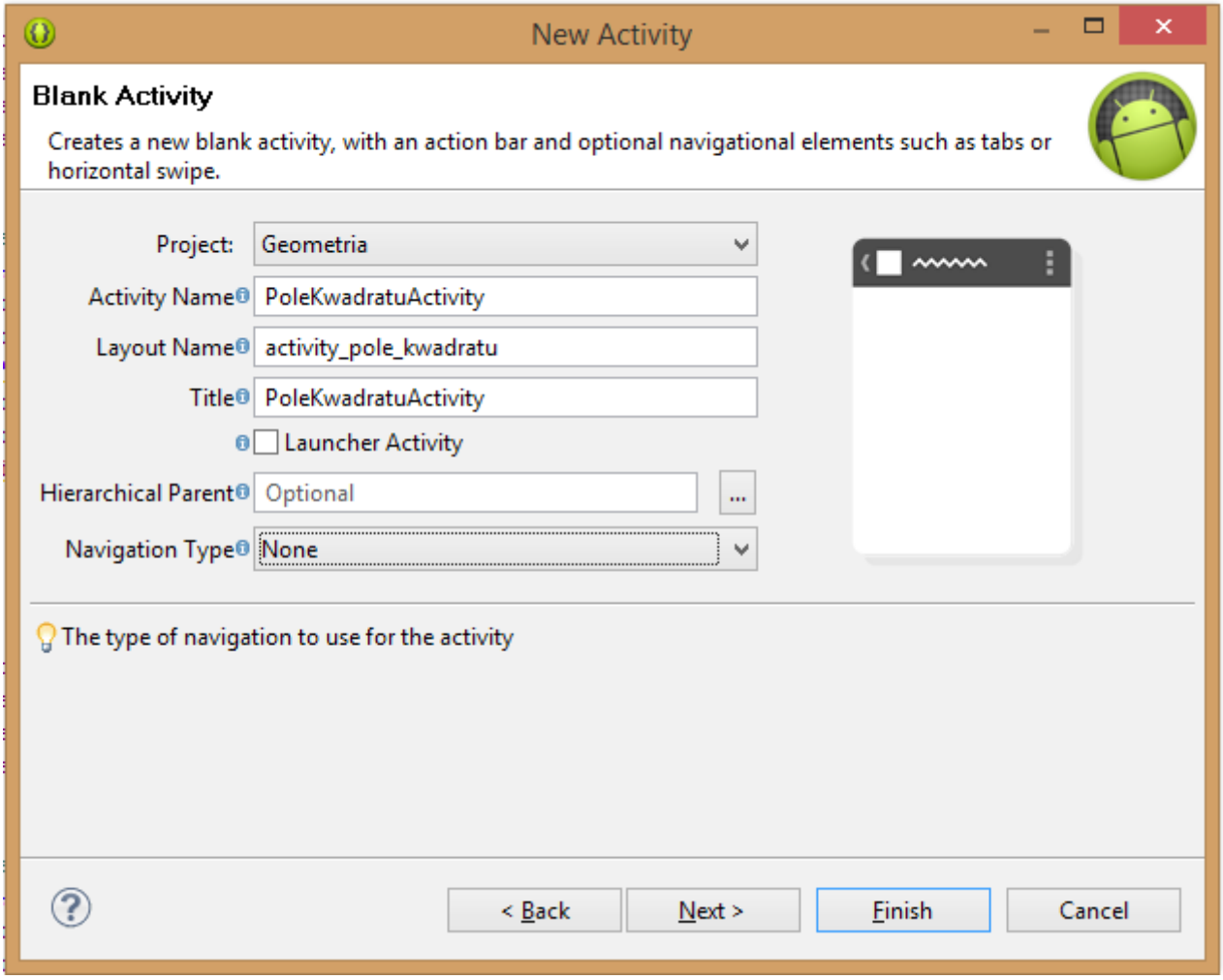
Przyszła pora na dodanie nowej aktywności – tj. ekranu na który aplikacja powinna przejść po kliknięciu na napis „pole kwadratu”. Klikam prawym przyciskiem myszy na pakiecie w którym znajduje się moja pierwsza aktywność i wybieram NEW--> OTHER:



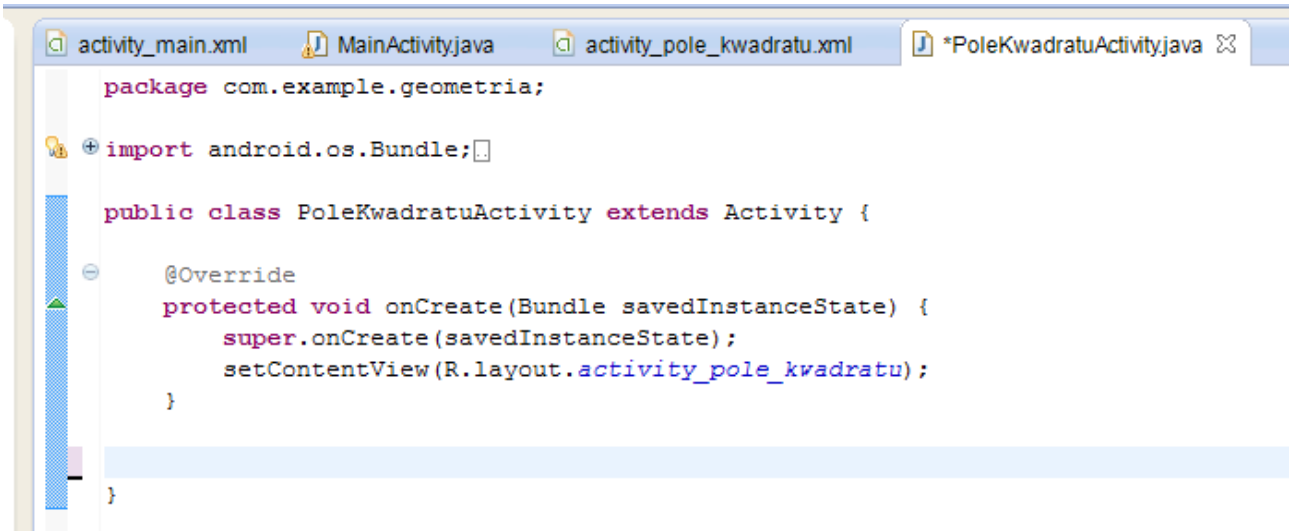
Dalej „Android Activity”:



Podobnie jak i wcześniej nadaje klasie aktywności i plikowi layoutu nazwy:

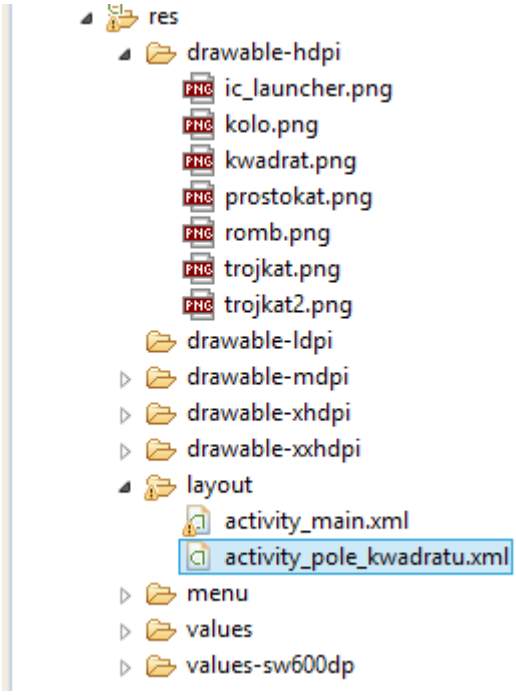


Klasa nowej aktywności:



Zauważ że w metodzie „onCreate” jest wywołanie metody setContentView, która służy do wiązania

aktywności z opisem wyglądu zawartym w pliku XML. Każda aktywność domyślnie będzie miała swój plik xml i po stworzeniu będzie wywołanie jej własnego pliku, ale możesz do zmienić wskazując w parametrze inny layout. Tutaj mamy R.layout.activity_pole_kwadratu, ponieważ taki właśnie plik został utworzony podczas generowania nowej aktywności:

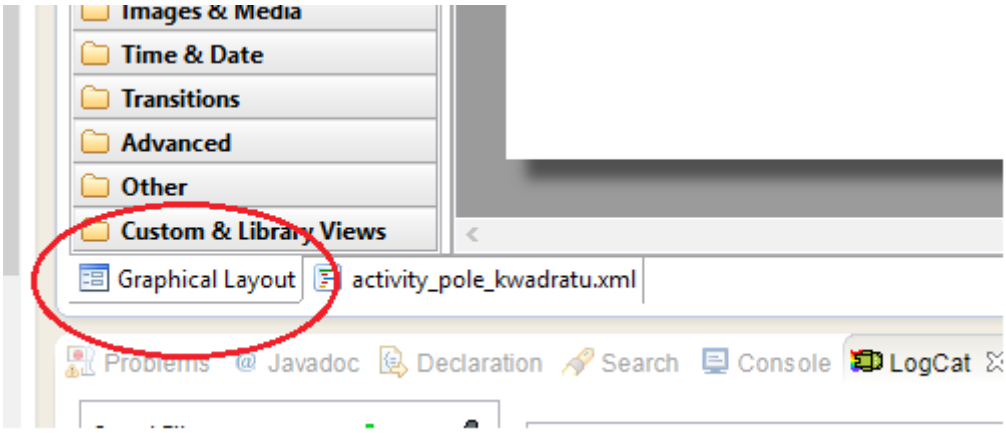


Domyślnie zawartość tego pliku wygląda tak:

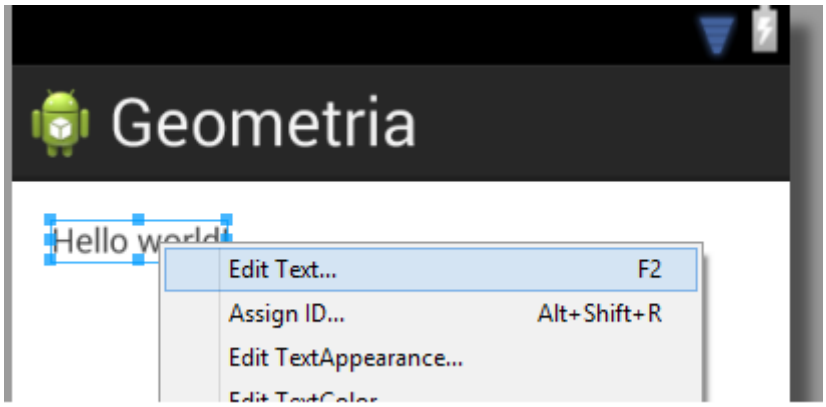
activity_main.xml | MainActivity.java | activity_pole_kwadratu.xml | PoleKwadratuActivity.java

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".PoleKwadratuActivity" >

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

</RelativeLayout>
```

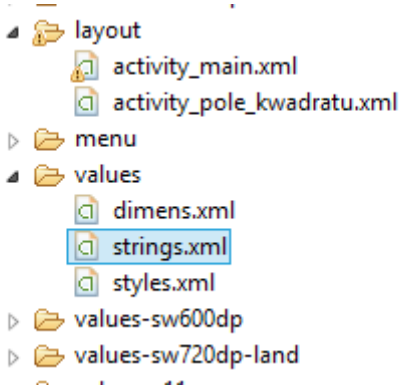
Na ten moment ta aktywność będzie po prostu wyświetlała napis „Hello World!”. My w tej aktywności dodamy parę elementów, będzie to formularz do obliczania pola kwadratu. Przechodzimy do graficznej formy edycji layoutu:



Napis który już tam się znajduje nieco zmodyfikujemy. Klikamy nań prawym przyciskiem myszy i wybieramy „Edit Text”:



Wcześniej tekst wpisywaliśmy niejako „bezpośrednio”, teraz przyjmiemy nieco inną konwencję. W projekcie obecny jest plik strings.xml:



Znajdują się w nim identyfikatory, oraz przypisane do nich teksty. Chcąc wyświetlić jakiś tekst na elemencie, będziemy się odwoływać poprzez identyfikator do tekstu zawartego w tym pliku.

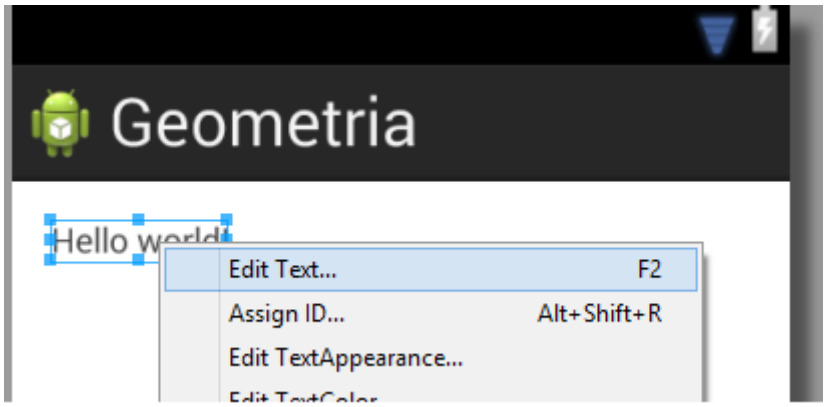
tivity_main.xml | MainActivity.java | *activity_pole_kwadratu.xml | PoleKwadratuActivity.java

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Geometria</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="title_activity_pole_kwadratu">PoleKwadratuActivity</string>
    <string name="pole_kwadratu">Pole kwadratu</string>

</resources>
```

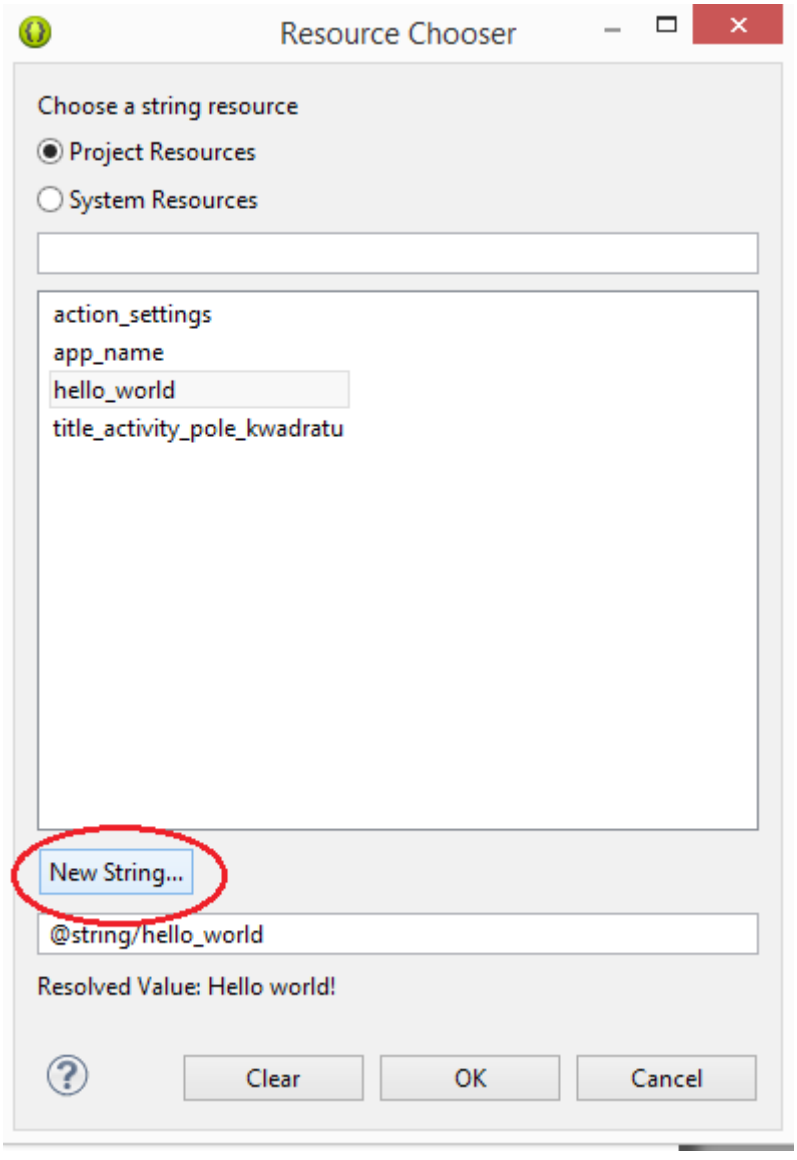
Po co nam to wszystko? Dzięki temu mamy wszystkie informacje tektstowe w jednym miejscu, dużo łatwiej jest je dzięki temu zmieniać. Ponadto znacznie łatwiej będzie później robić różne wersje językowe naszego programu, bo wystarczy potem tylko podmienić ten jeden plik.

\

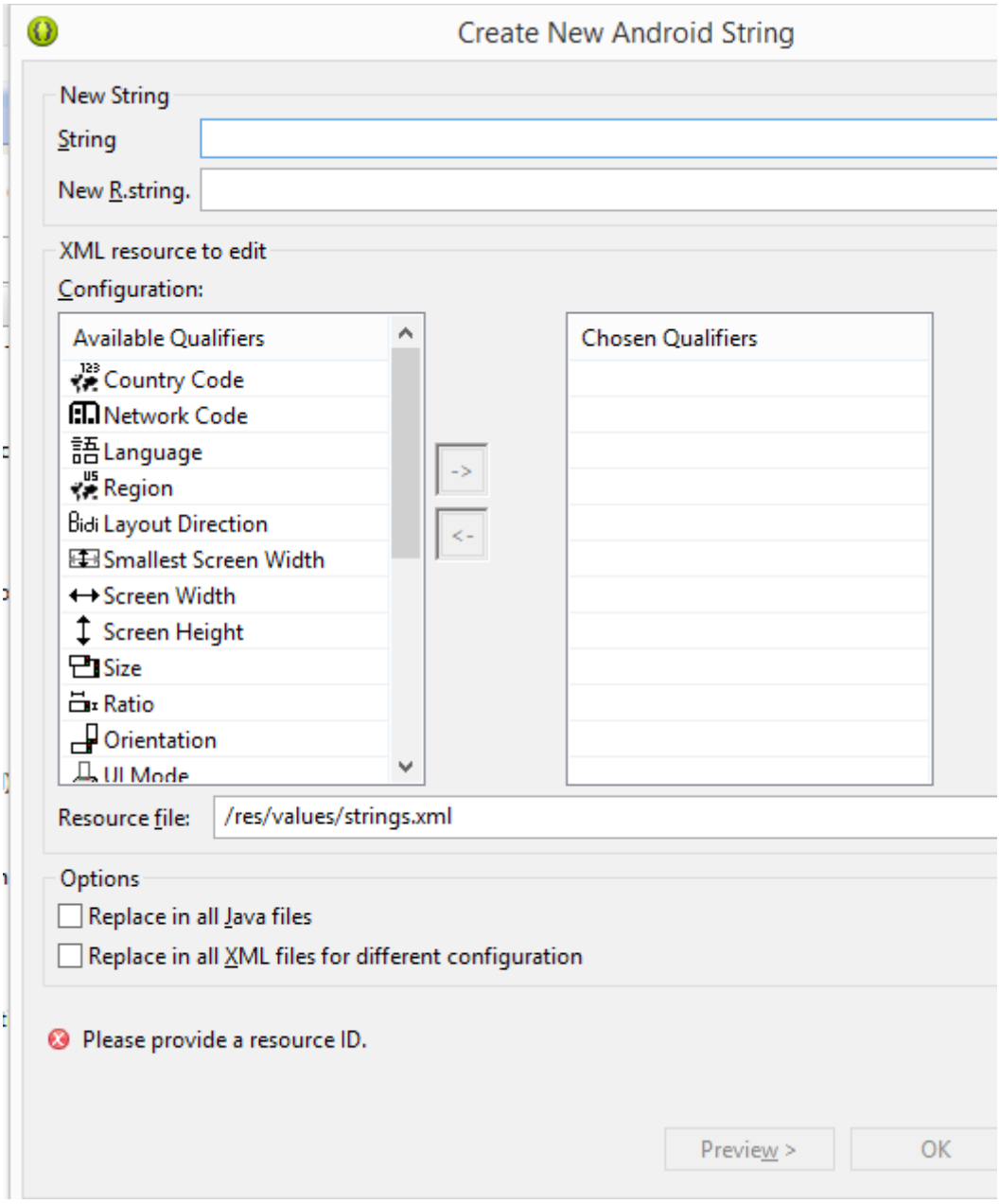
Ustawimy teraz napis na TextView, jednocześnie dodając go do pliku strings.xml:



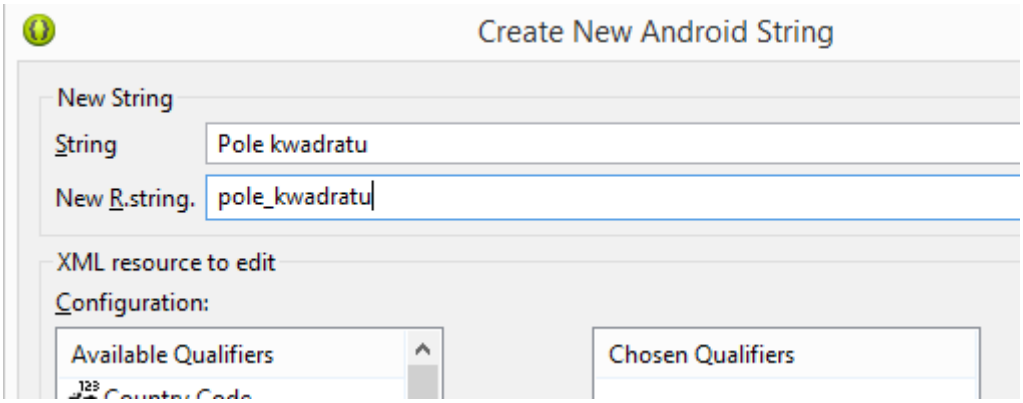
Wybieramy edycję i przechodzimy do takiego ekranu:



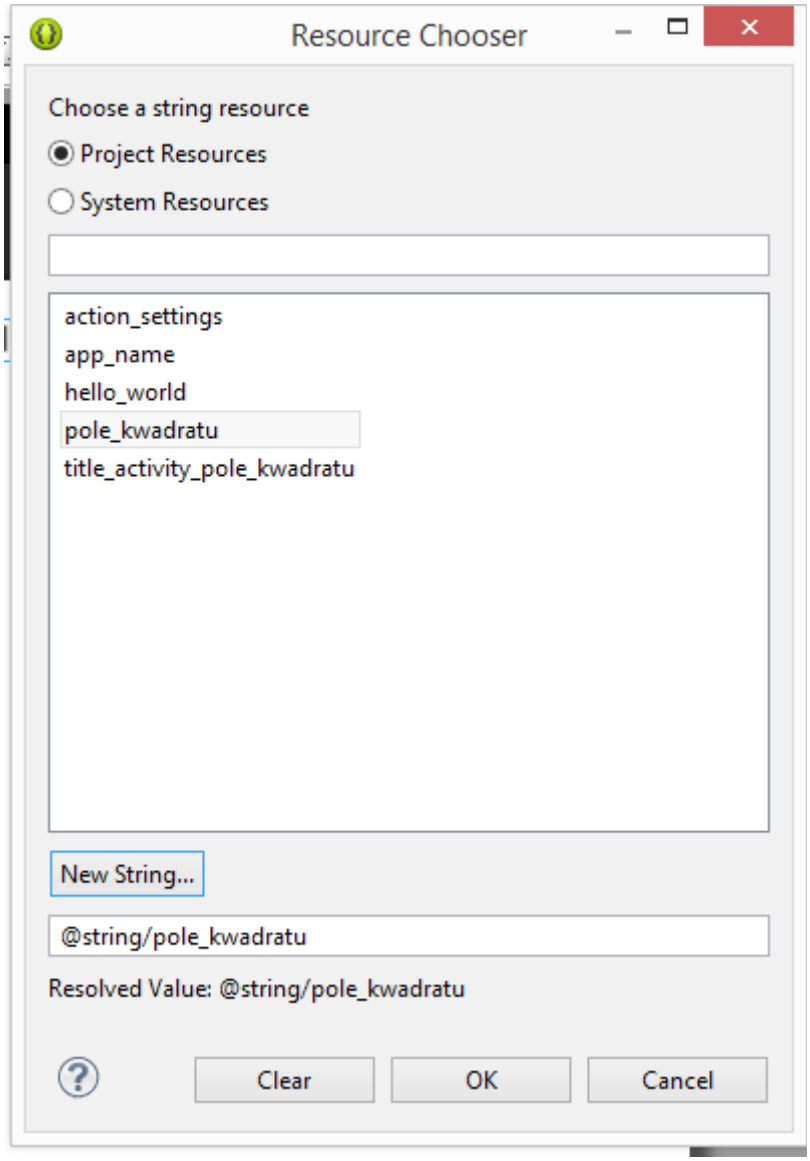
Klikamy przycisk „New String”. Powinniśmy teraz zobaczyć taki ekran:



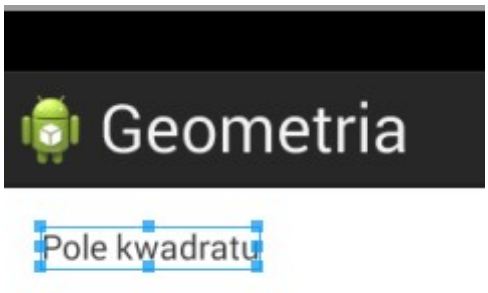
W polu „String” wpisujemy tekst który ma zostać dodany do pliku strings.xml. Oraz wyświetlony na komponencie. W polu „New R.string” podajemy identyfikator tekstu, pod który zostanie wpisany tekst do pliku.



Zatwierdzamy. Zobaczymy taki ekran:



Pojawi się nowy wpis w pliku strings.xml, a napis na komponencie się zmieni:

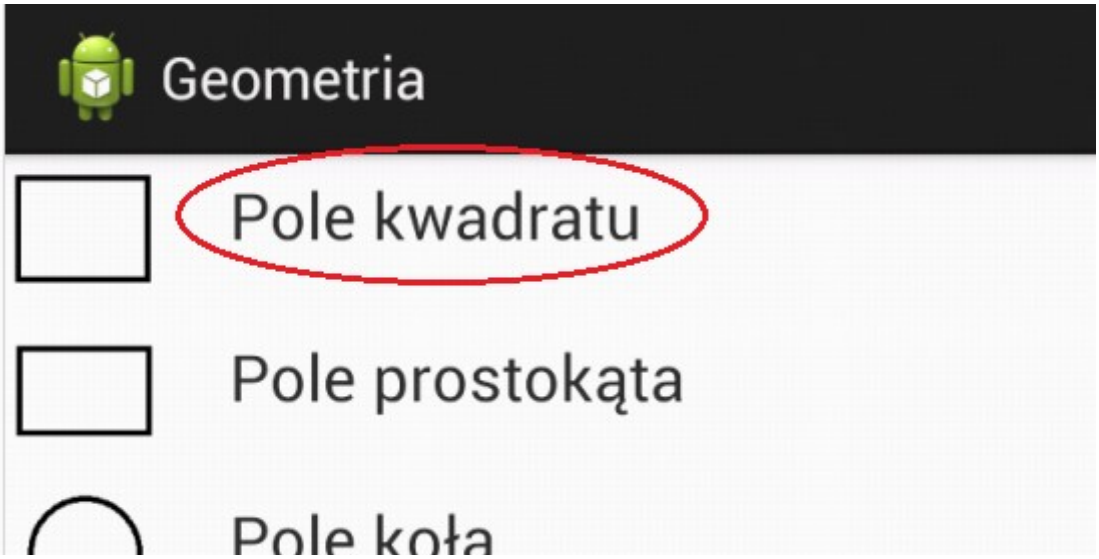


Dobrze. Mamy teraz ekran początkowy z listą elementów do wyboru, oraz zalążek nowego ekranu który docelowo będzie służył obliczaniu pola kwadratu. Trzeba teraz te elementy połączyć. Po kliknięciu napisu „pole kwadratu” na naszej aplikacji powinna pojawić się aktywność do obliczania pola kwadratu. Przyszedł czas na dodanie reakcji na kliknięcie napisu.

activity_main.xml | MainActivity.java | activity_pole_kwadratu.xml | PoleK

```
1  package com.example.geometria;
2
3  import android.os.Bundle;
9
10 public class MainActivity extends Activity {
11
12     TextView t1;
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18
19         t1 = (TextView) findViewById(R.id.textView1);
20         OnClickListener l = new OnClickListener() {
21
22             @Override
23             public void onClick(View v) {
24                 t1.setText("AŁA!!!");
25             }
26         };
27         t1.setOnClickListener(l);
28     }
29
30
31 }
32
```

Zmieniłem nieco zawartość klasy mojej głównej aktywności. Dodałem zmienną t1 która będzie reprezentowała komponent TextView z napisem „Pole kwadratu”. W linii 19 do tej zmiennej przypinam realny komponent. Robię to przy użyciu wbudowanej metody findViewById. Ma ją każda klasa dziedzicząca po Activity. Ta metoda służy do zwracania referencji komponentu z aktywności. Do zmiennej t1 zostaje przypisany element o identyfikatorze textView1 (czyli nasz napis „pole kwadratu” z ekranu głównego:



Obsługę zdarzenia kliknięcia realizujemy poprzez podstawienie listenera dla obiektu t1. Listener to taki proces nasłuchu który czeka na jakieś zdarzenie. Tworzymy nasz listener jako obiekt klasy OnClickListener i jednocześnie defuniujemy dla niego reakcję na kliknięcie (metoda onClick). Tymczasowo reakcja na kliknięcie komponentu ma polegać na zmianie napisu „Pole kwadratu na napis „AŁA!!!!”. Stworzony listener ustawiamy dla obiektu t1 poprzez metodę tego obiektu setOnClickListener.

Uruchamiam projekt i sprawdzam działanie. Reakcja na kliknięcie jest taka jaka być powinna. Nasz program jest nadwrażliwą beksą :)

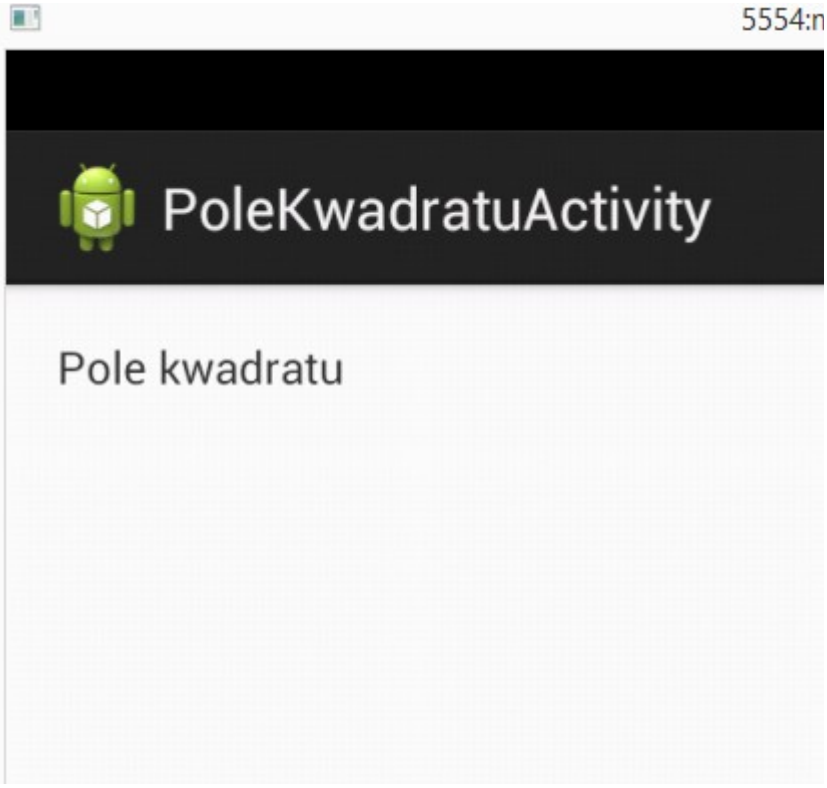
Skoro już działa obsługa kliknięcia, to teraz jako reakcję wepniemy uruchomienie innej aktywności w miejsce zmiany tekstu.

MainActivity.java | Geometria Manifest | MainActivity.java

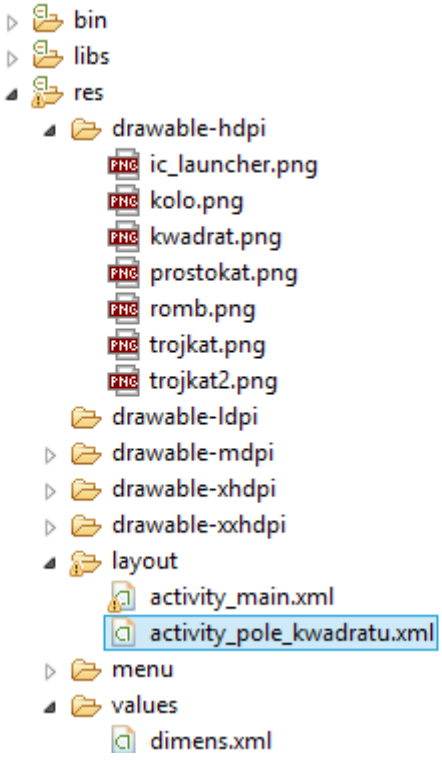
```
1  package com.example.geometria;
2
3  import android.os.Bundle;
4  import android.app.Activity;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.view.Menu;
8  import android.view.View;
9  import android.view.View.OnClickListener;
10 import android.widget.TextView;
11
12 public class MainActivity extends Activity {
13
14     TextView t1;
15     Context context;
16
17     @Override
18     protected void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.activity_main);
21
22         t1 = (TextView) findViewById(R.id.textView1);
23         OnClickListener l = new OnClickListener() {
24
25             @Override
26             public void onClick(View v) {
27                 //t1.setText("AŁA!!!");
28                 context = getApplicationContext();
29                 Intent intent = new Intent(context, PoleKwadratuActivity.class);
30                 startActivity(intent);
31             }
32         };
33         t1.setOnClickListener(l);
34     }
35
36
37 }
```

Wykomentowałem zmianę tekstu. Dodałem obiekt klasy Context do którego w linii 28 przypisuję kontekst aplikacji. W linii 29 tworzę nową intencję. Intencje to komunikaty łączące ze sobą różne komponenty.W tym przypadku nasza intencja będzie służyła wywołaniu innej aktywności. W tej samej linii podaję też informację, aktywność jakiej klasy ma zostać wywołana. W linii 30 odpalam tę aktywność.

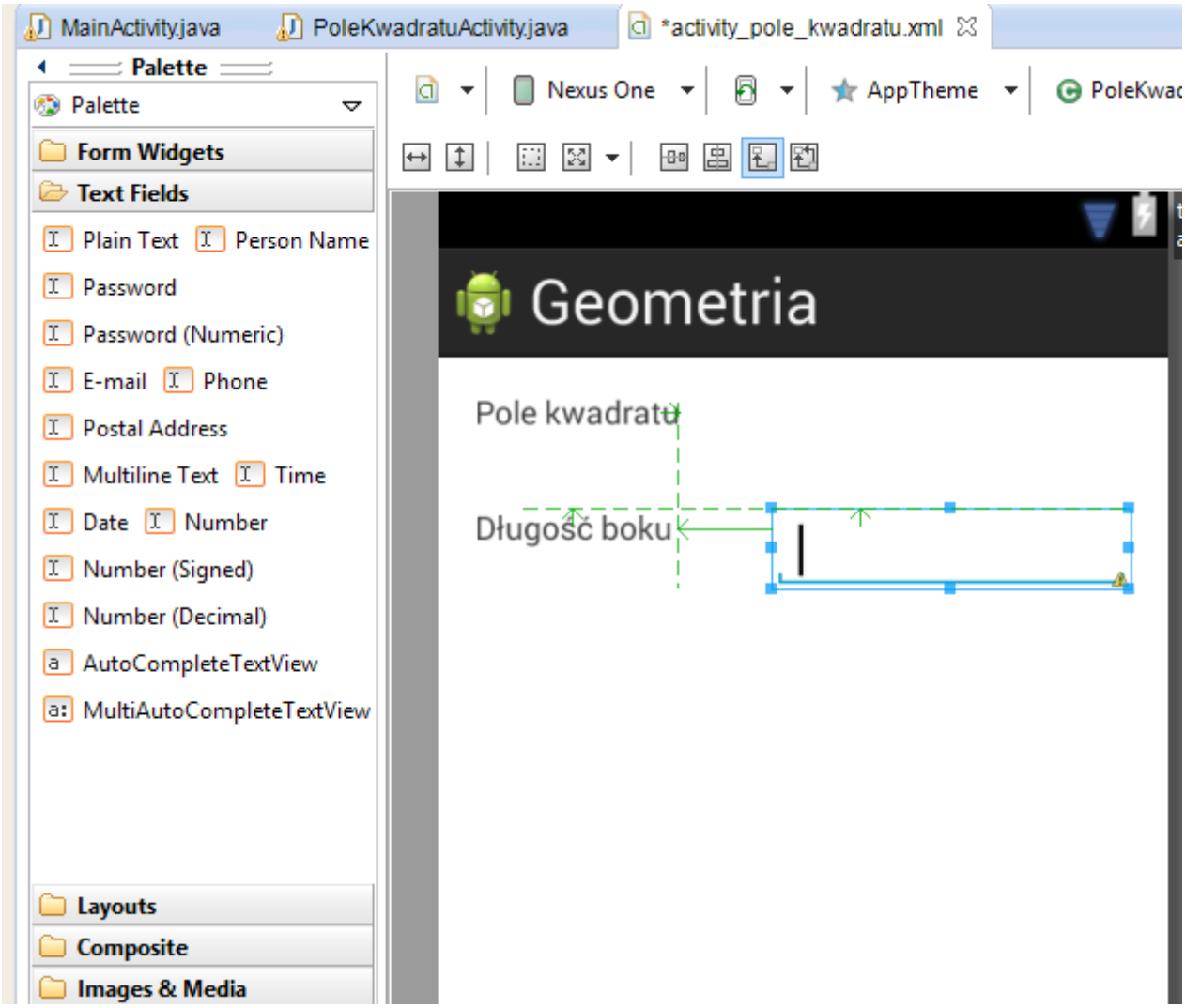
Sprawdzam działanie całości. Tym razem po kliknięciu napisu „Pole kwadratu” nie zmienia się tekst na komponencie, a zostaje wyświetlony ten ekran:

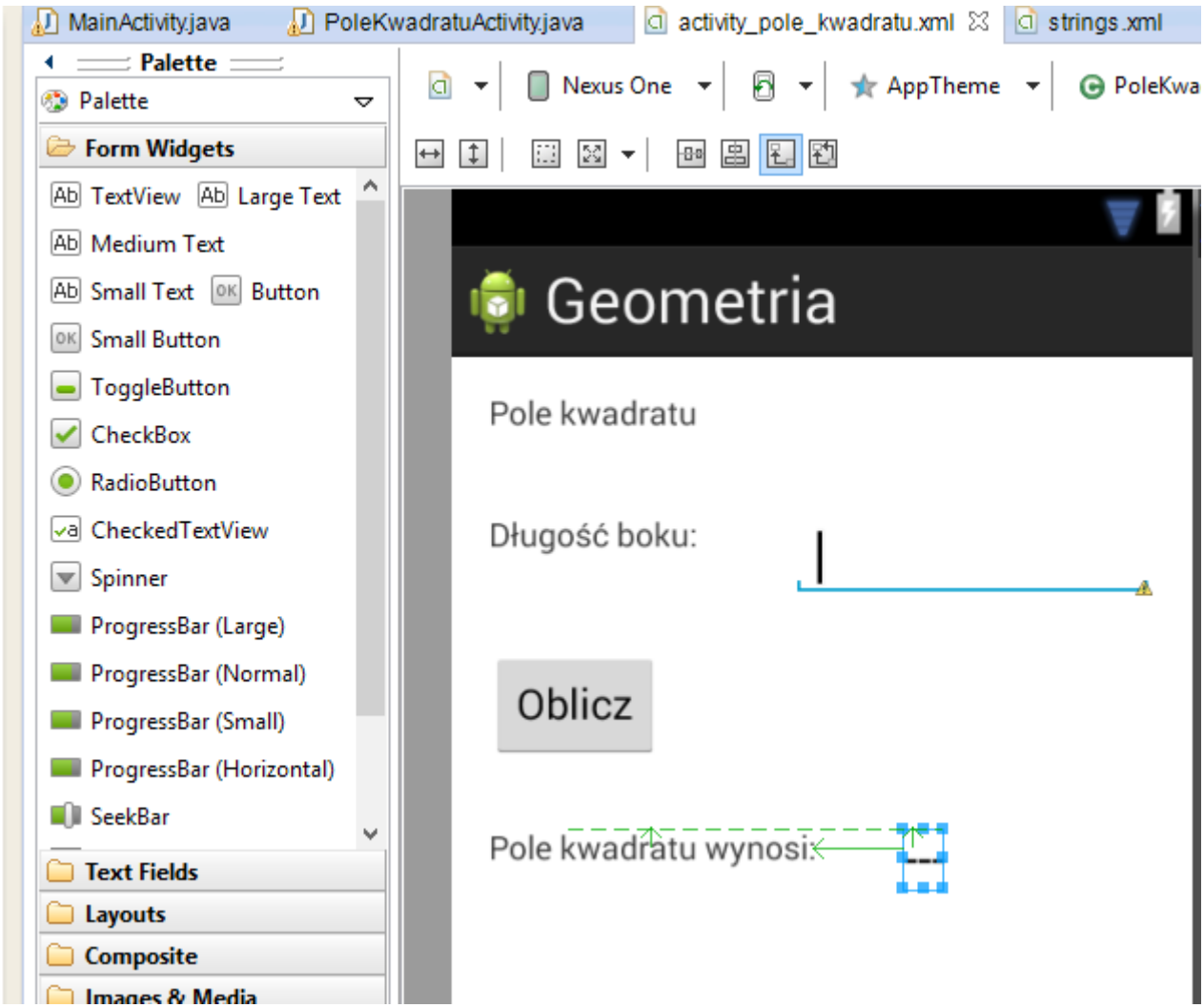


Skoro wywołania działają, trzeba przerobić tę aktywność tak, aby faktycznie służyła obliczaniu pola kwadratu. Przechodzę więc do pliku layoutu tej aktywności:



by następnie w trybie projektowania graficznego poprzyklejać do niej niezbędne komponenty.





Pojawiły się tu nowe rzeczy. Komponent klasy Button, komponent klasy LargeText, oraz komponent klasy EditText. Button to po prostu przycisk. LargeText to element wyświetlający tekst, ale o powiększonej czcionce. EditText to komponent w którym możemy wyświetlić jakiś tekst, ale użytkownik programu będzie mógł go zmienić.

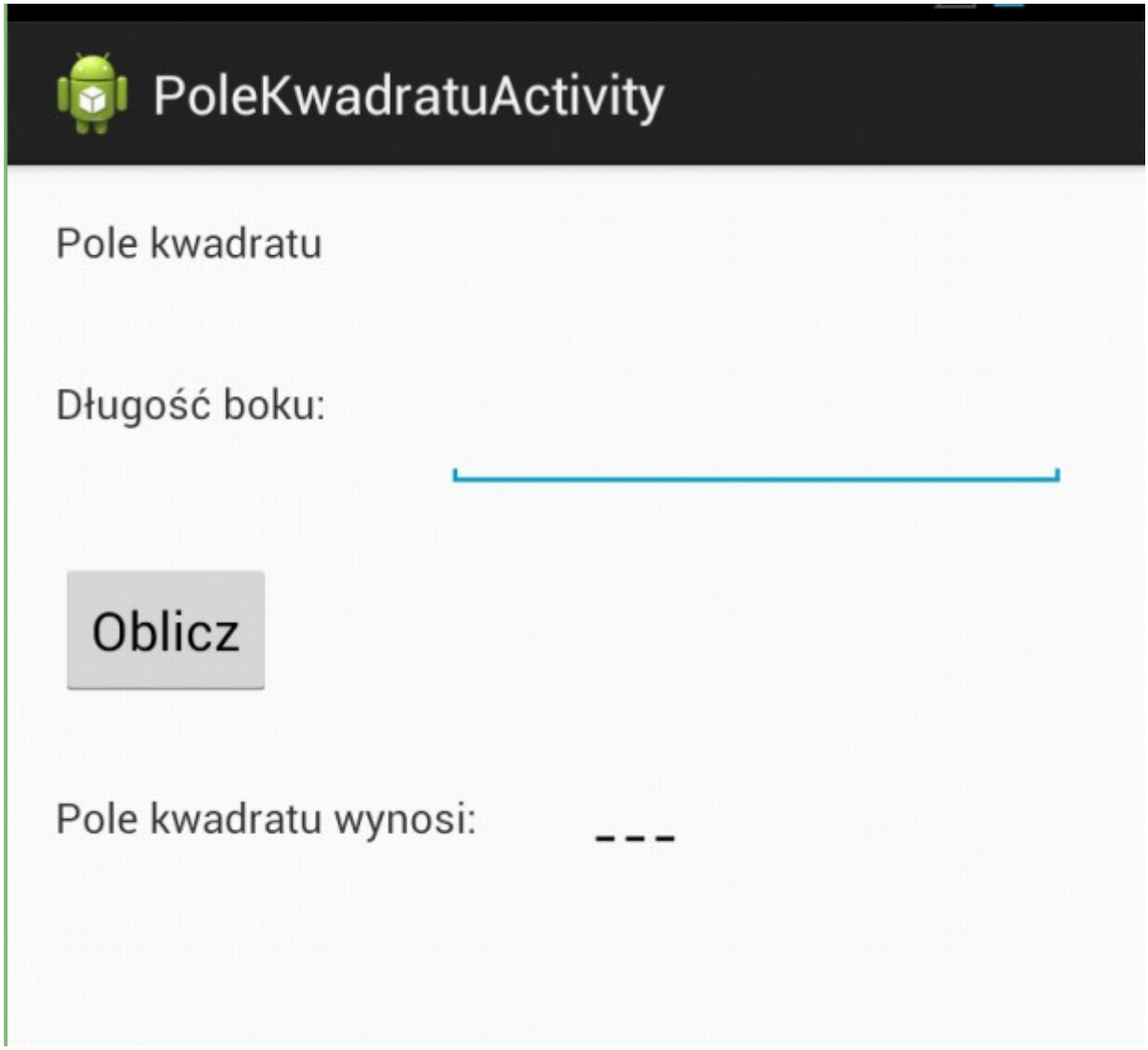
Po rozmieszczeniu elementów przechodzimy do edycji klasy aktywności :

MainActivity.java | PoleKwadratuActivity.java | activity_pole_kwadratu.xml | strings.xml

```
1  package com.example.geometria;
2
3  import android.os.Bundle;
11
12 public class PoleKwadratuActivity extends Activity {
13
14     Button guzik;
15     EditText bok;
16     TextView wynik;
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_pole_kwadratu);
22
23         guzik = (Button) findViewById(R.id.button1);
24         bok  = (EditText) findViewById(R.id.editText1);
25         wynik = (TextView) findViewById(R.id.textView4);
26
27         OnClickListener l = new OnClickListener() {
28
29             @Override
30             public void onClick(View v) {
31                     wynik.setText(  (   Double.parseDouble(bok.getText().toString())
32                             )*(
33                                     Double.parseDouble(bok.getText().toString())        ) +"" );
34             }
35         };
36         guzik.setOnClickListener(l);
37     }
38
39
40 }
41
```

W analogiczny sposób jak dla kliknięcia napisu „Pole kwadratu” oprogramowałem tutaj kliknięcie przycisku z napisem „Oblicz”. Zdefiniowałem w liniach 14-16 obiekty które będą reprezentować używane przeze mnie komponenty. W liniach 23-25 przypisuję do tych obiektów referencje do komponentów. W liniach 27-36 mam oprogramowaną reakcję na kliknięcie przycisku. Działa to analogicznie jak wcześniejszy przykład. Z nowości jest tutaj tylko rodzaj reakcji. W liniach 31-33 jest opisane co dokładnie ma się zdarzyć. Generalnie ustawiam tekst na komponencie **wynik** (to ten komponent klasy LargeText który ma domyślnie ustawione trzy myślniki. Ustawiany tekst to kwadrat wartości podanej poprzez obiekt **bok** – czyli obiekt klasy EditText, do którego użytkownik wprowadzi długość boku kwadratu. Po drodze pojawiają się też rzutowania z tekstu na liczbę i odwrotnie :)

Program po uruchomieniu:



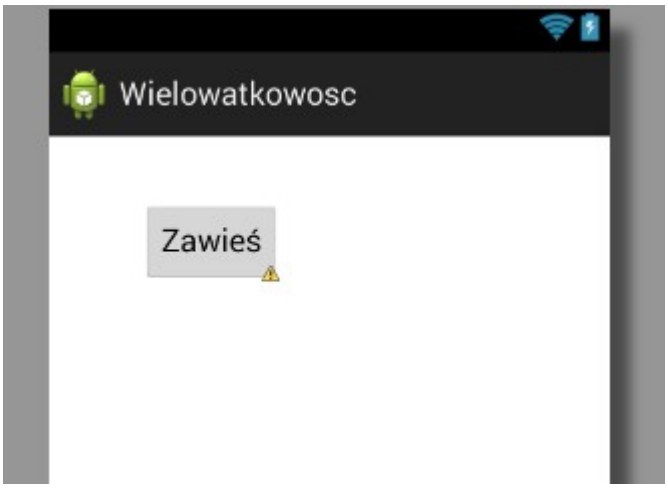
Po wprowadzeniu długości boku i kliknięciu przycisku „Oblicz” mamy taki efekt:

Jako ćwiczenie pozwalające nabyć nieco praktyki i oswoić się z nową technologią, proponuję dokończyć opisywany w tym rozdziale projekt.

Wielowątkowość w Androidzie

Wykorzystanie klas Thread i AsyncTask

Jak bardzo potrafi irytować zawieszanie się aplikacji wie każdy z nas. Teraz wyobraź sobie, że Twoja aplikacja na Androida ma do wykonania jakieś czasochłonne czynności, wystarczy że będzie musiała pobrać jakieś dane czy obraz z internetu. Jeśli zrobisz to w ramach głównej wątku programu, UI przestanie odpowiadać, a użytkownik Twojego programu będzie życzył Ci właśnie tego, czego zapewne Ty życzysz czasem programistom Internet Explorera :) Aby się o tym przekonać, wystarczy przykleić na ekranie przycisk, a jako jego reakcję oprogramować zdarzenie oczekiwania. Przyklej guzik:



a jako obsługę jego naciśnięcia wpisz taki kod:

```
15
16      @Override
17      protected void onCreate(Bundle savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_main);
20          b=(Button)findViewById(R.id.button1);
21
22          OnClickListener l = new OnClickListener() {
23              @Override
24              public void onClick(View v) {
25                  try {
26                      Thread.sleep(10000);
27                  } catch (InterruptedException e) {
28                      e.printStackTrace();
29                  }
30              }
31          };
32          b.setOnClickListener(l);
33      }
34
```

W zasadzie interesują nas tylko linie 24-32. Reszta to otoczka. Jak widzimy, w chwili naciśnięcia przycisku wywoływana jest instrukcja Thread.sleep(10000). To sprawi że aplikacja zaczeka 10 sekund. Spróbuj to uruchomić na dowolnym urządzeniu i np. wywołać menu pogramu (tym dedykowanym guzikiem do menu). Zobaczysz że nic się nie stanie, aplikacja po prostu się zawiesiła.

Trzeba więc wydzielić osobny wątek i puścić go w tle do realizowania czasochłonnych zadań. Można zrobić to „tradycyjnie” po javowemu:

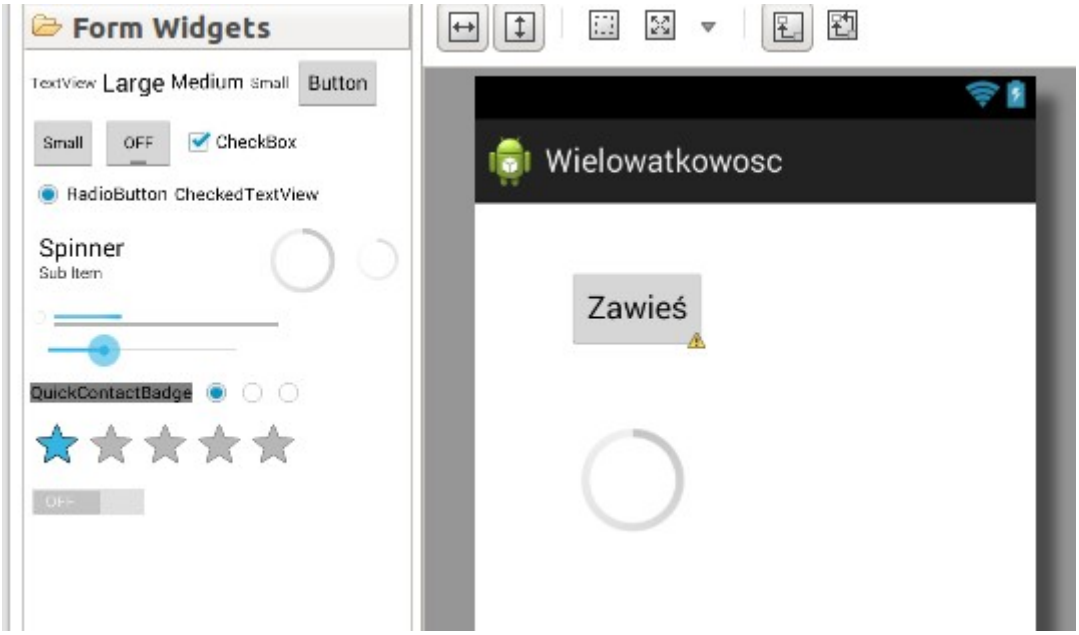
```
18      @Override
19      protected void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21          setContentView(R.layout.activity_main);
22          b=(Button)findViewById(R.id.button1);
23          OnClickListener l = new OnClickListener() {
24              @Override
25              public void onClick(View v) {
26
27                      new Thread(
28                              new Runnable() {
29                                  @Override
30                                  public void run() {
31                                      try {
32                                          Thread.sleep(10000);
33                                      } catch (InterruptedException e) {
34                                          e.printStackTrace();
35                                      }
36                                  }
37                              }
38                              ).start();
39              }
40          };
41          b.setOnClickListener(l);
42      }
```

Możesz uruchomić tak przerobiony program, a przekonasz się że wątek działający w tle nie blokuje innych elementów interfejsu. Jest oczywiście pewne „ale”. Jeśli spróbujesz sięgnąć z poziomu kodu do jakichkolwiek komponentów wizualnych z poziomu takiego wątku, dostaniesz taki wyjątek:

```
Text
threadid=11: thread exiting with uncaught exception (group=0x40e942 ⏎
a0)
FATAL EXCEPTION: Thread-1133
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the  ⏎
original thread that created a view hierarchy can touch its views.
        at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:4888)
        at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:943)
```

Myślę że komunikat tego wyjątku jest zrozumiały.

Na potrzeby Androida powstał zupełnie osobny mechanizm wielowątkowości. Wykorzystamy tutaj klasę AsyncTask która jak sama nazwa wskazuje, służy do wykonywania zadań asynchronicznie. Wykorzystamy też jeden ciekawy gadżet – ProgressBar. Jest dostępny na palecie Form Widgets.



Chcemy żeby użytownik widział, że program coś wykonuje, a nie po prostu zwisł. Taki ProgressBar domyślnie jest widoczny i wykonuje się animacja. My zrobimy tak, żeby na początku nie był widoczny, pokażemy go gdy puścimy wątek i schowamy gdy go zakończymy.

Wykorzystanie wcześniej wspomnianej klast AsyncTask polega na stworzeniu klasy która po niej dziedziczy i zaimplementowaniu paru metod. Do klasy naszej aktywności dodaję wewnętrzną klasę OsobnyWątek dziedziczącą po AsyncTask. Nic nie stoi na przeszkodzie by była to w ogóle osobna klasa. Pojawiło się tutaj też coś nowego – element <Void,Void,Void> oraz dziwna parametryzacja metod doInBackground i onPostExecute. Związane jest to z pojęciem klas generycznych, ogólnych. Tym zagadnieniem nie będziemy się tutaj zajmować, nie jest związane bezpośrednio z tematem wielowątkowości w Androidzie. Obiekty klasy OsobnyWatek będą stanowiły osobne wątki w ramach aplikacji. Zauważ że mamy tutaj trzy metody przesłaniające takie metody z klasy po której dziedziczymy. Metoda onPreExecute jest automatycznie wywoływana przy starcie wątku. Wyświetlam tutaj nasz ProgressBar. Kółko staje się widoczne i zaczyna „pracować”. Daję też komunikat na konsolę. W tej metodzie, oraz w metodzie onPostExecute możemy bez obaw odwoływać się do komponentów graficznych w naszej aktywności. Nie robimy tego za to w metodzie doInBackground. To co ma się dziać w ramach wątku – te długotrwałe czynności piszemy właśnie w metodzie doInBackground. Tutaj wyrzucam na konsolę informację, oraz wstawiam 10 sekundowe oczekiwanie. Metoda onPostExecute jest wywoływana automatycznie w chwili zakończenia wątku. Wyświetlam tutaj informację o zakończeniu i chowam ProgressBar.

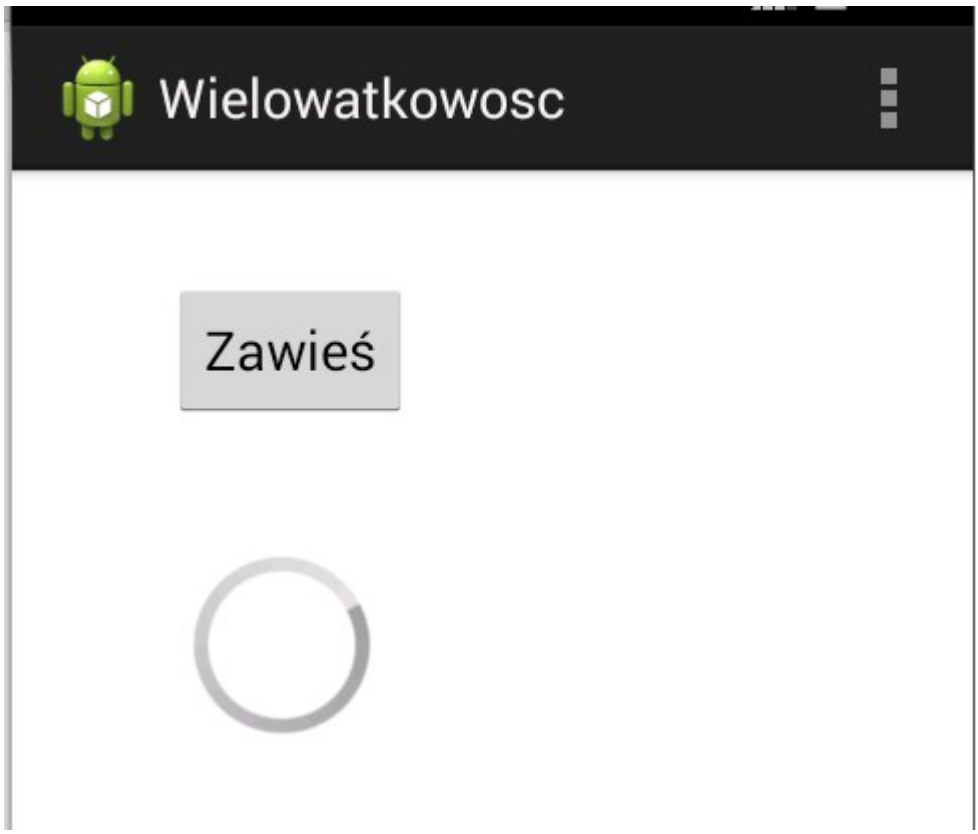
```
14 public class MainActivity extends Activity {
15
16
17     Button b;
18     ProgressBar pb;
19
20     private class OsobnyWatek extends AsyncTask<Void,Void,Void>{
21
22
23         @Override
24         protected void onPreExecute(){
25             Log.d("Cześć","Jestem wątkiem!");
26             pb.setVisibility(ProgressBar.VISIBLE);
27         }
28
29         @Override
30         protected Void doInBackground(Void... params) {
31             Log.d("Wiszę....","Sorry, taki mamy klimat...");
32             try {
33                 Thread.sleep(10000);
34             } catch (InterruptedException e) {
35                 e.printStackTrace();
36             }
37             return null;
38         }
39         @Override
40         protected void onPostExecute(Void result){
41             pb.setVisibility(ProgressBar.INVISIBLE);
42             Log.d("Do","zobaczenia");
43         }
44
45     }
46
```

Zobaczmy teraz co się dzieje w metodzie onCreate:

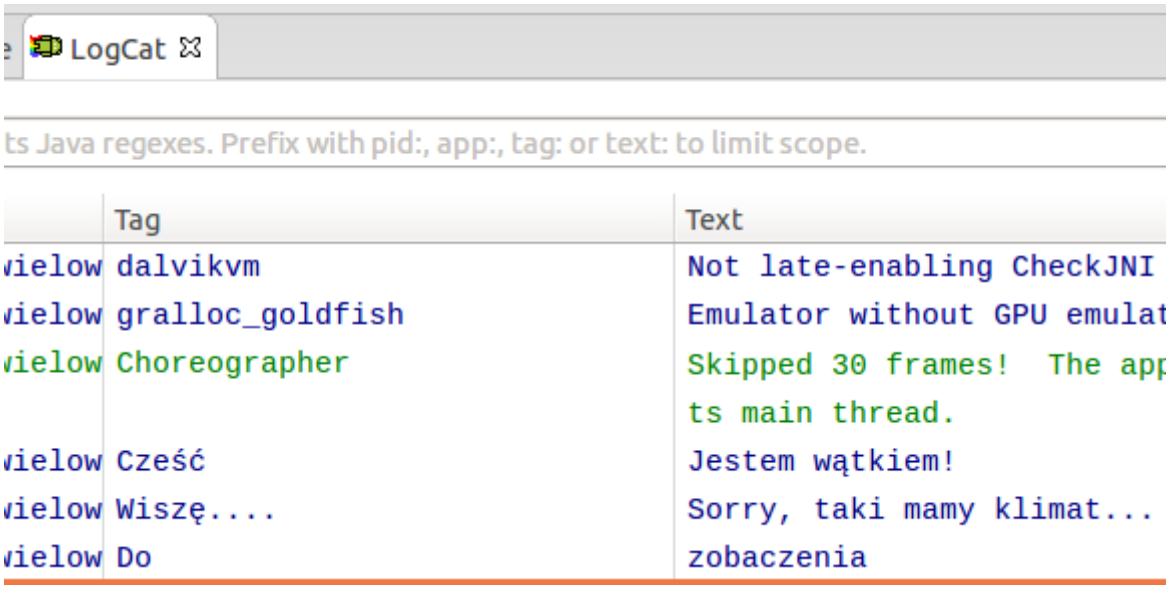
```
47
48      @Override
49      protected void onCreate(Bundle savedInstanceState) {
50          super.onCreate(savedInstanceState);
51          setContentView(R.layout.activity_main);
52          b=(Button)findViewById(R.id.button1);
53
54          pb=(ProgressBar) findViewById(R.id.progressBar1);
55          pb.setVisibility(ProgressBar.INVISIBLE);
56
57          OnClickListener l = new OnClickListener() {
58              @Override
59              public void onClick(View v) {
60                      new OsobnyWatek().execute();
61              }
62          };
63          b.setOnClickListener(l);
64      }
```

W linii 55 na początek chowam ProgressBar. Będzie on wyświetlany tylko w trakcie działania wątku w tle. Jako reakcję na naciśnięcie przycisku uruchamiam nasz wątek (linia 60).

Program po uruchomieniu i kliknięciu przycisku wygląda tak:



Zawartość konsoli LogCata po zakończeniu całego procesu:



Utrwalanie i przechowywanie danych

Wykorzystanie bazy danych SQLite w Androidzie

Android zapewnia wbudowaną obsługę baz SQLite. Nie potrzebujemy żadnych dodatkowych bibliotek, wszystko co niezbędne jest dla Androida natywne. SQLite jest łatwą w obsłudze lekką bazą danych i wykorzystywana jest nie tylko w systemie Android. Również w zwykłej Javie czy JEE możemy tę bazę wykorzystać. Jest jednak kilka różnic w obsłudze. Po pierwsze w zwykłej Javie/JEE potrzebujemy dodawać biblioteki SQLite, po drugie z samą bazą łączymy się z użyciem JDBC czy jakiegoś ORMa np. Hibernate. Tutaj cała baza zawiera się w jednym pliku który na dodatek jest generowany automatycznie :)

W niniejszym przykładzie stworzymy prostą bazę kontaktów. W pierwszej kolejności muszę stworzyć klasę dziedziczącą po SQLiteOpenHelper i przesłonić metody onCreate i onUpgrade, oraz stworzyć konstruktor sparametryzowany. Klasa ta będzie służyła do tworznia bazy danych i niezbędnych tabel.

```
1  package com.example.kontaktysqllite;
2
3  import android.content.Context;
4  import android.database.sqlite.SQLiteDatabase;
5  import android.database.sqlite.SQLiteDatabase.CursorFactory;
6  import android.database.sqlite.SQLiteOpenHelper;
7
8  public class ZarzadcaBazy extends SQLiteOpenHelper{
9
10     public ZarzadcaBazy(Context context) {
11         super(context, "kontakty.db", null, 1);
12     }
13
14     @Override
15     public void onCreate(SQLiteDatabase db) {
16         db.execSQL(
17                 "create table telefony(" +
18                 "nr integer primary key autoincrement," +
19                 "imie text," +
20                 "nazwisko text," +
21                 "telefon text);" +
22                 "");
23
24     }
25     @Override
26     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
27     }
28
29 }
30
```

Konstruktor jako taki przyjmuje tylko jeden parametr – kontekst, który zresztą przekazuje następnie do konstruktora klasy po której dziedziczy. „Kontakty.db” to nazwa pliku bazy danych który zostanie utworzony. Jedynka w konstruktorze to wersja bazy danych. Możemy tę wartość zwiększać przy kolejnych zmianach w strukturze bazy.

Podczas pierwszego skorzystania z obiektu naszej klasy ZarzadcaBazy, SQLiteOpenHelper zauważy brak pliku bazy (tutaj kontakty.db) i go utworzy a następnie wywoła metodę onCreate. W mojej metodzie onCreate, posługując się metodą execSQL obiektu klasy SQLiteDatabase tworzę tabelę w której będę przechowywał dane. Nic nadzwyczajnego – zwykły SQL. Metoda onUpgrade zostanie wywołana jeśli będziemy się odnosić do przestarzałej wersji bazy danych (silnik bazy będzie to wiedział na podstawie numeru wersji którą podajemy jako parametr konstruktora). Możemy tutaj np. zaimplementować wywołanie jakichś poleceń rozbudowujących czy zmieniających struktury bazy.

```
31
32
33      public void dodajKontakt(String imie,String nazwisko, String telefon){
34      SQLiteDatabase db = getWritableDatabase();
35          ContentValues wartosci = new ContentValues();
36          wartosci.put("imie", imie);
37          wartosci.put("nazwisko",nazwisko);
38          wartosci.put("telefon", telefon);
39          db.insertOrThrow("telefony",null, wartosci);
40      }
41
42      public Cursor dajWszystkie(){
43          String[] kolumny={"nr","imie","nazwisko","telefon"};
44          SQLiteDatabase db = getReadableDatabase();
45          Cursor kursor =db.query("telefony",kolumny,null,null,null,null,null);
46          return kursor;
47      }
48
49  }
```

Teraz dodamy funkcjonalność odpowiedzialną za dodawanie, modyfikowanie, kasowanie i wyświetlanie danych. Na razie jest bardzo ubogo :) Kolejne dwie metody również dodaję do klasy ZarządcaBazy. Oczywiście schemat ten jest bardzo uprowadzony, raczej powinniśmy stosować oddzielne klasy DAO do zadań tego typu. Taka uproszczona forma ułatwi zrozumienie samej zasady działania, potem każdy może sobie to przerobić wg własnego uznania. Dodałem metodę dodajKontakt która będzie służyła do dodawania nowych kontaktów. Metoda ta przyjmuje 3 parametry które stanową dane dodawane później do tabeli tj imię, nazwisko, telefon.

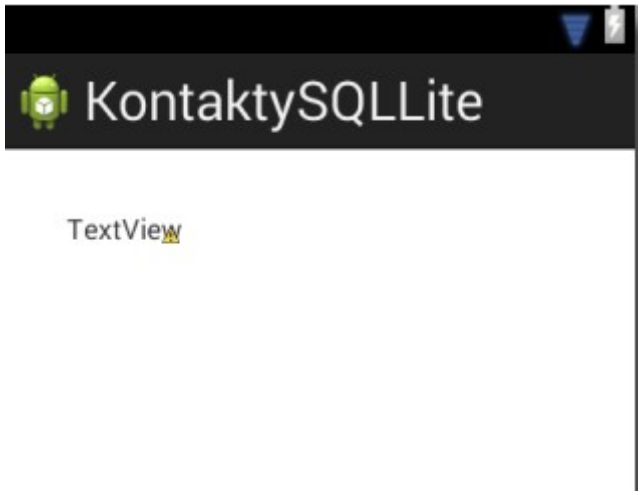
W linii 34 uzyskuję uchwyt do bazy. Korzystam tutaj z metody getWritableDatabase, ponieważ będę wprowadzał zmiany w bazie. W sytuacji gdybym chciał jedynie odczytywać dane, wywołałbym metodę getReadableDatabase. Linie 36-38 służą podstawieniu wartości do poszczególnych kolumn. Pierwszy parametr metody put to nazwa kolumny do której wstawiamy dane, drugi to wartość która ma do niej trafić. Nie muszę uzupełniać wszystkich. Swoją drogą przykładowo kolumny nr nie uzupełniam wcale. To jest kolumna typu „autoincrement”, a więc wartości w niej będą generowane automagicznie. W linii 16 wywołuję metodę insertOrThrow której przekazuję jako parametry nazwę tabeli do której wstawiane są dane, oraz listę wartości klasy ContentValues którą uzupełniałem wcześniej. Metoda ta służy do wstawienia wiersza do tabeli.

W razie problemów wyrzuci wyjątek typu SQLExcepion, ale możemy go wyłapać obejmując to wywołanie blokiem try catch.

Do klasy ZarzadcaBazy dodalem tez metodę dajWszystkie. Sluzy ona pobieraniu danych z bazy.

W linii 43 zdeklarowałem kolumny jakie chcę odczytać, oraz ich kolejność. Nie muszę odczytywać wszystkich kolumn, ani też nie muszę wyciągać ich w takiej kolejności w jakiej są w tabeli. W linii 44 pobieram uchwyt do bazy. Tym razem wywołuję jednak metodę getReadableDatabase, ponieważ nie będę w bazie nic modyfikował, a jedynie czytał. Zasadniczo pobranie danych jest realizowane metodą query w linii 45. Podaję tutaj nazwę tabeli z której będę czytał dane, oraz jako parametr listę kolumn które będę pobierał. Pozostałe parametry (które są tutaj nullami) służą do stsowania warunków WHERE, Having, grupowania, sortowania etc.

Przyszedł czas na dorobienie warstwy prezentacji. Przyklejam więc na swojej głównej aktywności komponent klasy TextView:



W ramach tego przykładu, do tabeli telefony dodam z użyciem dotychczas stworzonych funkcji 3 nowe kontakty, a następnie je wyświetlę.

W metodzie onCreate aktywności głównej zasadniczy najważniejszy dla nas element mieści się w liniach 18-28. Linia 18 to stworzenie obiektu zarządcy bazy danych – a więc tej klasy która jest odpowiedzialna za łączenie się z bazą i operacje na niej. Konstruktor tej klasy wymaga podania konktekstu. Tutaj jako kontekt wskazuję „this”, a więc bieżącą aktywność. Z użyciem metody „dodajKontakt” zawartej w owej klasie, dodaję trzy nowe kontakty. W dalszej kolejności chcę je pobrać z bazy i wyświetlić na ekranie. Korzystam z metody dajWszystkie, która jednak zwraca obiekt klasy Cursor. Przetwarzanie takiego kursora nie jest najwygodniejsze więc za chwilę zajmiemy się przerobieniem wszystkiego w taki sposób by metoda dajWszystkie zwracała listę obiektów np. klasy Kontakt. Krok po kroku. Na razie mamy spartańskie warunki.

W pętli while (linia 23) widzimy metodę moveToNext(), wykorzystanie tej metody pozwala nam na przesuwanie się po wyniku zapytania po linii w dół tak długo jak długo są jeszcze jakieś dane do przeczytania. W liniach 24-27 do definiowanych w pętli zmiennych przypisuję zawartość kolumn z kolejnych wierszy wg pozycji tej kolumny od lewej – stąd wywołania typu getInt(0), getString(3) – chodzi o numer kolumny od lewej strony wg kolejności jaką wymieniliśmy w metodzie dajWszystkie klasy ZarzadcaBazy. Zauważ że posługuję się też różnymi metodami w zależności od spodziewanego typu danych zwracanych z zapytania – mam na myśli metody getString i getInt. Pilnuj tego, ponieważ jeśli spróbujesz zrobić getInt lub np. getLong a w wyniku dostaniesz ciąg tekstowy , program wyrzuci wyjątek. W linii 28 zwyczajnie konkatenuję uzyskane informacje i wyświetlam je w kompononecie klasy TextView który przykleiłem tam przed momentem. Aby nie posklejało mi się to w jeden długi pozawijany ciąg – przed każdym kolejnym kontaktem doklejam „\n” czyli znak nowej linii.

activity_main.xml | MainActivity.java | ZarzadcaBazy.java

```
1  package com.example.kontaktysqllite;
2
3  import android.os.Bundle;
8
9  public class MainActivity extends Activity {
10
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16         TextView tv = (TextView)findViewById(R.id.textView1);
17
18         ZarzadcaBazy zb = new ZarzadcaBazy(this);
19         zb.dodajKontakt("Jan","Kowalski", "22 505 555 555");
20         zb.dodajKontakt("Krzysiu","Z Klanu", "42 545 666 554");
21         zb.dodajKontakt("Dziadek","Mróz", "112");
22         Cursor k = zb.dajWszystkie();
23         while(k.moveToNext()){
24             int nr=k.getInt(0);
25             String imie=k.getString(1);
26             String nazwisko=k.getString(2);
27             String telefon=k.getString(3);
28             tv.setText(tv.getText()+"\n"+imie+" "+nazwisko+" "+telefon);
29         }
30
31     }
32
33
34
35 }
```

Efekt :

Pokażę teraz ciekawą właściwość na bazie pewnego przykładu. Zawartość tej bazy danych będzie trwała pomiędzy kolejnymi uruchomieniami programu. Jeśli więc teraz uruchomię jeszcze raz ten program, a co za tym idzie po raz kolejny dodam trzy te same nowe kontakty, powinienem teraz zobaczyć zdublowane rekordy. Jedyne co powinno je odróżniać to identyfikatory rekordów tj. zawartość kolumny nr będącej kluczem głównym tabeli „telefony” a mającej własność „autoincrement”. Dokonuję więc drobnej kosmetycznej zmiany w kodzie:

```
23          while(k.moveToNext()){
24              int nr=k.getInt(0);
25              String imie=k.getString(1);
26              String nazwisko=k.getString(2);
27              String telefon=k.getString(3);
28              tv.setText(tv.getText()+"\n"+nr+" "+imie+" "+nazwisko+" "+telefon);
29          }
```

String nazwisko - com.examp

W linii 28 dodaję jedynie wyświetlanie kolumny nr Żadnych innych zmian nie nanoszę. Efekt jest więc taki jak przewidywaliśmy:

Aby program był w jakikolwiek sposób użyteczny, oraz aby kod był przejrzysty, trzeba będzie dokonać kilku kolejnych zmian:

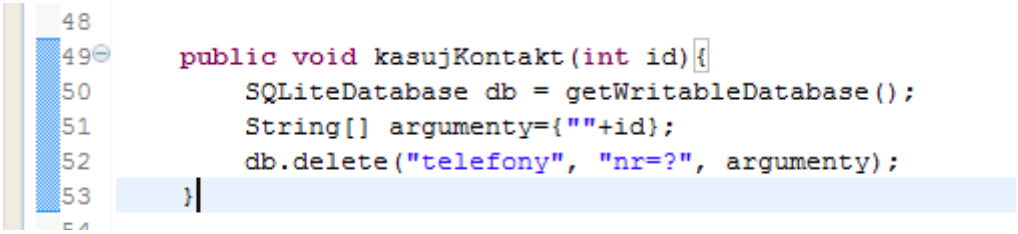
- Pozbyć się tego paskudnego „TextView” na samym początku listy.
- Dodać możliwość kasowania kontaktów
- Dodać możliwość aktualizowania kontaktów
- Stworzyć klasę „Kontakt” i posługiwać się obiektami tej klasy a nie listą elementów typu String.
- Dorobić metody pozwalające wyciągać dane kontakty wg wskazanych kryteriów (np. numeru identyfikacyjnego, numeru telefonu czy nazwiska).
- Dodać jakieś okienka umożliwiające wygodne dodawanianie, akatualizację i kasowanie kontaktów przez użytkownika.
- Odseparować kod odpowiedzialny za odczyt, zapis, aktualizację i kasowanie kontaktów do osobnej klasy.

Rozwiązanie problemu nr 1. sprowadza się tylko do wyczyszczenia wyświetlanego w komponencie klasy TextView tekstu przed jego uzupełnianiem w pętli. Widzimy to w linii nr 22 :

```
13      protected void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_main);
16          TextView tv = (TextView)findViewById(R.id.textView1);
17
18          ZarzadcaBazy zb = new ZarzadcaBazy(this);
19          zb.dodajKontakt("Jan","Kowalski", "22 505 555 555");
20          zb.dodajKontakt("Krzysiu","Z Klanu", "42 545 666 554");
21          zb.dodajKontakt("Dziadek","Mróz", "112");
22          tv.setText("");
23          Cursor k = zb.dajWszystkie();
24          while(k.moveToNext()){
25              int nr=k.getInt(0);
26              String imie=k.getString(1);
27              String nazwisko=k.getString(2);
28              String telefon=k.getString(3);
29              tv.setText(tv.getText()+"\n"+nr+" "+imie+" "+nazwisko+" "+telefon);
30          }
31
32      }
```

Przy okazji widzimy że nasze 3 kontakty zostały dodane po raz kolejny ;) Z tym też musimy coś zrobić. In plus że mamy różne numery identyfikacyjne.

Dodam więc prostą metodę służącą do kasowania zbędnych rekordów.



Oczywiście bazę podpinamy w trybie do zapisu, co widzimy w linii 50. W linii 51 definiuję listę elementów argumentów dla warunków where. Musi to być lista argumentów klasy String. Myślę że najlepiej będzie to wyjaśnić na przykładzie. Przyjrzyjmy się linii 52. Pierwszy warunek to nazwa tabeli z której chcemy kasować wiersze. Drugi to warunki które mają trafić po klauzuli where. Samego where już tutaj nie wymieniamy. Zamiast wartości liczbowych czy tekstowych które miałyby się pojawić w warunkach, stawiamy znak zapytania. Gdybyśmy mieli więcej niż jeden parametr, to ten ciąg tekstowy mógłby wyglądać np. tak: „department_id=? and manager_id=? and salary>?”. System oczekiwałby w takiej sytuacji podania mu trzech wartości które trafią pod nasze znaki zapytania. Listę tychże właśnie wartości podaję w postaci kolekcji String zdefiniowanej w linii 51 poprzez trzeci parametr metody delete. W moim przypadku lista składa się z całego jednego elementu który na dodatek w sposób mało elegancki, aczkolwiek popularny rzutuję na typ String. Czyli podsumowując:

Argument nr 1 to nazwa tabeli z której kasujemy

Argument 2 to warunki dla klauzuli where

Argument 3 to lista wartości które mają trafić do parametrów warunków where określonych jako argument nr 2 metody delete :)

Przetestujmy teraz działanie naszej nowej metody.

```
12      @Override
13      protected void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_main);
16          TextView tv = (TextView)findViewById(R.id.textView1);
17
18          ZarzadcaBazy zb = new ZarzadcaBazy(this);
19          /*  zb.dodajKontakt("Jan","Kowalski", "22 505 555 555");
20          zb.dodajKontakt("Krzysiu","Z Klanu", "42 545 666 554");
21          zb.dodajKontakt("Dziadek","Mróz", "112"); */
22          tv.setText("");
23          Cursor k = zb.dajWszystkie();
24
25          for(int i=4;i<=9;i++){
26              zb.kasujKontakt(i);
27          }
28          while(k.moveToNext()){
29              int nr=k.getInt(0);
30              String imie=k.getString(1);
31              String nazwisko=k.getString(2);
32              String telefon=k.getString(3);
33              tv.setText(tv.getText()+"\n"+nr+" "+imie+" "+nazwisko+" "+telefon);
34          }
35
36      }
```

Troszkę przerobiłem kod naszej głównej aktywności. Linie od 19 do 21 odpowiedzialne za dodawanie nwych wierszy po prostu wykomentowałem, za to 2 liniach 25-27 dodałem kasowanie wszystkich kontaktów o identyfikatorach od 4 do 9. Wyświetlanie kontaktów pozostało takie jakie było na początku. Aktualny stan :

Przydałaby się teraz metoda pozwalająca na modyfikowanie wpisów w bazie.

```
32
33      public void dodajKontakt(String imie,String nazwisko, String telefon){
34      SQLiteDatabase db = getWritableDatabase();
35          ContentValues wartosci = new ContentValues();
36          wartosci.put("imie", imie);
37          wartosci.put("nazwisko",nazwisko);
38          wartosci.put("telefon", telefon);
39          db.insertOrThrow("telefony",null, wartosci);
40      }
41
42      public void kasujKontakt(int id){
43          SQLiteDatabase db = getWritableDatabase();
44          String[] argumenty={""+id};
45          db.delete("telefony", "nr=?", argumenty);
46      }
47
48
49      public void aktualizujKontakt(int nr,String imie, String nazwisko, String telefon){
50          SQLiteDatabase db = getWritableDatabase();
51          ContentValues wartosci = new ContentValues();
52          wartosci.put("imie", imie);
53          wartosci.put("nazwisko",nazwisko);
54          wartosci.put("telefon", telefon);
55          String args[]={nr+""};
56          db.update("telefony", wartosci,"nr=?",args);
57      }
```

Troszkę przemieściłem metody w klasie „ZarzadcaBazy”, ponieważ nowo dodana metoda „aktualizujKontakt” ma parę elementów wspólnych z wcześniej napisanymi metodami „kasujKontakt” i „dodajKontakt”. W linii 49 określiłem jakie parametry przyjmuje metoda. Pierwszy to identyfikator wiersza (unikalny numer z kolumny nr) wg którego będę wybierał wiersz do aktualizacji. Pozostałe to wartości do aktualizacji. Przyjąłem że nie będę rozgraniczał aktualizji poszczególnych kolumn , robił osobnych metod ani skomplikowanych bloków warunkowych żeby aktualizować tylko te faktycznie zmienione kolumny. Po najmniejszej linii oporu. Zakładam że po prostu zczytam wiersz, zmienię jedno czy więcej pól i przekażę do tej metody tak by zaktualizowała wszystkie kolumny, niezależnie od tego czy faktycznie coś się w nich zmieniło czy nie.

Podobnie jak działo się to przy dodawaniu nowego wiersza, tak i tutaj w liniach 51-54 podaję listę wartości które mają trafić do poszczególnych kolumn. Są to wartości które ulegają zmianie – te które podaję przez parametry metody. Tak jak i wcześniej metoda put przyjmuje jako argumenty nazwę kolumny w której aktualizujemy wartość, oraz wartość dla tej kolumny. Podobnie jak w przypadku metody kasujKontakt, tak i tutaj w linii 55 wartości które zostaną podane jako argumenty warunku where. W linii 56 to już wywołanie metody update która przyjmje jako parametry kolejno: nazwę aktualizowanej tabeli,zaktualizowane wartości kolumn, warunek where, wartość lub wartości które mają trafić do warunku where w miejsce znaków zapytania.

Przerabiam teraz główną aktywność tak, aby przed wyświetleniem wywołać metodę aktualizującą i zmienić numer telefonu Dziadka Mroza:

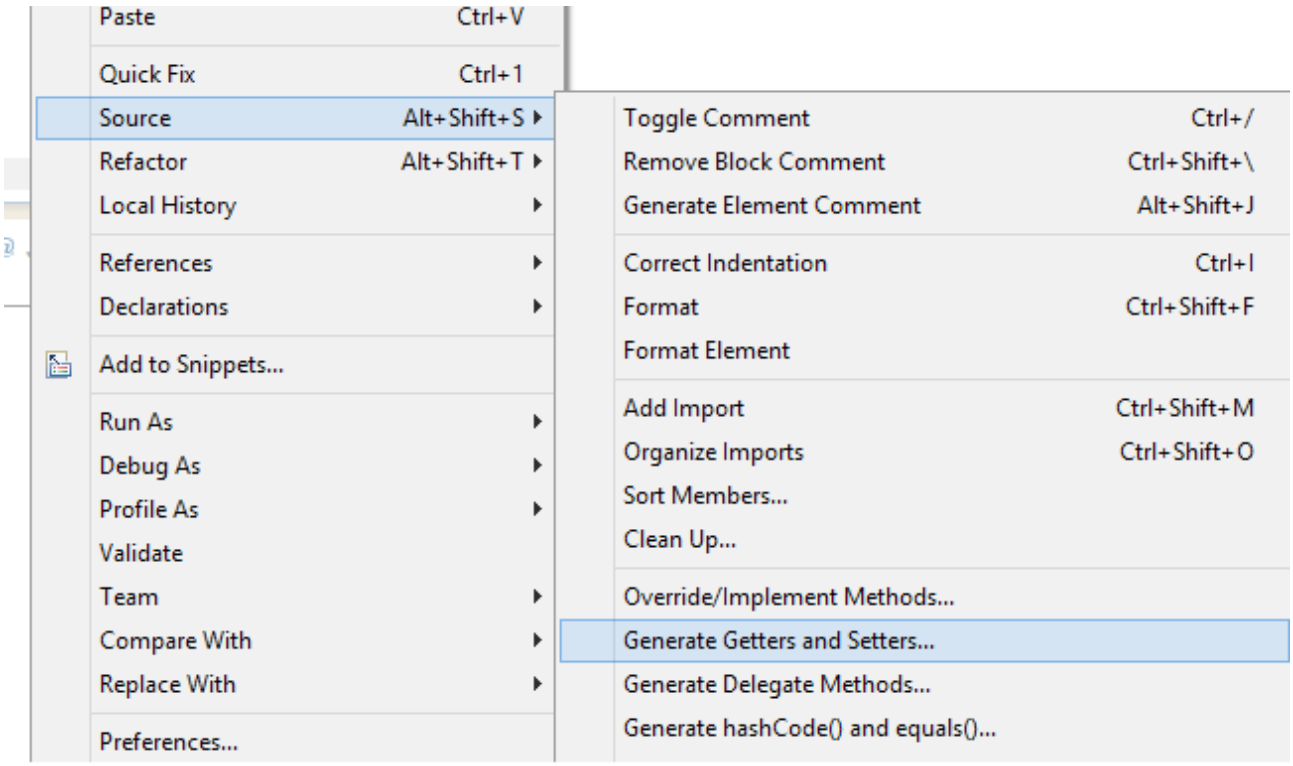
```
12      @Override
13      protected void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_main);
16          TextView tv = (TextView)findViewById(R.id.textView1);
17
18          ZarzadcaBazy zb = new ZarzadcaBazy(this);
19          /*  zb.dodajKontakt("Jan","Kowalski", "22 505 555 555");
20          zb.dodajKontakt("Krzysiu","Z Klanu", "42 545 666 554");
21          zb.dodajKontakt("Dziadek","Mróz", "112"); */
22          tv.setText("");
23          Cursor k = zb.dajWszystkie();
24          /*for(int i=4;i<=9;i++){
25              zb.kasujKontakt(i);
26          }*/
27          zb.aktualizujKontakt(3, "Dziadek", "Mróz", "997");
28          while(k.moveToNext()){
29              int nr=k.getInt(0);
30              String imie=k.getString(1);
31              String nazwisko=k.getString(2);
32              String telefon=k.getString(3);
33              tv.setText(tv.getText()+"\n"+nr+" "+imie+" "+nazwisko+" "+telefon);
34          }
35
36      }
```

Efekt:

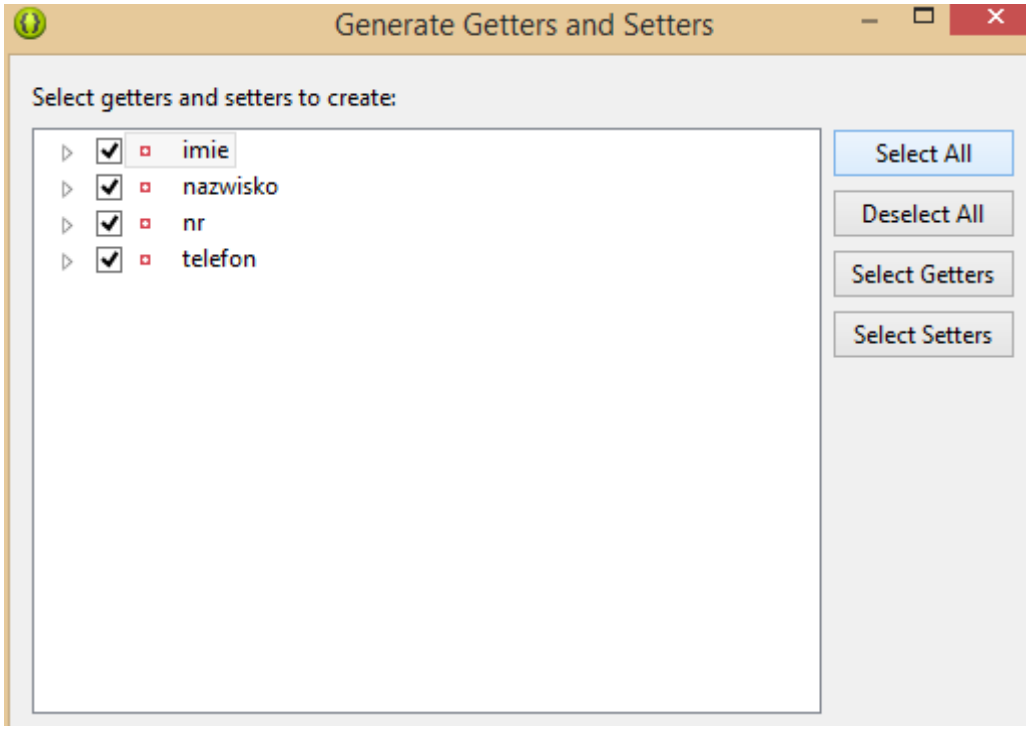
Oczywiście programistyczni puryści podniosą zarzut, że jakto tak z palca podawać wszystkie wartości włącznei z tymi nie zmienianymi, że te dane powinny być pobrane z bazy , zwrócone przez jakąś metodę w postaci obiektu, a następnie po zmianie wartości wybranego pola przekazane jako obiekt do metody aktualizującej w bazie. I jak najbardziej będą mieli rację :) Z tym, że aby to osiągnąć potrzebujemy najpierw przerobić to na troszkę bardziej obiektową formę, oraz stworzyć metodę zwracającą wybrany wiersz wg identyfikatora :). Zacznijmy więc od zdefiniowania klasy która będzie reprezentować pojedynczy wiersz. Starałem się tak definiować żeby pola w klasie odpowiadały nazwom kolumn w tabeli:

```
1  package com.example.kontaktysqllite;
2
3  public class Kontakt {
4      private Long nr;
5      private String imie;
6      private String nazwisko;
7      private String telefon;
8
9
10 }
```

Jest taka dobra praktyka programistyczna, by pola w klasach typu POJO nie były publiczne, a prywatne, natomiast dostęp do nich uzyskiwać poprzez tak zwane gettery i settery. Dzięki temu mamy większą kontrolę nad zawartością tych pól. W środowisku Eclipse jest nawet specjalna funkcja do automatycznego generowania tego typu metod. Wybieramy z menu dostępnego pod prawym przyciskiem myszki:



Gdy pojawi się okno, zaznaczamy wszystkie opcje:



i klikamy dostępny na dole przycisk „OK”.

W tej chwili nasza klasa powinna wyglądać mniej więcej tak:

```
1  package com.example.kontaktysqllite;
2
3  public class Kontakt {
4      private Long nr;
5      private String imie;
6      private String nazwisko;
7      private String telefon;
8      public Long getNr() {
9          return nr;
10     }
11     public void setNr(Long nr) {
12         this.nr = nr;
13     }
14     public String getImie() {
15         return imie;
16     }
17     public void setImie(String imie) {
18         this.imie = imie;
19     }
20     public String getNazwisko() {
21         return nazwisko;
22     }
23     public void setNazwisko(String nazwisko) {
24         this.nazwisko = nazwisko;
25     }
26     public String getTelefon() {
27         return telefon;
28     }
29     public void setTelefon(String telefon) {
30         this.telefon = telefon;
31     }
32
33
34 }
```

Stworzę teraz w naszym ZarzadcyBazy metodę odpowiedzialną za pobieranie z bazy i oddawanie wiersza w postaci obiektu nowo stworzonej klasy Kontakt. Dodaję do klasy ZarzadcaBazy metodę dajKontakt. W linii 67 tworzę na razie pusty obiekt klasy kontakt, który po uzupełnieniu zostanie z metody zwrócony. W linii 68, tak jak i w każdym dotychczasowym przypadku podpinam zaczep do bazy. W linii 69 definiuję kolumny które będę czytał. Widzimy że analogiczny fragment pojawia się po raz kolejny, trzeba więc będzie go odseparować w przyszłości. Linia 70 to tak jak i wcześniej podstawienie wartości która trafi pod znak „?” w warunku where. Linia 71 to zasadnicze wykonanie zapytania. Argumenty odpowiedzialne za grupowanie, warunek having, sortowanie i limitowanie ilości zwracanych wierszy (wg wymienionej kolejności) pozostawiam nullowe, ponieważ żadnej z tych operacji nie będę wykonywał. Jeśli w wyniku zapytania zostanie zwrócony przynajmniej jeden wiersz (co sprawdzam w linii 72), przechodzę do czytania danych z pól wiersza i uzupełniania obiektu. W linii 73 przechodzę do pierwszego (i w tym przypadku jedynego) wiersza.

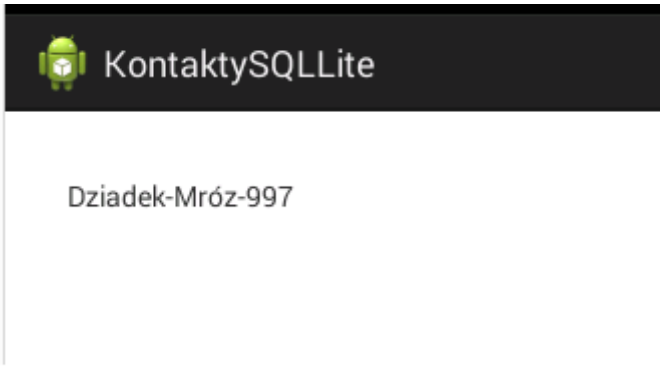
Linie 74-77 to znane już z wcześniejszych przypadków czytanie danych, tym razem jednak nie przypisuję ich do zmiennych, a przy użyciu setterów uzupełniam obiekt klasy kontakt. W linii 79 uzupełniony już (miejmy nadzieję – mogliśmy przecież poszukiwać kontaktu po numerze który w tabeli nie występuje) obiekt klasy kontakt zwracam z metody. Cała metoda:

```
66    public Kontakt dajKontakt(int nr){
67        Kontakt kontakt=new Kontakt();
68            SQLiteDatabase db = getReadableDatabase();
69            String[] kolumny={"nr","imie","nazwisko","telefon"};
70            String args[]={nr+""};
71            Cursor kursor=db.query("telefony",kolumny," nr=?",args,null,null,null,null);
72            if(kursor!=null){
73                kursor.moveToFirst();
74                kontakt.setNr(kursor.getLong(0));
75                kontakt.setImie(kursor.getString(1));
76                kontakt.setNazwisko(kursor.getString(2));
77                kontakt.setTelefon(kursor.getString(3));
78            }
79        return kontakt;
80    }
81
```

Aby przetestować nową metodę przerabiam ponownie główną aktywność:

```
12    @Override
13    protected void onCreate(Bundle savedInstanceState) {
14        super.onCreate(savedInstanceState);
15        setContentView(R.layout.activity_main);
16        TextView tv = (TextView)findViewById(R.id.textView1);
17
18        ZarzadcaBazy zb = new ZarzadcaBazy(this);
19        tv.setText("");
20        Kontakt k = zb.dajKontakt(3);
21        tv.setText(k.getImie()+"-"+k.getNazwisko()+"-"+k.getTelefon());
22
```

Efekt:



Przerobię teraz pozostałe metody z klasy ZarzadcaBazy, tak by całość była jako tako obiektowa :)

Zacząłem od zmiany metody dodajKontakt. Zamiast przyjmować wartości pojedynczych pól jako osobne argumenty metody, przekazuję po prostu jeden obiekt klasy Kontakt, który zostaje dodany do bazy. Jeszcze jedna drobna różnica jest widaczna w liniach 36-39. Zamiast korzystać z osobnych zmiennych reprezentujących wartości do poszczególnych pól, po prostu wydobywam niezbędne dane z obiektu, z użyciem zdefiniowanych w klasie Kontakt getterów.

```
32
33      public void dodajKontakt(Kontakt kontakt){
34      SQLiteDatabase db = getWritableDatabase();
35          ContentValues wartosci = new ContentValues();
36          wartosci.put("imie", kontakt.getImie());
37          wartosci.put("nazwisko",kontakt.getNazwisko());
38          wartosci.put("telefon", kontakt.getTelefon());
39          db.insertOrThrow("telefony",null, wartosci);
40      }
```

W podobny sposób przerobiłem metodę aktualizującą wiersze:

```
48
49      public void aktualizujKontakt(Kontakt kontakt){
50          SQLiteDatabase db = getWritableDatabase();
51          ContentValues wartosci = new ContentValues();
52          wartosci.put("imie", kontakt.getImie());
53          wartosci.put("nazwisko",kontakt.getNazwisko());
54          wartosci.put("telefon", kontakt.getTelefon());
55          String args[]={kontakt.getNr()+""};
56          db.update("telefony", wartosci,"nr=?",args);
57      }
```

Bardzo nie podoba mi się również dotychczasowe rozwiązanie pobierania wszystkich kontaktów:

```
59      public Cursor dajWszystkie(){
60          String[] kolumny={"nr","imie","nazwisko","telefon"};
61          SQLiteDatabase db = getReadableDatabase();
62          Cursor kursor =db.query("telefony",kolumny,null,null,null,null,null);
63          return kursor;
64      }
```

Metoda zwraca obiekt klasy Cursor, którego przetwarzanie nie należy do najwygodniejszych. Tą metodę przerobię też tak, by zwracała listę (klasy LinkedList) obiektów.

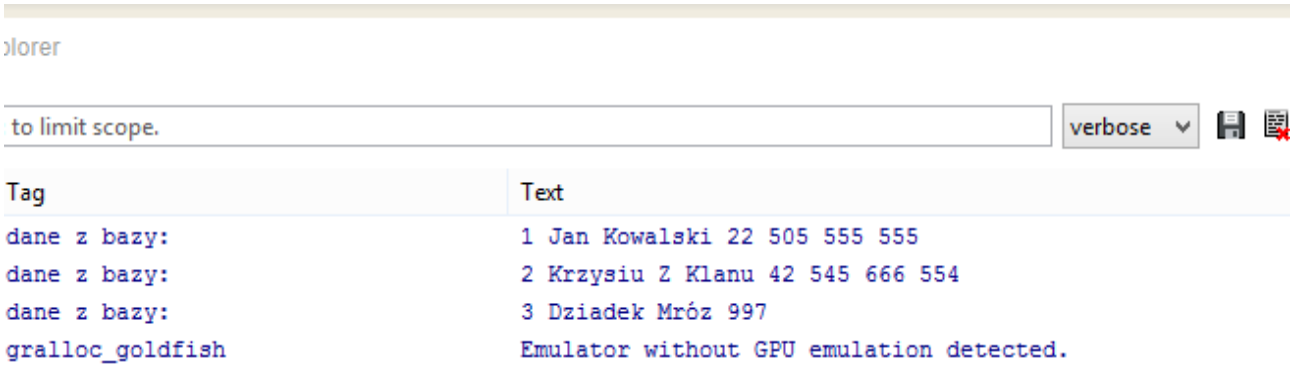
Po paru poprawkach metoda dajWszystkie działa tak jak sobie tego życzyliśmy. Sposób pobierania danych jest taki sam, jedynie dodałem w liniach 67-73 przetwarzanie kursora i ładowanie danych do przygotowanej wcześniej (w linii 63) listy kontaktów.

```
62      public List<Kontakt> dajWszystkie(){
63          List<Kontakt> kontakty = new LinkedList<Kontakt>();
64          String[] kolumny={"nr","imie","nazwisko","telefon"};
65          SQLiteDatabase db = getReadableDatabase();
66          Cursor kursor =db.query("telefony",kolumny,null,null,null,null,null);
67          while(kursor.moveToNext()){
68              Kontakt kontakt = new Kontakt();
69              kontakt.setNr(kursor.getLong(0));
70              kontakt.setImie(kursor.getString(1));
71              kontakt.setNazwisko(kursor.getString(2));
72              kontakt.setTelefon(kursor.getString(3));
73              kontakty.add(kontakt);
74          }
75          return kontakty;
76      }
```

Przetwarzanie takiej listy jest później dużo wygodniejsze. Poniżej wykorzystanie nowej wersji metody dajWszystkie w głównej aktywności:

```
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17          TextView tv = (TextView)findViewById(R.id.textView1);
18
19          ZarzadcaBazy zb = new ZarzadcaBazy(this);
20          tv.setText("");
21
22          for(Kontakt k:zb.dajWszystkie()){
23              Log.d("dane z bazy:", k.getNr()+" "+k.getImie()+" "+k.getNazwisko()+" "+k.getTelefon());
24          }
25
26      }
```

Przy okazji pojawia się nam tutaj nowa metoda: d klasy Log. Wywołuje się ją w taki sposób: Log.d(„tag”,”text”); a efekt wygląda tak:



Sprawdza się to zwłaszcza w zastępstwie System.out.println którego może nieco brakować programistom Javy ;)

Taki jeszcze drobny, acz istotny element. W tych przykładach tego nie wykorzystamy, ale może nastąpić konieczność pogrupowania, zastosowania klauzuli having, sortowania, czy limitowania ilości wierszy w zapytaniu. Trzeba odpowiednio podać parametry w wywołaniu metody query. Najlepiej będzie pokazać na przykładzie. Gdybym zechciał np. odebrać wynik takiego zapytania:

SELECT AVG(SALARY) , DEPARTMENT_ID FROM EMPLOYEES GROUP BY DEPARTMENT_ID ORDER BY DEPARTMENT_ID DESC;

To wywołanie metody query musiałoby wyglądać tak:

```
SQLiteDatabase db = getReadableDatabase();
String[] kolumny={"avg(salary) as srednia","department_id"};
Cursor kursor =
db.query("telefony",kolumny,null,null,”department_id”,null,
”department_id desc”);
```

Parametry wg kolejności :

1. Nazwa tabeli z której czytamy
2. Kolumny które czytamy
3. Warunki where
4. Wartości do warunków where
5. Kolumna/kolumny po których grupuję
6. Warunek do klauzuli having
7. Sposób sortowania

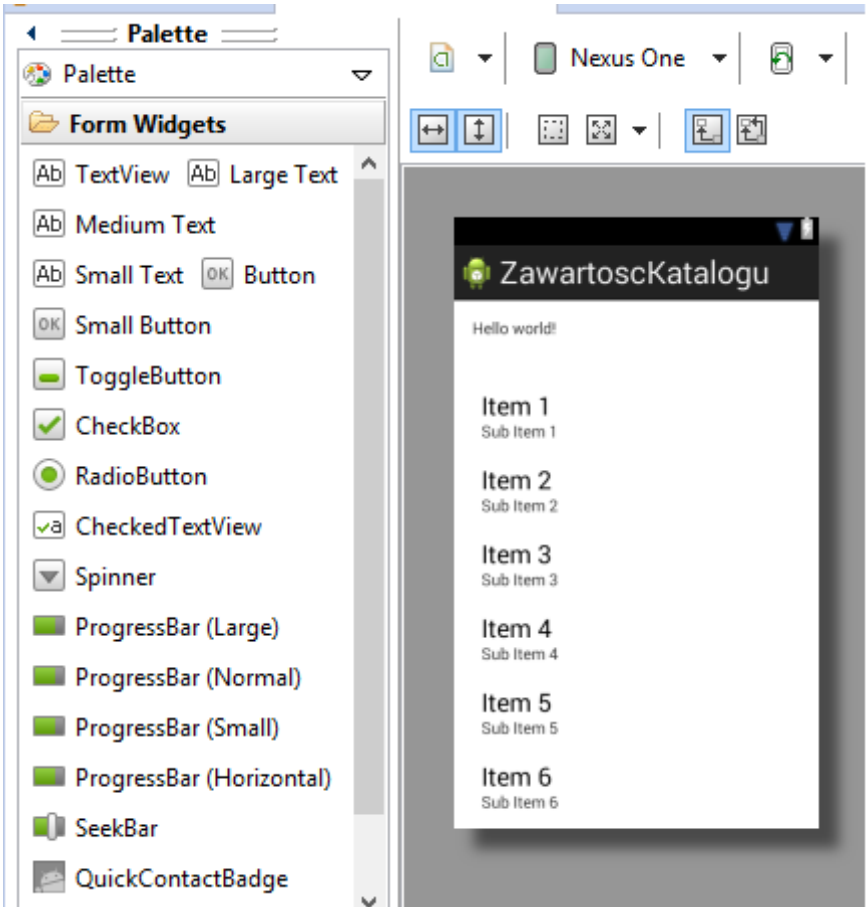
Tutaj już każdy może uznać czy chce taką postać wykorzystywać czy nie. Moim zdaniem to średnio wygodne i mało intuicyjne. Poza tym, co jeśli chciałbym stosować złączenia między tabelami, podzapytania etc? Zamiast takiej nieco abstrakcyjnej formy wywoływania zapytań z użyciem metody query, możemy wykorzystać metodę rawQuery. Do naszego zarządcy bazy dopisałem metodę wyciągającą kontakty o nazwisku które przekazuję jako parametr.

```
94      public List<Kontakt> dajPoNazwisku(String nazwisko){
95          List<Kontakt> kontakty = new LinkedList<Kontakt>();
96          String[] kolumny={"nr","imie","nazwisko","telefon"};
97          SQLiteDatabase db = getReadableDatabase();
98          Cursor kursor =db.rawQuery("select nr,imie,nazwisko,telefon from telefony where nazwisko='"
99                                      +nazwisko+
100                                     "' order by imie asc", null);
101         /*Alternatywne wywołanie metody rawQuery
102          *
103          * Cursor kursor =db.rawQuery
104          * ("select nr,imie,nazwisko,telefon from telefony where nazwisko=? order by imie asc",  nazwisko );
105          * */
106         while(kursor.moveToNext()){
107             Kontakt kontakt = new Kontakt();
108             kontakt.setNr(kursor.getLong(0));
109             kontakt.setImie(kursor.getString(1));
110             kontakt.setNazwisko(kursor.getString(2));
111             kontakt.setTelefon(kursor.getString(3));
112             kontakty.add(kontakt);
113         }
114         return kontakty;
115     }
```

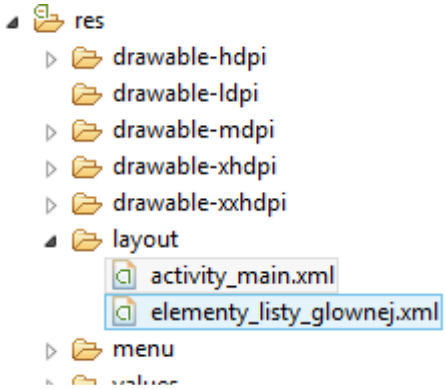
W przypadku metody rawQuery, nie używamy żadnych dodatkowych parametrów do określania grupowania, czy sortowania. Wszyskie tego typu rzeczy określamy w samym zapytaniu SQL bez żadnych udziwnień. Jak widać w liniach 98-100 potrzebny parametr zwyczajnie dołączyłem na zasadzie sklejania tekstu, warunek WHERE i sortowanie określone jest w samym SQL. Zdecydowanie preferuję tę formę, jako bardziej przyjazną :) W liniach 101-105 wstawiłem też zakomentowaną alternatywną postać tego samego wywołania. W tym przypadku zamiast sklejać zapytanie, posłużyłem się znakami zapytania i podstawiam wartości które pod te znaki zapytania mają trafić w trakcie wykonania z listy podanej jako drugi parametr metody rawQuery (tutaj lista jest po prostu jednym elementem typu String). Samo pobieranie danych z takiego kursora przebiega dokładnie tak jak wcześniej. Ponieważ wartości z kolejnych kolumn pobieram podając jej numer zaczynac liczyć od 0 od lewej strony, nie muszę nawet robić żadnych aliasów dla kolumn które wynikają np. z działania funkcji.

Sprawdzanie zawartości katalogu i własności plików

Rozpoczniemy od sprawdzania zawartości katalogu. Wyniki wyświetlę na komponencie ListView. W pierwszej kolejności dodaję więc taki komponent do mojej aktywności:



Do tego komponentu trzeba będzie dodać layout dla pojedynczych elementów listy (jest to niezbędne na późniejszych etapach), na razie tworzę pusty plik XML w katalogu layout.



Jego zawartość przedstawia się następująco:

MainActivity.java | activity_main.xml | elementy_listy_glownej.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/Row"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:textSize="15sp" >
</TextView>
```

Nic specjalnego. Plik opisuje po prostu w jaki sposób ma wyglądać pojedynczy element listy. Tutaj będzie to po prostu napis, ale równie dobrze możemy tutaj zastosować choćby TableLayout i ułożyć sobie bardziej skomplikowane struktury (np. obrazek i obok tekst).

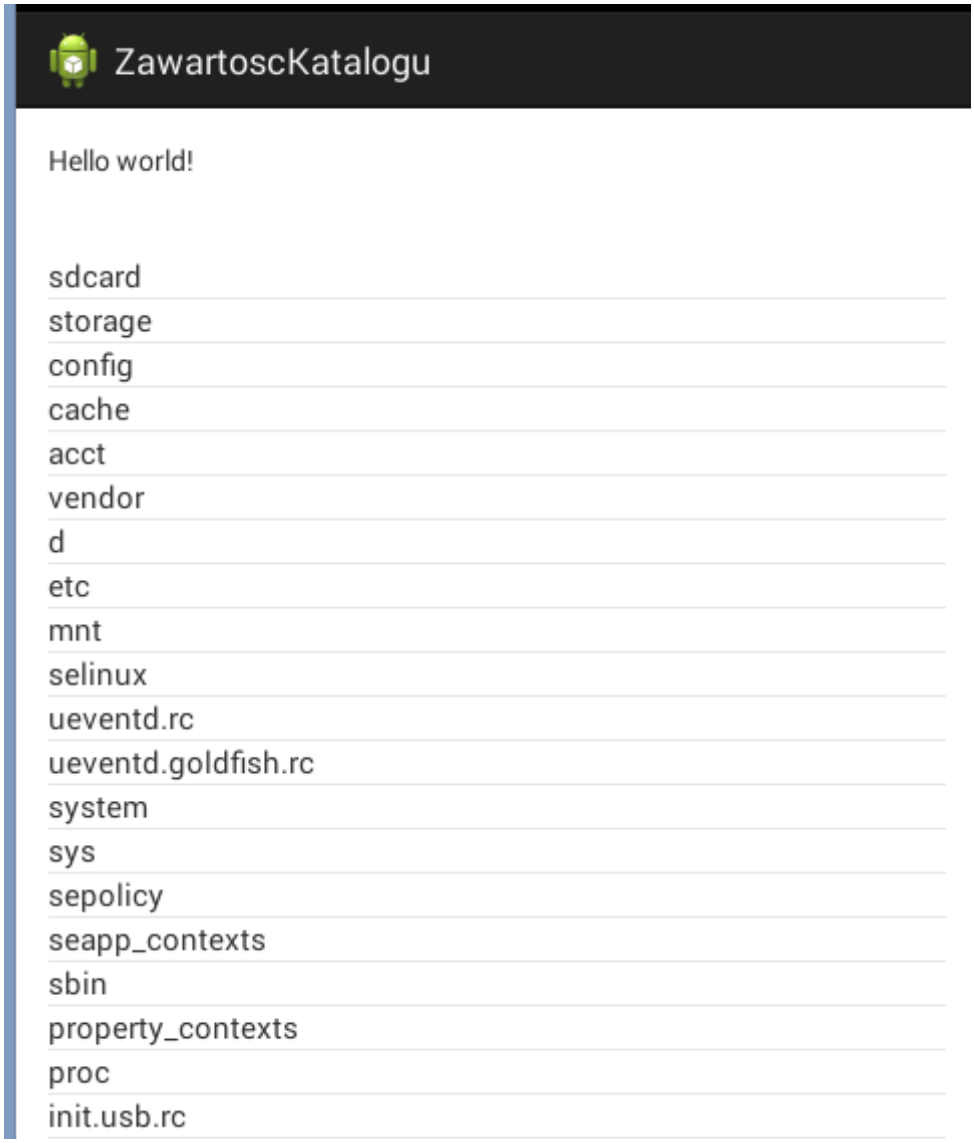
Przyszła pora na kod:

MainActivity.java | activity_main.xml

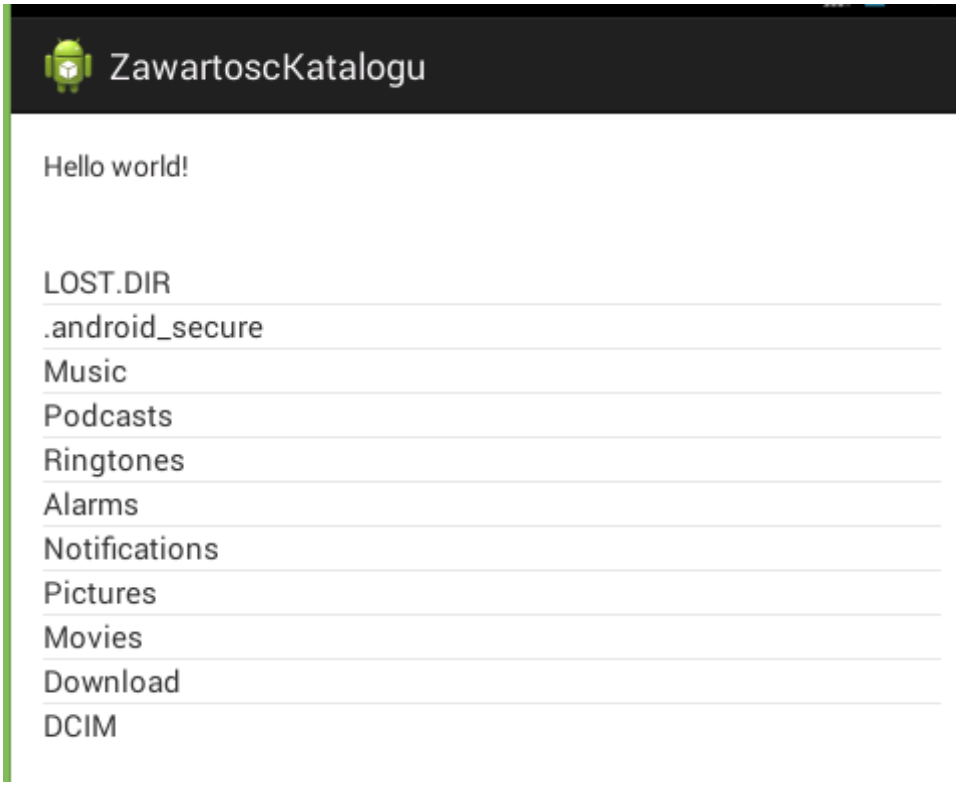
```
package com.example.zawartosckatalogu;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {

    private ListView listaKomponent;
    private ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listaKomponent = (ListView) findViewById(R.id.listView1);

        String[] elementy = new File(".").list();
        ArrayList<String> listaArray = new ArrayList<String>();

        listaArray.addAll(Arrays.asList(elementy));
        adapter = new ArrayAdapter<String>(this, R.layout.elementy_listy_glownej, listaArray);
        listaKomponent.setAdapter(adapter);
    }


}
```

Najważniejsze elementy znajdują się w liniach 23-28. W linii 23 do listy elementów typu String przypisuję listę plików z wskazanego katalogu. Klasa File to zwykła javowa File, nie żadna Androidowa interpretacja. W Javie każdy katalog też jest plikiem, stąd taki może nieco dziwny zapis. Katalog którego zawartość pobieram to „.” , czyli katalog root systemu. Równie dobrze może to być dowolny inny katalog, podajemy go jako parametr konstruktora klasy File. W liniii 24 tworzę obiekt klasy ArrayList. Będę musiał do adaptera (czyli elementu dzięki któremu uzupełniam zawartość komponentu klasy ListView) podać listę właśnie takiego typu. W linii 26 robię „konwersję” z dotychczas używanej listy stringów na obiekt klasy ArrayList. Linia 27 to inicjalizacja wcześniej wspomnianego adaptera. Mamy trzy parametry, pierwszy to kontekst, drugi do wskaźnik do zawartości pliku XML charakteryzującego wygląd pojedynczego elementu listy, trzeci to lista typu ArrayList zawierająca dane do wyświetlenia.

Efekt:



Zmieniłem katalog którego zawartość wyświetlam na „/sdcard” , czyli zawartość karty SD. Wynik wygląda tak:

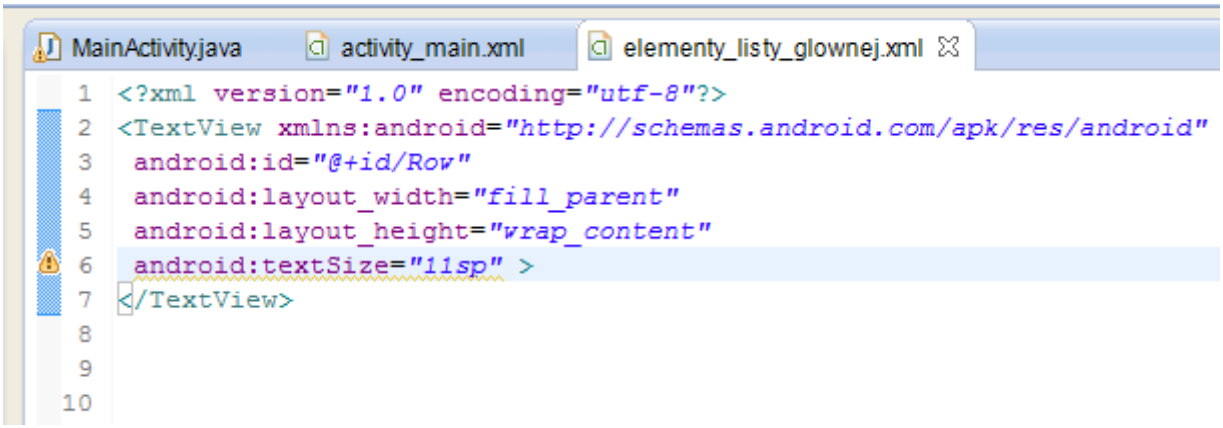


Dodajemy teraz wyświetlanie własności plików i katalogów.

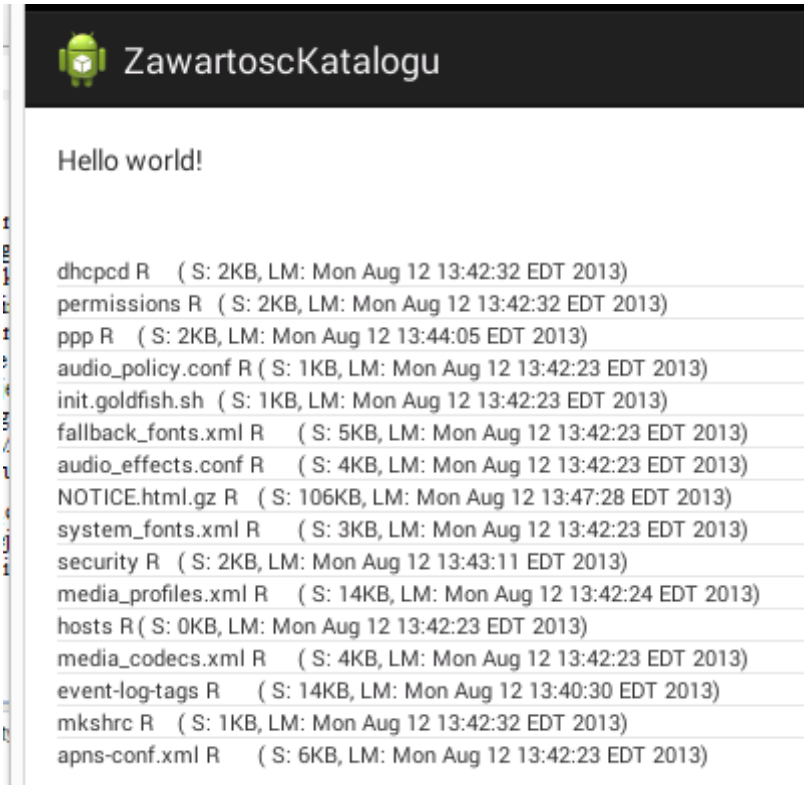
```
14 public class MainActivity extends Activity {
15
16     private ListView listaKomponent;
17     private ArrayAdapter<String> adapter;
18     String katalog="/etc/";
19     int i=0;
20
21     @Override
22     protected void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.activity_main);
25         listaKomponent = (ListView) findViewById(R.id.listView1);
26
27         String[] elementy = new File(katalog).list();
28
29         for(String element:elementy){
30             File f = new File(katalog+element);
31             if(f.canRead())elementy[i]=elementy[i]+" R";
32             if(f.canWrite())elementy[i]=elementy[i]+" W";
33             Date d=new Date();
34             d.setTime(f.lastModified());
35             elementy[i]=elementy[i]+"    ( S: "+f.length()/1024+"KB, LM: "+d+")";
36             i++;
37         }
```

Wprowadziłem kilka zmian w kodzie. W linii 27 zmieniłem troszkę wywołanie konstruktora klasy File, tak by korzystał ze zmiennej typu String o nazwie katalog, a nie podanej bezpośrednio ścieżki. Zrobiłem tak, ponieważ nieco dalej znowu korzystam ze ścieżki do pliku i nie chcę powielać tego samego kodu. Od linii 29 do 37 iteruję po elementach listy stringów zawierających nazwy plików i katalogów. W zależności od własności pliku/katalogu do jego nazwy doklejam różne informacje. Dalsza część kodu pozostaje bez zmian. Linia 31 to doklejenie do nazwy litery R, jeśli plik/katalog możemy odczytywać. Linia 32 to dodanie do nazwy litery W jeśli mamy możliwość pisania do pliku. Linie 33-35 służą dodaniu daty do nazwy pliku/katalogu. Korzystam tutaj z obiektu klasy java.util.Date, ponieważ wartość zwracana przez metodę lastModified() wyrażona jest jako int. W linii 36 powiększam wartość iteratora którego używam do poruszania się po liście.

W pliku XML elementy_listy_glownej.xml opisującym wygląd pojedynczego elementu listy również dokonałem małej zmiany (czysta kosmetyka). Zmniejszyłem wielkość czcionki z 15 na 11, ponieważ większa ilość informacji powodowałaby zawijanie linii, a tak mamy wszystkie informacje dotyczące pliku/katalogu w jednej linijcie :)



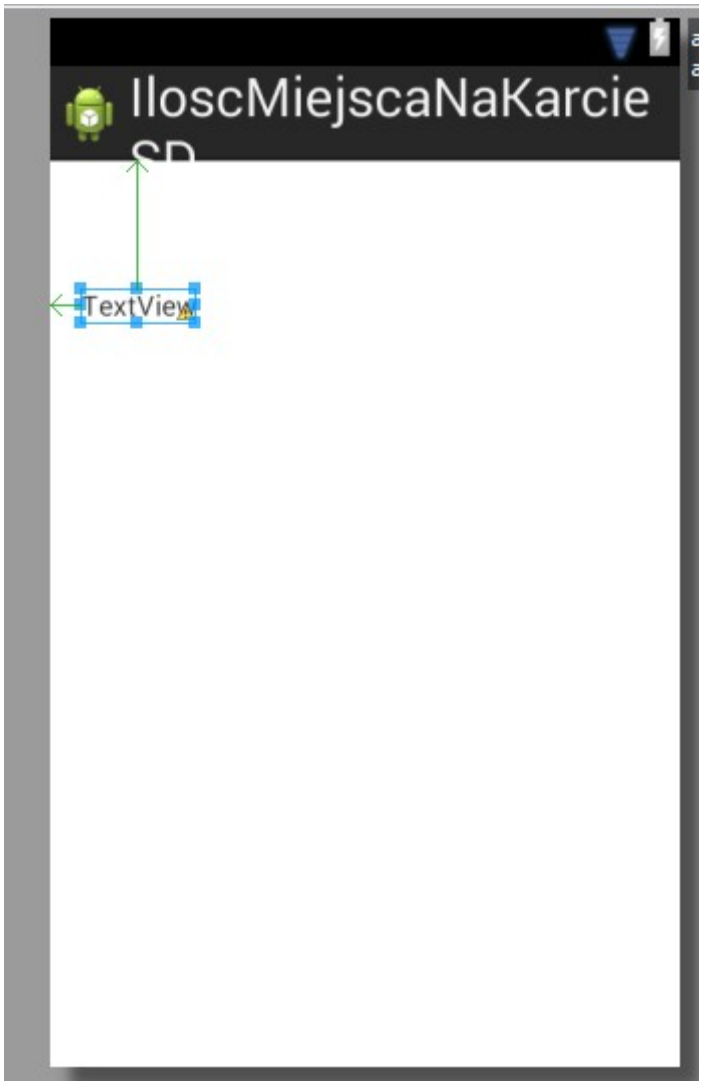
Efekt:


ZawartoscKatalogu
Hello world!
dhcpcd R (S: 2KB, LM: Mon Aug 12 13:42:32 EDT 2013)
permissions R (S: 2KB, LM: Mon Aug 12 13:42:32 EDT 2013)
ppp R (S: 2KB, LM: Mon Aug 12 13:44:05 EDT 2013)
audio_policy.conf R (S: 1KB, LM: Mon Aug 12 13:42:23 EDT 2013)
init.goldfish.sh (S: 1KB, LM: Mon Aug 12 13:42:23 EDT 2013)
fallback_fonts.xml R (S: 5KB, LM: Mon Aug 12 13:42:23 EDT 2013)
audio_effects.conf R (S: 4KB, LM: Mon Aug 12 13:42:23 EDT 2013)
NOTICE.html.gz R (S: 106KB, LM: Mon Aug 12 13:47:28 EDT 2013)
system_fonts.xml R (S: 3KB, LM: Mon Aug 12 13:42:23 EDT 2013)
security R (S: 2KB, LM: Mon Aug 12 13:43:11 EDT 2013)
media_profiles.xml R (S: 14KB, LM: Mon Aug 12 13:42:24 EDT 2013)
hosts R (S: 0KB, LM: Mon Aug 12 13:42:23 EDT 2013)
media_codecs.xml R (S: 4KB, LM: Mon Aug 12 13:42:23 EDT 2013)
event-log-tags R (S: 14KB, LM: Mon Aug 12 13:40:30 EDT 2013)
mkshrc R (S: 1KB, LM: Mon Aug 12 13:42:32 EDT 2013)
apns-conf.xml R (S: 6KB, LM: Mon Aug 12 13:42:23 EDT 2013)

Sprawdzanie ilości wolnego miejsca na karcie SD

Jeśli chcemy sprawdzić ilość wolnego miejsca na karcie SD lub w katalogu , możemy skorzystać z klasy StatFs. Niestety nie działa to dla katalogu root.

W pierwszej kolejności przyklejam na ekranie element typu TextView. Będzie na nim wyświetlona ilość miejsca na karcie.

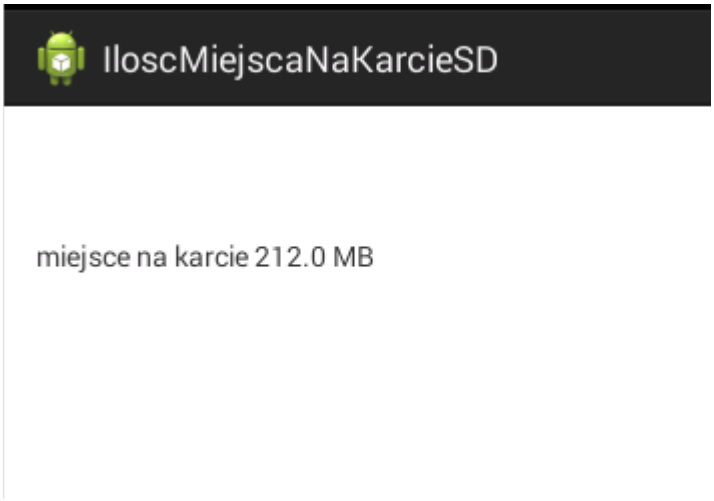


Teraz trzeba dodać nieco kodu który to obsłuży:

activity_main.xml | MainActivity.java

```
1  package com.example.iloscmiejscanakarciesd;
2
3  import android.os.Bundle;
8
9  public class MainActivity extends Activity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15         TextView t = (TextView) findViewById(R.id.textView1);
16         StatFs statFs = new StatFs("/etc/");
17         double miejsce = (statFs.getBlockSize()*statFs.getBlockCount());
18         double miejsceMB=Math.round(miejsce/1024/1024);
19         t.setText("miejsce na karcie "+miejsceMB+" MB");
20     }
21
22
23 }
24
```

Zasadniczo to co najważniejsze mieści się w liniach 16 i 17. W 16 z użyciem konstruktora klasy StatFs wskazuję katalog dla którego będzie sprawdzana ilość wolnego miejsca. W linii 17 wykorzystuję metody getBlockSize i getBlockCount do sprawdzenia ilości miejsca na karcie wyrażonej w bajtach. Ponieważ bardziej czytelne jest wyświetlenie tego w megabajtach, dzielę dwukrotnie uzyskaną wartość przez 1024 i zaokrąglam w linii 18. Efekt działania programu:



Lokalizacja i mapy

Wykorzystanie GPS

Jeśli nasz telefon / urządzenie z którego korzystamy posiada czujniki GPS, możemy wykorzystać je do sprawdzania naszej pozycji programowo. Możemy również skorzystać z darmowych map projektu OpenStreetMap (http://openstreetmap.org) i np. stworzyć własną nawigację.

Zaczniemy od utworzenia nowego projektu i wyedytowania pliku AndroidManifest.xml tego projektu. Musimy dodać prośbę o uprawnienia do korzystania z czujników GPS. Bez takiej autoryzacji ze strony użytkownika, nasz program nie będzie działał. Dodajemy linie:

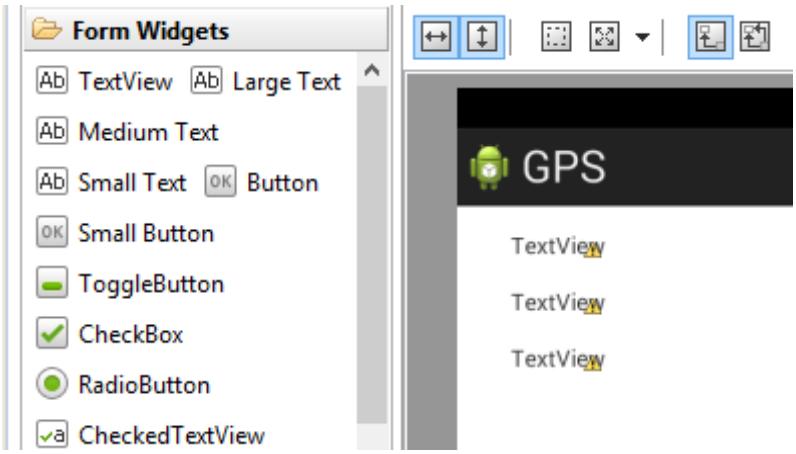
```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Tuż pod zamknięciem elementu <application>. Podglądzik:

```
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.gps.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

</manifest>
```

Mamy już możliwość korzystania z czujników GPS. Czas przejść do właściwej wersji kodu. Zaczniemy jak zawsze od wersji najprostszej tj. po uruchomieniu programu ma się na ekranie wyświetlić nasze położenie tzn. długość i szerokość geograficzna.

Przechodzimy teraz do naszej głównej aktywności. Przyklejam na ekranie trzy elementy klasy TextView. Na długość i szerokość geograficzną, oraz na nazwę wybranego dostawcy (wyjaśni się to nieco później).



Od razu w metodzie onCreate podpinam do nich uchwyty. Będę się przecież do tych elementów odwoływał, więc będę też potrzebował elementów reprezentujących te komponenty.

Do określania lokalizacji będzie nam służył obiekt klasy LocationManager. Definiuję go podobnie jak komponenty klasy TextView – jako pole klasy a nie jako zmienna prywatna w metodzie onCreate. Robię tak, bo być może będę się do tego obiektu owoływał z innych metod na dalszych etapach tworzenia programu.

```
1  package com.example.gps;
2
3  import android.os.Bundle;
7
8  public class MainActivity extends Activity {
9
10     TextView t1;
11     TextView t2;
12     TextView t3;
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18         t1=(TextView) findViewById(R.id.textView1);
19         t2=(TextView) findViewById(R.id.textView2);
20         t3=(TextView) findViewById(R.id.textView3);
21
22     }
23 }
```

Obiekt kr klasy Criteria (linia 29) służy do wyszukiwania najlepszego dostawcy informacji o położeniu GPS. Mogę np. poprzez te kryteria określić że interesują mnie wyłącznie bezpłatni dostawcy (metoda setCostAllowed klasy Criteria), czy poziom zapotrzebowania energetycznego wymaganego dla danego sposobu określania lokalizacji (setPowerRequirement). Chcąc zastosować któryś z wymienionych warunków, po utworzeniu obiektu klasy Criteria, uruchamiamy dla tego obiektu wybrane metody. Sam obiekt klasy Criteria jako element warunkujący wybór dostawcy stosujemy później przy pobieraniu nazwy najlepszego dostawcy. W linii 30 inicjalizuję obiekt przy uzyciu usługi zwracanej przez metodę getSystemService (metoda dziedziczona po klasie Activity). Z użyciem tej metody możemy podpinać się do najprzeróżniejszych usług systemowych : np. korzystać z różnych czujników (temperatury, ciśnienia etc), korzystać z usług lokalizacyjnych, pobierać dane z internetu, drukować, korzystać z portu USB i wiele innych. Ponieważ będziemy korzystać z usług lokalizacyjnych, jako parametr metody getSystemService podaję LOCATION_SERVICE. Linia 31 to pobranie nazwy najlepszego dostawcy informacji o położeniu (może to być np. GPS, położenie określone wg triangulacji, lub sieć WIFI). Zwrócony może być tylko dostawca dla którego wywołująca aktywność ma niezbędne pozwolenia (chodzi o wpisy w AndroidManifest.xml). Jeśli kilku dostawców spełnia założone kryteria, to zwrócona zostanie nazwa tego który charakteryzuje się najwyższą dokładnością położenia. Przykładowo, jeśli możemy określić nasze położenie na podstawie GPS, lub na podstawie triangulacji, przy czym GPS określi to z dokładnością do 10 metrów a triangulacja z dokładnością do 1 kilometra, to zostanie zwrócona nazwa dostawcy gps. Pierwszy parametr metody getBestProvider to nasze kryteria, drugi określa czy zwracani mają być tylko aktywni dostawcy (w tym przypadku oczywiście tylko tacy nas interesują).

```
11 public class MainActivity extends Activity {
12
13     TextView t1;
14     TextView t2;
15     TextView t3;
16     LocationManager lm;
17     Criteria kr;
18     Location loc;
19     String najlepszyDostawca;
20
21     @Override
22     protected void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.activity_main);
25         t1=(TextView) findViewById(R.id.textView1);
26         t2=(TextView) findViewById(R.id.textView2);
27         t3=(TextView) findViewById(R.id.textView3);
28
29         kr=new Criteria();
30         lm=(LocationManager) getSystemService(LOCATION_SERVICE);
31         najlepszyDostawca=lm.getBestProvider(kr, true);
32         loc=lm.getLastKnownLocation(najlepszyDostawca);
33
34         t1.setText("najlepszy dostawca: "+najlepszyDostawca);
35         t2.setText("długość geograficzna: "+loc.getLongitude());
36         t3.setText("szerokość geograficzna: "+loc.getLatitude());
37
38     }
39 }
```

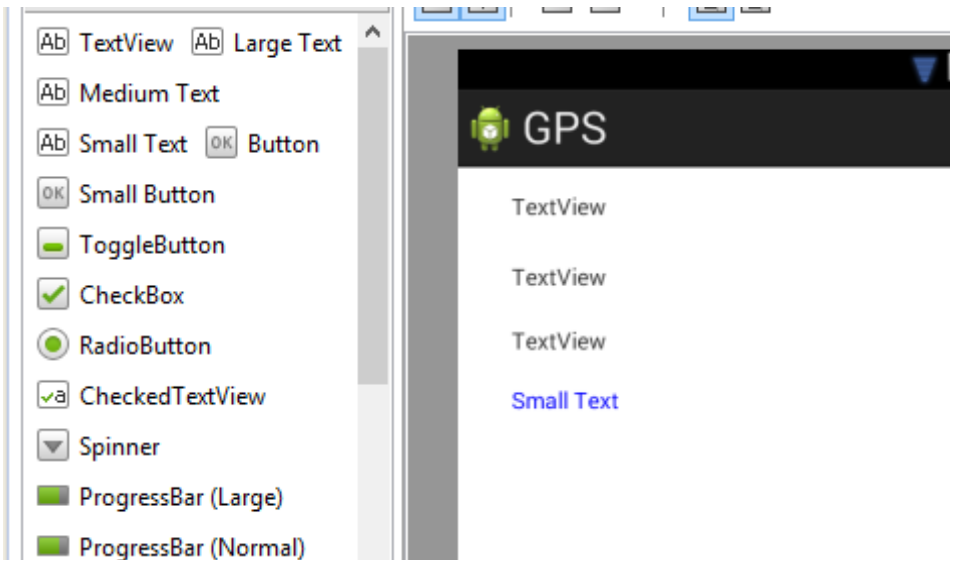
Obiekt klasy Location będzie reprezentował naszą lokalizację w przestrzeni, będziemy z niego później pobierać np. naszą długość i szerokość geograficzną. Najpierw jednak musimy określić nasze położenie :) Robimy to w linii 32 przy użyciu metody getLastKnownLocation obiektu klasy LocationManager. Jako parametr podajemy nazwę wybranego dostawcy informacji o położeniu. Linie 34-36 to po prostu wyświetlenie uzyskanych danych. Warte uwagi mogą tutaj być metody getLongitude i getLatitude zwracające długość i szerokość geograficzną. Zwracają liczby typu double, z dosyć dużą dokładnością. Będąc na obrzeżach Warszawy dostałem swoje położenie z dokładnością do 7 miejsca po przecinku stopnia geograficznego. Do tego prostego programu warto byłoby ewentualnie dorobić obsługę sytuacji w której nie byłoby w danym położeniu żadnego dostawcy, lub nie można byłoby określić położenia. Nie będziemy się tym tutaj jednak zajmować ponieważ obsługa nulla zwracanego z metody (tutaj z metod getBestProvider i getLastKnownLocation) to podstawy Javy. Chcąc testować rzeczy związane z GPS, najlepiej robić to na realnym urządzeniu a nie żadnych emulatorach. Niby na emulatorach jest możliwość ustawienia fikcyjnego położenia GPS, ale też nie zawsze to działa. U mnie po uruchomieniu programu na telefonie wyświetliło się:

najlepszy dostawca: network

szerokość geograficzna: 52.2996591

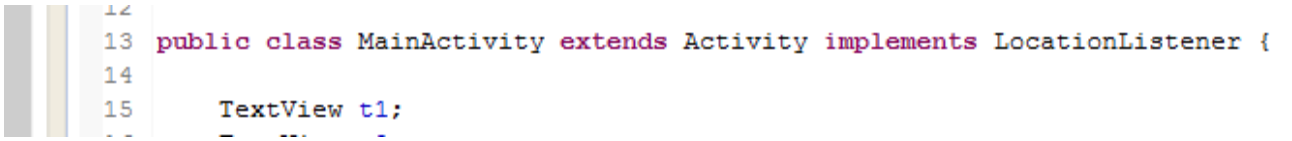
długość geograficzna: 20.9930961

Program działa, jednak jeśli zmienimy położenie, nasz program tego nie odnotuje. Warto byłoby wzbogacić program o funkcjonalność która by uwzględniała zmiany naszego położenia. Zacznę od dodania do ekranu elementu na którym będzie pojawiała się historia lokalizacji. Przykleiłem komponent klasy SmallText.



Żeby historia troszkę się odróżniała, zmieniłem tekstu w tym nowym komponencie poprzez edycję pliku layoutu i dopisani linijki : android:textColor=”#300FD”

Kodowi źródłowemu naszej aktywności będziemy przyglądać się poczynając od góry. W definicji dopisałem „implements LocationListener”, dzięki czemu będę miał dostępną metodę onLocationChanged która jest wywoływana za każdym razem kiedy zostanie wykryta nowa lokalizacja. Ponieważ jest to interfejs, będę musiał zaimplementować cztery wymagane przez niego metody. Tym się zajmiemy za chwilę.



W linii 18 widzimy nowe pole – t4. To jest obiekt reprezentujący komponent do wyświetlania historii. W liniach 25-28 widzimy nową metodę „odswiez”. Ponieważ będę musiał odświeżać swoje położenie przynajmniej dwukrotnie tj. przy uruchomieniu aplikacji i przy zmianie lokalizacji, postanowiłem kod odpowiedzialny za odświeżanie oddelegować do osobnej metody.

```
13 public class MainActivity extends Activity implements LocationListener {
14
15     TextView t1;
16     TextView t2;
17     TextView t3;
18     TextView t4;
19
20     LocationManager lm;
21     Criteria kr;
22     Location loc;
23     String najlepszyDostawca;
24
25     private void odswiez(){
26         najlepszyDostawca=lm.getBestProvider(kr, true);
27         loc=lm.getLastKnownLocation(najlepszyDostawca);
28     }
29
30     @Override
31     protected void onCreate(Bundle savedInstanceState) {
```

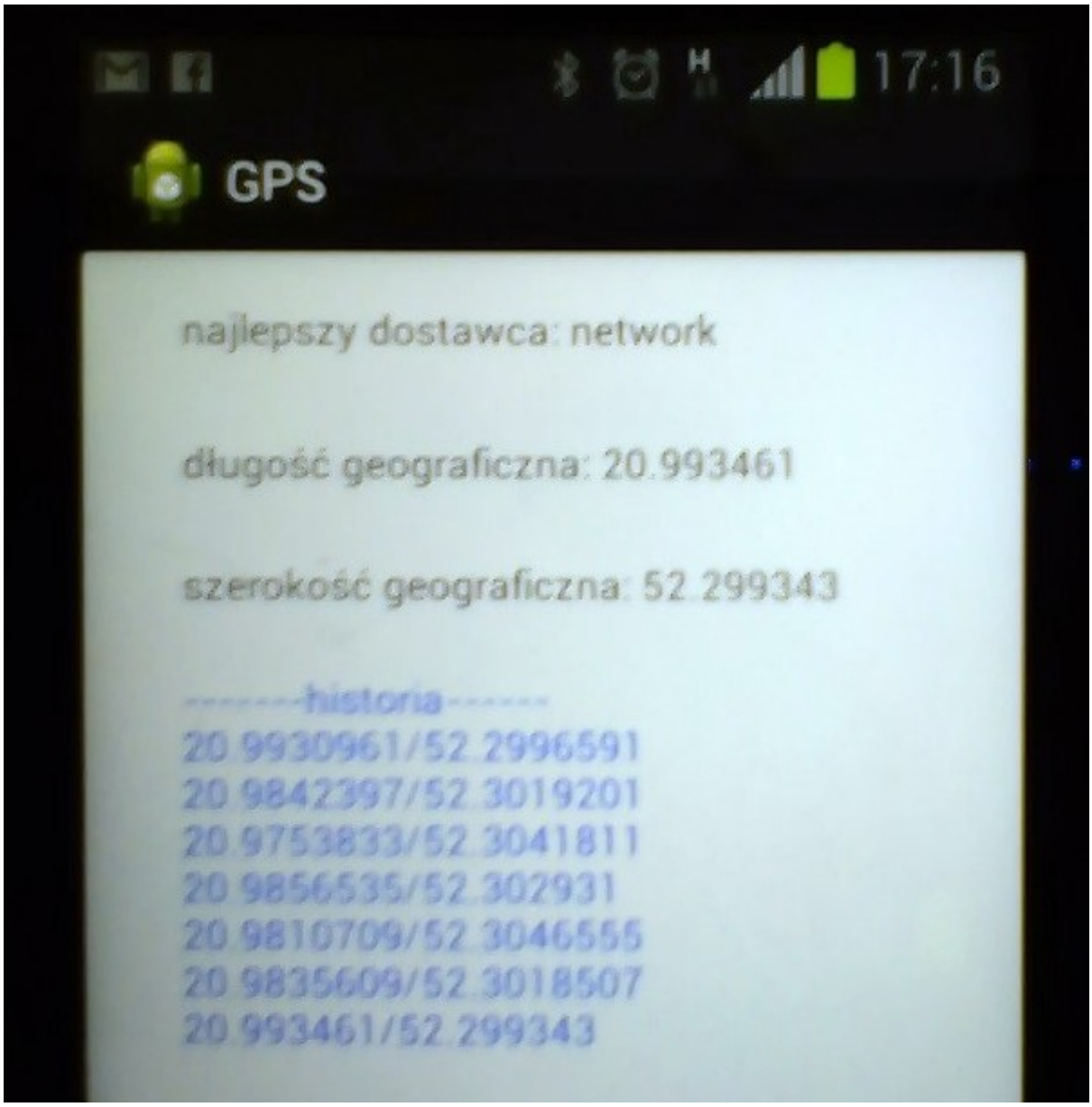
W linii 37 widzimy podpięcie do obiektu t4 referencji do naszego nowego komponentu (tego napisu który będzie robił za historię). Nic nadzwyczajnego, ale musimy o tym pamiętać. W linii 41 zamiast wywoływać osobno szukanie najlepszego dostawcy i pobieranie położenia mamy wywołanie metody która właśnie to robi i aktualizuje nasze obiekty. Linia 42 zawiera ustawienie własności odświeżania. Konfigurujemy tutaj co jaki czas i przy jakiej zmianie położenia system ma odświeżać lokalizację (czyli jak często ma być automatycznie wywoływana metoda onLocationChanged którą będziemy za chwilę implementować - w liniach 50-57). Pierwszy parametr to nazwa dostawcy z którego korzystamy, drugi to czas w milisekundach co ile ma być odświeżana lokalizacja, trzeci to co jaki dystans (wyrażony w metrach), a czwarty to obiekt implementujący interfejs LocationListener (tutaj akurat dotyczy to obiektu w którym się znajdujemy). Linie 43-46 to ustawienie domyślnych napisów, czyli tego co ma się pojawić zaraz po uruchomieniu programu. Tutaj tak jak wcześniej wyświetlamy dostawcę i współrzędne. W linii 46 ustawiam pierwszą i na razie jedyną linię historii.

```
30     @Override
31     protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         setContentView(R.layout.activity_main);
34         t1=(TextView) findViewById(R.id.textView1);
35         t2=(TextView) findViewById(R.id.textView2);
36         t3=(TextView) findViewById(R.id.textView3);
37         t4=(TextView) findViewById(R.id.textView4);
38
39         kr=new Criteria();
40         lm=(LocationManager) getSystemService(LOCATION_SERVICE);
41         odswiez();
42         lm.requestLocationUpdates(najlepszyDostawca, 1000, 1, this);
43         t1.setText("najlepszy dostawca: "+najlepszyDostawca);
44         t2.setText("długość geograficzna: "+loc.getLongitude());
45         t3.setText("szerokość geograficzna: "+loc.getLatitude());
46         t4.setText("-------historia------\n");
47     }
```

Poniżej mamy cztery metody które musiałem zaimplementować z racji implementacji interfejsu LocationListener. Trzy tymczasowo zostawiam w spokoju. Interesuje mnie teraz tylko metoda onLocationChanged. Jest ona wywoływana co interwał czasowy lub dystans określone (linia 42) w metodzie requestLocationUpdates. W ramach mojej implementacji metody onLocationChanged odświeżam położenie wywołując metodę odswiez, a następnie w liniach 52-54 wyświetlam na komponentach aktualne położenie. W linii 55 dodaję do historii kolejną linię.

```
50      public void onLocationChanged(Location location) {
51          odswiez();
52          t1.setText("najlepszy dostawca: "+najlepszyDostawca);
53          t2.setText("długość geograficzna: "+loc.getLongitude());
54          t3.setText("szerokość geograficzna: "+loc.getLatitude());
55          t4.setText(t4.getText()+""+loc.getLongitude()+"/"+loc.getLatitude()+"\n");
56
57      }
58
59      @Override
60      public void onProviderDisabled(String provider) {
61          // TODO Auto-generated method stub
62      }
63
64      @Override
65      public void onProviderEnabled(String provider) {
66          // TODO Auto-generated method stub
67
68      }
69
70      @Override
71      public void onStatusChanged(String provider, int status, Bundle extras) {
72          // TODO Auto-generated method stub
73
74      }
```

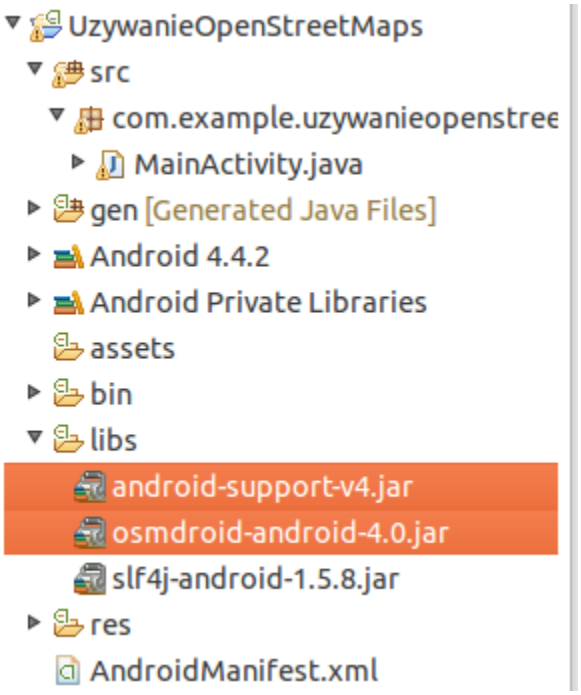
Byłem ciekaw na ile sprawnie t o działa, więc wgrałem program na telefon i poszedłem sprawdzić czy nie ma mnie za rogiem. Oto wyniki:



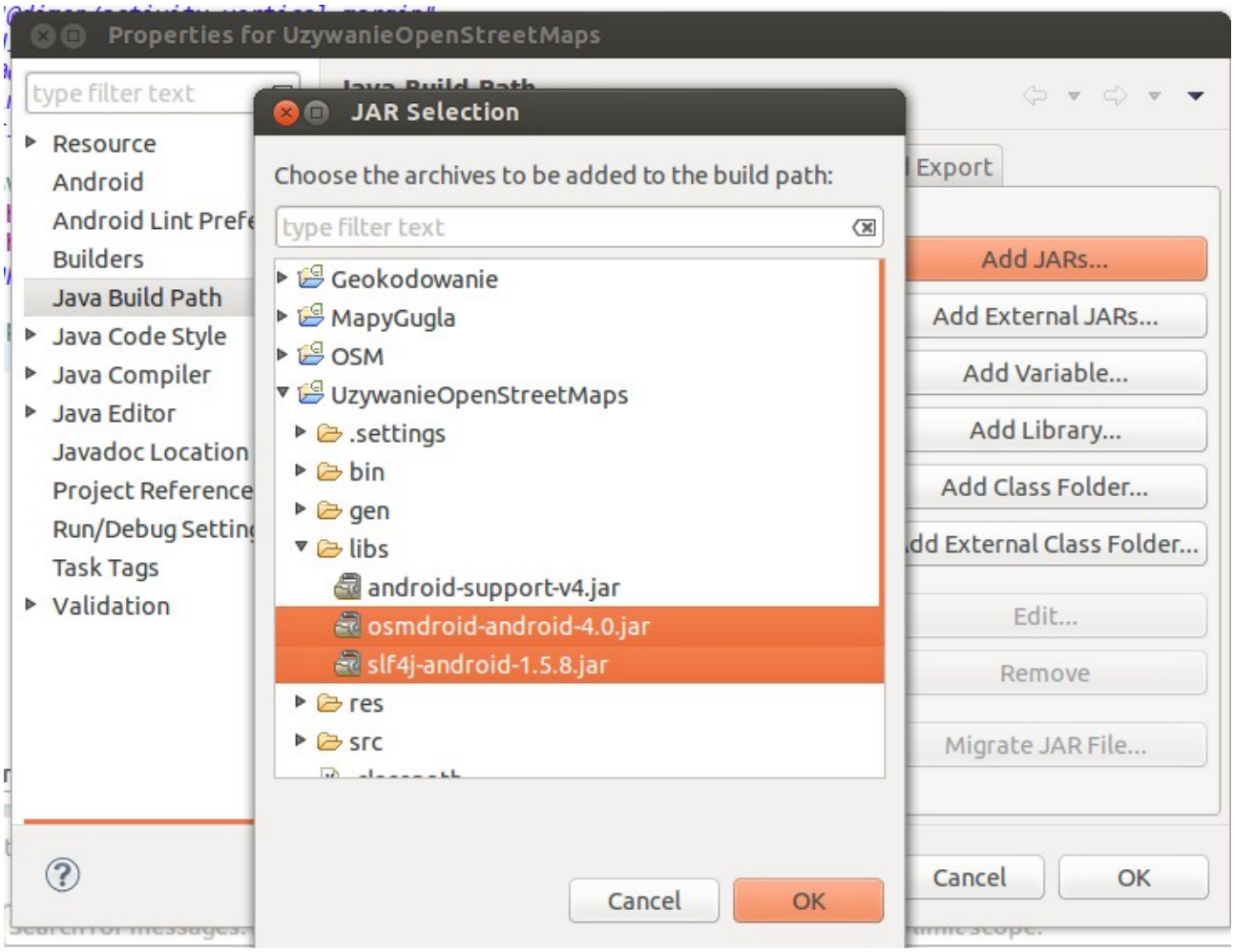
Używanie map OpenStreetMaps

Jedną z alternatyw dla stosowania w swoich aplikacjach map od Google jest wykorzystaniem map zod OpenStreetMaps. Są bezpłatne i działają całkiem sprawnie.

Zaczynamy od dodania do projektu niezbędnych blibliotek tj. osmdroid-android-4.0.jar i slf4j-android-1.5.8.jar (lub ich nowszych wersji jeśli takie się pojawią):



Musimy też dodać te biblioteki do Build Path projektu. Robimy to wchodząc do właściwości projektu, następnie przechodząc do „Java Build Path” i wybierając „Add Jars”:



Wybieramy te dwie nowo dodane biblioteki i zatwierdzamy. Czas wstawić obiekt mapy do naszego layoutu. Dodajemy widoczny na poniższej ilustracji element org.osmdroid.views.MapView:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <org.osmdroid.views.MapView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/mapview"
        >
    </org.osmdroid.views.MapView>
</RelativeLayout>
```

Ponieważ aplikacja będzie musiała pobierać dane (mapy) z internetu, musimy zadbać o odpowiednie pozwolenia. Do pliku manifestu dodajemy więc poniższe linie:

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

tuż za tagiem </application>

To są wystarczające uprawnienia niezbędne do działania map OpenStreetMaps. My w naszym przykładzie wykorzystamy naszą realną pozycję z GPS i wyświetlimy fragment mapy z okolicą naszej pozycji. Z tego powodu trzeba będzie dodać jeszcze dodatkowe uprawnienia:

```
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

W przypadku gdybyśmy nie korzystali z pozycji GPS, tych ostatnich uprawnień nie dodajemy. Ogólnie w suemie całość powinna wyglądać mniej więcej tak:

```
        </activity>
    </application>

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

</manifest>
```

Przechodzimy teraz do aktywności którą wykorzystamy do zaprezentowania mapy. Dodaję kilka pól do klasy. Pola w liniach 17-20 są związane z wykorzystaniem GPS, więc jeśli nie zamierzamy z niego korzystać, nie definiujemy ich. Potrzebne nam będą tylko obiekty z linii 23,24:

```
15 public class MainActivity extends Activity {
16
17     LocationManager locationManager;
18     String bestDostawca;
19     Criteria criteria;
20     Location location;
21
22
23     MapView mapView;
24     MapController mapController;
25
```

```
27    @Override
28    protected void onCreate(Bundle savedInstanceState) {
29        super.onCreate(savedInstanceState);
30        setContentView(R.layout.activity_main);
31
32
33
34
35        mapView=(MapView) findViewById(R.id.mapview);
36        mapView.setTileSource(TileSourceFactory.MAPQUESTOSM);
37        mapView.setBuiltInZoomControls(true);
38        mapView.setMultiTouchControls(true);
39        mapController=(MapController) mapView.getController();
40        mapController.setZoom(2);
41
42        criteria=new Criteria();
43        locationManager=(LocationManager) getSystemService(LOCATION_SERVICE);
44        bestDostawca=locationManager.getBestProvider(criteria, true);
45        location=locationManager.getLastKnownLocation(bestDostawca);
46        GeoPoint punkt =   new GeoPoint(location.getLatitude(),location.getLongitude());
47
48
49        mapController.setCenter(punkt);
50        mapView.invalidate();
51
52    }
```

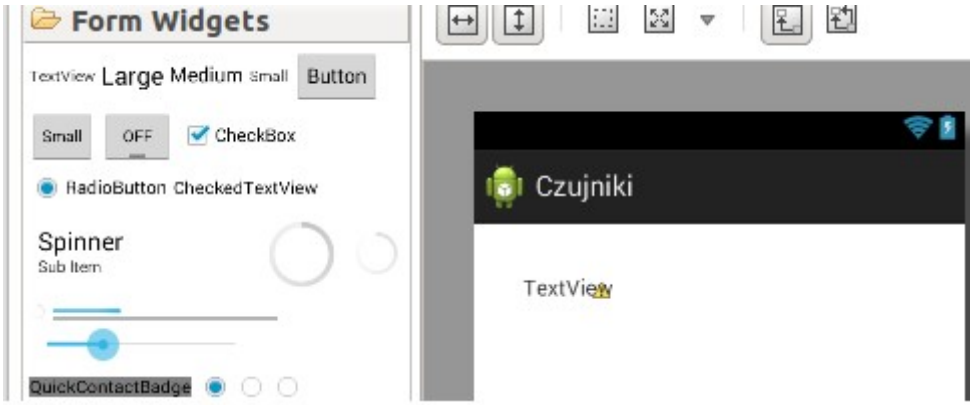
Kod z linii 42-45 jest związany z pobraniem pozycji GPS, omawiałem w jednym z poprzednich rozdziałów dokładnie sposób działania takiego kodu i nie będę się tutaj tym zajmował. Nas interesują zasadniczo linie 35-40 i 46-50. Linia 35 to element nie związany z samą mapą jako taką, służy do podpięcia referencji do komponentu mapy. W linii 36 wskazujemy źródło z którego pobieramy mapy. Linie 37 i 38 określają zachowanie mapy. Metoda setBuildInZoomControls (linia 37) ustawia, czy ma być możliwość przybliżania i oddalania na mapie (jeśli podamy true, będą dostępne przyciski + i – na mapie). Metoda setMultiTouchControls (linia 38) ustawia możliwość stosowania gestów dotykowych na ekranie – jak np. zbliżanie i oddalanie widoku dwoma palcami. W linii 40 określamy domyślne przybliżenie mapy. W linii 46 tworzymy obiekt klasy GeoPoint reprezentujący nasze położenie. W linii 49 centrujemy mapę na punkcie wskazanym przez obiekt klasy GeoPoint ustawiony w linii 46. Wywołanie metody invalidate w linii 50 to odświeżenie widoku.


UzywanieOpenStreetMaps

Android też czuje – czujniki

Sprawdzanie jakie czujniki mamy dostępne

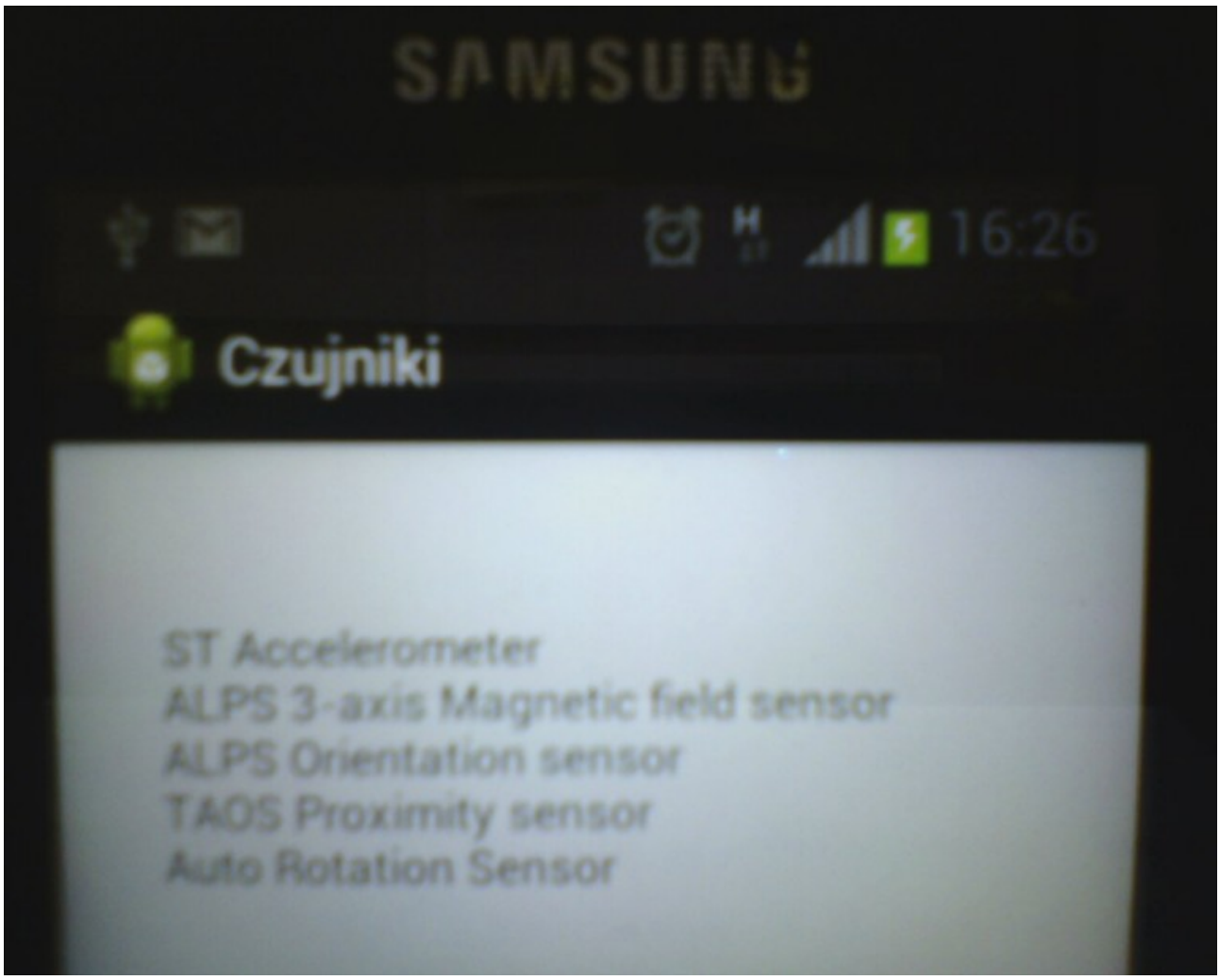
Zanim zacznę korzystać z czujników jakiegoś urządzenia, chciałbym wiedzieć jakie w ogóle są dostępne. W pierwszej kolejności przyklejam na ekranie komponent typu TextView na którym wyświetlę wyniki:



W metodzie onCreate aktywności głównej dodaję parę linii kodu. Kod który ma realizować zadanie znajduje się właśnie w tej metodzie, ponieważ chcę aby wyniki wyświetliły się od razu po uruchomieniu programu. Linie 21,22 to podpięcie referencji do komponentu TextView i wyczyszczenie jego zawartości (napisu). Klasa SensorManager do której obiektu odwołuję się w linii 25 (sama deklaracja obiektu jest w linii 14) daje nam dostęp do czujników wbudowanych w urządzenie. Pobranie instancji tej klasy następuje poprzez wywołanie metody getSystemService z parametrem SENSOR_SERVICE. Aby odczytać nazwy czujników które są dostępne w urządzeniu, wykorzystuję metodę getSensorList klasy SensorManager (linia 26). Metoda ta zwraca dostępne (czyli znajdujące się na danym urządzeniu) czujniki wskazanego typu. Tutaj podaję jako typ Sensor.TYPE_ALL, dzięki czemu w efekcie dostaję czujniki wszystkich typów. Mógłbym w tym miejscu wstawić np. Sensor.TYPE_GYROSCOPE i dostałbym listę wszystkich dostępnych w urządzeniu żyroskopów. Linie 27-29 to już iteracja po zwróconych obiektach i dodawanie kolejnych linii z nazwami kolejnych czujników do kompontentu tekstowego.

```
 1 package com.example.czujniki;
 2
 3 import java.util.List;
11
12 public class MainActivity extends Activity {
13
14     SensorManager manager;
15     TextView tv1;
16
17     @Override
18     protected void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.activity_main);
21         tv1=(TextView)findViewById(R.id.textView1);
22         tv1.setText("");
23
24
25         manager=(SensorManager)getSystemService(SENSOR_SERVICE);
26         List<Sensor> sensory = manager.getSensorList(Sensor.TYPE_ALL);
27             for(int x=0;x<sensory.size();x++){
28                 tv1.setText(tv1.getText()+"\n"+sensory.get(x).getName());
29             }
30
31     }
32
33
34 }
35
```

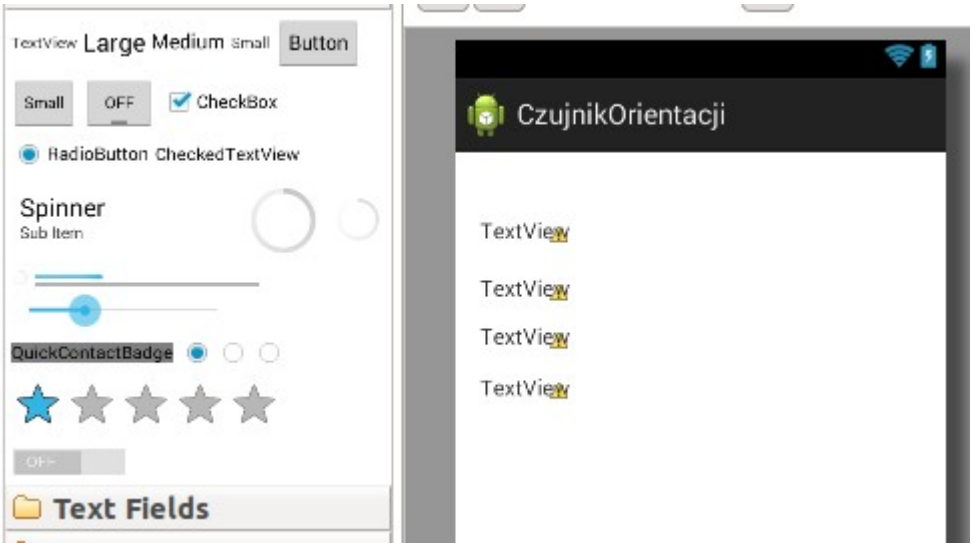
Efekt na Samsung ACE3:


SAMSUNG
16:26
Czujniki
ST Accelerometer
ALPS 3-axis Magnetic field sensor
ALPS Orientation sensor
TAOS Proximity sensor
Auto Rotation Sensor

Czujnik orientacji (poziomica + kompas)

Android umożliwia korzystanie z wbudowanego w urządzenie czujnika orientacji. Dzięki temu czujnikowi jesteśmy w stanie określić pozycję telefonu – tj. pochylenie góra-dół i lewa-prawa. Można dzięki temu zaprogramować np. elektroniczną poziomicę. Ten sam czujnik orientacji pozwala też na określenie azymutu na który zwrócony jest telefon. W tym przykładzie wykorzystamy wszystkie te możliwości.

Zaczynamy od przyklejenia na ekranie czterech elementów klasy TextView:



Do przyklejonych elementów musimy podpiąć referencje aby z nich korzystać, tutaj niczego zaskakującego nie ma. Pojawia się za to implementacja interfejsu SensorEventListener. O co tu chodzi? Nasz czujnik będzie co chwila podawał nowe odczyty, aby wyświetlać te zmiany musimy zaimplementować ten interfejs. W związku z tym, będziemy musieli przesłonić metody onSensorChanged i onAccuracyChanged. O tym za momencik.

```
15 public class MainActivity extends Activity implements SensorEventListener{
16
17     TextView tv1;
18     TextView tv2;
19     TextView tv3;
20     TextView tv4;
21
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_main);
27         tv1=(TextView) findViewById(R.id.textView1);
28         tv2=(TextView) findViewById(R.id.textView2);
29         tv3=(TextView) findViewById(R.id.textView3);
30         tv4=(TextView) findViewById(R.id.textView4);
```

Zanim zaczniemy korzystać z naszego czujnika, musimy zarejestrować wybraną klasę (musi ona implementować interfejs SensorEventListener) jako odbiornik dla czujnika. Ja to zrobiłem w metodzie onCreate mojej aktywności, ponieważ ta jest uruchamiana w momencie startu programu. Jako odbiornik zarejestrowałem aktywność, ponieważ to w niej chcę reagować na zmiany odczytów (a zadbałem o to by ta aktywność implementowała interfejs SensorEventListener).

```
32
33          SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
34          sm.registerListener(this,sm.getDefaultSensor(Sensor.TYPE_ORIENTATION),0,null);
35
```

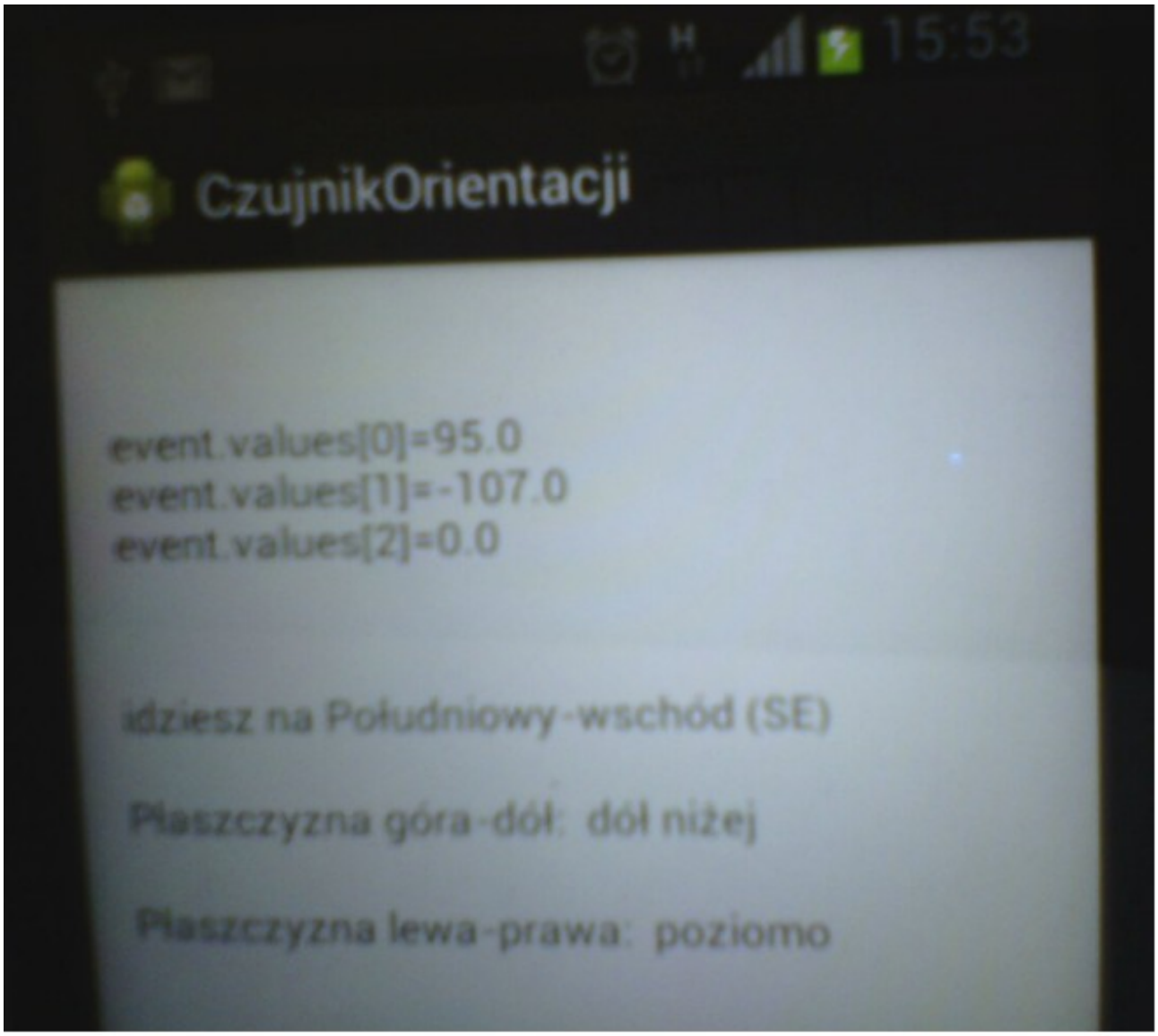
W związku z implementacją interfejsu SensorEventListener muszę zaimplementować też metody onAccuracyChanged i onSensorChanged. Pierwsza jest wywoływana kiedy dokładność czujnika ulega zmianie (tutaj raczej nam się to nie przyda), druga jest wywoływana automatycznie kiedy odczyty z czujników ulegną zmianie (ta już nas interesuje). Do tej metody przekazywane są odczyty z czujnika. Pod indeksem 0 (event.values[0]) znajdziemy azymut na jaki zwrócona jest góra telefonu, pod indeksem 1 (event.values[1]) wychylenie w płaszczyźnie góra-dół, pod indeksem 2 (event.values[2]) wychylenie lewa-prawa. Wartości wszystkich trzech wypisuję na pierwszym komponencie.

```
38      @Override
39      public void onAccuracyChanged(Sensor arg0, int arg1) {
40          // TODO Auto-generated method stub
41
42      }
43
44      @Override
45      public void onSensorChanged(SensorEvent event) {
46
47          tv1.setText("event.values[0]="+event.values[0]+"\n" +
48                  "event.values[1]="+event.values[1]+"\n"+
49                  "event.values[2]="+event.values[2]+"\n");
```

Korzystając z tych odczytów wyświetlam na pozostałych komponentach stosowne komentarze. Mają one sprawić, że odczyty będą bardziej czytelne. Nie jest to niezbędny element, czysta estetyka. Całą metodę onSensorChanged widać na poniższej ilustracji.

```
44      @Override
45      public void onSensorChanged(SensorEvent event) {
46
47          tv1.setText("event.values[0]="+event.values[0]+"\n" +
48                  "event.values[1]="+event.values[1]+"\n"+
49                  "event.values[2]="+event.values[2]+"\n");
50
51          String kierunek="";
52          if(event.values[0]==0){
53              kierunek="Północ (N)";
54          }else if(event.values[0]<90){
55              kierunek="Północny-wschód (NE)";
56          }else if(event.values[0]==90){
57              kierunek="Wschód (E)";
58          }else if(event.values[0]<180){
59              kierunek="Południowy-wschód (SE)";
60          }else if(event.values[0]==180){
61              kierunek="Południe (S)";
62          }else if(event.values[0]<270){
63              kierunek="Południowy-zachód (SW)";
64          }else if(event.values[0]==270){
65              kierunek="Zachód (W)";
66          }else if(event.values[0]<360){
67              kierunek="Północny-zachód (NW)";
68          }
69          tv2.setText("idziesz na "+kierunek);
70
71          String pochylenie_gd="";
72          if(event.values[1]==0){
73              pochylenie_gd="poziomo";
74          }else if(event.values[1]>0){
75              pochylenie_gd="góra niżej";
76          }else{
77              pochylenie_gd="dół niżej";
78          }
79          tv3.setText("Płaszczyzna góra-dół:  "+pochylenie_gd);
80
81          String pochylenie_lp="";
82          if(event.values[2]==0){
83              pochylenie_lp="poziomo";
84          }else if(event.values[2]>0){
85              pochylenie_lp="lewa niżej";
86          }else{
87              pochylenie_lp="prawa niżej";
88          }
89          tv4.setText("Płaszczyzna lewa-prawa:  "+pochylenie_lp);
90      }
```

Efekt działania (Samsung ACE3):



Do programu warto byłoby dodać jeszcze obsługę sytuacji gdyby urządzenie nie posiadało czujnika, nie każdy telefon jest w niego wyposażony.

Wysyłanie i odbieranie SMSów

Wysyłanie pojedynczego SMSa

Wysyłanie SMSów w Androidzie jest bardziej niż proste. Do pliku manifestu dodajemy odpowiednie uprawnienie:

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

A następnie przechodzimy do kodu:

```
11 public class MainActivity extends Activity {
12
13     SmsManager smsManager=null;
14     String odbiorca="533cenzura:)";
15     String wiadomosc="Jestem wiadomością!";
16
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_main);
22         smsManager=SmsManager.getDefault();
23         smsManager.sendTextMessage(odbiorca,null, wiadomosc, null, null);
24     }
25
26
27
28
29
30 }
31
```

Obiekt klasy SmsManager to po prostu narzędzie służące do wysyłania SMS. Znaczenia pozostałych 2 zmiennych – odbiorca i wiadomosc nie trudno jest się domyślić. Przyjrzyjmy się teraz liniom 22 i 23. Metoda getDefault oddaje nam instancję klasy SmsManager. Metoda sendTextMessage służy do wysyłania SMSów składających się z nie więcej niż 160 znaków – tj. pojedynczy sms. Istnieje też metoda „sendMultipartTextMessage” która umożliwia wysłanie tekstu na który składa się więcej niż jeden SMS. Pierwszy parametr metody sendTextMessage to numer telefonu odbiorcy,trzeci to treść wiadomości. Musimy pamiętać, że urządzenie z którego wysyłamy wiadomości musi posiadać kartę SIM :)

Wysyłanie wieloczęściowego SMSa

Wysyłanie wieloczęściowego SMSa jest bardzo podobne do wysyłania pojedynczego, z tą różnicą że smsa trzeba po drodze podzielić na kilka fragmentów, ale nawet do tego mamy dedykowaną gotową metodę. Zacznijmy od dodania niezbędnego uprawnienia:

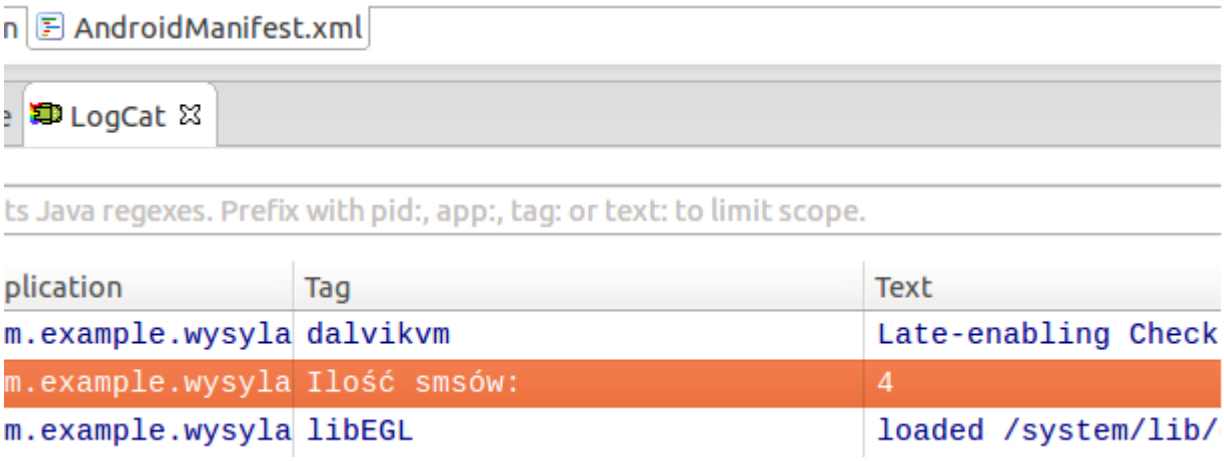
```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Potem przechodzimy do kodu:

```
13 public class MainActivity extends Activity {
14     SmsManager smsManager=null;
15     String odbiorca="530cenzurka :)";
16     ArrayList<String> fragmenty=null;
17     String wiadomosc="Mądrości Kubusia Puchatka:"+
18     "Kochać człowieka ,to znaczy mieć czas, nie spieszyć się , być obecnym dla niego."+
19     "Ojcem może być każdy ,ale trzeba  być   kimś  wyjątkowym ,aby  być TATĄ."+
20     "Miłość wszystko zwycięży ,nawet chrapanie"+
21     "Wyważona dieta ,to ciasteczka w obu rączkach.";
22
23
24     @Override
25     protected void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_main);
28         smsManager=SmsManager.getDefault();
29         fragmenty=smsManager.divideMessage(wiadomosc);
30         Log.d("Ilość smsów: ",fragmenty.size()+"");
31         smsManager.sendMultipartTextMessage(odbiorca, null, fragmenty, null, null);
32
33
34     }
35
```

Obiekt klasy SmsManager jest elementem wysyłającym smsy. Zmienna odbiorca to numer telefonu na który chcemy wysłać wiadomość. Lista elementów fragmenty, zdefiniowana jako pusta to pojemnik do którego wrzucona zostanie wiadomość SMS podzielona na fragmenty. Zmienna wiadomosc służy do tymczasowego przechowania wiadomości o długości większej niż 160 znaków. W metodzie onCreate , w linii 28 odbieram instancję klasy SmsManager. W linii 29 wykorzystuję metodę divideMessage której podaję długą wiadomość, a ta zwraca nam ją już podzieloną na fragmenty do pojemnika „fragmenty”. Samo wysłanie wiadomości to wywołanie metody sendMultipartTextMessage, której podaję numer telefonu odbiorcy i wiadomość podzieloną na fragmenty. Po drodze (w linii 30) wyświetlam ilość elementów jakie wyjdą z naszej wiadomości.

Wyszły mi cztery, co widać na poniższym obrazku:



Wiadomość przychodzi jako całość. Pamiętamy oczywiście o włożeniu karty SIM do urządzenia z którego chcemy wysyłać smsy, ponieważ wbrew pozorom nie jest to metoda na darmowe wysyłanie smsów ;)

Odbieranie SMSów

Jeśli chcemy napisać aplikację która będzie reagowała na nadejście SMS, musimy ją zarejestrować jako taki odbiornik i w ten sposób zdeklarować że potrafi obsługiwać takie akcje. W tym celu,gdzieś pomiędzy tagami <Application> w pliku manifestu musimy wstawić taki fragment:

```
<receiver android:name=".OdbiornikSMS" android:enabled="true">
        <intent-filter>
            <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
   </receiver>
```

Oczywiście w pierwszej linii podajemy nazwę swojej klasy która ma sesemesy odbierać. Wstawiając taki kawałek kodu do naszego pliku manifestu, deklarujemy że nasz program potrafi obsługiwać pewne operacje – np. w tym przypadku android.provider.Telephony.SMS_RECEIVED, ale równie dobrze może to być odbieranie nadchodzących połączeń czy obsługa wywołań otwarcia strony internetowej.

Skoro deklarujemy, że jesteśmy w stanie obsługiwać odbieranie sms, to musimy jeszcze mieć do tego uprawnienia ;) Pamiętamy więc o dodaniu takiej linii do pliku manifestu:

```
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

Jeszcze kodzik który to wszystko obsłuży:

```
12 public class OdbiornikSMS extends BroadcastReceiver {
13
14     @Override
15     public void onReceive(Context arg0, Intent arg1) {
16         Bundle bundle = arg1.getExtras();
17         SmsMessage[] msgs=null;
18         String wiadomosc="";
19         Object[] pdus =(Object[])bundle.get("pdus");
20         msgs=new SmsMessage[pdus.length];
21         for(int x=0;x<msgs.length;x++){
22             msgs[x]=SmsMessage.createFromPdu((byte[]) pdus[x]);
23             wiadomosc=msgs[x].getMessageBody();
24             Toast.makeText(arg0, wiadomosc, Toast.LENGTH_LONG).show();
25         }
26     }
27
28
29
30 }
```

Musimy zaimplementować interfejs BroadcastReceiver i posiadać metodę onReceive w której określimy co ma się zdarzyć przy odbieraniu sms. Nie musisz się martwić że od teraz nasza aplikacja „zasłoni” domyślną obsługę odbierania smsów. Platforma Android w momencie odebrania SMS wywoła wszystkie aplikacje zarejestrowane jako odbiorniki smsów :)

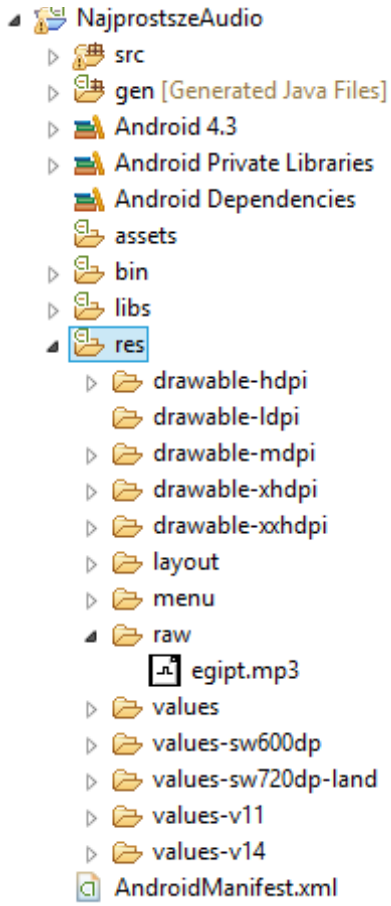
Taką funkcjonalność można fajnie wykorzystać do stworzenia systemu zarządzanego z dowolnego miejsca na ziemi poprzez SMS :)

Multimedia

Odwarzanie dźwięku

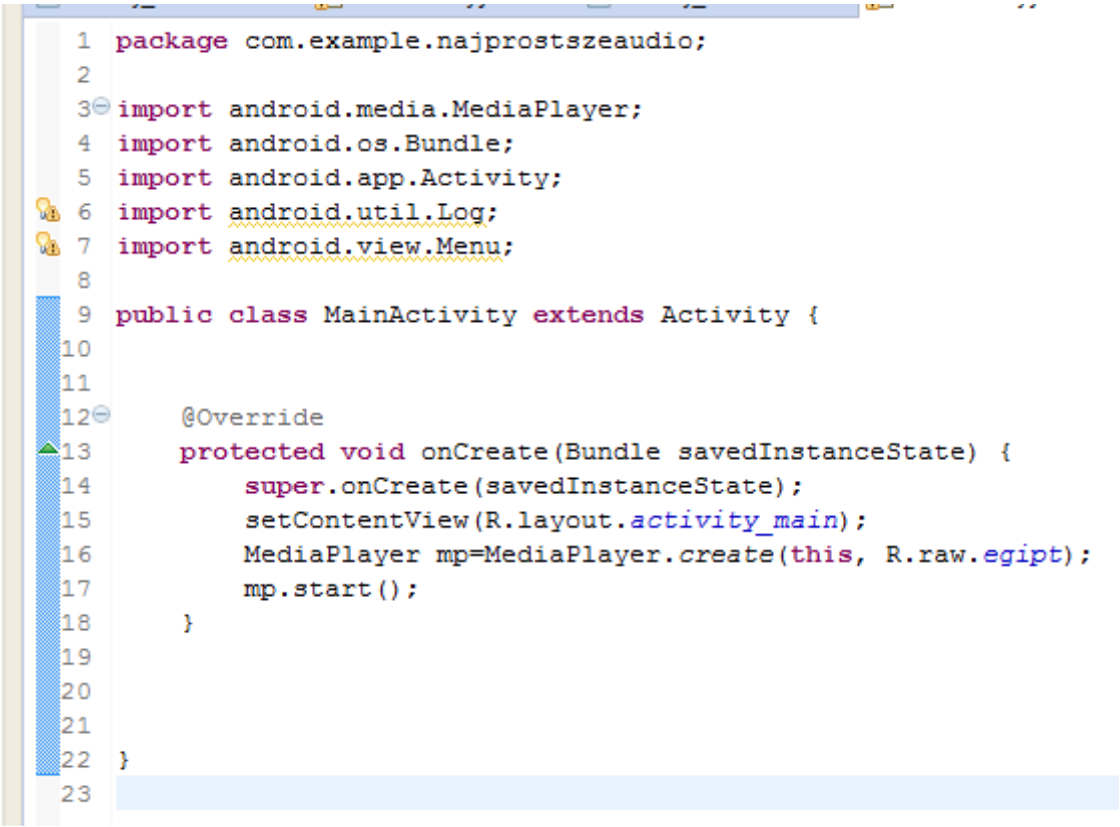
Zaczniemy od najprostszego przykładu. Program zaraz po uruchomieniu odtworzy plik MP3 który będzie znajdował się w ramach samego programu.

Zaczynamy od stworzenia projektu, oraz utworzenia podkatalogu o nazwie „raw” w katalogu „res” projektu. Następnie do podkatalogu raw wrzucamy utwór muzyczny w którymś z formatów: wav, aac, mp3, wma, amr, ogg, midi



Wszystko co wrzucimy do podkatalogu res, jest automagicznie rejestrowane jako zasób dostępny z poziomu kodu.

Teraz już tylko mały szlif kodu głównej aktywności:



```
1  package com.example.najprostszeaudio;
2
3  import android.media.MediaPlayer;
4  import android.os.Bundle;
5  import android.app.Activity;
6  import android.util.Log;
7  import android.view.Menu;
8
9  public class MainActivity extends Activity {
10
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16         MediaPlayer mp=MediaPlayer.create(this, R.raw.egipt);
17         mp.start();
18     }
19
20
21
22 }
23
```

Kluczowe są tutaj linie 19-21. Żartowałem :) Interesują nas linie 16 i 17. W linii 16 tworzymy obiekt klasy MediaPlayer która jako taka służy do odtwarzania multimediów. Podczas tworzenia przekazujemy jako argumenty metody create : kontekts i zasób. Jak widać nie podajemy rozszerzenia pliku.

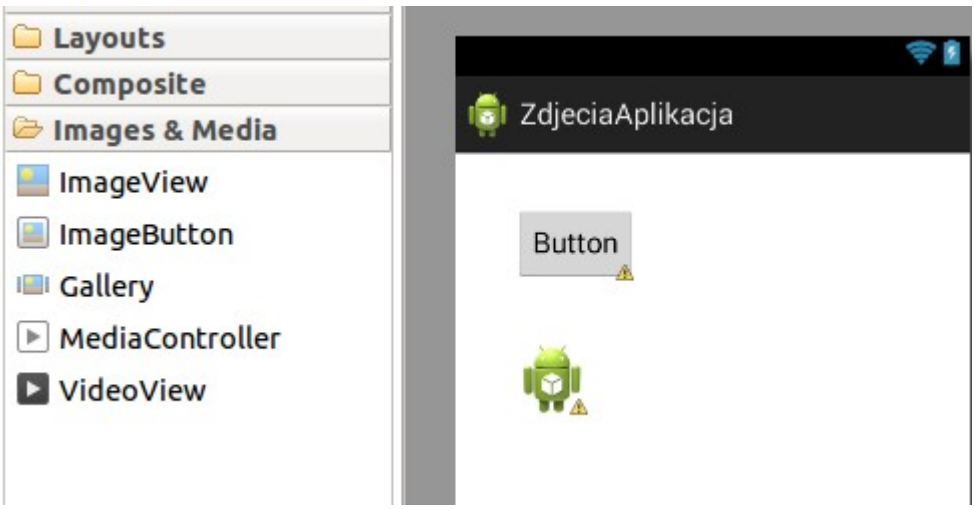
Linia 17 to uruchomienie odtwarzania zasobu któy wskazaliśmy w linii 16.

W sytuacji gdyby plik muzyczny który chcemy odtwarzać znajdował się poza aplikacją, np. na karcie SD nasz kod wyglądać by musiał tak:

```
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(”/sdcard/rammstein/raise_raise.mp3”);
mp.prepare();
mp.start();
```

Wykorzystanie aparatu fotograficznego do robienia zdjęć

Podobnie jak w przypadku czujników, do kamery również mamy gotowy interfejs. Zostały stworzone odpowiednie klasy i metody, wystarczy tylko z nich skorzystać. Zaczniemy od najprostszego programu robiącego zdjęcie i wyświetlające je na komponencie klasy ImageView. Przyklejam na głównej aktywności jeden przycisk i jeden imageView:



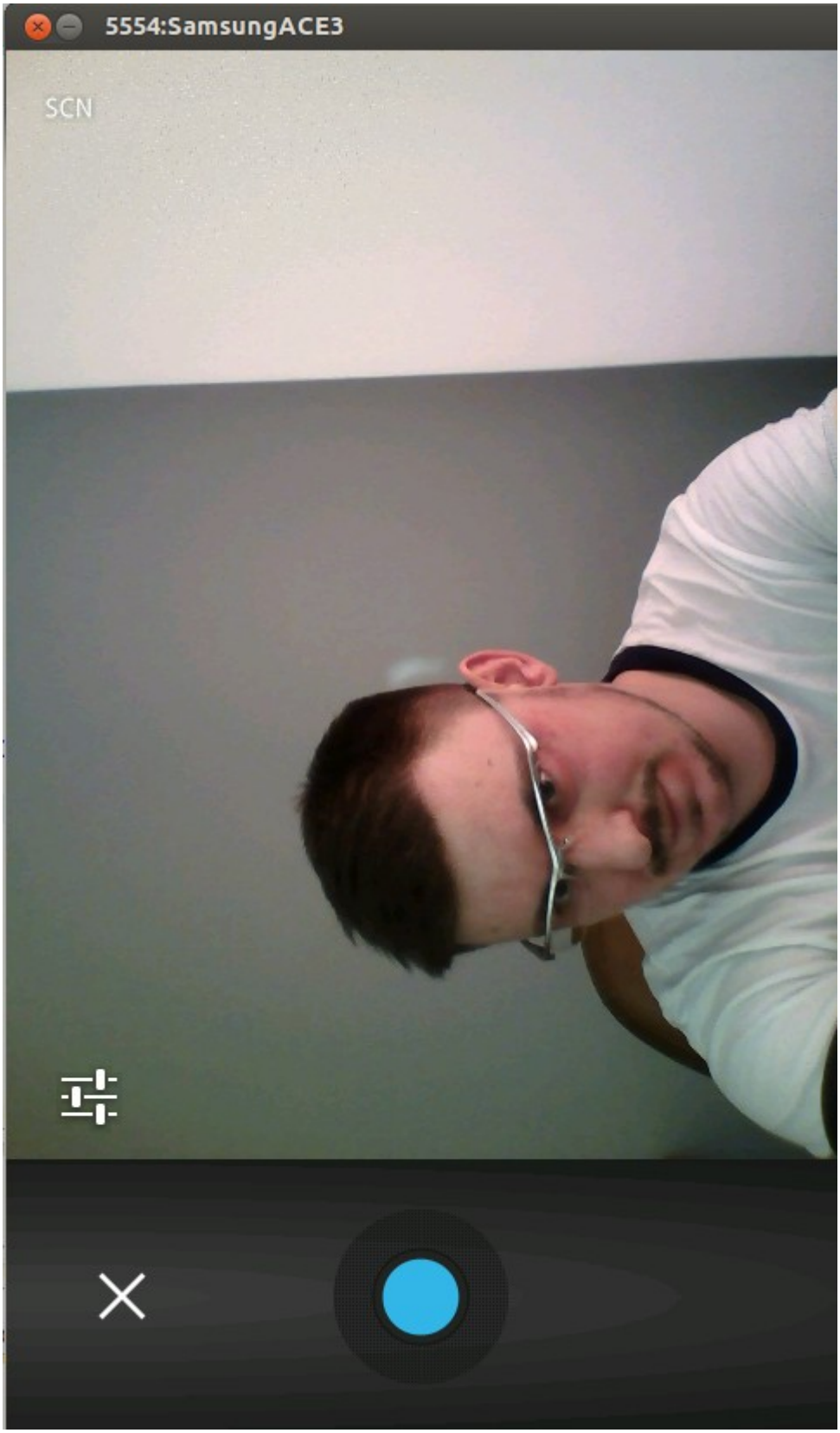
Po naciśnięciu tego przycisku zostanie uruchomiony wbudowany program do robienia zdjęć, a następnie odbierzemy zrobione zdjęcie i wyświetlimy je na ImageView. Poniżej wklejam kod aktywności związanej z tym ekranem. W liniach 26-34 oprogramowuję zdarzenie naciśnięcia przycisku. Część która jest związana z samym robieniem zdjęć jako takim to linie 30 i 31. Tworzę nową intencję której zadaniem jest wywołanie wbudowanego w Androida programu do robienia zdjęć (linia 30). W kolejnej linii uruchamiam tę intencję. Wywołanie takiej intencji wygląda nieco inaczej niż to do czego się przyzwyczailiśmy. Zazwyczaj wywoływaliśmy metodę startActivity(Intent i), a tutaj pojawia się startActivityForResult(Intent i, int x). Sama w sobie metoda zakłada że wywoływana intencja coś nam odda. Przy zakończeniu działania intencji i zwrocie danych wywoływana jest metoda onActivityResult (o tym za chwilę).Jako parametry podajemy intencję która ma zostać uruchomiona, oraz identyfikator wywołania. Poprzez drugi parametr podajemy wartość (parametr typu int) która później zostanie zwrócona po zrobieniu zdjęcia. Przydatne gdybyśmy np. wywołali kilka różnych intencji oczekując różnych zwracanych wyników. Każda z takich intencji wywoła metodę onActivityResult (która parę linii niżej implementujemy) w której możemy później po takim identyfikatorze dojść z której to intencji/wywołania intencji zostały zwrócone dane.

```
13
14 public class MainActivity extends Activity {
15
16     Button b;
17     ImageView i;
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23         b=(Button) findViewById(R.id.button1);
24         i=(ImageView) findViewById(R.id.imageView1);
25
26         OnClickListener l = new OnClickListener() {
27
28             @Override
29             public void onClick(View v) {
30                 Intent fotkejszyn = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
31                 startActivityForResult(fotkejszyn, 1);
32             }
33         };
34         b.setOnClickListener(l);
35     }
36
37     @Override
38     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
39         if (requestCode == 1 && resultCode == RESULT_OK) {
40             Bundle extras = data.getExtras();
41             Bitmap bitmap = (Bitmap) extras.get("data");
42             i.setImageBitmap(bitmap);
43         }
44     }
45 }
46
```

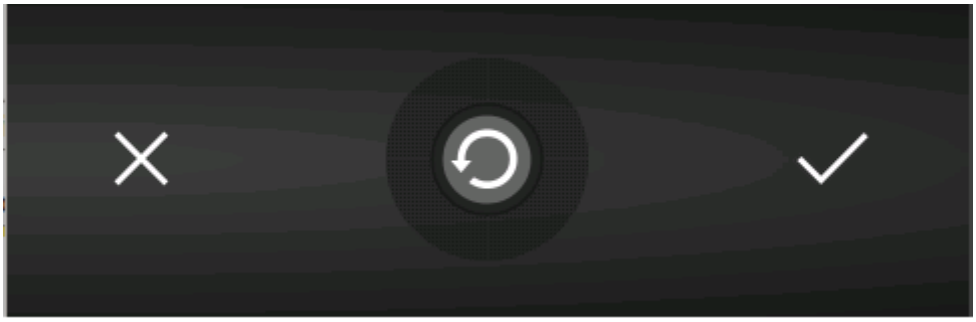
W liniach 38-44 implementuję metodę onActivityResult która jest automagicznie wywoływana w momencie kiedy nasza intencja wywołana przy użyciu metody startActivityForResult kończy swój przebieg. W tej właśnie metodzie odbieramy zrobione zdjęcie. Przyjrzyjmy się warunkom w linii 39. Sprawdzam parametr requestCode. To jest właśnie parametr który przekazałem przy wywoływaniu intencji (linia 31). Sprawdzam czy requestCode jest równe 1, by upewnić się że to jest wywołanie intencji związane z robieniem zdjęcia (mogłem przeciez wywołać wcześniej kilka różnych intencji i oczekiwać na wynik). Drugi parametr – resultCode jest zwracany przez wywoływaną aktywność – tutaj wbudowany program do robienia zdjęć. RESULT_OK jest zwracane w przypadku zatwierdzenia zdjęcia. Może też zostać zwrócone RESULT_CANCEL jeśli ktoś anuluje robienie zdjęcia. W liniach 40 i 41 odbieram zrobione zdjęcie. Aby odebrać fotkę muszę zawsze odwołać się w wiązce do identyfikatora „data”. Linia 42 to już po prostu ustawienie odebranego zdjęcia jako obrazek dla komponentu ImageView. Wykorzystanie kamery wymaga odpowiednich pozwoleń, dlatego dodajemy je do pliku manifestu (AndroidManifest.xml) aplikacji w ten sposób:

```
    </application>

     <uses-permission android:name="android.permission.CAMERA" />

</manifest>
```

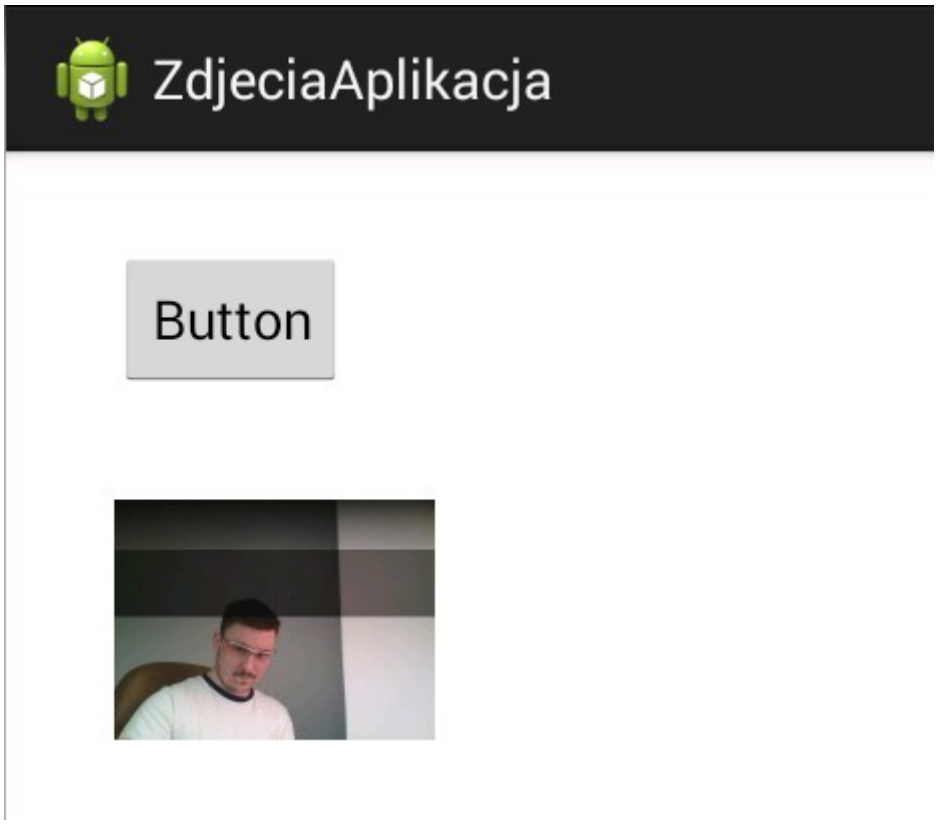
Po uruchomieniu i naciśnięciu przycisku pojawia nam się wbudowany programik do robienia fotografii:



Zapomniałem „dziubka” zrobić, to bym sobie na „fejsbuczka” wstawił „słit focię” ;) . Po zrobieniu zdjęcia, dolny panel powinien nam się zamienić na taki:



Mogę tutaj anulować operację, powtórzyć robienie zdjęcia, lub zatwierdzić. Zatwierdzamy, a po powrocie z tej aktywności nasz ekran wygląda tak:



Za wyjątkiem może zdjęcia, na swoim zobaczysz pewnie coś ładniejszego :p Przy użyciu tej metody możemy niestety robić tylko takie małe zdjęcia. Za chwilę zrobimy to troszkę inaczej i dostaniemy pełnowymiarowe zdjęcie.

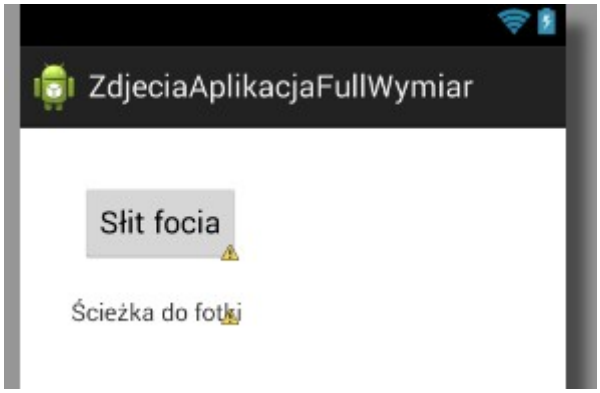
Stworzyłem nową aplikację. Aby zrobić pełnowymiarowe zdjęcie, Android będzie musiał zapisać je w pliku. Podobnie jak wcześniej dodaję prośbę o uprawnienia do kamery pliku manifestu, teraz jednak muszę też uzyskać pozwolenie na korzystanie z karty pamięci, lub jakiegoś innego folderu.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Android udostępnia publicznie dostępny katalog (dostępny dla wszystkich aplikacji). Aby się do niego dobrać możemy zastosować np. taką konstrukcję:

```
File katalog =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
```

Jak zawsze rozpoczynam pracę od przyklejenia komponentów:



Dałem jeden guzik który uruchomi intencję, oraz jeden TextView na którym znajdzie się ścieżka do zrobionej fotografii.

W aktywności, w metodzie onCreate podpinam referencję do komponentów i jako obsługę naciśnięcia przycisku wywołuję metodę obslugaGuzika().

```
23
24 public class MainActivity extends Activity {
25
26     Button b;
27     TextView t;
28
29
30     @Override
31     protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         setContentView(R.layout.activity_main);
34         b=(Button) findViewById(R.id.button1);
35         t=(TextView) findViewById(R.id.textView1);
36
37         OnClickListener l = new OnClickListener() {
38             @Override
39             public void onClick(View v) {
40                 obslugaGuzika();
41             }
42         };
43         b.setOnClickListener(l);
44     }
45
```

Sama metoda „obslugaGuzika” jest przestawiona poniżej:

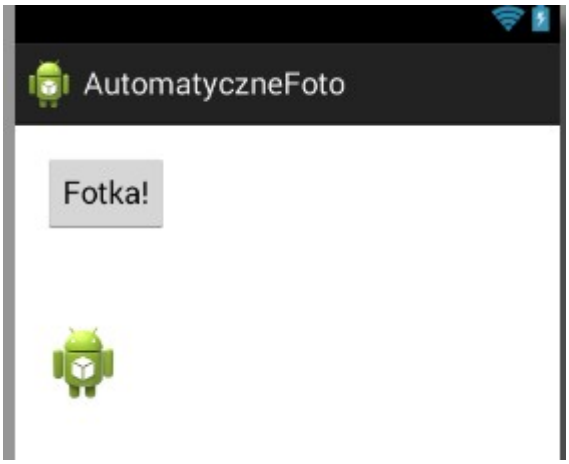
```
48     public void obslugaGuzika(){
49         Intent fotejszyn = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
50         File fotka=null;
51         Calendar c = Calendar.getInstance();
52         String nazwaPliku="fotka_"+c.get(Calendar.YEAR)+"_"+c.get(Calendar.MONTH)+"_"
53                         +c.get(Calendar.DAY_OF_MONTH)+"_"+c.get(Calendar.HOUR_OF_DAY)
54                         +"_"+c.get(Calendar.MINUTE)+"_"+c.get(Calendar.SECOND);
55         String rozszerzenie=".jpg";
56         String pelnaSciezka="";
57         File katalog = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
58         try {
59              fotka = File.createTempFile(nazwaPliku,rozszerzenie,katalog);
60         } catch (IOException e) {
61             e.printStackTrace();
62         }
63         fotejszyn.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(fotka));
64         startActivityForResult(fotejszyn, 1);
65         pelnaSciezka=katalog.getAbsolutePath()+"/"+nazwaPliku+rozszerzenie;
66         Log.d("LOG:",pelnaSciezka);
67         t.setText(pelnaSciezka);
68
69     }
```

Wyjaśniam rzeczy których nie omawiałem wcześniej. W linii 50 tworzę plik który wykorzystam za chwilę do zapisania fotografii na dysku. Linie 51-54 służą jedynie nazwaniu pliku tak, by nazwa zawierała datę i czas zrobienia zdjęcia. Jeśli chcesz, możesz je nazwać po prostu „fotka”. Linia 57 to pobranie referencji do kyatalogu przeznaczonego na obrazki. W linii 59 tworzę plik w którym znajdzie się zdjęcie. W linii 63 do intencji podaję dodatkowy parametr tj. ścieżkę do pliku w którym ma się znaleźć zdjęcie. Po zakończeniu cyklu intencji wyświetlam na textView ścieżkę do utworzonego pliku.

Efekt:

Może nam nie odpowiadać to, że musimy dodatkowo klikać na androidowej aktywności do robienia zdjęć. Chcielibyśmy by przykładowo program sam robił zdjęcie w reakcji na jakieś zdarzenie – np. pobudzenie czujnika ruchu. Konieczność dodatkowego kliknięcia przez człowieka w takim wypadku nie wchodzi w grę. Jest możliwość automatycznego robienia zdjęć, bez wywoływania dodatkowej systemowej aktywności, ale wymaga to więcej pracy.

Stworzyłem nowy projekt i przykleiłem na ekranie button i imageView:



Założenie jest takie, że zaraz po kliknięciu przycisku zostanie wykonana fotografia i wyświetlona na imageView. Muszę też zadbać o wymagane pozwolenie:

```
<uses-permission android:name="android.permission.CAMERA"/>
```

Przechodzimy teraz do kodu.

```
25 @SuppressLint("NewApi")
26 public class MainActivity extends Activity  implements PictureCallback {
27
28     private Camera camera;
29     private int cameraId = 0;
30     ImageView iv;
31     Button but;
32
33     @Override
34     protected void onCreate(Bundle savedInstanceState) {
35         super.onCreate(savedInstanceState);
36         setContentView(R.layout.activity_main);
37         iv=(ImageView) findViewById(R.id.imageView1);
38         but =(Button) findViewById(R.id.button1);
39
40
41         if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)) {
42             Log.d("Logi","Kamery niet!");
43         } else {
44             Log.d("Logi","Jest kamera... jakaś...");
45             cameraId =znajdzKamere();
46             if (cameraId < 0) {
47                 Log.d("Logi","Nie ma kamery z tyłu....");
48             } else {
49                 camera = Camera.open(cameraId);
50             }
51         }
```

Aby automatycznie odbierać robione zdjęcie, musimy w klasie która ma je odbierać implementować PictureCallback (linia 26). To sprawi, że w momencie zrobienia zdjęcia zostanie wywołana metoda onPictureTaken do której zostanie przekazane zdjęcie w postaci tablicy elementów typu byte... Do tej metody przejdziemy za kilka chwil. Linie 37,38 to podpięcie referencji do komponentów, nic nadzwyczajnego. Kod z linii 41-51 ma na celu sprawdzenie czy urządzenie w ogóle posiada kamerę (linia 41) oraz znalezienie jej identyfikatora i uruchomienie. W linii 45 do zmiennej liczbowej cameraId przypisuję identyfikator kamery zwracany jest przez metodę znajdzKamere. Zwraca ona wartość większą niż -1 jeśli zostanie znaleziona tylna kamera, jeśli nie zostanie znaleziona -metoda zwraca wartość -1. To mam nadzieję że wyjaśnia warunek if z linii 46. Kodem samej metody zajmiemy się za chwilę. Linia 49 to uruchomienie kamery dla identyfikatora zwróconego przez metodę znajdzKamere. Przejdźmy dalej:

```
50              }
51          }
52
53          OnClickListener l = new OnClickListener() {
54
55              @Override
56              public void onClick(View v) {
57                  onC(v);
58              }
59          };
60          but.setOnClickListener(l);
61      }
62
63
64      public void onC(View view) {
65          camera.takePicture(null, null,this);
66          Log.d("OK","OK");
67      }
68
69
70      private int znajdzKamere() {
71          int cameraId = -1;
72          for (int i = 0; i < Camera.getNumberOfCameras(); i++) {
73              CameraInfo info = new CameraInfo();
74              Camera.getCameraInfo(i, info);
75              if (info.facing == CameraInfo.CAMERA_FACING_BACK) {
76                  Log.d("Trololo", "Znaleziono kamerę tylną");
77                  cameraId = i;
78                  break;
79          }
80          }
81          return cameraId;
82      }
```

Linie 53-61 to podpięcie wywołania mojej metody onC pod kliknięcie przycisku. W liniach 64-67 znajdziemy metodę, która powoduje wykonanie zdjęcia przez kamerę. Istotnym z punktu widzenia funkcjonalności jest tutaj wywołanie metody takePicture dla obiektu klasy Camera dla którego wcześniej (linia 49) podpinaliśmy identyfikator fizycznego urządzenia. Linie 70 – 81 to metoda zwracająca identyfikator tylnej kamery. Wywoływaliśmy ją wcześniej (linia 45) w celu odnalezienia identyfikatora kamery. Zasada działania jest prosta. Jest pętla, której zawartość jest wykonywana tyle razy, ile jest kamer w systemie. Przy każdym „obrocie” sprawdzam czy właśnie sprawdzana kamera to ta odpowiadająca identyfikatorowi tylnej kamery (CameraInfo.CAMERA_FACING_BACK w linii 75). Gdybyśmy zechcieli robić zdjęcia z użyciem kamery przedniej, możemy wykorzytać identyfikator CAMERA_FACING_FRONT. Przejdźmy do dalszej części kodu:

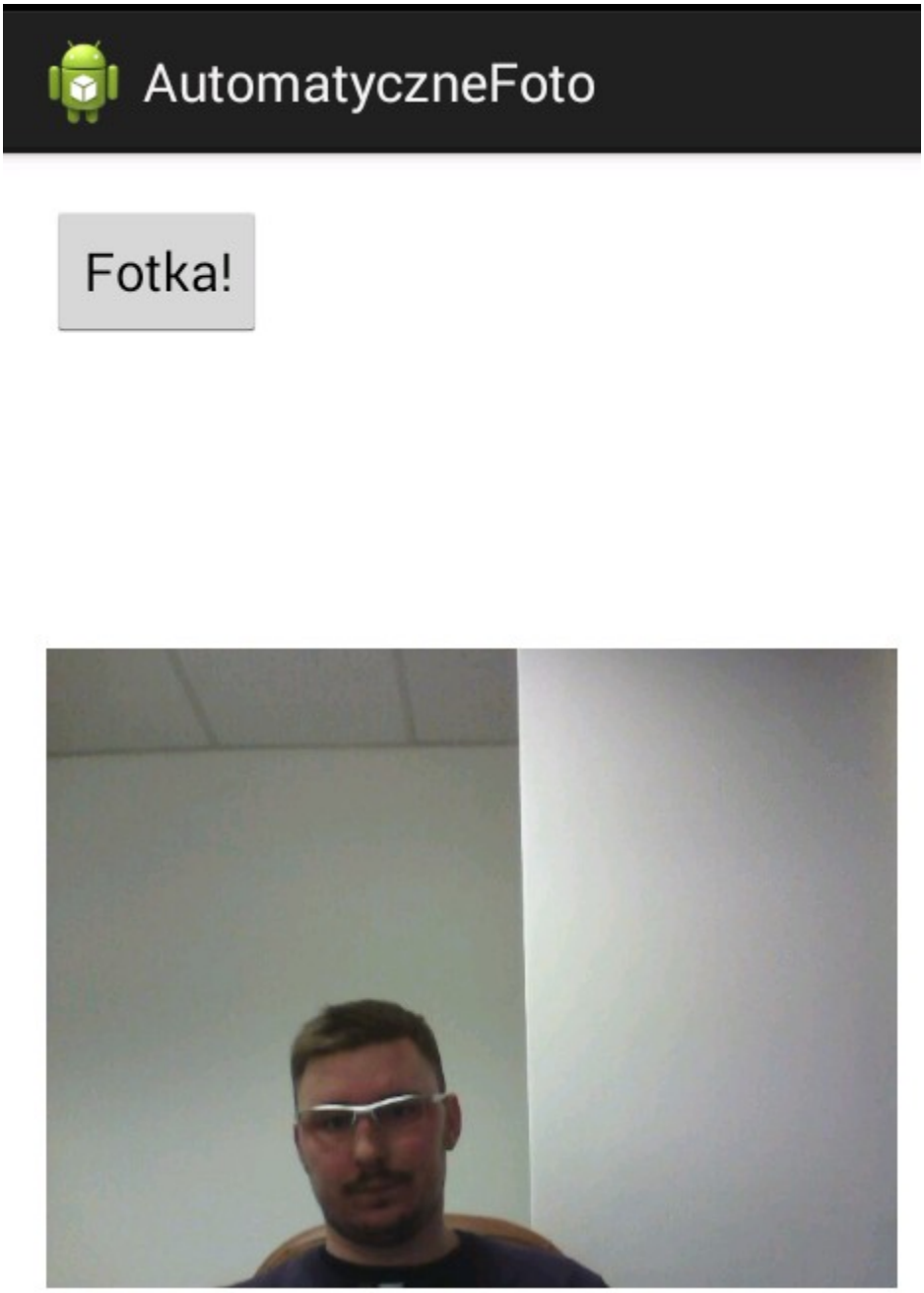
```
84      @Override
85      protected void onPause() {
86          if (camera != null) {
87              camera.release();
88              camera = null;
89          }
90          super.onPause();
91      }
92
93      @Override
94      public void onPictureTaken(byte[] arg0, Camera arg1) {
95          Bitmap bitmap = BitmapFactory.decodeByteArray(arg0 , 0, arg0.length);
96          iv.setImageBitmap(bitmap);
97      }
98
99  }
```

Linie 85-90 to odblokowanie kamery w przypadku np. „minimalizacji” programu, tak by inne programy również mogły korzystać z tego urządzenia. Przesłaniam tutaj domyślną metodę onPause klasy Activity. Linie 94-97 to implementacja metody onPictureTaken (konieczność wymaga z faktu że implementujemy interfejs PictureCallBack. Ta metoda jest automatycznie wywoływana w momencie zrobienia zdjęcia. Przy użyciu metody decodeByteArray klasy BitmapFactory (linia 95) konwertuję obraz w postaci tablicy elementów typu byte do bitmapy. Linia 96 to wyświetlenie fotki na komponencie klasy ImageView. Co z tego wszystkiego jest najważniejsze? Najważniejsze elementy:

- Implementujemy interfejs PictureCallback (linia 26)
- Podpięcie urządzenia pod obiekt klasy camera (linia 45)
- Wywołanie metody takePicture obiektu klasy Camera wtedy kiedy chcemy zrobić zdjęcie (linia 65)
- Implementacja automatycznie wywoływanej metody onPictureTaken i przetworzenie przekazanego nam przez argument zdjęcia (linia 94).

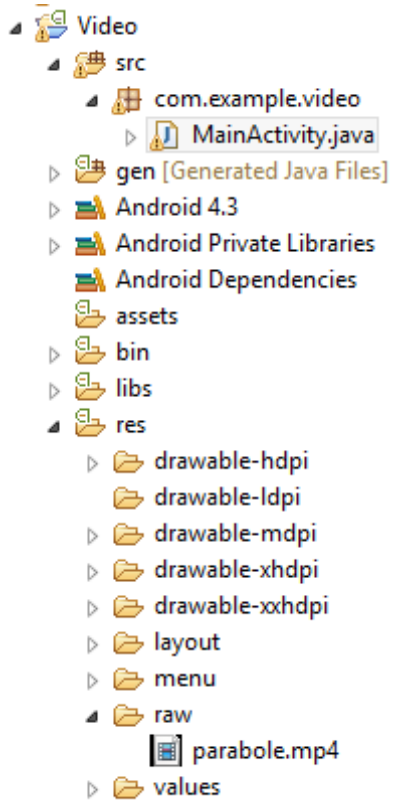
Te elementy to nasz „engine”, reszta to poboczny osprzęt.

Efekt po uruchomieniu i kliknięciu przycisku:

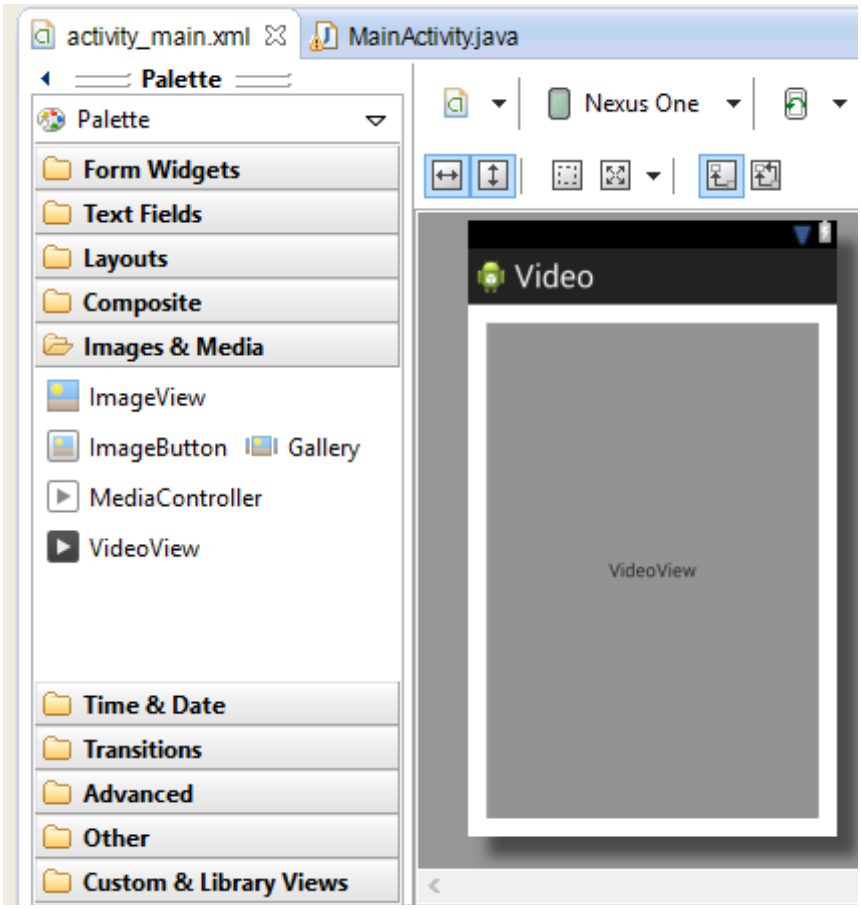


Odtwarzanie Video

Odtwarzać możemy formaty MP4 (MPEG-4), AVC, 3GP. Mamy gotowy komponent który wystarczy przykleić gdzieś na aktywności. Sam plik video może znajdować się już w aplikacji, ale może się też znajdować np. na karcie SD. W tym przykładzie uruchomię plik video znajdujący się w aplikacji. W katalogu „res” aplikacji dodaję podkatalog „raw”. Do niego wrzucam plik video. Wklejony przeze mnie filmik to 2 i pół minuty video z tańczącymi i śpiewającymi parabolami. Nie wrzucaj zbyt dużego pliku video, ponieważ przy każdej aktualizacji kodu i ponownym uruchamianiu programu , całość będzie uploadowana na emulator lub telefon. Gdyby plik był duży, trwałoby to niemiłosiernie długo.



W kolejnym kroku przyklejam na aktywności komponent VideoView:



Następnie wprowadzam kilka zmian w metodzie onCreate aktywności głównej:

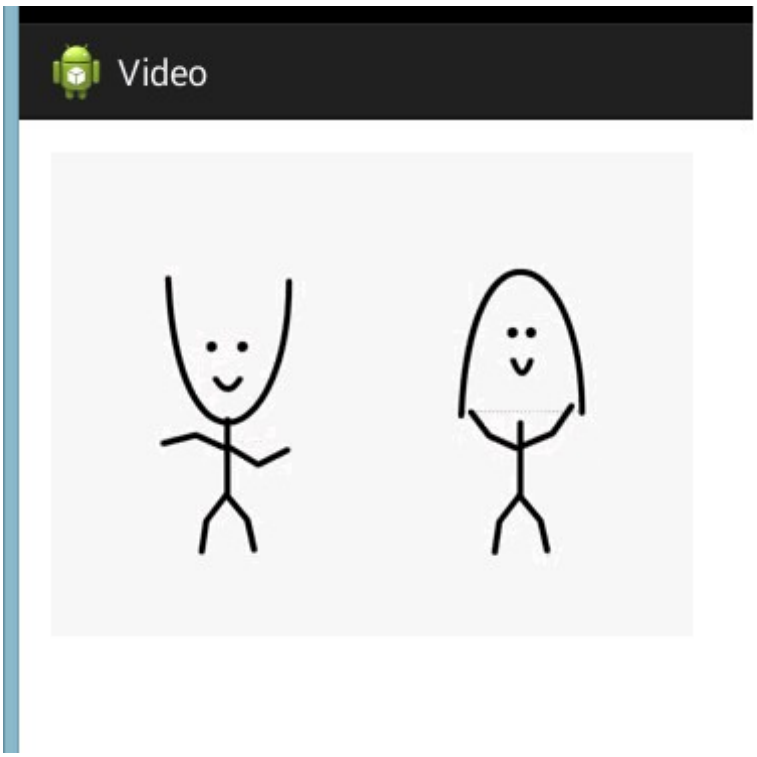
activity_main.xml | MainActivity.java

```
1  package com.example.video;
2
3  import android.net.Uri;
8
9  public class MainActivity extends Activity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15         VideoView vv = (VideoView) findViewById(R.id.videoView1);
16         Uri uri = Uri.parse("android.resource://com.example.video/raw/parabole");
17         vv.setVideoURI(uri);
18         //vv.setVideoPath("/sdcard/parabole.mp4");
19         vv.start();
20
21     }
22
23 }
```

Linia 15 do podpięcie referencji do komponentu VideoView. Muszę to zrobić by w jakikolwiek sposób móc się do tego elementu odnosić (np. wskazać mu plik video). Ponieważ chcę wskazać plik video zawarty w samej aplikacji, muszę stworzyć do niego referencję w postaci obiektu klasy Uri i przekazać go do metody setVideoURI komponentu klasy VideoView (linie 16 i 17). Średnio to wygodne moim zdaniem . Szkoda że nie ma możliwości przekazania po prostu ścieżki jako Stringa. Ścieżkę do pliku muszę podać z przedrostkiem „android:resource://”, pakietem w którym znajduje się dana aktywność, katalogiem raw i nazwą pliku video bez rozszerzenia.

W linii 18 widać wykomentowany alternatywny sposób. Podaję tutaj ścieżkę do pliku wideo znajdującego się poza aplikacją, z użyciem metody setVideoPath. Tutaj dla odmiany możemy podać ścieżkę jako zwykłego Stringa.

Po uruchomieniu aplikacji wszystko działa w spodziewany sposób. Jedyna drobna uwaga – filmik nie dopasowuje się do ustawionej wielkości komponentu VideoView, a wyświetla się w swoich oryginalnych proporcjach. W związku z powyższym w orientacji pionowej u mnie (Samsung ACE 3) przy propocji ekranu 480:800, wideo zajmuje jaką 1/3 ekranu.



Nagrywanie video

Podczas nagrywania, potrzebny nam będzie podgląd na nagrywany obraz. W tym celu w pliku layoutu aktywności wstawiam komponent SurfaceView. Będzie robił za mini ekran kamery. Wielkość i szerokość tego podglądu ustawiłem na na tyle małe aby zmieściło się na większości ekranów telefonów z Androidem (parametry layout_width i layout_height). Nie daję fill_parent, ponieważ chcę obok jeszcze umieścić guzik służący do rozpoczynania i przerywania nagrywania.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<LinearLayout
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<SurfaceView
android:id="@+id/videoview"
android:layout_width="360px"
android:layout_height="240px"/>
<Button
android:id="@+id/mybutton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Nagrywaj"
/>
</LinearLayout>
</LinearLayout>
```

Trzeba też zadbać o niezbędne uprawnienia aplikacji. Moja kamera będzie rejestrowała nie tylko obraz, ale i dźwięk – stąd też uprawnienia do nagrywania audio. Sam plik z nagraniem zostanie zapisany na zewnętrznej karcie pamięci – stąd uprawnienie WRITE_EXTERNAL_STORAGE. Jeśli zamierzasz zapisywać nagrany film w pamięci wbudowanej, tego uprawnienia nie potrzebujesz.

```
   <uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Przejdźmy teraz do kodu który będzie całość obsługiwać. Aby nasza aktywność mogła otrzymywać informacje o tym że powierzchnia podglądu została przygotowana (a będzie trzeba ją przygotować przed rozpoczęciem nagrywania), nasza aktywność musi implementować interfejs SurfaceHolder.CallBack. W związku z implementacją tego interfejsu musimy też posiadać zaimplementowaną metodę onSurfaceChanged (ale nie będziemy jej tutaj rozwijać). W linii 20 definiuję obiekt którego użyję do obsługi przycisku (dalej podpinam pod niego referencję do buttona zdefiniowanego w layoucie). W linii 21 tworzę obiekt klasy MediaRecorder. To klasa służąca do rejestracji obrazu i dźwięku. W linii 22 tworzę obiekt klasy SurfaceHolder. Z użyciem obiektu tej klasy możemy np. konfigurować wielkość i format nagrania, ale ogólnie służy do

„obserwacji” tego co się dzieje z obiektem surfaceView – czyli ekranem podglądu.

```
17
18 public class MainActivity extends Activity implements SurfaceHolder.Callback {
19
20     Button guzik;
21     MediaRecorder mediaRecorder;
22     SurfaceHolder surfaceHolder;
23     SurfaceView surfaceView;
24     boolean nagrywanie;
25     String sciezkaPliku = "/sdcard/nagranie2.mp4";
26
27     public void klikniecie() {
28         if (nagrywanie) {
29             mediaRecorder.stop();
30             mediaRecorder.release();
31             finish();
32             File f = new File(sciezkaPliku);
33             if (f.exists()) {
34                 Log.d("KAMERA", "Jest plik!");
35             } else {
36                 Log.d("KAMERA", "Nie ma plika");
37             }
38         } else {
39             mediaRecorder.start();
40             nagrywanie = true;
41             guzik.setText("Koniec");
42         }
43     }
```

W linii 23 tworzę obiekt który będzie reprezentował ekran podglądu nagrania. Na ekranie będę miał też guzik, początkowo będzie na nim napis „Nagrywaj”. Kiedy nacisnę go pierwszy raz, ma się rozpocząć nagrywanie a napis na nim ma się zmienić na „Koniec”. Kiedy kliknę po raz drugi, ma zostać przerwane nagranie i zapisany plik. Potrzebuję więc zmiennej (linia 24), tutaj typu boolean której stan będzie informował o tym czy trwa nagranie czy nie. Zdefiniowana w linii 25 zmienna sciezkaPliku ma wskazywać nazwę i ścieżkę do pliku z nagraniem który ma zostać utworzony. Linie 27-43 to metoda która ma obsługiwać naciscięnia przycisku. Za pierwszym kliknięciem wykonywany jest kod z linii 39-41, rozpoczyna się nagrywanie a napis na przycisku zmienia się na „Koniec”. Przy drugim kliknięciu, wykonywany jest kod z linii 28-38. Przerywane jest nagrywanie, a dodatkowo sprawdziam czy utworzył się plik z nagraniem. Jeśli zostanie faktycznie stworzony, dostanę na konsoli LogCata informację o tym.

Przechodzimy teraz do metody onCreate, a więc uruchamianej w momencie startu aktywności.

```
45    @Override
46    public void onCreate(Bundle savedInstanceState) {
47        super.onCreate(savedInstanceState);
48        setContentView(R.layout.activity_main);
49        nagrywanie = false;
50        mediaRecorder = new MediaRecorder();
51        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
52        initMediaRecorder();
53        surfaceView = (SurfaceView) findViewById(R.id.videoview);
54        surfaceHolder = surfaceView.getHolder();
55        surfaceHolder.addCallback(this);
56        surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
57        guzik = (Button) findViewById(R.id.mybutton);
58        OnClickListener l = new OnClickListener() {
59            @Override
60            public void onClick(View v) {
61                klikniecie();
62            }
63        };
64        guzik.setOnClickListener(l);
65    }
```

Podpinam tutaj referencje do komponentów którymi będziemy się posługiwać, ustawiam nagrywanie tylko w poziomie (linia 51), rejestruję obiekt aktywności jako odbiornik dla zdarzeń związanych z nagrywaniem i przygotowywaniem ekranu podglądu (linie 54-56), a jako obsługę kliknięcia przycisku ustawiam wywołanie wcześniej omawianej metody klikniecie() (linia 61). W linii 52 wywołuję też metodę initMediaRecorder(), której zawartość wygląda tak:

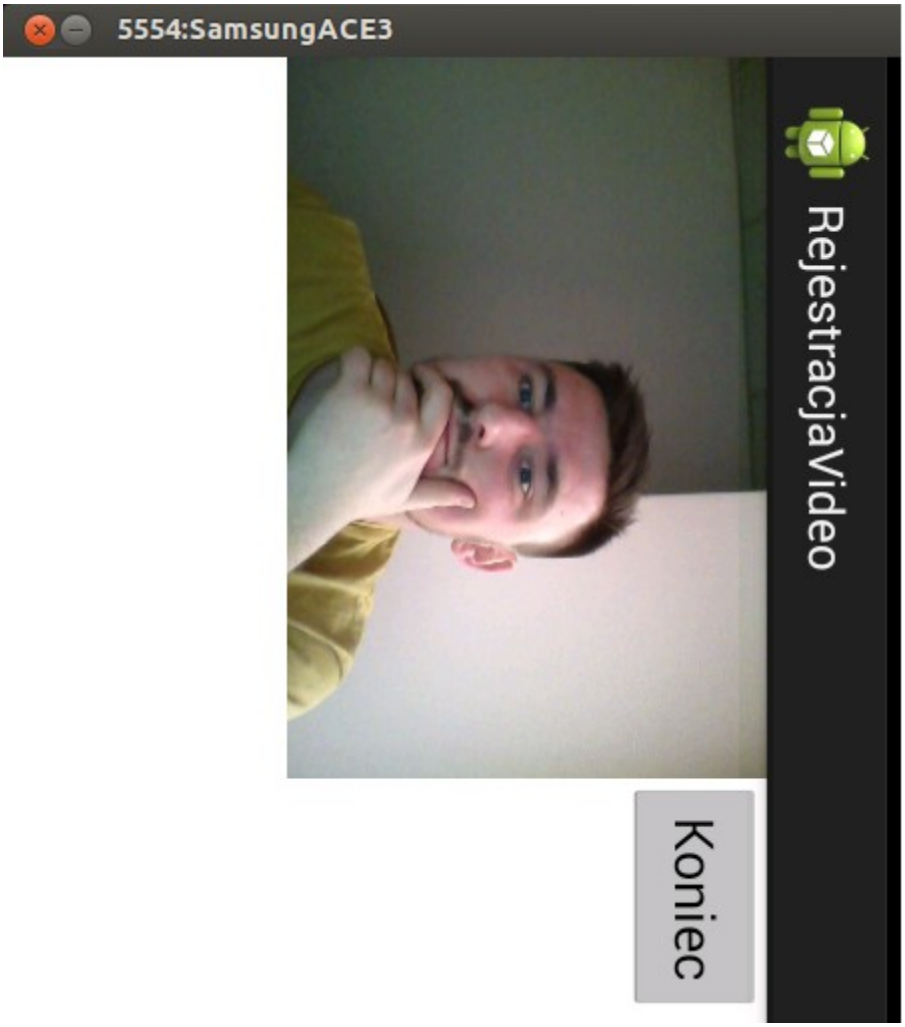
```
87    private void initMediaRecorder() {
88        mediaRecorder.setAudioSource(MediaRecorder.AudioSource.DEFAULT);
89        mediaRecorder.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
90        CamcorderProfile hq = CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH);
91        mediaRecorder.setProfile(hq);
92        mediaRecorder.setOutputFile(sciezkaPliku);
93        mediaRecorder.setMaxDuration(180000); //trzy minuty
94        mediaRecorder.setMaxFileSize(10000000); //10MB
95    }
```

Służy ona ustawieniu źródeł dźwięku i obrazu, jakości obrazu, maksymalnego czasu nagrania i maksymalnej wielkości pliku.

Przy uruchomieniu aktywności, tworzony jest też podgląd nagrania. Musimy go przy tej okazji przygotować. W metodzie surfaceCreated (wymaganej w związku z implementacją interfejsu Callback) wywołuję więc własna metodę prepareMediaRecorder, której jedynym zadaniem jest wykonanie wyżej wspomnianej czynności.

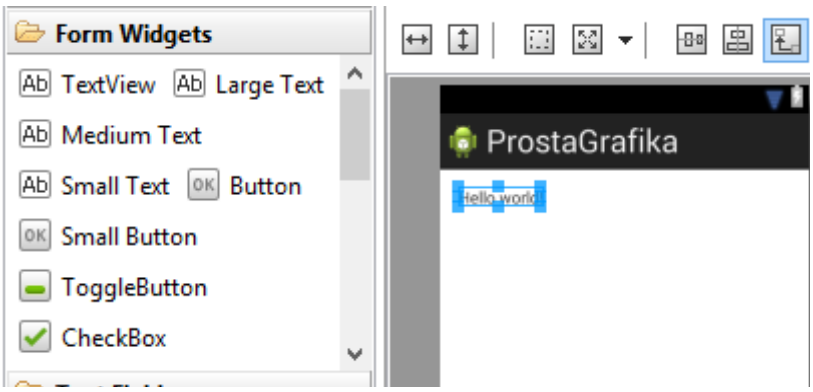
```
74
75      @Override
76      public void surfaceCreated(SurfaceHolder arg0) {
77          // TODO Auto-generated method stub
78          prepareMediaRecorder();
79      }
80
81      @Override
82      public void surfaceDestroyed(SurfaceHolder arg0) {
83          // TODO Auto-generated method stub
84
85      }
86
87      private void initMediaRecorder() {
88          mediaRecorder.setAudioSource(MediaRecorder.AudioSource.DEFAUL
89          mediaRecorder.setVideoSource(MediaRecorder.VideoSource.DEFAUL
90          CamcorderProfile hq = CamcorderProfile.get(CamcorderProfile.Q
91          mediaRecorder.setProfile(hq);
92          mediaRecorder.setOutputFile(sciezkaPliku);
93          mediaRecorder.setMaxDuration(180000); //trzy minuty
94          mediaRecorder.setMaxFileSize(10000000); //10MB
95      }
96
97      private void prepareMediaRecorder() {
98          mediaRecorder.setPreviewDisplay(surfaceHolder.getSurface());
99          try {
100             mediaRecorder.prepare();
101
102         } catch (Exception e) {
103             // TODO Auto-generated catch block
104             e.printStackTrace();
105         }
106     }
107 }
```

Aplikacja po uruchomieniu na emulatorze:



Grafika 2D

Aby wykonywać jakiekolwiek rysunki, będziemy potrzebowali płótna na którym będziemy rysować. Takie płótno będzie obiektem klasy Canvas. Tworzę zwyczajny projekt, a z ekranu głównego usuwam domyślnie pojawiający się tam kompontent textView1. Zasadniczo nasze płótno przykryje całość ekranu włącznie z tym komponentem, więc kasowanie go nie jest konieczne.



Obiekt klasy Canvas początkowo będzie pusty. Możemy rysować obiekty korzystając z wyznaczania kolejnych linii wg współrzędnych, ale możemy też skorzystać z gotowych w klasie Canvas metod które narysują dla nas wybrane figury geometryczne – typu prostokąt czy okrąg. Aby stworzyć i wykorzystywać nasze „płótno”, nasza aktywność musi posiadać wewnętrzną klasę dziedziczącą po klasie View dla której przesłaniamy metodę „onDraw” przyjmującej jako parametr obiekt klasy canvas z którego będziemy korzystać. Myślę że posłużymy się tutaj obrazem. Pierwsze co robię, to do aktywności na której chcę rysować dodaję klasę wewnętrzną CanvasView (nazwa dowolna), dziedziczącą po klasie View (android.view.View). Musimy dodać do niej konstruktor przyjmujący jako parametr kontekst. W zasadzie z kontekstem nic szczególnego tutaj nie robimy, jedynie przekazujemy je do super klasy, niemniej taki konstruktor po prostu musi się tutaj pojawić (wymogi implementacyjne klasy View). Nas bardziej interesuje metoda „onDraw”, bo to w niej właśnie opisujemy wszystko co ma zostać narysowane na „dzień dobry” na naszym płótnie.

```
1  package com.example.prostagrafika;
2
3  import android.os.Bundle;
9
10 public class MainActivity extends Activity {
11
12     class CanvasView extends View{
13         public CanvasView(Context context) {
14             super(context);
15         }
16         @Override
17         protected void onDraw(Canvas canvas){
18
19         }
20     }
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.activity_main);
26     }
27
28
29 }
```

Jak na razie, nic szczególnego się nie dzieje. Musimy sprawić, by zamiast domyślnego layoutu (tego określanego w pliku XML), na ekranie pojawiło się nasze „płótno”.

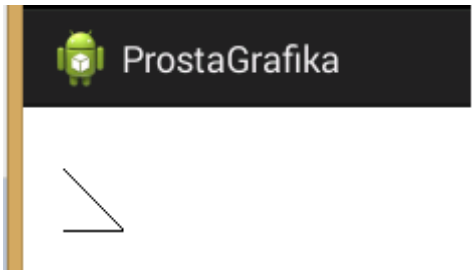
```
 7 import android.view.Menu;
 8 import android.view.View;
 9
10 public class MainActivity extends Activity {
11
12     class CanvasView extends View{
13         public CanvasView(Context context) {
14             super(context);
15         }
16         @Override
17         protected void onDraw(Canvas canvas){
18
19         }
20     }
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         //setContentView(R.layout.activity_main);
26         setContentView(new CanvasView(this));
27     }
28
29
30 }
```

Zmiany tej dokonałem podmieniając parametr dla metody setContentView w metodzie onCreate naszej aktywności. Widzimy to na powyższej ilustracji w linii 26. W linii 25 pozostawiłem (dla przykładu) wykomentowane domyślne ustawienie.

Skoro już wszystko mamy popodpinane, zajmiemy się teraz samym rysowaniem. Dodałem trzy linie do metody onDraw. W linii 19 definiuję obiekt klasy Paint. Jest to pędzel którym będziemy rysować różne kształty. Możemy dla niego ustawiać różne parametry – np. kolor – ale o tym za chwilę. Linie 20 i 21 to wywołanie metody drawLine , służącej rysowaniu linii. Przyjmuje ona 5 parametrów. Wg. Kolejności: X początkowe, Y początkowe, X końcowe, Y końcowe, obiekt klasy paint którym rysujemy (nasz pędzel). Narysowałem więc po prostu dwie linie złączone na końcach:

```
17         @Override
18         protected void onDraw(Canvas canvas){
19             Paint pedzel = new Paint();
20             canvas.drawLine(20, 30, 50, 60, pedzel);
21             canvas.drawLine(50, 60, 20, 60, pedzel);
22         }
```

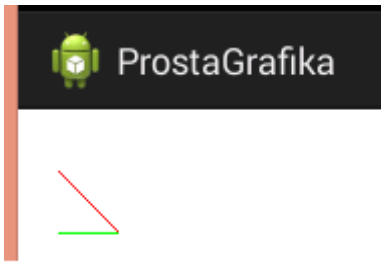
Efekt:



Pobawimy się teraz kolorami pędzla. Do zmiany jego koloru, służy metoda setARGB klasy Paint (której nasz pędzel jest obiektem). Przyjmuje ona cztery parametry. Pierwszy to stopień przeźroczystości linii / figury. 0 to całkowita przeźroczystość – czyli w praktyce niewidoczność, 255 to całkowita nieprzezroczystość. Kolejne trzy parametry to RGB - stopień nasycenia trzech kolorów wg kolejności : Red, Green,Blue. Przyjmowane wartości to 0-255. Przed narysowaniem pierwszej linii ustawiam kolor pędzla na czerwień (255 dla RED, reszta 0), następnie rysuję linię. Ponownie zmieniam kolor pędzla, tym razem na zielony (255 dla GREEN, reszta 0) i znowu rysuję linię:

```
18          protected void onDraw(Canvas canvas){
19              Paint pedzel = new Paint();
20              pedzel.setARGB(255, 255,0,0);
21              canvas.drawLine(20, 30, 50, 60, pedzel);
22              pedzel.setARGB(255, 0,255,0);
23              canvas.drawLine(50, 60, 20, 60, pedzel);
24          }
25      }
```

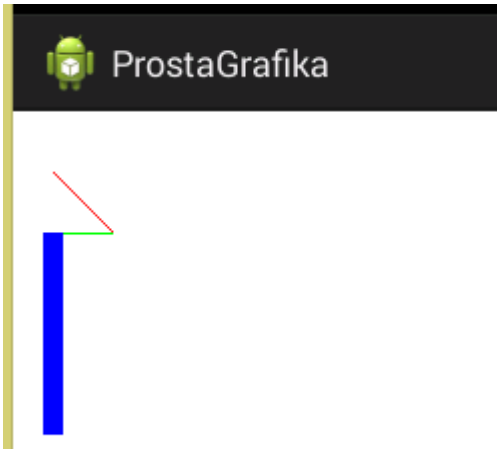
Efekt:



Wzbogaciłem kod o kolejne kilka linii. Zmieniłem kolor pędzla na niebieski i dorysowałem kolejną linię. Doszło nam wywołanie metody setStrokeWidth dla pędzla (linia 25). Ustawia ona grubość linii pędzla:

```
16          }
17          @Override
18          protected void onDraw(Canvas canvas){
19              Paint pedzel = new Paint();
20              pedzel.setARGB(255, 255,0,0);
21              canvas.drawLine(20, 30, 50, 60, pedzel);
22              pedzel.setARGB(255, 0,255,0);
23              canvas.drawLine(50, 60, 20, 60, pedzel);
24              pedzel.setARGB(255, 0,0,255);
25              pedzel.setStrokeWidth(10);
26              canvas.drawLine(20, 60, 20, 160, pedzel);
27
28          }
```

Efekt:

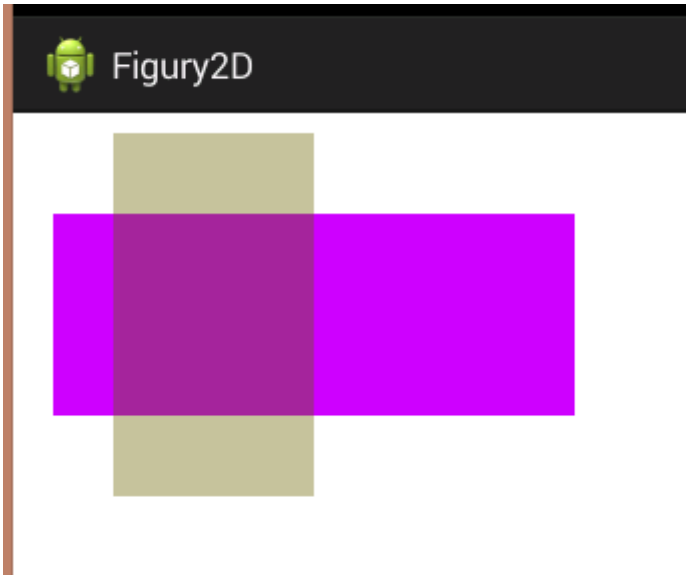


Zajmiemy się teraz rysowaniem figur geometrycznych. Zaczniemy od prostokątów (i kwadratów, wszak kwadrat też jest prostokątem tyle że o równych ramionach :)).

Stworzyłem nowy projekt, wszystko na identycznej zasadzie jak w poprzednim przykładzie. Różni się tylko zawartością metody onDraw:

```
19        @Override
20        protected void onDraw(Canvas c){
21            Paint p = new Paint();
22            p.setARGB(255, 200, 0, 255);
23            c.drawRect(20, 50, 280, 150, p);
24            p.setARGB(100, 100, 100, 0);
25            c.drawRect(50, 10, 150, 190, p);
26        }
```

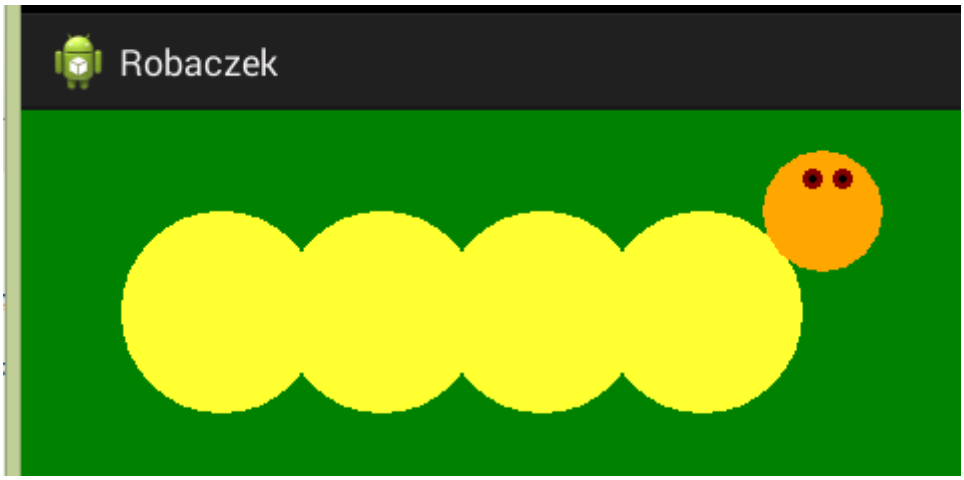
Metoda setARGB jest już znana z poprzednich przykładów, jednak tutaj pobawiłem się troszkę przezroczystością. Z nowych rzeczy mamy tutaj dwukrotne wywołanie metody drawRect klasy Canvas. Metoda ta służy rysowaniu prostokątów. Pierwsze dwa argumenty to współrzędne (x,y) jednego z rogów, kolejne dwa to współrzędne(x,y) rogu przeciwległego. W tym przypadku według kolejności – lewy górny i prawy dolny. Ostatni argument to obiekt klasy Paint, czyli pędzel którym będziemy rysować. Tutaj narysowałem dwa prostokąty – jeden nieprzezroczysty fioletowy ((?) sorry, mężczyźni nie rozróżniają kolorów zbyt szczegółowo) , drugi częściowo przezroczysty w kolorze szybki przyciemnianych okularów (?). Drugi ma ustawioną częściową przezroczystość. W przypadku rysowania pierwszego prostokąta przezroczystość dla pędzla ustawiłem na 0, w drugim na 100. Efekt wygląda tak:



Rysowanie kółek nie sprawia również wielu problemów. Stworzyłem kolejny projekt, tym razem metoda onDraw wygląda tak:

```
19          protected void onDraw(Canvas c){
20              c.drawRGB(0,128,0);
21              Paint paint = new Paint();
22              paint.setARGB(255, 255,255,51);
23              for(int x=100;x<=400;x=x+80){
24                  c.drawCircle(x, 100, 50, paint);
25              }
26              paint.setARGB(255, 255,165,0);
27              c.drawCircle(400, 50, 30, paint);
28              paint.setARGB(255, 128,0,0);
29              c.drawCircle(395, 34, 5, paint);
30              c.drawCircle(410, 34, 5, paint);
31              paint.setARGB(255, 0,0,0);
32              c.drawCircle(395, 34, 2, paint);
33              c.drawCircle(410, 34, 2, paint);
34          }
35
36      }
```

Generalnie do rysowania kół służy metoda drawCircle która przyjmuje cztery parametry. Pierwsze dwa to X i Y środka koła, trzeci to promień, a czwarty to pędzel którego parametry ustawiamy na identycznych zasadach co wcześniej. Zaprzęgnąłem tutaj pętle do wyrysowania mi kilku kół, które częściowo na siebie nachodzą. W zasadzie cały kod to rysowane w rożnych miejscach koła, ze zmianą właściwości pędzla. Efekt wygląda tak :



Robacek :) Ponieważ uznałem że przeciętny robaczek żyje na listku/trawce/gałązce/czołgu/czymś innym zielonym, musiałem zmienić kolor tła. Do tego celu użyłem (linia 20) metody drawRGB obiektu klasy Canvas. Ta metoda nie przyjmuje atrybutu przezroczystości tak metoda drawARGB dla obiektów klasy Paint. Płótno samo w sobie nie może być przezroczyste :). Kolejność kolorów jak sama nazwa wskazuje RGB - Red, Green, Blue.

Bluetooth – czyli niebieskie pogaduszki

O korzyściach płynących z możliwości korzystania z Bluetooth nie trzeba chyba nikogo przekonywać. Dziś zajmiemy się podłączaniem urządzeń z Bluetooth i komunikacją z nimi. Tradycyjnie zaczynamy od dodania niezbędnych uprawnień do pliku manifestu:

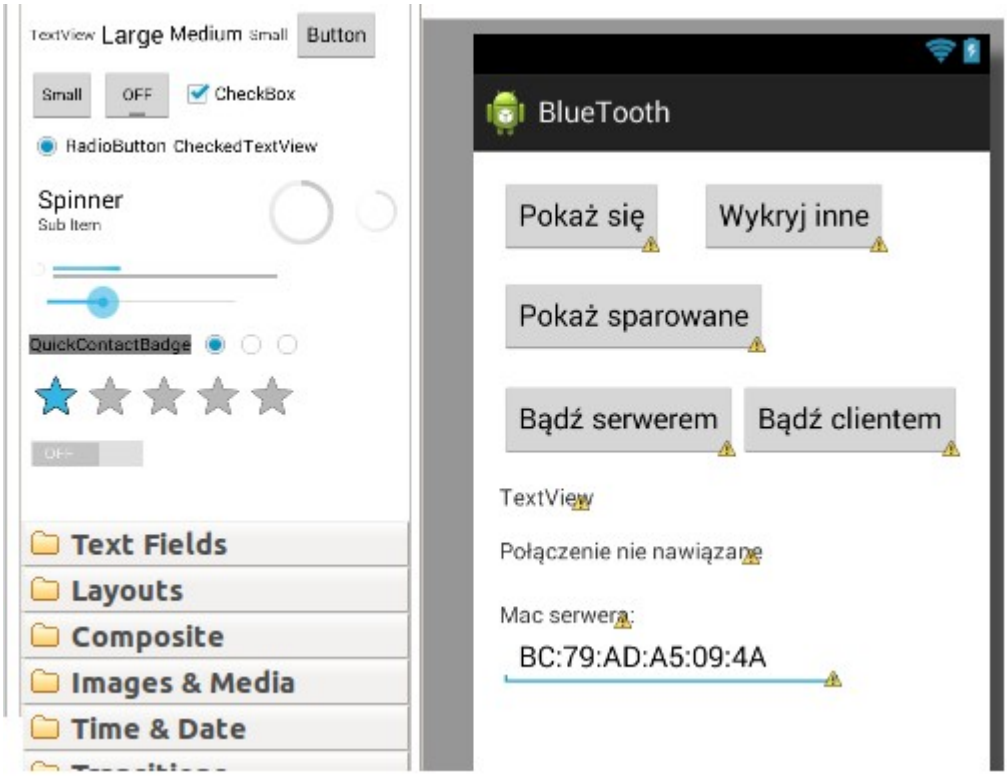
```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Pierwsze uprawnienie daje nam możliwość korzystania z Bluetooth, drugie jego włączania i wyłączania (a musimy mieć taką możliwość w razie gdyby w urządzeniu BT było wyłączone).

Napiszemy aplikację która będzie w stanie :

- Rozgłosić fakt swojego istnienia, tak by inne urządzenia skanując w poszukiwaniu sprzętu z BlueTooth mogły nas zobaczyć. Jeśli urządzenia nie są ze sobą sparowane, urządzenie nie siejące informacji o swoim istnieniu nie będzie widoczne.
- Wyświetlić na konsoli listę sparowanych urządzeń
- Wykryć inne urządzenia pozwalające się wykryć (czyli rozgłaszające fakt swojego istnienia).
- Utworzyć serwer sieciowy oparty na BlueTooth, który będzie oczekiwał na podłączenie się clienta, a gdy to nastąpi prześle mu komunikat.
- Połączyć się jako client do innego urządzenia na którym będzie uruchomiony serwer.

Nasza aplikacja będzie mogła działać zarówno jako client, jak i jako serwer. Zależeć to będzie od tego, który przycisk naciśniemy. Aby móc testować aplikację najlepiej będzie byś wyposażył się w dwa fizyczne urządzenia wyposażone w BlueTooth. Przyklejam parę guzików, które będą uruchamiać wcześniej omówione funkcje. Na komponencie TextView na którym aktualne wyświetlam napis „TextView” w momencie uruchomienia aplikacji pokaże się adres mac naszego urządzenia. Na TextView na którym początkowo wyświetlać się będzie napis „Połączenie nie nawiązane”, wyświetli się później tryb działania aplikacji – jako serwer lub jako client. W polu edycyjnym będziemy wpisywać mac adres urządzenia które będzie pracowało jako serwer. Wstawiłem tutaj adres swojego telefonu, żeby później nie musieć każdorazowo go podawać.



Przejdźmy teraz do kodu głównej aktywności. Na początku w metodzie onCreate podpinam komponenty wizualne do obiektów. Nic nowego ani nadzwyczajnego:

```
19
20 public class MainActivity extends Activity {
21
22     Button b;
23     Button b2;
24     Button b3;
25     Button b4;
26     Button b5;
27     TextView t1;
28     TextView t2;
29     EditText et1;
30
31     @Override
32     protected void onCreate(Bundle savedInstanceState) {
33         super.onCreate(savedInstanceState);
34         setContentView(R.layout.activity_main);
35         b=(Button) findViewById(R.id.button1);
36         b2=(Button) findViewById(R.id.button2);
37         b3=(Button) findViewById(R.id.button3);
38         b4=(Button) findViewById(R.id.button4);
39         b5=(Button) findViewById(R.id.button5);
40         t1=(TextView) findViewById(R.id.textView1);
41         t2=(TextView) findViewById(R.id.textView2);
42         et1=(EditText) findViewById(R.id.editText1);
43
```

Dalej w metodzie onCreate podpinam wywołania metod realizujących poszczególne funkcjonalności do przycisków.

```
44          Log.d("INFO","Uruchomiono program");
45
46          b.setOnClickListener(new OnClickListener() {
47              @Override
48              public void onClick(View v) {
49                  dajSieWykryc();
50              }
51          });
52          b2.setOnClickListener(new OnClickListener() {
53              @Override
54              public void onClick(View v) {
55                  wykryjInne();
56              }
57          });
58
59          b3.setOnClickListener(new OnClickListener() {
60              @Override
61              public void onClick(View v) {
62                  pokazSparowane();
63              }
64          });
65          b4.setOnClickListener(new OnClickListener() {
66              @Override
67              public void onClick(View v) {
68                  t2.setText("Jo sem serwer!");
69                  new SerwerBluetooth().start();
70              }
71          });
72          b5.setOnClickListener(new OnClickListener() {
73              @Override
74              public void onClick(View v) {
75                  t2.setText("Jo sem client!");
76                  BluetoothAdapter ba = BluetoothAdapter.getDefaultAdapter();
77                  BluetoothDevice serwer = ba.getRemoteDevice(et1.getText().toString());
78                  new ClientBluetooth(serwer).start();
79              }
80          });
81
```

Wyjaśnienia mogą wymagać jedynie linie 67-69 i 75-78. W zależności od tego czy aplikacja na danym urządzeniu będzie pracować jako serwer czy client, wywoływane są wątki realizujące zadania danej strony. Oparłem to na wątkach, ponieważ np. oczekiwanie na połączenie , czy proces łączenia są operacjami które blokowałyby wątek głównej aktywności. W ten sposób wątki pracują sobie asynchronicznie, a aplikacja nie „blokuje się”. Obiekt ba z linii 76 reprezentuje fizyczny adapter Bluetooth urządzenia. Linia 77 to pobranie mac adresu wpisanego w pole edycyjne. Będzie nam ten adres potrzebny, po przekazujemy go przez parametr konstruktora do wątku clienta. Musi on wiedzieć jaki jest adres urządzenia z którym ma się łączyć. W liniach 68 i 75 w zależności od trybu działania aplikacji, wyświetlam na stosownym polu na ekranie, jaką rolę spełnia aplikacja (czy jest serwerem czy clientem).

Przejdźmy dalej. Linie 85,86 to wyświetlenie mac adresu urządzenia na ekranie oraz konsoli. W dalszej części metody onCreate aktywności (linie 87-91) obsługuję sytuację gdyby okazało się, że BlueTooth na urządzeniu jest wyłączony. Aktywność BT weryfikuję metodą isEnabled() klasy BluetoothAdapter (linia 87). Jeśli okaże się że jest wyłączony, wyświetlam komunikat z prośbą o zgodę na włączenie BT. Ponieważ intencję z prośbą wywołuję z użyciem startActivityForResult, dodałem też metodę onActivityResult która jest automatycznie wywoływana przez system kiedy użytkownik udzieli zgody na włączenie BT (albo i nie udzieli :p). Jeśli użytkownik udzieli zgody, sam system uruchomi BT a nam pozostaje tylko zainicjalizowanie obiektu którego będziemy używać do komunikowania się z fizycznym adapterem BT i ewentualnie wyświetlenie informacji na konsoli.

```
82
83
84          BluetoothAdapter ba = BluetoothAdapter.getDefaultAdapter();
85          t1.setText("Twój mac: "+ba.getAddress());
86          Log.d("INFO","Twój adres urządzenia: "+ba.getAddress());
87          if(!ba.isEnabled()){
88              Intent i=new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
89              startActivityForResult(i, 1);
90          }
91      }
92
93      @Override
94      protected void onActivityResult(int requestCode,int resultCode,Intent i){
95          if(resultCode==Activity.RESULT_OK){
96              Log.d("INFO","Mamy zgodę!");
97              BluetoothAdapter ba = BluetoothAdapter.getDefaultAdapter();
98          }
99      }
```

Przeszkoczymy na razie definicję obiektu klasy BroadcastReceiver (linie 102-117), wrócimy do niej za chwilę, a na razie przyjrzymy się metodom na końcu klasy.

```
118
119     public void dajSieWykryc(){
120         Log.d("INFO","Daję się wykryć!");
121         Intent pokazSie = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
122         pokazSie.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
123         startActivity(pokazSie);
124     }
125
```

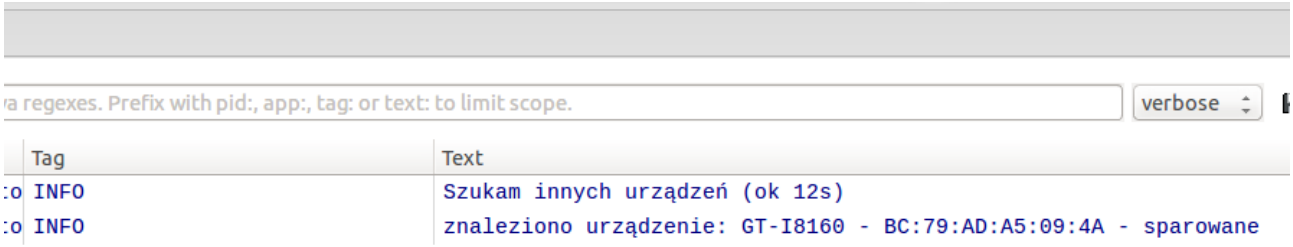
Metoda dajSieWykryc() sprawia, że nasze urządzenie jest widoczne dla innych urządzeń które skanują w poszukiwaniu „kolegów” :). Jeśli o to nie zadbamy a urządzenia nie będą sparowane, nasze nie będzie widoczne dla innych. Linie 121 i 123 są odpowiedzialne za wyświetlenie zapytania o możliwość „ujawnienia się” i rozpoczęcie rozgłaszania swojego istnienia. Linia 122 to opcjonalna konfiguracja czasu rozgłaszania. Domyślnie urządzenie rozgłaszałoby przez 2 minuty, natomiast podałem mu wartość 300 (sekund) , dzięki czemu będzie widoczny przez 5 minut.

```
126
127     public void wykryjInne(){
128         Log.d("INFO","Szukam innych urządzeń (ok 12s)");
129         IntentFilter filtr = new IntentFilter(BluetoothDevice.ACTION_FOUND);
130         this.registerReceiver(odbiorca, filtr);
131         BluetoothAdapter ba = BluetoothAdapter.getDefaultAdapter();
132         ba.startDiscovery();
133     }
```

Metoda wykryjInne() powoduje rozpoczęcie poszukiwania innych urządzeń. Trwa to ok 12 sekund. Co istotne – widoczne będą tylko te urządzenia które będą rozgłaszały swoje istnienie. Kiedy jakieś urządzenie zostanie znalezione trzeba będzie obsłużyć takie zdarzenie nie przerywając przeszukiwania. Z tego powodu rejestrujemy odbiorcę komunikatu rozgłoszeniowego informującego o znalezieniu urządzenia (linia 130). Odbiorca to obiekt klasy BroadcastReceiver którego metoda onReceive jest automatycznie wywoływana w przypadku znalezienia urządzenia z włączonym rozgłaszaniem sygnału. W ramach tej metody wyciągam obiekt reprezentujący dane urządzenie fizyczne, sprawdzam czy to urządzenie zostało wcześniej sparowane i wyświetlam o nim informacje.

```
102     private final BroadcastReceiver odbiorca= new BroadcastReceiver() {
103         @Override
104         public void onReceive(Context context, Intent i) {
105             String akcja =  i.getAction();
106             if(BluetoothDevice.ACTION_FOUND.equals(akcja)){
107                   BluetoothDevice device = i.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
108                   String status="";
109                   if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
110                       status="nie sparowane";
111                   }else{
112                       status="sparowane";
113                   }
114                       Log.d("INFO","znaleziono urządzenie: "+device.getName()+" - "+device.getA
115           }
116         }
117     };
```

Łącznie działanie tych dwóch elementów przedstawia się w ten sposób, że metoda wykryjInne() rozpoczyna poszukiwanie innych urządzeń, a gdy jakiś znajdzie to zostaje automatycznie wywołana metoda onReceive obiektu klasy BroadcastReceiver zarejestrowanego jako odbiorca takiego zdarzenia. W sumie, na konsoli zostaną wyświetlone informacje o dostępnych w okolicy urządzeniach. Po uruchomieniu na konsoli widzę znalezione urządzenie (jeśli zostaną wykryte, pojawi się ich więcej ;)) :



W ramach tej klasy mamy jeszcze metodę pokazSparowane():

```
134
135     public void pokazSparowane(){
136         /*
137          * Wyświetlanie listy sparowanych urządzeń, niezależnie od tego czy są
138          * właśnie podłączone czy w ogóle włączone.
139          * */
140         Log.d("INFO","Sparowane dla tego urządzenia");
141         BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
142         Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
143         if (pairedDevices.size() > 0) {
144             for (BluetoothDevice device : pairedDevices) {
145                 Log.d("INFO",device.getName()+" - "+device.getAddress());
146             }
147         }
148     }
149
150
151
152
153 }
```

Jej zadaniem jest sprawdzenie listy urządzeń które wcześniej sparowaliśmy i wyświetlenie informacji o nich na ekranie. Ta metoda nie sprawdza czy urządzenia te są dostępne , a jedynie wyświetla informacje o zarejestrowanych urządzeniach które są przechowywane w systemie:

Tag	Text
libEGL	loaded /system/lib/egl/libEGL_mali.so
libEGL	loaded /system/lib/egl/libGLESv1_CM_mali.
libEGL	loaded /system/lib/egl/libGLESv2_mali.so
OpenGLRenderer	Enabling debug mode 0
INFO	Sparowane dla tego urządzenia
INFO	SM-T211 - C8:14:79:17:39:50
INFO	PLT_M165 - 48:C1:AC:52:8E:ED

W przypadku mojego telefonu zostały wyświetlone informacje o słuchawce na Bluetooth której używam (więc siłą rzeczy musi być sparowana), oraz tablecie który musiał zostać sparowany by odbywać po BT pogaduszki z moim telefonem na potrzeby niniejszego artykułu.

Przejdźmy teraz do elementu komunikacji. Jeśli programowałeś kiedyś sieciowe połączenia client-server w Javie, na pewno rozpoznasz wiele elementów. W zasadzie jest to zwykłe połączenie po socketach z tą różnicą, że nasza komunikacja nie będzie się odbywała po sieci, a po Bluetooth. Najpierw weźmiemy na tapetę klasę odpowiadającą za działanie aplikacji jako serwer.

```
14 public class SerwerBluetooth extends Thread {
15     private final BluetoothServerSocket mmServerSocket;
16
17     public SerwerBluetooth() {
18         BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
19         BluetoothServerSocket tmp = null;
20         try {
21             UUID uuid=UUID.fromString("550e8400-e29b-41d4-a716-446655440000");
22             tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord("Usługa witająca", uuid);
23         } catch (IOException e) { }
24         mmServerSocket = tmp;
25     }
26
27     public void run() {
28         Log.d("INFO","Uruchamiam serwer");
29         BluetoothSocket socket = null;
30         while (true) {
31             try {
32                 Log.d("INFO","Czekam na połączenie od clienta");
33                 socket = mmServerSocket.accept();
34                 Log.d("INFO","Mam clienta!");
35                 /*Utworzenie strumieni wejściowego i wyjściowego*/
36                 PrintWriter out = new PrintWriter(socket.getOutputStream(),true);
37                 out.println("Witaj kolego!");
38                // BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getIn
39             } catch (IOException e) {
40                 break;
41             }
42             if (socket != null) {
43                /*
44                 * Jakieś dodatkowe operacje na połączeniu
45                 * */
46                 try {
47                     mmServerSocket.close();
48                 } catch (Exception e) {
49                     e.printStackTrace();
50                 }
51                 break;
52             }
53         }
54     }
55
56 }
```

W linii 15 definiuję socket przy użyciu którego będzie odbywała się komunikacja. Zagadkowe mogą się wydać linie 21 i 22. Usługa musi mieć unikalny identyfikator, dzięki któremu będzie jednoznacznie rozpoznawana (http://en.wikipedia.org/wiki/Universally_unique_identifier). W tych liniach naszą usługę rejestruję pod nazwą „Usługa witająca” pod identyfikatorem

`550e8400-e29b-41d4-a716-446655440000`

Ten akurat identyfikator jest całkiem przypadkowy, skopiowałem jakiś przykład takiego identyfikatora z internetu. Linie 27-54 to klasyczna implementacja procesu nasłuchu, różnica polega na tym, że nie sieciowego a na bluetooth. Wątek czeka na kontakt ze strony serwera, po czym otwiera strumień i wysyła tekst „Witaj kolego!”.

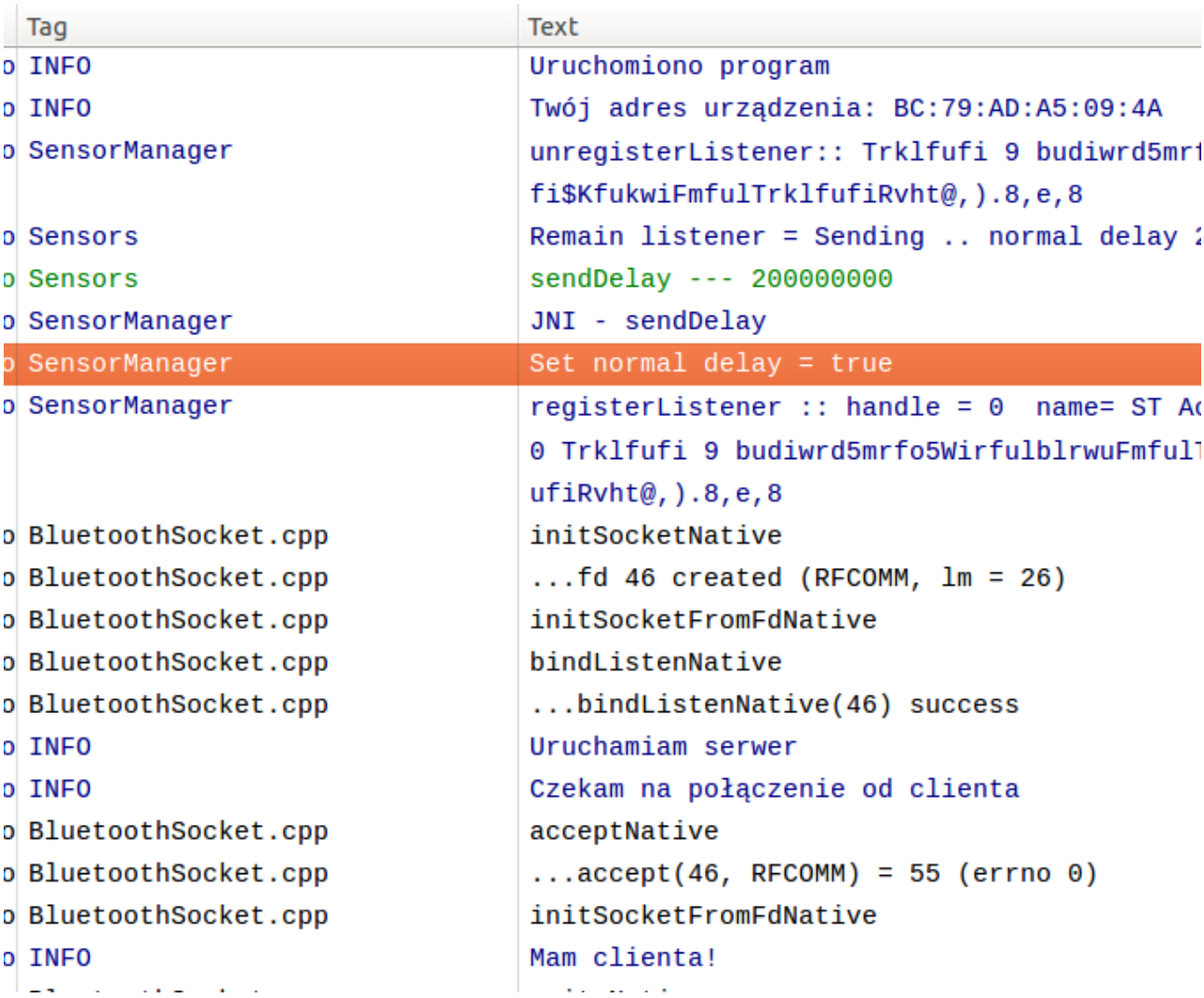
Strona kliencka:

```
public class ClientBluetooth extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ClientBluetooth(BluetoothDevice device) {
        BluetoothSocket tmp = null;
        mmDevice = device;
        try {
            UUID uuid = UUID.fromString("550e8400-e29b-41d4-a716-446655440000");
            tmp = device.createRfcommSocketToServiceRecord(uuid);
        } catch (Exception e) { }
        mmSocket = tmp;
    }

    public void run() {
         try {
            Log.d("INFO","Próba połączenia....");
            mmSocket.connect();
            Log.d("INFO","Połączono z serwerem!");
            BufferedReader in = new BufferedReader(new InputStreamReader(mmSocket.getInputStream()));
            String input = in.readLine();
            Log.d("INFO","Serwer mówi: "+input);

        } catch (Exception ce) {
            try {
                mmSocket.close();
            } catch (Exception cle) { }
        }

    }
}
```

Po tej stronie otwieram socket dla połączenia z serwerem (podobnie jak po stronie serwerowej nie socket sieciowy a po bluetooth) i odczytuję co mi serwer przysłał.

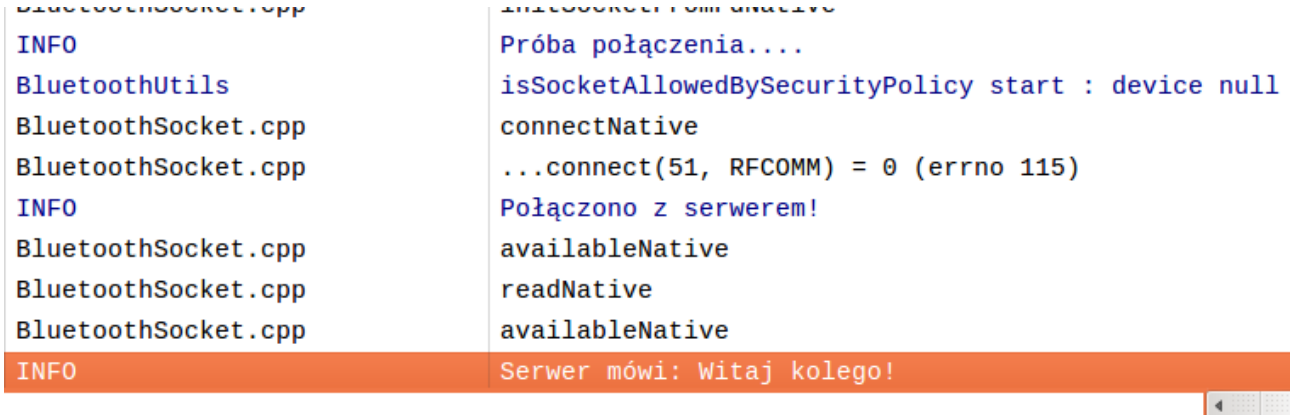
Uruchamiam teraz program na obu urządzeniach, telefon robi za serwer, tablet za clienta.

Efekty działania serwera:



Tag	Text
INFO	Uruchomiono program
INFO	Twój adres urządzenia: BC:79:AD:A5:09:4A
SensorManager	unregisterListener:: Trklfufi 9 budiwrd5mrf
	fi$KfukwiFmfulTrklfufiRvht@,).8,e,8
Sensors	Remain listener = Sending .. normal delay 2
Sensors	sendDelay --- 200000000
SensorManager	JNI - sendDelay
SensorManager	Set normal delay = true
SensorManager	registerListener :: handle = 0 name= ST Ac
	0 Trklfufi 9 budiwrd5mrfo5WirfulblrwuFmfulT
	ufiRvht@,).8,e,8
BluetoothSocket.cpp	initSocketNative
BluetoothSocket.cpp	...fd 46 created (RFCOMM, lm = 26)
BluetoothSocket.cpp	initSocketFromFdNative
BluetoothSocket.cpp	bindListenNative
BluetoothSocket.cpp	...bindListenNative(46) success
INFO	Uruchamiam serwer
INFO	Czekam na połączenie od clienta
BluetoothSocket.cpp	acceptNative
BluetoothSocket.cpp	...accept(46, RFCOMM) = 55 (errno 0)
BluetoothSocket.cpp	initSocketFromFdNative
INFO	Mam clienta!

Po stronie clienta:



INFO	Próba połączenia....
BluetoothUtils	isSocketAllowedBySecurityPolicy start : device null
BluetoothSocket.cpp	connectNative
BluetoothSocket.cpp	...connect(51, RFCOMM) = 0 (errno 115)
INFO	Połączono z serwerem!
BluetoothSocket.cpp	availableNative
BluetoothSocket.cpp	readNative
BluetoothSocket.cpp	availableNative
INFO	Serwer mówi: Witaj kolego!

Gdybyś zechciał rozwijać ten program, lub na jego bazie napisać coś swojego, pamiętaj że mamy do czynienia z nieco innym niż sieciowy protokołem i usługa BT z jednej strony może się łączyć tylko z jedną inną usługą.