**Android – Szkolenie Podstawowe**

**Future Processing**

**Central Mobile Team**

***„Projekt Jaszczur”* 2014**

**Android – Szkolenie Podstawowe**

[Zad 1 - Utworzenie nowego projektu 3](#_Toc396374415)

[Cykl życia aktywności 3](#_Toc396374416)

[Struktura projektu 6](#_Toc396374417)

[Gradle 7](#_Toc396374418)

[AndroidManifest 8](#_Toc396374419)

[Zad 2 – Tworzymy pierwszą aktywność 9](#_Toc396374420)

[Zad 3 – Odkrywamy magię cyklu życia 10](#_Toc396374421)

[Zad 4 - Tworzymy drugą aktywność z listą 11](#_Toc396374422)

[Zad 5 – Zapisanie wybranej waluty do pamięci trwałej 17](#_Toc396374423)

[Zad 6 – Odczyt z pamięci 19](#_Toc396374424)

[Zad 7 – Asynchroniczne ładowanie bitmap 20](#_Toc396374425)

[Zad 8 – Pobieranie walut z internetu i parsowanie xml’a 21](#_Toc396374426)

[Zad 9 – Dodawanie menu kontekstowego oraz akcji do ActionBar’a 24](#_Toc396374427)

[Zad 10 – Dialog do edycji aktualnej waluty 25](#_Toc396374428)

[Zad 11 – Przeliczanie walut 27](#_Toc396374429)

[Przydatne linki 27](#_Toc396374430)

### Zad 1 - Utworzenie nowego projektu

### Cykl życia aktywności



**onCreate()** – wywoływana gdy aktywność została utworzona po raz pierwszy. To jest mniejsce gdzie należy wykonywać podstawowe czynności, jak tworzenie widoków, bindowanie danych itp.

**onRestart()** – wywoływana gdy aktywność została zatrzymana i uruchomiona ponownie

**onStart()** – wywoływana, gdy aktywność staje się widoczna dla użytkownika.

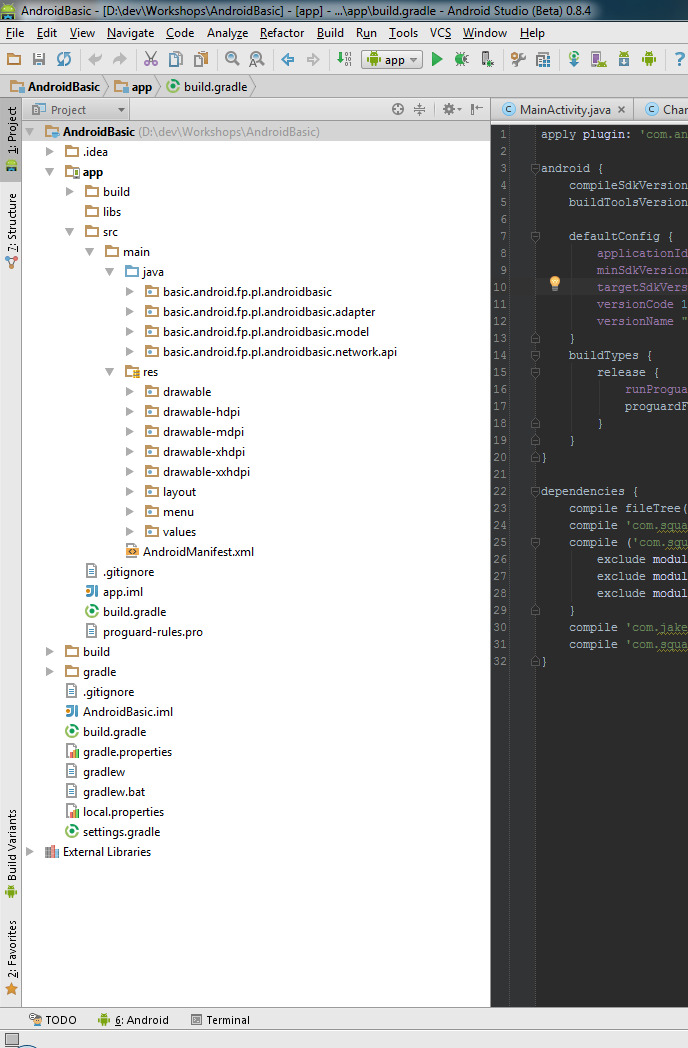
**onResume()** – wywoływana gdy aktywność rozpoczyna interakcję z użytkownikiem. W tym omencie aktywność jest na szczycie sotsu aktywności.

**onPause()** - wywoływana, gdy system rozpoczyna przywracanie poprzedniej aktywności. Jest zazwyczaj używana do zapisania trwałych danych, zatrzymania animacji i innych rzeczy, które mogą obciążać procesor.

**onStop()** – wywoływana gdy aktywność przestaje być widoczna dla użytkownik. Dzieje się tak zazwyczaj gdy otwierana jest nowa aktywność lub aktualna zostaje zniszczona.

**onDestroy()** – Ostatnia metoda wywoływana przed zniszczeniem aktywności.

### Struktura projektu



### Gradle

**Plik gradle.build w module:**

*apply plugin: 'com.android.application'*

*android {*

*compileSdkVersion 20*

*buildToolsVersion "20.0.0"*

*defaultConfig {*

*applicationId "basic.android.fp.pl.androidbasic"*

*minSdkVersion 15*

*targetSdkVersion 20*

*versionCode 1*

*versionName "1.0"*

*}*

*buildTypes {*

*release {*

*runProguard false*

*proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'*

*}*

*}*

*}*

*dependencies {*

*compile fileTree(dir: 'libs', include: ['\*.jar'])*

*}*

### AndroidManifest

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

*<manifest xmlns:android="http://schemas.android.com/apk/res/android"*

*package="fp.pl.bestworkshop"*

*android:versionCode="1"*

*android:versionName="1.0" >*

*<uses-sdk*

*android:minSdkVersion="8"*

*android:targetSdkVersion="17" />*

*// Permission required if you want to use internet connection*

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

*<application*

*android:allowBackup="true"*

*android:icon="@drawable/ic\_launcher"*

*android:label="@string/app\_name"*

*android:theme="@style/AppTheme" >*

*<activity*

*android:name="fp.pl.bestworkshop.MainActivity"*

*android:configChanges="orientation"*

*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>*

*</manifest>*

### Zad 2 – Tworzymy pierwszą aktywność

**Pamiętaj:** Każda aktywność musi być zadeklarowana w manifeście!!

**MainActivity:**

import android.os.Bundle;

import android.widget.Button;

public class MainActivity extends Activity {

private Button button;

private Button button2;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

button = (Button) findViewById(R.id.button);

button2 = (Button) findViewById(R.id.button2);

}

}

**Activity\_main.xml:**

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:gravity="center">

<Button

android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/button"/>

<Button

android:id="@+id/button2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/button2"/>

</LinearLayout>

### Zad 3 – Odkrywamy magię cyklu życia

public class MyActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

Log.i("LIFECYCLE", "-OnCreate");

}

@Override

protected void onStart() {

super.onStart();

Log.i("LIFECYCLE", "--OnStart");

}

@Override

protected void onResume() {

super.onResume();

Log.i("LIFECYCLE", "---OnResume");

}

@Override

protected void onPause() {

super.onPause();

Log.i("LIFECYCLE", "---OnPause");

}

@Override

protected void onStop() {

super.onStop();

Log.i("LIFECYCLE", "--OnStop");

}

@Override

protected void onDestroy() {

super.onDestroy();

Log.i("LIFECYCLE", "-OnDestroy");

}

@Override

protected void onRestart() {

super.onRestart();

Log.i("LIFECYCLE", "=====>OnRestart");

}

}

### Zad 4 - Tworzymy drugą aktywność z listą

**Activity\_change\_currency.xml:**

<ListView xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/list"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

**ChangeCurrencyActivity:**

public class ChangeCurrencyActivity extends Activity {

@InjectView(R.id.list)

ListView currencyListView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_change\_currency);

ButterKnife.inject(this);

}

}

**Dodajemy do build.gradle w tagu dependencies:**

compile 'com.jakewharton:butterknife:+'

**Dodajemy do MainActivity otwieranie nowej aktywności po naciśnięciu buttona.**

**W metodzie onCreate() dodajemy:**

button.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent(getApplicationContext(), ChangeCurrencyActivity.class);

startActivity(i);

}

});

**Tworzymy layout dla pojedyńczego elementu listy:**

**item\_currency\_list.xml:**

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="horizontal">

<ImageView

android:id="@+id/flag"

android:layout\_width="36dp"

android:layout\_height="36dp"/>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

android:id="@+id/currencyName"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:paddingLeft="6dp"

android:text="Name"

android:textAppearance="?android:attr/textAppearanceMedium"/>

<TextView

android:id="@+id/averageRate"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:paddingLeft="6dp"

android:text="1 EUR 4 PLN"

android:textAppearance="?android:attr/textAppearanceSmall"/>

</LinearLayout>

</LinearLayout>

**Informacje na temat walut pobierać będziemy ze strony:**

<http://www.nbp.pl/kursy/xml/LastA.xml>

**Na podstawie otrzymanego xml’a tworzymy klasy modelu:**

**CurrencyTable:**

public class CurrencyTable {

private String type;

private String id;

private String number;

private Date publicationDate;

private List<Currency> currencyList = new ArrayList<Currency>();

public CurrencyTable(String type, String id, String number, Date publicationDate, List<Currency> currencyList) {

this.type = type;

this.id = id;

this.number = number;

this.publicationDate = publicationDate;

this.currencyList = currencyList;

}

public List<Currency> getCurrencyList() {

return currencyList;

}

public Date getPublicationDate() {

return publicationDate;

}

}

**Currency:**

public class Currency {

private String name;

private int converter;

private String code;

private String averageRate;

public Currency(String name, int converter, String code, String averageRate) {

this.name = name;

this.converter = converter;

this.code = code;

this.averageRate = averageRate;

}

public String getName() {

return name;

}

public String getCode() {

return code;

}

public double getAverageRate() {

return Double.valueOf(averageRate.replaceAll(",", "."));

}

public int getConverter() {

return converter;

}

}

**Tworzymy adapter dla listy:**

**CurrencyListAdapter:**

public class CurrencyListAdapter extends BaseAdapter {

private Context context;

private CurrencyTable currencyTable;

private final LayoutInflater inflater;

public CurrencyListAdapter(Context context, CurrencyTable currencyTable) {

this.context = context;

this.currencyTable = currencyTable;

inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

}

@Override

public int getCount() {

return currencyTable.getCurrencyList().size();

}

@Override

public Currency getItem(int position) {

return currencyTable.getCurrencyList().get(position);

}

@Override

public long getItemId(int position) {

return position;

}

@Override

public View getView(int position, View convertView, ViewGroup parent) {

return convertView;

}

}

**Dodajemy klasę wewnętrzną.**

**ViewHolder:**

static class ViewHolder {

@InjectView(R.id.currencyName)

TextView currencyName;

@InjectView(R.id.averageRate)

TextView averageRate;

private ViewHolder(View rootView) {

ButterKnife.inject(this, rootView);

}

}

**Definiujemy jak wypełniany ma być element listy.**

**W metodzie getView() dodajemy:**

@Override

public View getView(int position, View convertView, ViewGroup parent) {

ViewHolder vh;

if (convertView == null) {

convertView = inflater.inflate(R.layout.item\_currency\_list, parent, **false**);

vh = new ViewHolder(convertView);

convertView.setTag(vh);

} else {

vh = (ViewHolder) convertView.getTag();

}

Currency currency = currencyTable.getCurrencyList().get(position);

vh.currencyName.setText(currency.getName());

vh.averageRate.setText(String.valueOf(currency.getAverageRate()));

return convertView;

}

**Ważne, by przy inflatowaniu podać jako parametr attacheToRoot „false”(pogrubione w powyższym listingu), ponieważ adapter pod spodem robi to za nas, więc podpięcie samodzielnie spowalnia cały proces.**

**Tworzymy dane testowe i dodajemy adapter do listy w ChangeCurrencyActivity w metodzie onCreate():**

List<Currency> list = new ArrayList<Currency>();

list.add(new Currency("Polski złoty", 1, "PLN", "99"));

list.add(new Currency("Euro", 1, "EUR", "55"));

list.add(new Currency("Dolar", 1, "USD", "44"));

list.add(new Currency("Polski złoty", 1, "PLN", "1.54"));

list.add(new Currency("Polski złoty", 1, "PLN", "2.5"));

CurrencyTable currencies = new CurrencyTable("Normal", "1", "1", new Date(), list);

CurrencyListAdapter adapter = new CurrencyListAdapter(this, currencies);

currencyListView.setAdapter(adapter);

### Zad 5 – Zapisanie wybranej waluty do pamięci trwałej

**Dodajemy klasę pomocniczą do zapisu i odczytu z pamięci trwałej.**

public class SharedPreferencesSupporter {

private static final String CURRENCY\_MAIN\_KEY = SharedPreferencesSupporter.class.getName() + ".currency";

private static final String NAME = ".name";

private static final String CODE = ".code";

private static final String AVERAGE\_RATE = ".averageRate";

private static final String CONVERTER = ".converter";

private static Currency defaultCurrency = new Currency("default", 1, "def", "1");

public static Currency loadCurrentCurrency(Context context) {

SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context);

int converter = preferences.getInt(CURRENCY\_MAIN\_KEY + CONVERTER, defaultCurrency.getConverter());

float averageRate = preferences.getFloat(CURRENCY\_MAIN\_KEY + AVERAGE\_RATE, defaultCurrency.getAverageRate());

String code = preferences.getString(CURRENCY\_MAIN\_KEY + CODE, defaultCurrency.getCode());

String name = preferences.getString(CURRENCY\_MAIN\_KEY + NAME, defaultCurrency.getName());

return new Currency(name, converter, code, String.valueOf(averageRate));

}

public static void saveCurrentCurrency(Currency currency, Context context) {

SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context);

SharedPreferences.Editor editor = preferences.edit();

editor.putInt(CURRENCY\_MAIN\_KEY + CONVERTER, currency.getConverter());

editor.putFloat(CURRENCY\_MAIN\_KEY + AVERAGE\_RATE, currency.getAverageRate());

editor.putString(CURRENCY\_MAIN\_KEY + CODE, currency.getCode());

editor.putString(CURRENCY\_MAIN\_KEY + NAME, currency.getName());

editor.apply();

}

}}

**Następnie obsługujemy kliknięcie na elemencie listy.**

**W klasie ChangeCurrencyActivity w metodzie onCreate dodajemy:**

currencyListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

CurrencyListAdapter currencyAdapter = (CurrencyListAdapter)parent.getAdapter();

Currency currency = currencyAdapter.getItem(position);

SharedPreferencesSupporter.saveCurrentCurrency(currency, ChangeCurrencyActivity.this);

Toast.makeText(this, "Currency saved to SharedPreferences", Toast.LENGTH\_SHORT).show();

}

});

**Lub używając Butterknifa:**

@OnItemClick(R.id.list)

void onListItemClick(AdapterView<?> parent, View view, int position){

CurrencyListAdapter currencyAdapter = (CurrencyListAdapter)parent.getAdapter();

Currency currency = currencyAdapter.getItem(position);

SharedPreferencesSupporter.saveCurrentCurrency(currency, ChangeCurrencyActivity.this);

Toast.makeText(this, "Currency saved to SharedPreferences", Toast.LENGTH\_SHORT).show();

}

### Zad 6 – Odczyt z pamięci

**By zobaczyć rezultat zapisu w pamięci dodamy pole tekstowe w MainActivity z aktualnie wybraną walutą.**

**Edytujemy activity\_main.xml:**

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:gravity="center">

<TextView

android:id="@+id/currentCurrency"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="20sp"

android:textStyle="bold"/>

<Button

android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/fancy\_adapter"/>

<Button

android:id="@+id/button2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/fancy\_adapter"/>

</LinearLayout>

**Oraz obsługujemy pobieranie z pamięci w MainActivity w metodzie onCreate:**

currentCurrency = (TextView) findViewById(R.id.currentCurrency);

@Override

protected void onResume() {

super.onResume();

currency = SharedPreferencesSupporter.loadCurrentCurrency(this);

currentCurrency.setText("Current currency is: " + currency.getName() + "\nRate: " + currency.getAverageRate());

}

### Zad 7 – Asynchroniczne ładowanie bitmap

**Do pliku build.gradle w tagu dependecsies dodajemy:**

compile 'com.squareup.picasso:picasso:+'

**Dodajemy do folderu drawable obrazki z flagami.**

**Następnie do CurrencyListAdaptera dodajemy metodę pozwalającą dynamicznie wydobyć referencję do drawable z pliku R na podstawie kodu waluty.**

private int getDrawableId(String drawableCode) {

if (drawableCode.toLowerCase().equals("try")) {

drawableCode += "\_";

}

int drawableId = context.getResources().getIdentifier(drawableCode.toLowerCase(), "drawable", context.getPackageName());

if (drawableId == 0) {

return R.drawable.money;

}

return drawableId;

}

**Uaktualniamy ViewHolder**

static class ViewHolder {

@InjectView(R.id.currencyName)

TextView currencyName;

@InjectView(R.id.averageRate)

TextView averageRate;

@InjectView(R.id.flag)

ImageView flag;

private ViewHolder(View rootView) {

ButterKnife.inject(this, rootView);

}

}

**W metodzie getView() dodajemy pobranie obrazków za pomocą Picasso:**

int drawableId = getDrawableId(currency.getCode());

Picasso.with(context).load(drawableId).placeholder(R.drawable.money).into(vh.flag);

### Zad 8 – Pobieranie walut z internetu i parsowanie xml’a

**Informacje na temat walut pobierać będziemy ze strony:**

<http://www.nbp.pl/kursy/xml/LastA.xml>

**W pliku gradle.build w tagu dependencies dodajemy:**

compile ('com.squareup.retrofit:converter-simplexml:+'){

exclude module: 'stax'

exclude module: 'stax-api'

exclude module: 'xpp3'

}

**Tworzymy API które definiuje punkty dostępu do serwisu.**

public interface NbpApi {

@GET("/kursy/xml/LastA.xml")

CurrencyTable getCurrencyTable();

}

**W ChangeCurrencyActivity w metodzie onCreate() konfigurujemy adapter serwisu.**

RestAdapter adapter = new RestAdapter.Builder()

.setEndpoint("http://www.nbp.pl")

.setConverter(new SimpleXMLConverter())

.build();

service = adapter.create(NbpApi.class);

**Używamy biblioteki Simple do parsowania XML’a dlatego trzeba zmodyfikować pliki modelu.**

**CurrencyTable:**

@Root

public class CurrencyTable {

@Attribute(name = "typ")

private String type;

@Attribute(name = "uid")

private String id;

@Element(name = "numer\_tabeli")

private String number;

@Element(name = "data\_publikacji")

private Date publicationDate;

@ElementList(inline = true)

private List<Currency> currencyList = new ArrayList<Currency>();

**Oraz dodajemy pusty konstruktor:**

public CurrencyTable() {

}

**Currency:**

@Root(name = "pozycja")

public class Currency {

@Element(name = "nazwa\_waluty")

private String name;

@Element(name = "przelicznik")

private int converter;

@Element(name = "kod\_waluty")

private String code;

@Element(name = "kurs\_sredni")

private String averageRate;

**Oraz dodajemy pusty konstruktor:**

public Currency() {

}

**Pobieranie z internet wywoływane musi być na osobnym wątku, inaczej rzucony zostanie NetworkOnMainThreadException. Dlatego oddelegujemy wywołanie operacji sieciowej do osobnego wątku. Zdefiniujemy wewnętrzną klasę rozszerzającą AsyncTask.**

private class GetCurrencyTableTask extends AsyncTask<Void, Void, CurrencyTable> {

private ProgressDialog dialog;

public GetCurrencyTableTask(Context context) {

dialog = new ProgressDialog(context);

dialog.setMessage("Please wait");

}

@Override

protected void onPreExecute() {

super.onPreExecute();

dialog.show();

}

@Override

protected CurrencyTable doInBackground(Void... params) {

return service.getCurrencyTable();

}

@Override

protected void onPostExecute(CurrencyTable currencies) {

super.onPostExecute(currencies);

dialog.dismiss();

currencyListView.setAdapter(new CurrencyListAdapter(ChangeCurrencyActivity.this, currencies));

}

}

**Pozostaje w metodzie onCreate wywołać AsyncTask’a:**

new GetCurrencyTableTask(this).execute();

### Zad 9 – Dodawanie menu kontekstowego oraz akcji do ActionBar’a

**By zdefiniować menu kontekstowe oraz akcje w ActionBarze, należy utworzyć nowy plik w folderze menu.**

**Change\_currency.xml:**

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

<item

android:id="@+id/action\_refresh"

android:title="@string/action\_refresh"

android:showAsAction="always"

android:icon="@drawable/ic\_menu\_refresh"/>

<item

android:id="@+id/menu\_refresh"

android:title="@string/action\_refresh"

android:showAsAction="never"

android:icon="@drawable/ic\_menu\_refresh"/>

</menu>

**Wracamy do ChangeCurrencyActivity i dodajemy metody:**

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.change\_currency, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

int id = item.getItemId();

if (id == R.id.action\_refresh || id == R.id.menu\_refresh) {

new GetCurrencyTableTask(this).execute();

return true;

}

return super.onOptionsItemSelected(item);

}

### Zad 10 – Dialog do edycji aktualnej waluty

**Dodajemy klasę definiującą dialog do edycji waluty:**

public class RateChangeDialog {

public interface OnCurrencyChangedListener {

void onCurrencyChanged(Currency currency);

}

private Context context;

private AlertDialog dialog;

private Currency currency;

private EditText inputEditText;

private Button positiveButton;

private OnCurrencyChangedListener listener;

public RateChangeDialog(Context context, OnCurrencyChangedListener listener, Currency currency) {

this.context = context;

this.listener = listener;

this.currency = currency;

}

public void show() {

if (dialog == null) {

dialog = buildDialog();

}

dialog.show();

getPositiveButton();

setTextInInputEditText();

}

private AlertDialog buildDialog() {

inputEditText = createInputEditText();

return new AlertDialog.Builder(context) //

.setIcon(R.drawable.ic\_launcher) //

.setTitle(R.string.dialog\_title) //

.setMessage(R.string.dialog\_message) //

.setPositiveButton(R.string.ok, new PositiveOnClickListener()) //

.setNegativeButton(R.string.cancel, new NullOnClickListener()) //

.setView(inputEditText) //

.create();

}

private void getPositiveButton() {

positiveButton = dialog.getButton(Dialog.BUTTON\_POSITIVE);

}

private void setTextInInputEditText() {

inputEditText.setText(String.valueOf(currency.getAverageRate()));

}

private class PositiveOnClickListener implements DialogInterface.OnClickListener {

@Override

public void onClick(DialogInterface dialog, int which) {

changeExchangeRate();

}

}

private void changeExchangeRate() {

CharSequence inputText = inputEditText.getText();

float rate = Float.parseFloat(inputText.toString());

currency.setAverageRate(rate);

listener.onCurrencyChanged(currency);

}

private class NullOnClickListener implements DialogInterface.OnClickListener {

@Override

public void onClick(DialogInterface dialog, int which) {

}

}

private EditText createInputEditText() {

EditText input = new EditText(context);

input.setInputType(InputType.TYPE\_CLASS\_NUMBER | InputType.TYPE\_NUMBER\_FLAG\_DECIMAL);

input.addTextChangedListener(new EmptyInputValidator());

return input;

}

private class EmptyInputValidator implements TextWatcher {

@Override

public void onTextChanged(CharSequence s, int start, int before, int count) {

}

@Override

public void beforeTextChanged(CharSequence s, int start, int count, int after) {

}

@Override

public void afterTextChanged(Editable s) {

boolean positiveButtonEnable = s.toString().trim().length() > 0;

positiveButton.setEnabled(positiveButtonEnable);

}

}

}

**W MainActivity dodajemy obsługę klikania na drugi button:**

button2.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

new RateChangeDialog(MainActivity.this, currency).show();

}

});

**Oraz implementacje interfejsu OnCurrencychangedListener**

@Override

public void onCurrencyChanged(Currency currency) {

currentCurrency.setText("Current currency is: " + currency.getName() + "\nRate: " + currency.getAverageRate());

SharedPreferencesSupporter.saveCurrentCurrency(currency, this);

}

### Zad 11 – Przeliczanie walut

### Przydatne linki

* <http://square.github.io/retrofit/>
* <http://simple.sourceforge.net/download/stream/doc/tutorial/tutorial.php>
* <http://jakewharton.github.io/butterknife/>
* <http://square.github.io/picasso/>
* <http://gradleplease.appspot.com/>
* <http://developer.android.com/index.html>