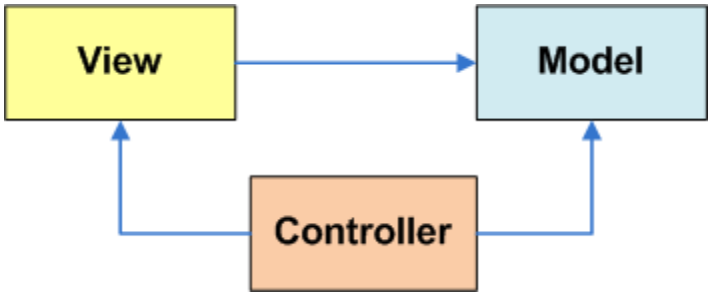
Programowanie urządzeń mobilnych

Graficzny interfejs użytkownika

Tłumaczenie i adaptacja materiałów: dr Tomasz Xięski.
Na podstawie prezentacji udostępnionych przez Victor Matos, Cleveland State University.

Wzorzec MVC

Wzorzec Model-Widok-Kontroler *(MVC)* jest istotnym wzorcem projektowym, którego głównym zadaniem jest odseparowanie (1) interfejsu użytkownika, (2) biznesowej oraz (3) operacyjnej logiki.



Jak to wygląda z punktu widzenia programisty Android?

- **Model**. Składa się z kodu w języku JAVA i odwołań API, które zarządzają zachowaniem danych aplikacji.
- **Widok**. Zestaw ekranów, które użytkownik widzi i wchodzi w interakcję.
- **Kontroler**. Implementowany m. in. poprzez system operacyjny, odpowiedzialny za interpretację zdarzeń użytkownika i systemowych. Zdarzenia mogą pochodzić z różnorakich źródeł: trackball, klawiatura, ekran dotykowy, moduł GPS, usługi działające w tle. Nakazuje modelowi i/lub widokowi (zwykle przez wywołania zwrotne i nasłuchiwaczy) by dokonały odpowiednich zmian.

[Burbeck92] Burbeck, Steve. "Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC)."*University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive.* Available at: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

Wzorzec MVC c. d.

Graficzne interfejsy użytkownika zwykle implementowane są jako pliki XML (aczkolwiek mogą one być tworzone również w sposób dynamiczny poprzed kod języka Java).

Zachowanie kontrolera:

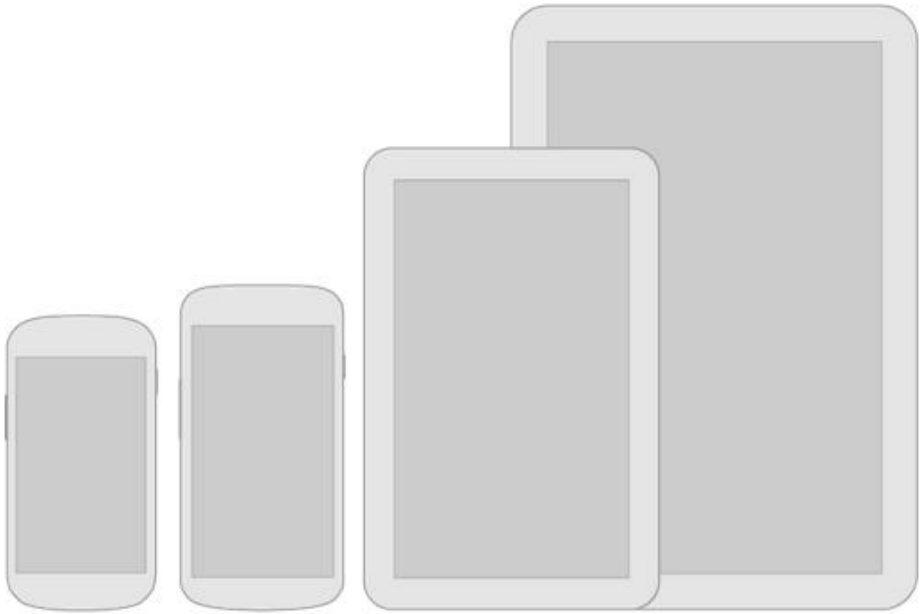
- Bezpośrednia interakcja
  Gdy użytkownik dotknie określonego miejsca na ekranie, kontroler dokonuje interpretacji tego zdarzenia i określa, jaki dokładnie fragment ekranu oraz gest miał miejsce. Na podstawie tej informacji, przekazywane jest do modelu wywołanie określonej funkcji (callback) lub konieczność zmiany stanu aplikacji.

- Niejawna interakcja
  Usługa działająca w tle, może bez ingerowania użytkownika powiadomić kontroler odnośnie zmian stanu (np. osiągnięto miejsce docelowe w nawigacji), co w konsekwencji może spowodować zmianę widoku.

Wzorce projektowe GUI

Devices and Displays

Android powers more than a billion phones, tablets, and other devices in a wide variety of screen sizes and form factors. By taking advantage of Android's flexible layout system, you can create apps that gracefully scale from large tablets to smaller phones.



Be flexible

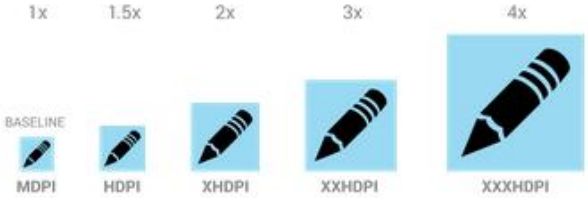
Stretch and compress your layouts to accommodate various heights and widths.

Optimize layouts

On larger devices, take advantage of extra screen real estate. Create compound views that combine multiple views to reveal more content and ease navigation.

Assets for all

Provide resources for different screen densities (DPI) to ensure that your app looks great on any device.



https://developer.android.com/design/patterns/index.html

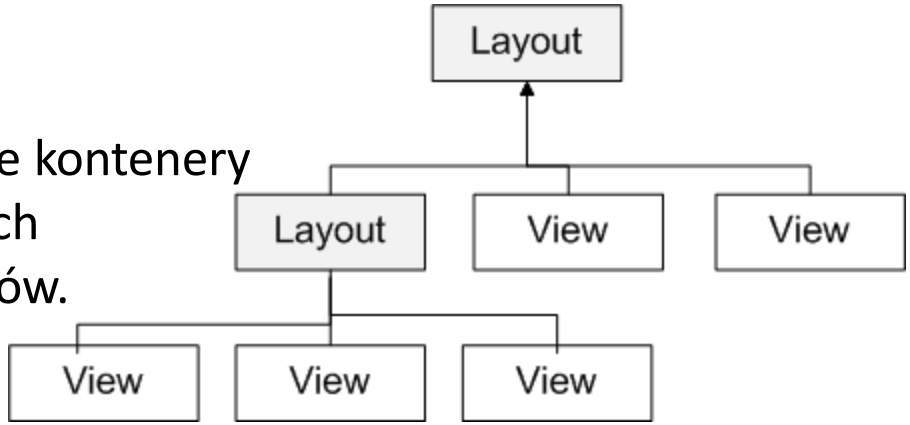
Klasa View

- Klasa **View reprezentuje podstawowy komponent Androida** za pomocą którego można tworzyć graficzne interfejsy użytkownika. Stanowi kontener dla wszystkich elementów, które można wyświelić.

- **Widok** zajmuje prostokątną przestrzeń na ekranie i jest odpowiedzialny za *rysowanie* jak również *przetwarzanie zdarzeń*.

- **Widżety (ang. widgets)** są potomkami klasy View. Używane są do stworzenia interaktywnych komponentów graficznych takich jak przyciski, etykiety, pola tekstowe itp.

- **Układy (ang. layouts)** to niewidzialne kontenery umożliwiające pozycjonowanie innych widoków oraz zagnieżdżonych układów.



Interfejs GUI ⟷ Układ XML

Rzeczywisty interfejs aplikacji

Wersja tekstowa: *activity_main.xml* ⟶

```
<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" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="35dp"
        android:layout_marginTop="35dp"
        android:ems="10"
        android:hint="Enter your NAME here" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginLeft="54dp"
        android:layout_marginTop="26dp"
        android:text="Go" />

</RelativeLayout>
```

Wykorzystanie Widoków

- Plik **XML** z widokiem składa się z układem (**layout)** tworzącym hierarchiczną strukturę zawartych w nim elementów.
- Wewnętrzne elementy mogą być zwykłymi widżetami bądź zagnieżdżonymi widokami zawierające skomplikowane hierarchie elementów.
- Aktywność używa `setContentView(R.layout.xmlfilename)` by wyświetlić widok na ekranie urządzenia.

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


</LinearLayout>
```

Widżety i zagnieżdżone układy

Wykorzystanie widoków

Wykorzystanie widoków i układów z reguły składa się z następującego szeregu czynności:

1. **Ustaw właściwości:** Przykładowo kolor tła, tekstu, czcionki i wielkości komponentów.

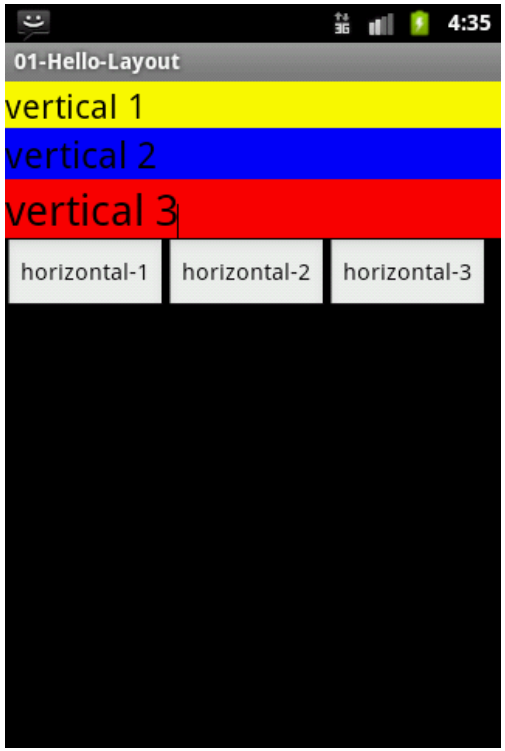
2. **Ustaw nasłuchiwacze (ang. listeners):** Przykładowo obraz może być skonfigurowany by reagował na zdarzenia kliknięcia, dłuższego przytrzymania palca itp.

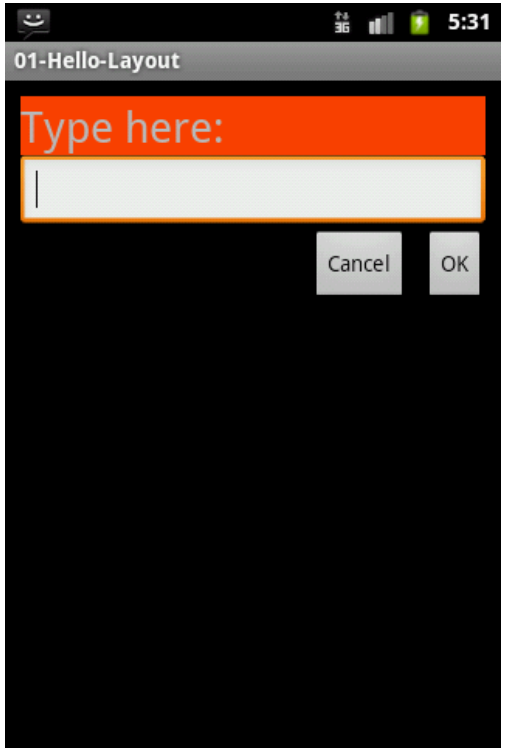
3. **Ustaw focus:** By ustawić focus na określonym komponencie należy użyć metody requestFocus() lub znacznika XML <requestFocus/>

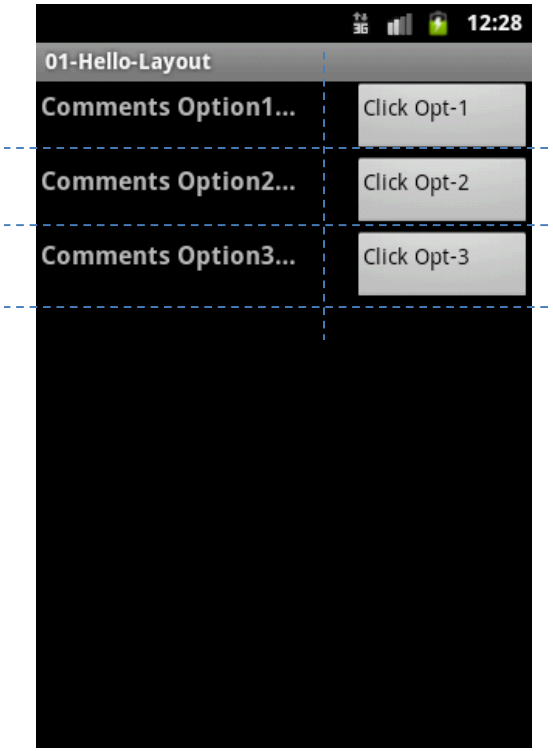
4. **Ustaw widoczność:** Można pokazywać lub ukrywać elementy wykorzystując metodę setVisibility(…).

Przykładowe komponenty GUI

Układy







Linear Layout
LinearLayout rozmieszcza widoki horyzontalnie lub wertykalnie.

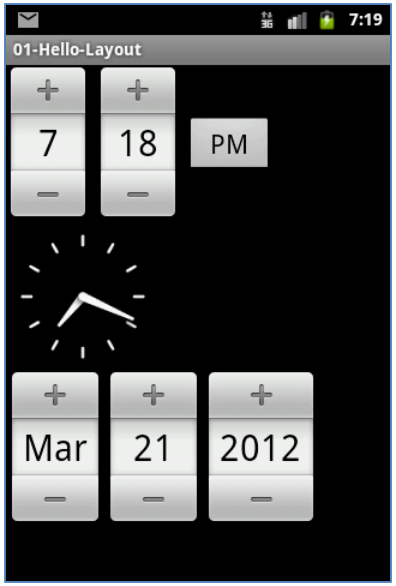
Relative Layout
RelativeLayout jest grupą elementów, która umożliwia rozmieszczenie widoków w sposób względny.

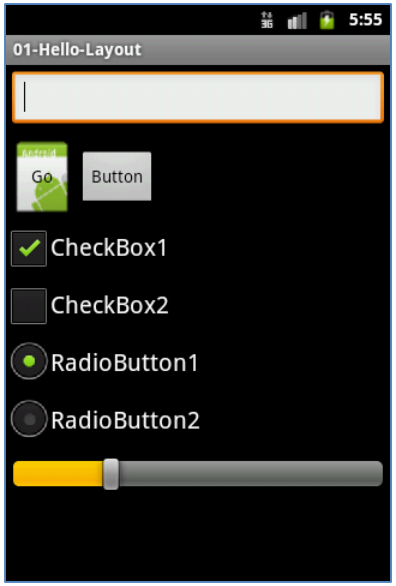
Table Layout
TableLayout jest grupą elementów, która rozmieszcza elementy w wierszu bądź kolumnie wirtualnej tabeli.

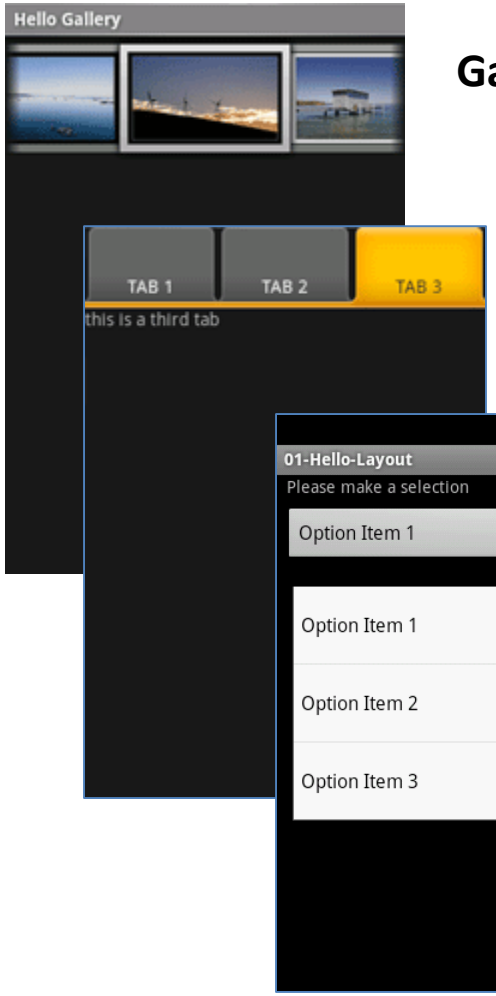
Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

Przykładowe komponenty GUI

Widżety







GalleryView

TabWidget

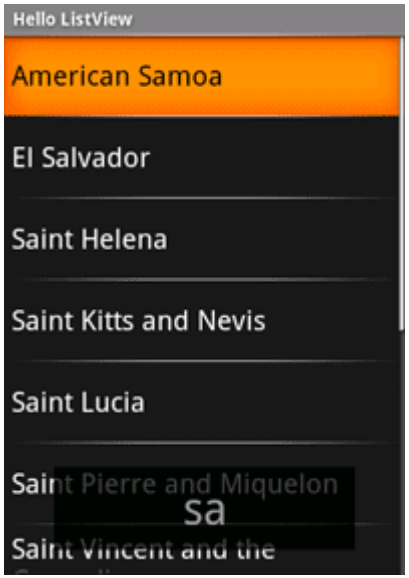
Spinner

TimePicker
AnalogClock
DatePicker
DatePicker jest komponentem umożlwiającym wybór miesiąca, dnia i roku.

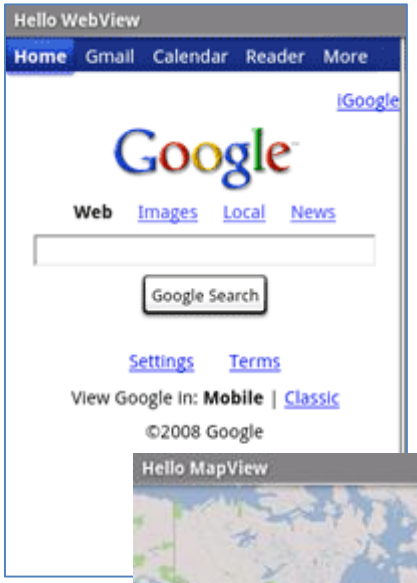
Kontrolki formularza
Grupa komponentów do wpisania danych: *przyciski z grafiką, pola jednokrotnego i wielokrotnego wyboru itp.*

Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

Przykładowe komponenty GUI



WebView



MapView

AutoCompleteTextView

Jest specyficzną wersją pola tekstowego (*EditText), który pokazuje sugestie użytkownikowi podczas pisania*. Sugestie pochodzą z kolekcji typu tablica ciągów.

ListView

ListView jest widokiem który pokazuje dane w formie pionowej, przewijalnej listy. Dane pochodząc z obiektu typu *ListAdapter*.

Reference: http://developer.android.com/guide/topics/ui/layout-objects.html

Układy XML w Android Studio

W kontekście platformy Android, pliki XML wchodzą w skład **zasobów (ang. *resources*)**, więc pliki z interfejsem graficznym znajdują się w katalogu **res/layout** projektu.

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

app

SQLite1 > app > src > main > res > layout > activity_main.xml

Android

app
- manifests
- java
- res
  - drawable
  - layout
    - activity_main.xml
    - car_list.xml
    - car_list_item.xml
    - car_new.xml
    - car_update.xml
    - cost_list.xml
    - cost_list_item.xml
    - cost_new.xml
    - cost_update.xml
    - drawer_layout.xml
    - drawer_nav_header.xml
    - refuel_list.xml
    - refuel_list_item.xml

RefuelListActivity.java | activity_main.xml | refuel_list.xml

android.support.design.widget.CoordinatorLayout | android.support.design.widget.AppBarL

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:fab="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="com.example.pietrzyk.sqlite1.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/cost_update_toolbar"
            android:layout_width="match_parent"
```

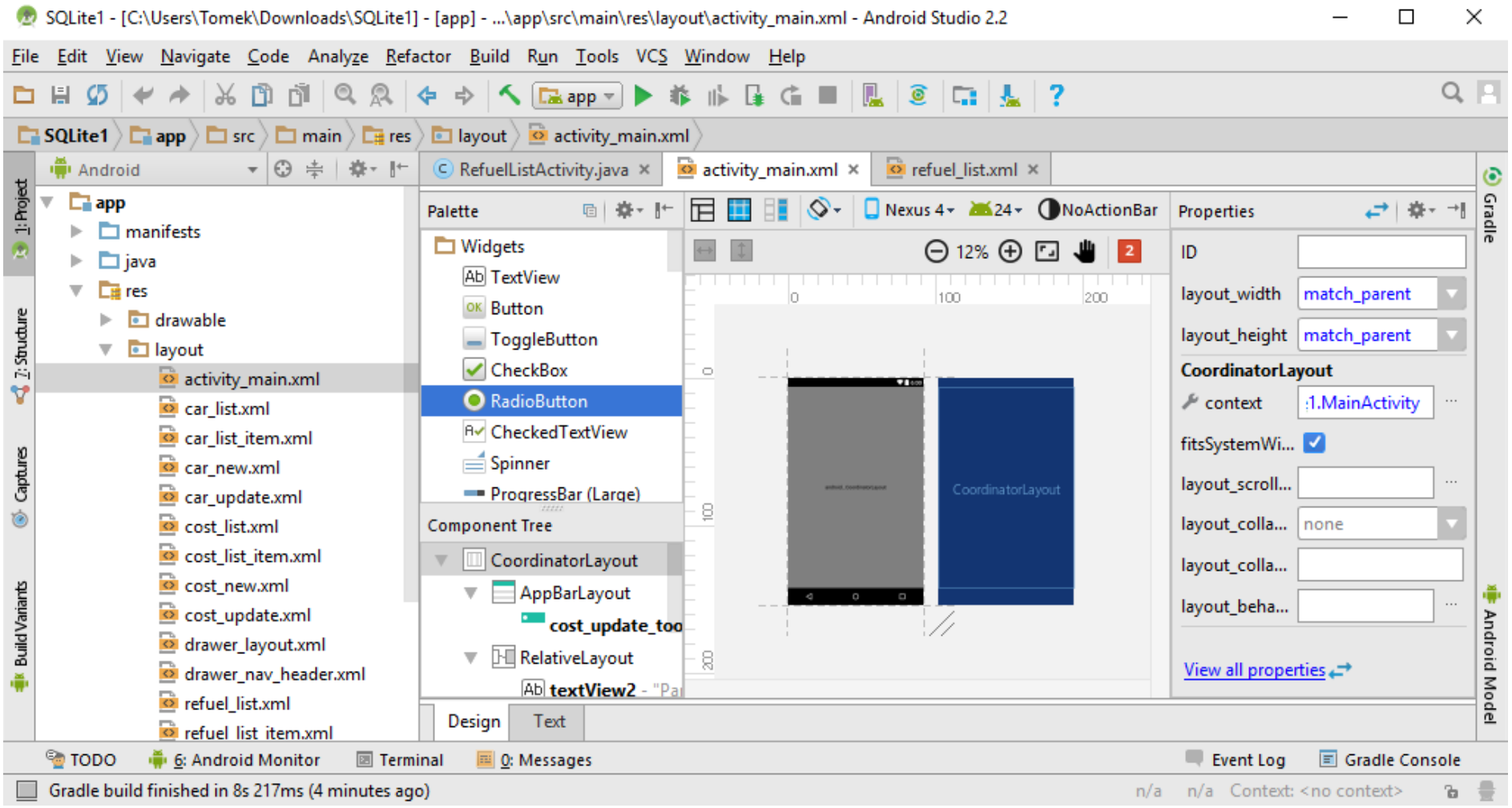
Design | Text

TODO | 6: Android Monitor | Terminal | 0: Messages | Event Log | Gradle Console

Copy to clipboard | 12 chars | 13:43 | CRLF | UTF-8 | Context: <no context>

Układy XML w Android Studio

Android Studio zawiera również edytor typu WYSIWIG, który umożliwia tworzenie interfejsów metodą przeciągij-i-upuść.

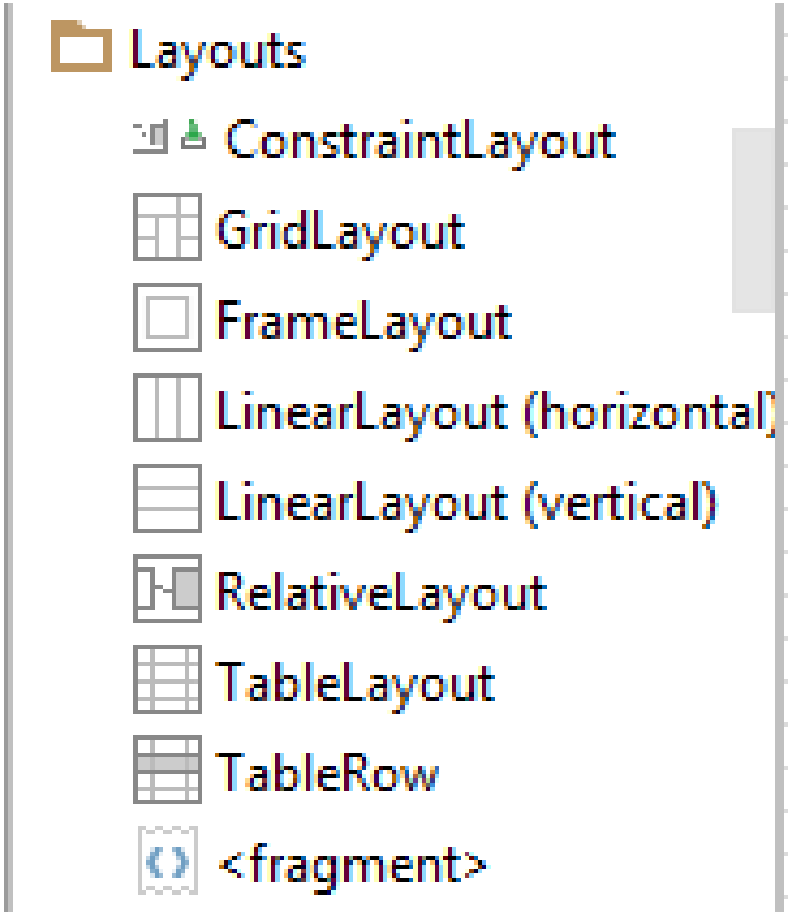


Jak tworzyć GUI w Androidzie

- Układy w Androidzie są kontenerami GUI, posiadające ściśle zdefiniowaną strukturę oraz właściwości dotyczące rozłożenia elementów.

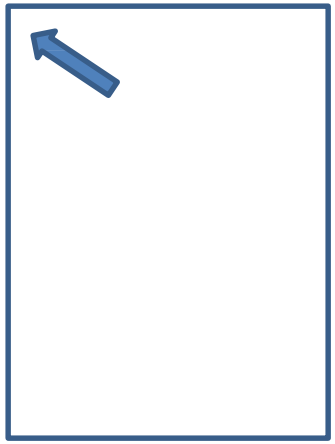
- **Układy mogą być zagnieżdżane**, dlatego komórka, wiersza albo kolumna może być początkiem dla innego widoku bądź układu.

- W Android Studio dostępne są następujące typy układów (co ewoluuje z każdą wersją Androida):



Podstawowe układy

FrameLayout

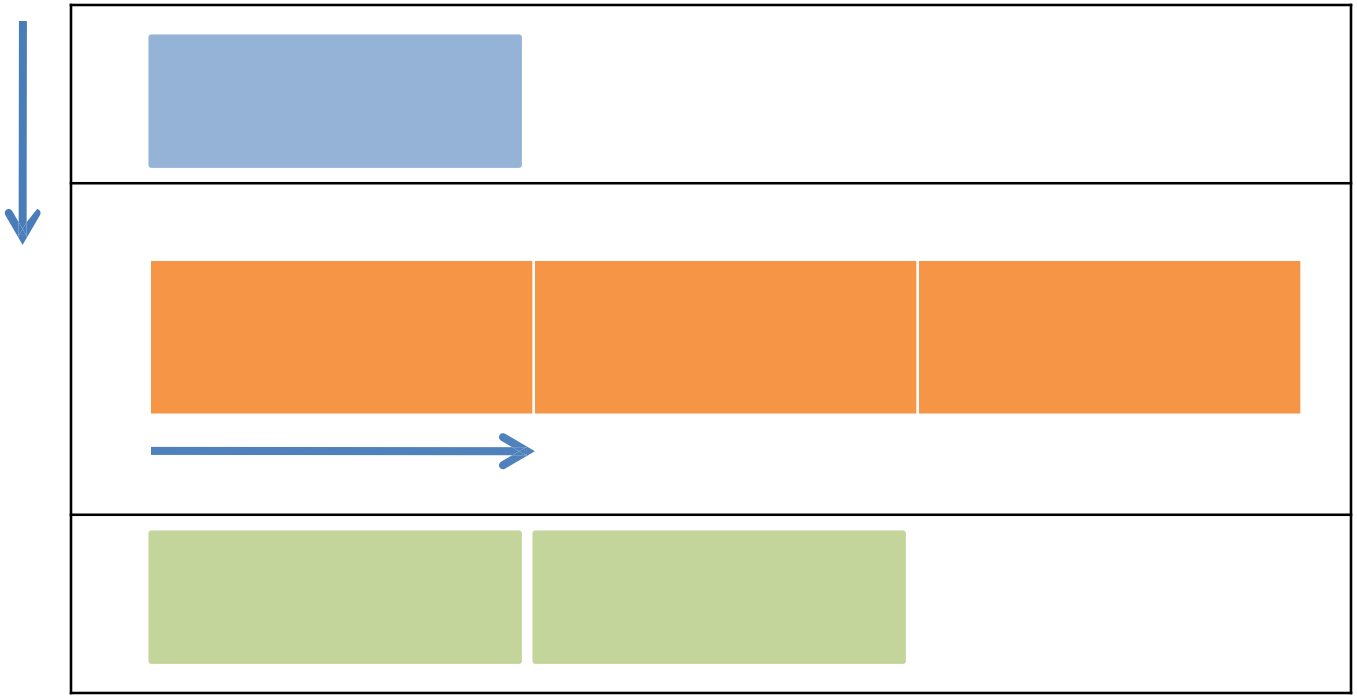


- FrameLayout jest najprostszym typem układu.
- Najczęściej wykorzystywany jest jako najbardziej zewnętrzny układ.
- Umożliwia zdefiniowanie jak duża część ekranu (wysokość, szerokość) ma zostać wykorzystana.
- Wszystkie jego elementy potomne pozycjonowane są względem górnego lewego końca ekranu.

Linear Layout

1. Linear Layout

- **LinearLayout** wspiera strategię wypełnienia według której elementy są ułożone w sposób horyzontalny lub wertykalny.
- Jeżeli orientacja układu ustawiona jest na pionową nowe wiersze (widoki) układane są jeden na drugim.
- Orientacja horyzontalna wykorzystuje rozmieszczenie jeden obok drugiego.



Linear Layout

1. LinearLayout: Ustawianie właściwości

Konfiguracja **LinearLayout** sprowadza się do ustawienia następujących właściwości:

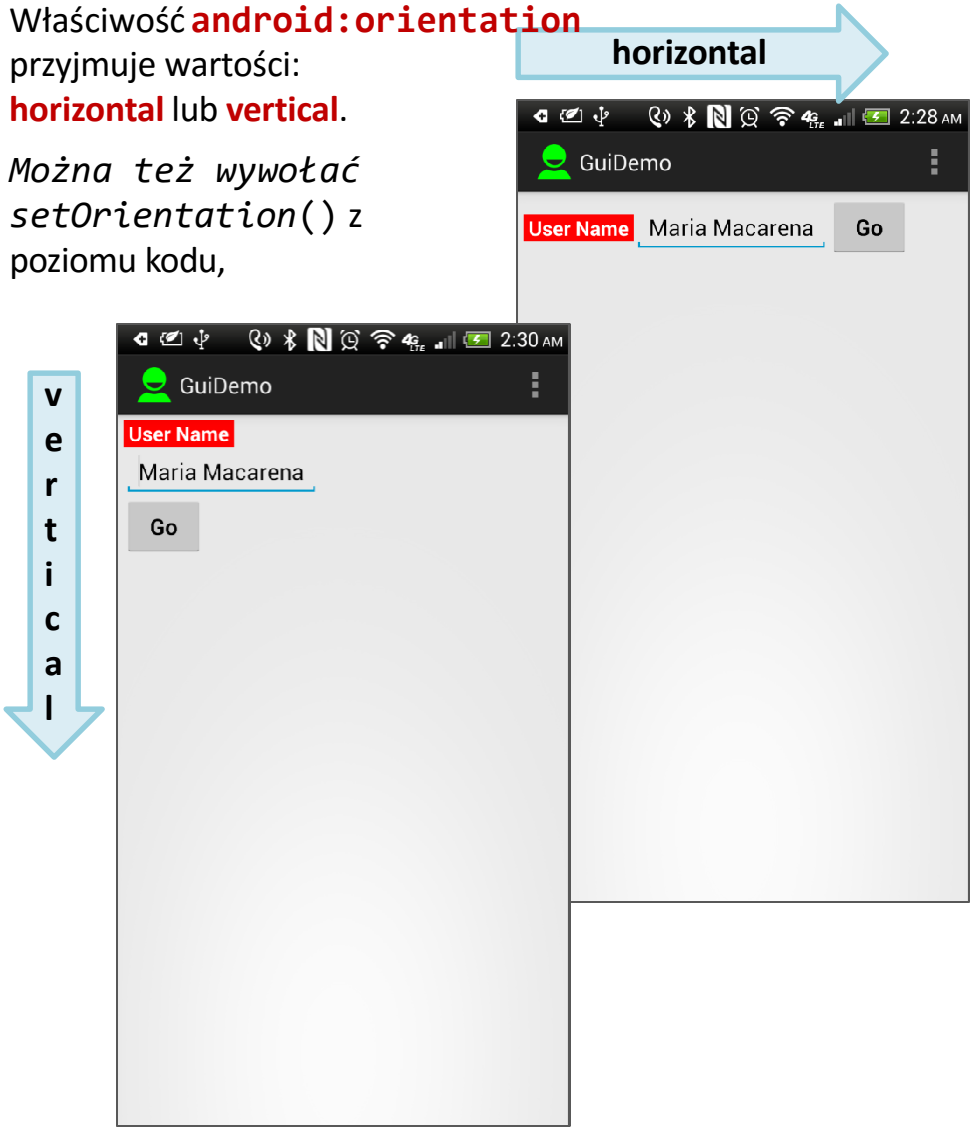
- orientation (*vertical, horizontal)*
- fill model (*match_parent, wrap_contents*)
- weight (*0, 1, 2, …n*)
- gravity (*top, bottom, center,…*)
- padding (*dp – dev. independent pixels*)
- margin (*dp – dev. independent pixels*)

LinearLayout - Orientation

1.1 Atrybut Orientation

Właściwość **android:orientation** przyjmuje wartości: **horizontal** lub **vertical**.

Można też wywołać setOrientation() z poziomu kodu,

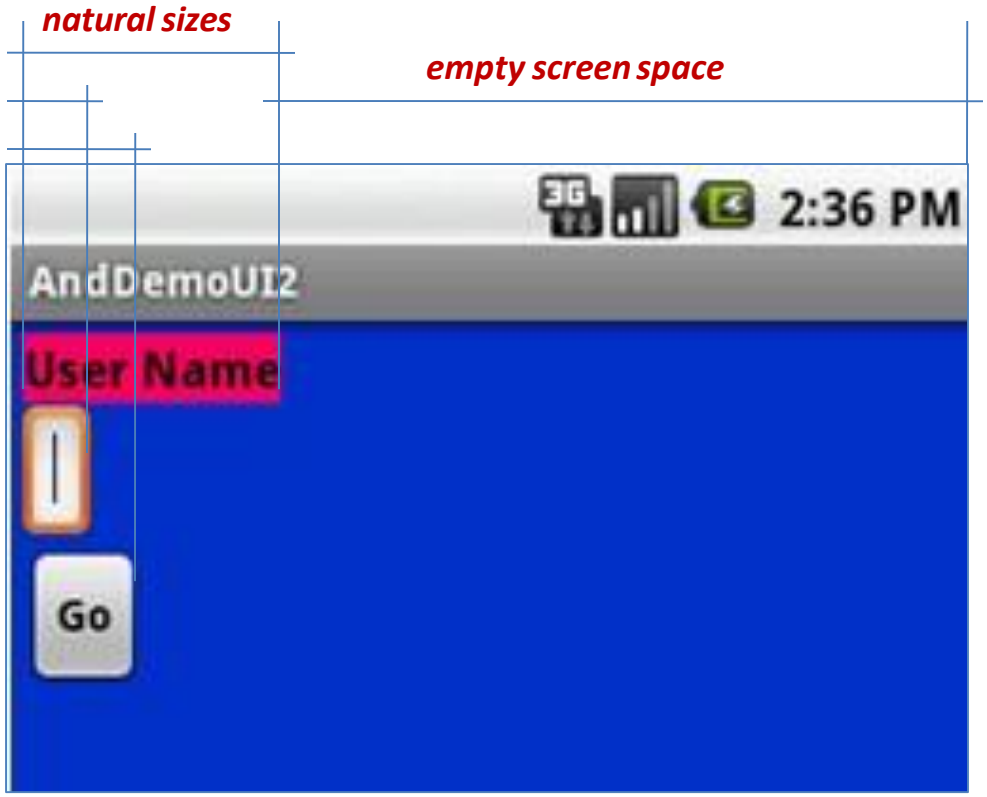



```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="4dp" >

    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#ffff0000"
        android:text=" User Name "
        android:textColor="#ffffffff"
        android:textSize="16sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/ediName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Maria Macarena"
        android:textSize="18sp" />

    <Button
        android:id="@+id/btnGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold" />

</LinearLayout>
```

LinearLayout – Fill Model

1.2 Fill Model

- Widżety posiadają "naturalny rozmiar“ określony na podstawie ich zawartości.
- W pewnych okolicznościach widżet powinien posiadać określony rozmiar (szerokość, wysokość) nawet jeżeli nie wprowadzono żadnego tekstu (tak jak na poniższym rysunku).



LinearLayout – Fill Model

1.2 Fill Model

Wszystkie widżety wewnątrz LinearLayout **muszą** posiadać nadane atrybuty ‘width’ i ‘height’.

```
android:layout_width
android:layout_height
```

Wartości używane do określenia szerokości czy wysokości to:

1. Określony rozmiar jak **125dp** (device independent pixels, a.k.a. **dip**)

2. **wrap_content** wskazuje, że widżet zachowuje swój naturalny rozmiar.

3. **match_parent** (kiedyś ‘**fill_parent**’) wskazuje, że widżet powinien być tak samo wielki jak jego rozdzic

LinearLayout – Fill Model

1.2 Fill Model

Medium resolution is: 320 x 480 dpi.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff0033cc
    "
    android:orientation="vertical
    "  android:padding="4dp" >

    <TextView
        android:layout_width="_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textColor="#ff000000"
        android:textSize="16sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/ediName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp" />

    <Button
        android:id="@+id/btnGo"
        android:layout_width="125dp"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold" />

</LinearLayout>
```

Row-wise

Use all the row

Specific size: 125dp

LinearLayout – Weight

1.2 Weight

Określa ile miejsca w LinearLayout powinien mieć zaalokowany widok. Należy użyć **0** jeżeli widok nie ma się rozciągać. Im większa waga, tym więcej miejsca posiada dany widżet na swoim poziomie hierarchii.

Przykład

Specyfikacja XML okna jest podobna jak w poprzednim przykładzie.

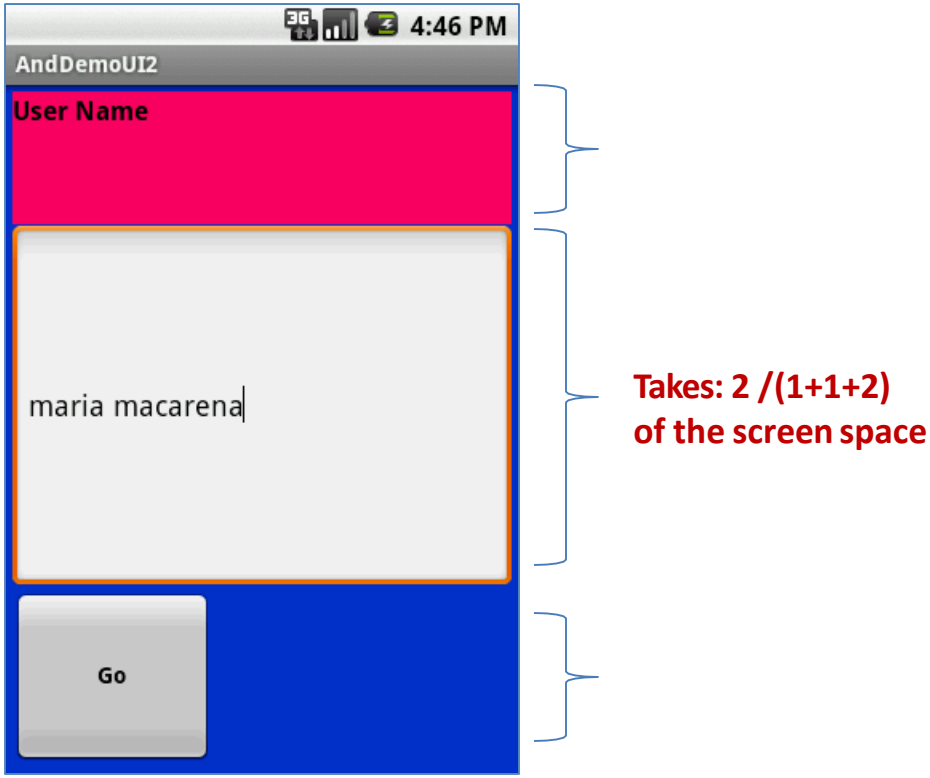
Kontroki TextView oraz Button mają dodatkowo właściwość:

```
android:layout_weight="1"
```

Gdzie komponent EditText ma

```
android:layout_weight="2"
```

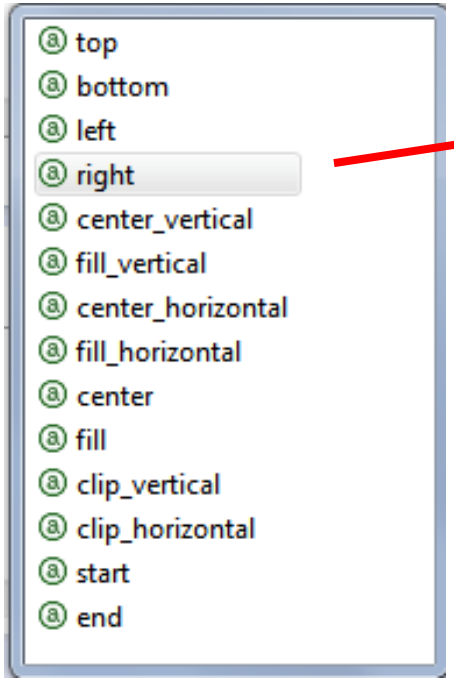
Domyślna wartość to 0

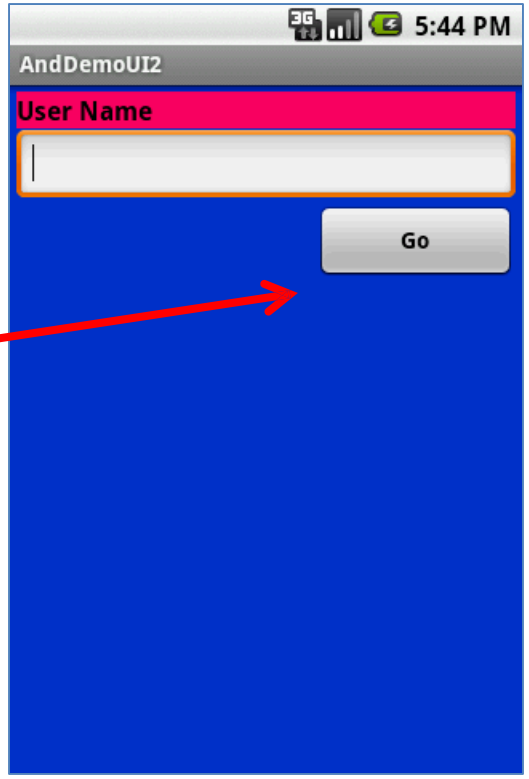


LinearLayout – Gravity

1.3 Layout_Gravity

- Używana by wskazać, jak wyrównane są elementy.
- Zwykle widżety pozycjonowane są do *lewej, górnej strony*.
- Wykorzystując w pliku XML właściwość
  `android:layout_gravity="…"`
  można ustawić inne wyrównanie:
  left, center, right, top, bottom, itp.





Button has **right** layout_gravity

The LinearLayout – Gravity

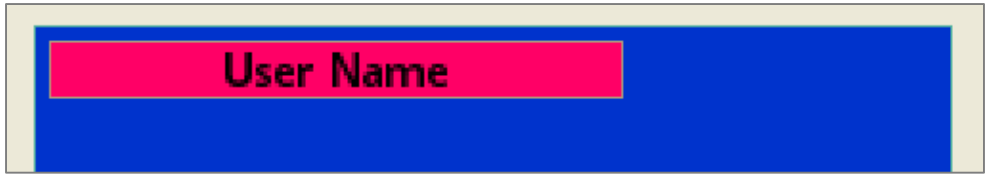
1.3 UWAGA: gravity vs. layout_gravity

Różnica względem:

android:gravity

Wskazuje jak zachowuje się obiekt wewnątrz kontenera. Tutaj tekst jest wycentrowany

```
android:gravity="center"
```



android:layout_gravity

Pozycjonuje widżet względem rodzica:

```
android:layout_gravity="center"
```

LinearLayout – Padding

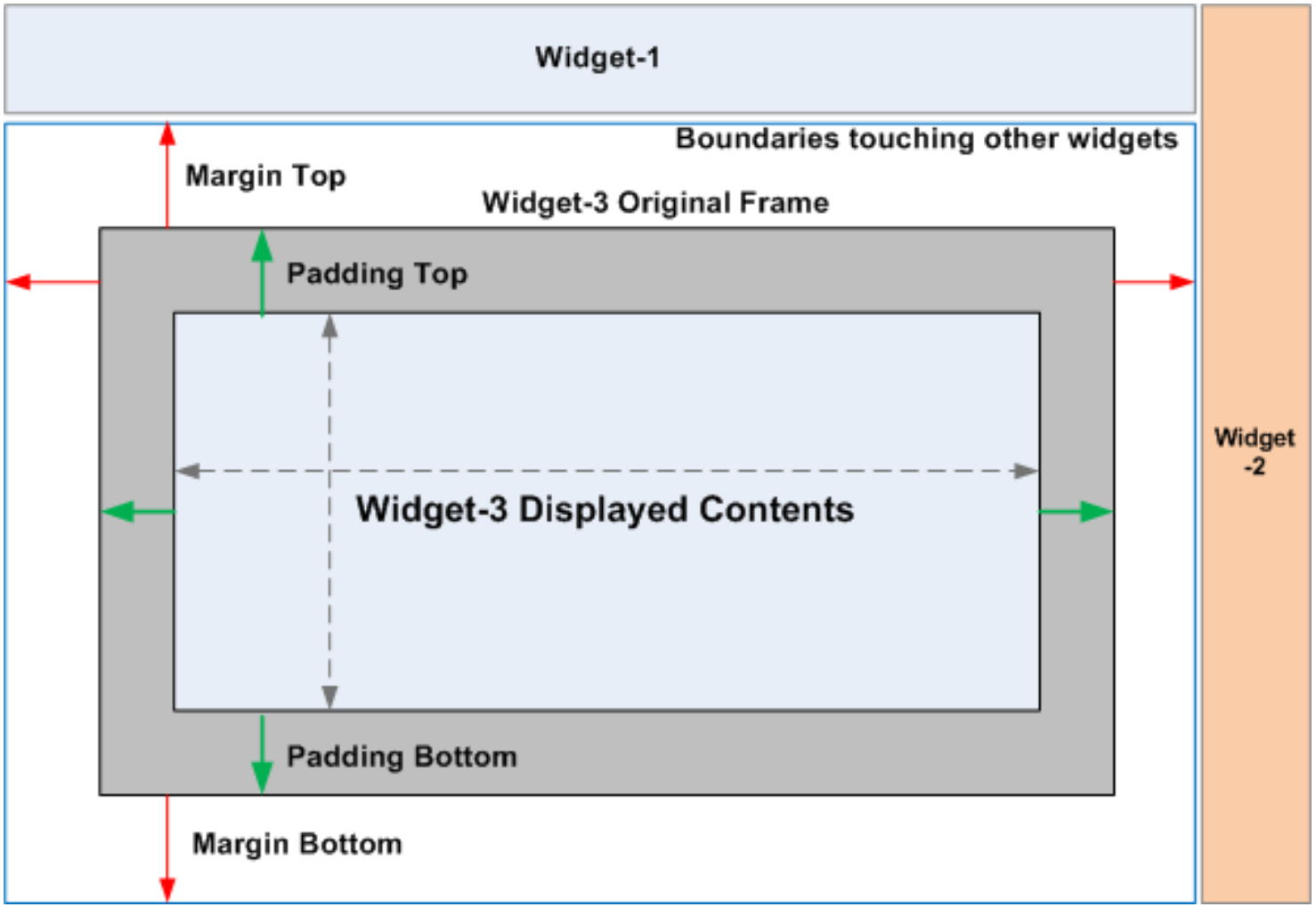
1.4 Padding

- Atrybut **padding** określa wewnętrzny margines widżetu (w **dp**).
- Wewnętrzny margines dodaje dodatkowe miejsce między obramowaniem widżetu a jego faktyczną zawartością.
- Używaj
  - `android:padding` (właściwości)
  - lub metody `setPadding()` z poziomu kodu

LinearLayout – Padding

1.3 Padding i Margin

Różnice między wewnętrznym i zewnętrznym marginesem

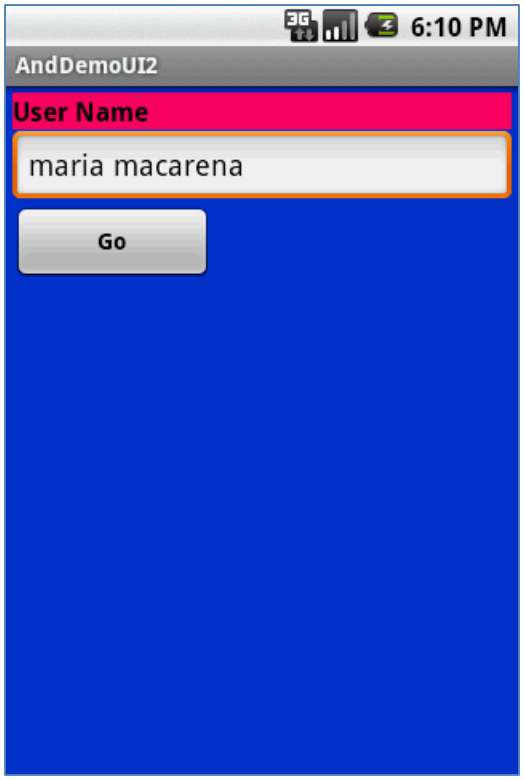


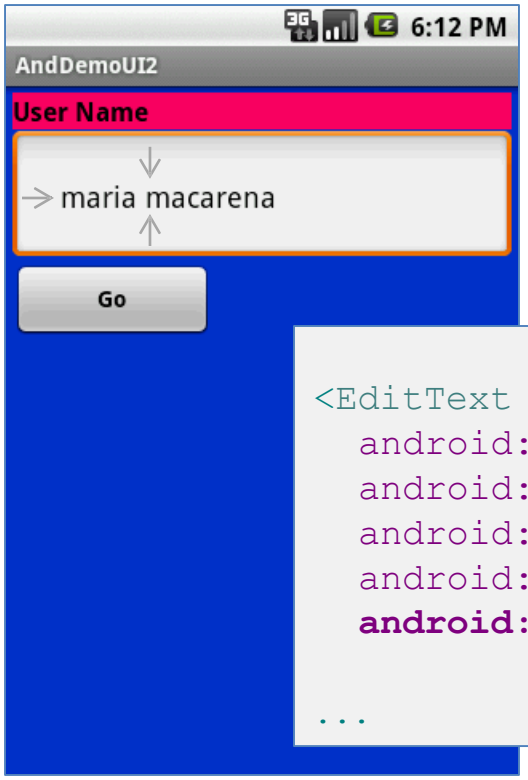
LinearLayout – Padding

1.3 Wewnętrzny margines używając padding

Przykład:

Komponentowi EditText nadano margines o wielkości 30dp



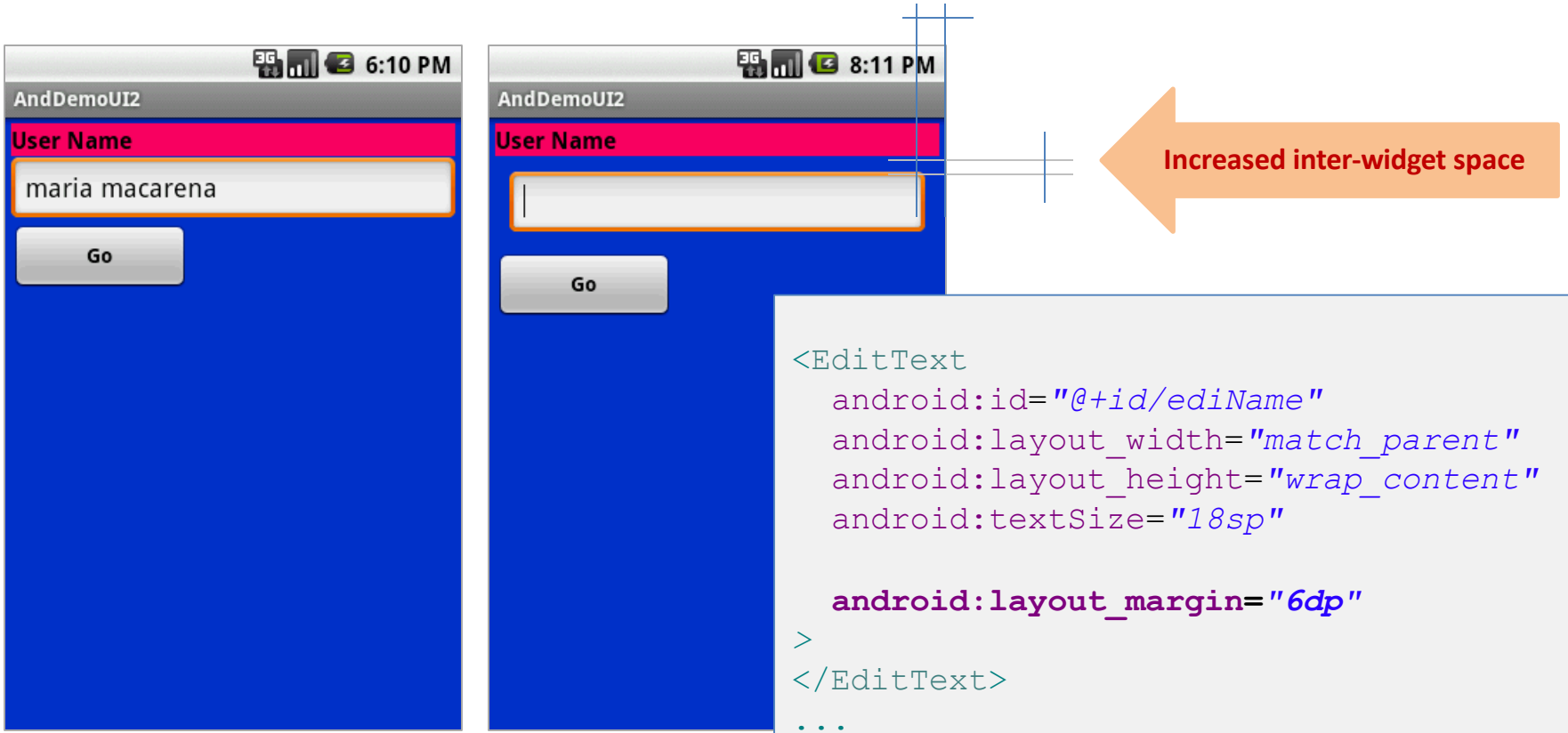



```
<EditText
  android:id="@+id/ediName"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"
  android:padding="30dp"   />

...
```

LinearLayout – Margin

1.4 Zewnętrzny margines

- Widżety domyślnie pozycjonowane są tuż obok siebie.
- Atrybut **android:layout_margin** umożliwia zwiększenie tego miejsca

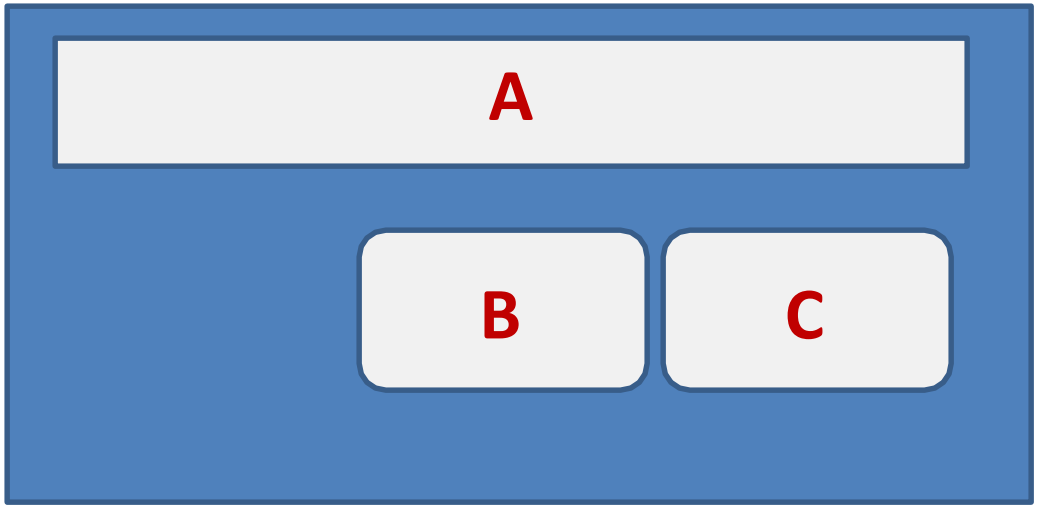



```
<EditText
  android:id="@+id/ediName"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"

  android:layout_margin="6dp"
>
</EditText>
...
```

Relative Layout

2. Relative Layout

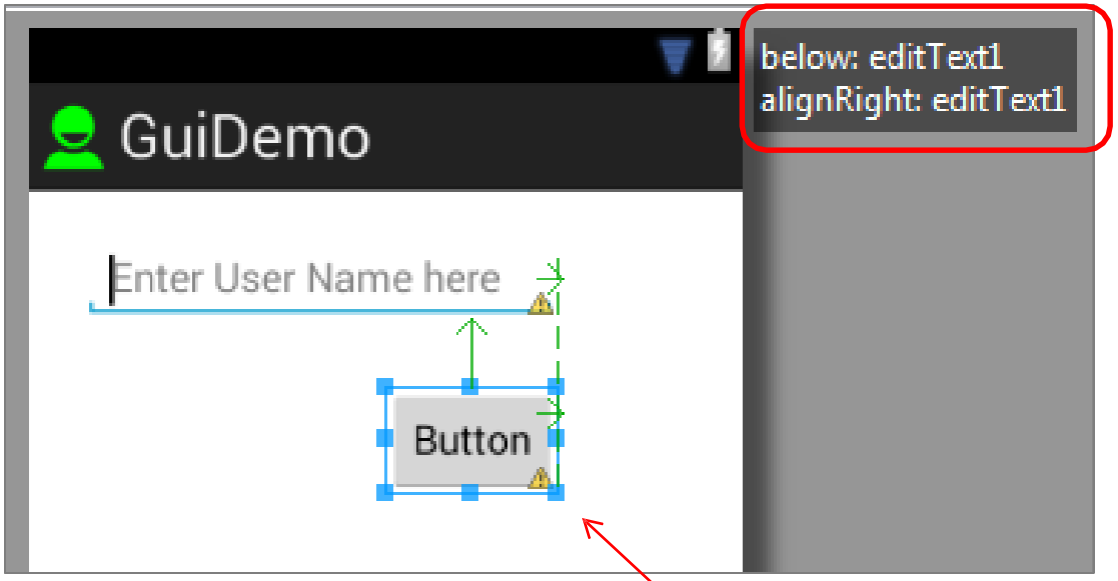
Rozmieszczenie widżetów w **RelativeLayout** uwzględnia relację sąsiedztwa do innych *widżetów w kontenerze* oraz w przodku danej hierarchii.



Przykład:
A jest w górnej części rodzica
C jest pod A, po jego prawej stronie
B jest pod A, na lewo od C

Relative Layout

2. Przykład: Relative Layout



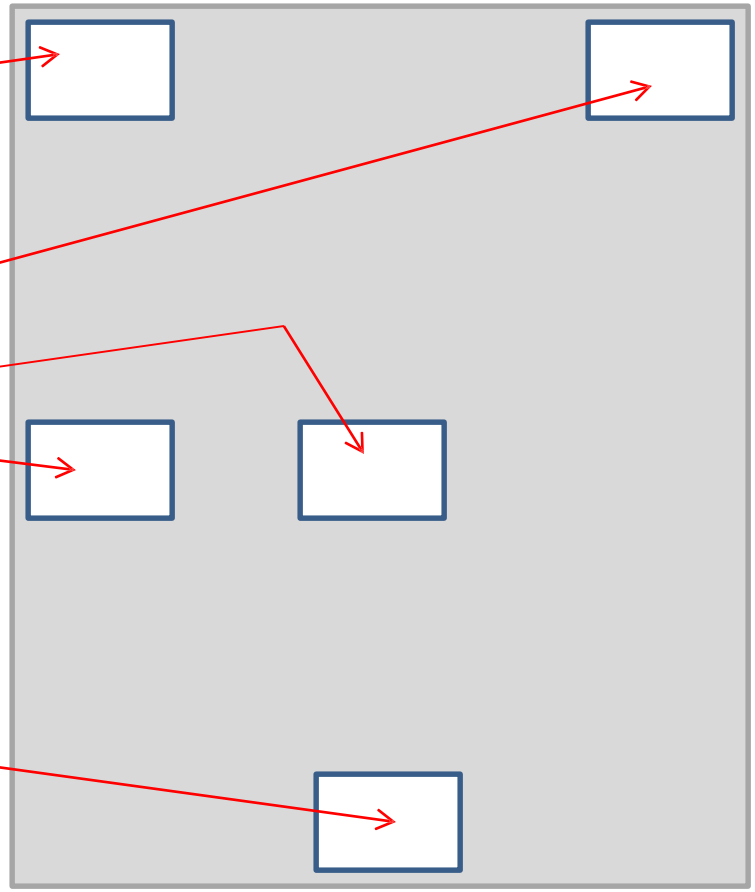
Lokalizacja przycisku jest wyrażona w odniesieniu do (relatywnej) pozycji pola tekstowego (EditText).

Relative Layout

2. Odniesienie do kontenera

Poniżej znajduje się lista atrybutów XML typu **boolean** (=“true/false”) określających pozycję widżetów względem jego **rodzica (kontenera)**.

```
android:layout_alignParentTop
android:layout_alignParentBottom

android:layout_alignParentLeft
android:layout_alignParentRight

android:layout_centerInParent
android:layout_centerVertical
android:layout_centerHorizontal
```

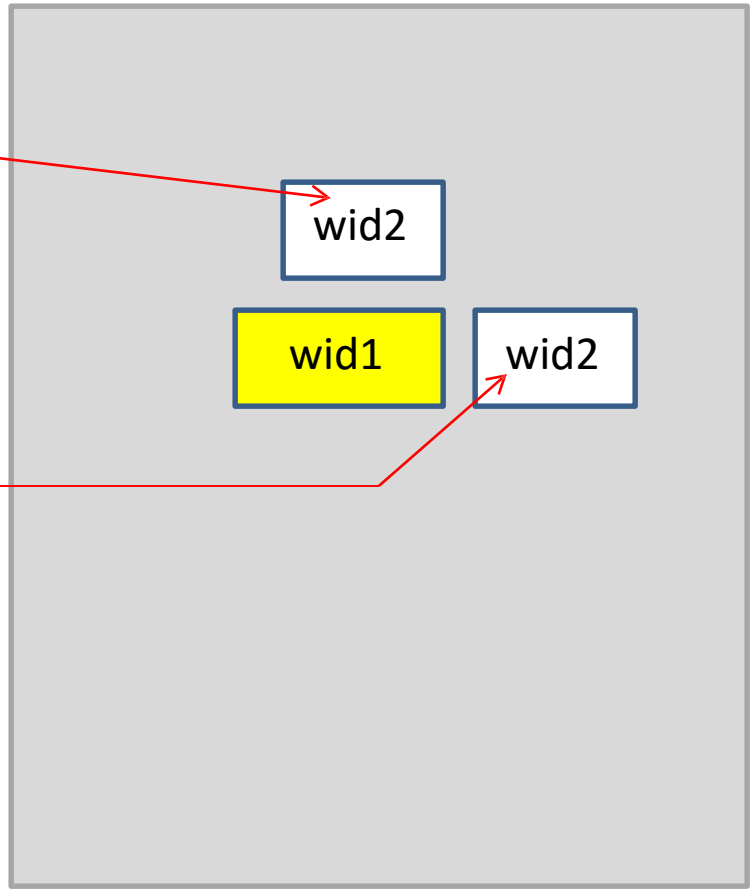


Relative Layout

2. Odniesienie do innych widżetów

Następujące atrybuty określają położenie widżetu **w odniesieniu do innych widżetów:**

```
android:layout_above="@+id/wid1"

android:layout_below

android:layout_toLeftOf

android:layout_toRightOf
```

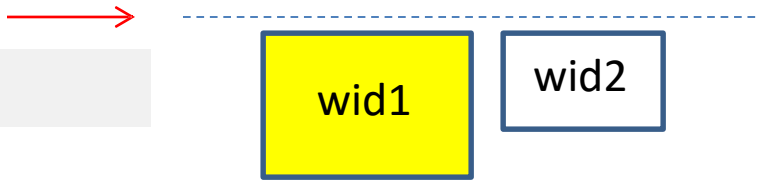


W tym przykładzie “wid2” jest w relacji sąsiedztwa z wid1 (“@+id/wid1”)

Relative Layout

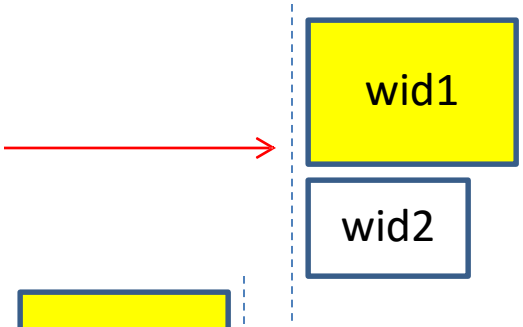
2. Odniesienie do innych widżetów c.d.

```
android:layout_alignTop="@+id/wid1"
```

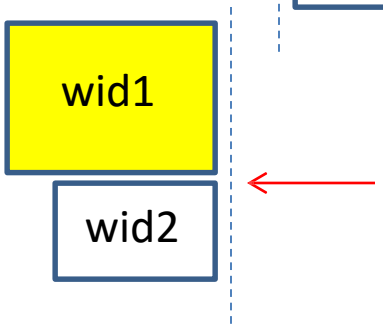



```
android:layout_alignBottom ="@+id/wid1"
```


```
android:layout_alignLeft="@+id/wid1"
```




```
android:layout_alignRight="@+id/wid1"
```



Relative Layout

2. Odniesienie do innych widżetów c.d.

Używając pozycjonowania relatywnego należy zadbać o:

1. Nadanie identyfikatorów (atrybut **`android:id`**) dla wszystkich elementów do których będziemy się odnosili.

2. Elementy XML są nazywane poprzez zapis: **`@+id/...`** Przykładowo pole tekstowe może być nazwane: `android:id="@+id/txtUserName"`

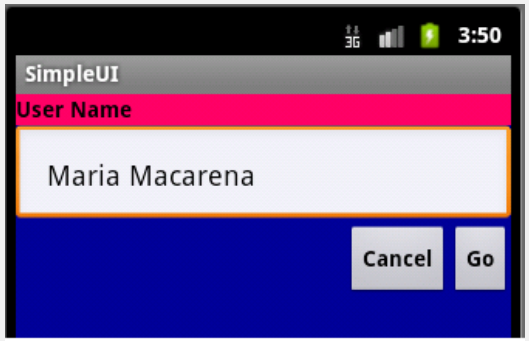
3. Można odwoływać się jedynie do uprzednio zdefiniowanych widżętów. Przykładowo, nowy komponent który znajduje się pod polem tekstowym *txtUserName można wypozycjonować jako*:

   `android:layout_below="@+id/txtUserName"`

Relative Layout

2. Przykład

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myRelativeLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff000099" >

    <TextView
        android:id="@+id/lblUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:background="#ffff0066
        "  android:text="User Name"
        android:textColor="#ff000000"
        android:textStyle="bold" >
    </TextView>
```




```
    <EditText
        android:id="@+id/txtUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/lblUserName"
        android:padding="20dp" >
    </EditText>

    <Button
        android:id="@+id/btnGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

android:layout_alignRight="@+id/txtUserName"
        android:layout_below="@+id/txtUserName"
        android:text="Go"
        android:textStyle="bold" >
    </Button>

    <Button
        android:id="@+id/btnCancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtUserName"
        android:layout_toLeftOf="@+id/btnGo"
        android:text="Cancel"
        android:textStyle="bold" >
    </Button>

</RelativeLayout>
```

Table Layout

3. Table Layout

1. Układ **TableLayout** wykorzystuje siatkę do pozycjonowania widżetów.
2. Podobnie jak w macierzy, komórki siatki identyfikowane są po numerach wierszy i kolumn.
3. Kolumny są responsywne – mogą się zwężąć bądź rozciągać by dostosować się do ich zawartości.
4. Klasa **TableRow** wykorzystywana jest do stworzenia nowego wiersza w którym widżety mogą zostać umieszczone.
5. Liczba kolumn TableRow jest wyznaczana przez liczbę widżetów umieszczonych w danym wierszu.

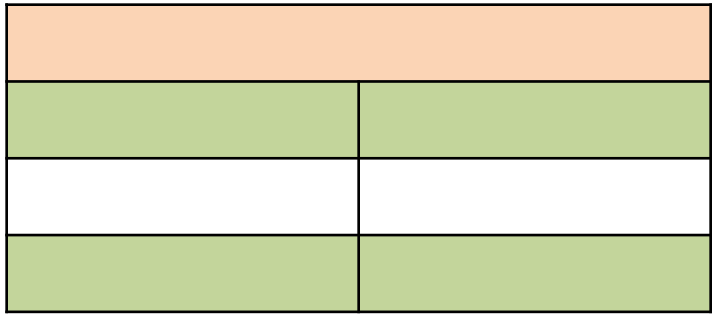


Table Layout

3. Table Layout – Ustawienie liczby kolumn

Liczba kolumn w ramach wiersza wyznacza jest przez system Android.

Przykład: Jeżeli TableLayout ma trzy wiersze, jeden z dwoma widżetami, jeden z trzema i jeden z czterema, zostaną stworzone przynajmniej 4 kolumny.

<table>
<tr><td colspan="2">0</td><td colspan="2">1</td></tr>
<tr><td colspan="2">0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td></tr>
</table>

Table Layout

3. Table Layout – Rozciąganie kolumn

- Pojedynczy widżet wewnątrz TableLayout może zajmować więcej niż jedną kolumnę.

- Właściwość `android:layout_span` wskazuje na jaką liczbę kolumn widżet może się rozciągnąć.

```
<TableRow>
    <TextView android:text="URL:" />
    <EditText android:id="@+id/txtData"
              android:layout_span="3" />
</TableRow>
```

Table Layout

3. Table Layout – Rozciąganie kolumn

Widżety w wierszu tabeli ustawiane są od lewej do prawej, rozpoczynając od pierwszej wolnej kolumny. Każda kolumna w układzie rozciąga się zgodnie z początkową zawartością danego widżetu.

Przykład: Pokazana tabela ma 4 kolumny (*indeksy*: 0,1,2,3). Nazwa (“*ISBN*”) tworzy pierwszą kolumnę (*indeks 0*). Pole tekstowe wykorzystuje atrybut `layout_span` by zostać rozciągniętym na 3 kolumny.

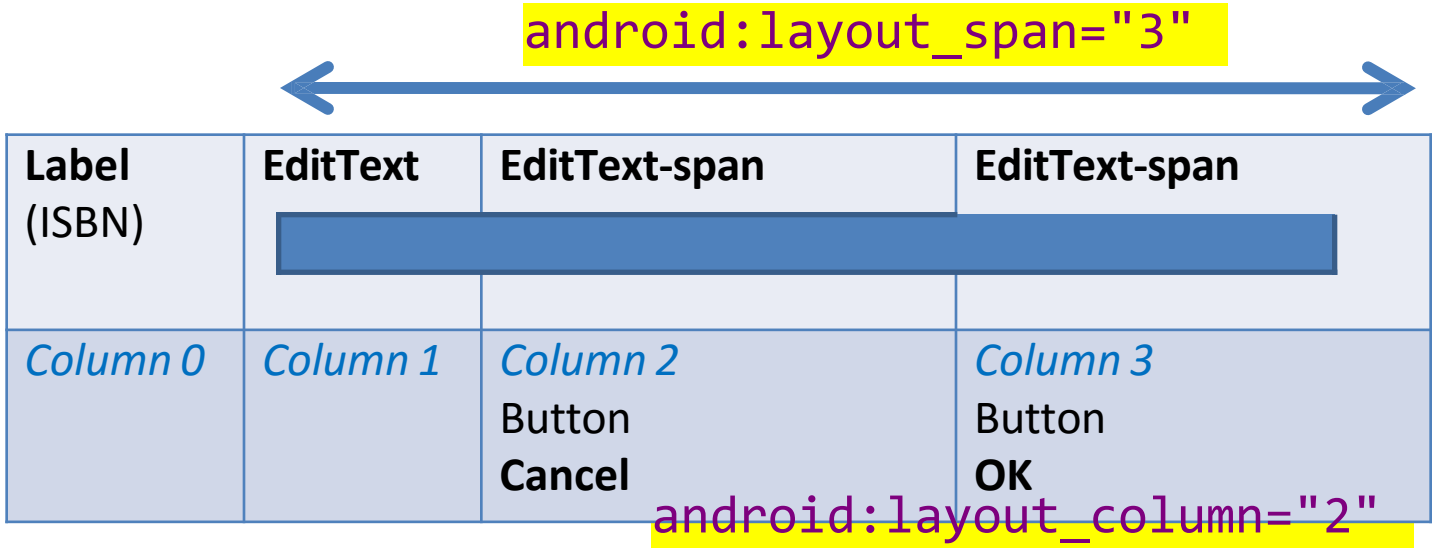


Table Layout – przykład 2

```
<TableLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myTableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >

    <TableRow>
        <TextView
            android:background="#FF33B5E5"  android:text="Item " />
        <TextView
            android:layout_marginLeft="5dp"
            android:background="#FF33B5E5"
            android:text="Calories " />
        <TextView
            android:layout_marginLeft="5dp"
            android:background="#FF33B5E5"
            android:text="Price $ " />
    </TableRow>

    <View
        android:layout_height="1dp"
        android:background="#FF33B5E5" />

    <TableRow>

        <TextView
            android:text="Big Mac" />
        <TextView
            android:gravity="center"
            android:text="530" />
        <TextView
            android:gravity="center"
            android:text="3.99" />
        <Button
            android:id="@+id/btnBuyBigMac"
            android:gravity="center"
            android:text="Buy" />
    </TableRow>

    <View
        android:layout_height="1dp"
        android:background="#FF33B5E5" />

    <!-- other TableRows ommitted --!>

</TableLayout>
```

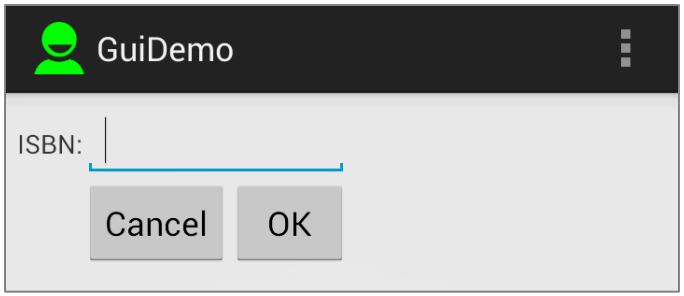
Table Layout

3. Table Layout - przykład

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myTableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="6dp"
    android:orientation="vertical" >
    <TableRow>
        <TextView android:text="ISBN:" />
        <EditText
            android:id="@+id/ediISBN"
            android:layout_span="3" />
    </TableRow>

    <TableRow>
        <Button
            android:id="@+id/cancel"
            android:layout_column="2"
            android:text="Cancel" />
        <Button
            android:id="@+id/ok"
            android:text="OK" />
    </TableRow>

</TableLayout>
```

Zajmij 3 kolumny

Pomiń kolumny 0, 1



Spróbuj zmienić
layout_span na 1, 2, 3

Table Layout

3. Rozciąganie całej tabeli

- Domyślnie, kolumna jest szeroka jak “naturalny” rozmiar najszerszego widżetu zawartego w tej kolumnie (przykładowo kolumna z przyciskiem “Go” jest węższa niż kolumna z przyciskiem “Cancel”).

- Tabela niekoniecznie musi zajmować całą dostępną przestrzeń w poziomie.

- Jeżeli tabela ma (w poziomie) być zrównana z kontenerem można skorzystać z opcji:

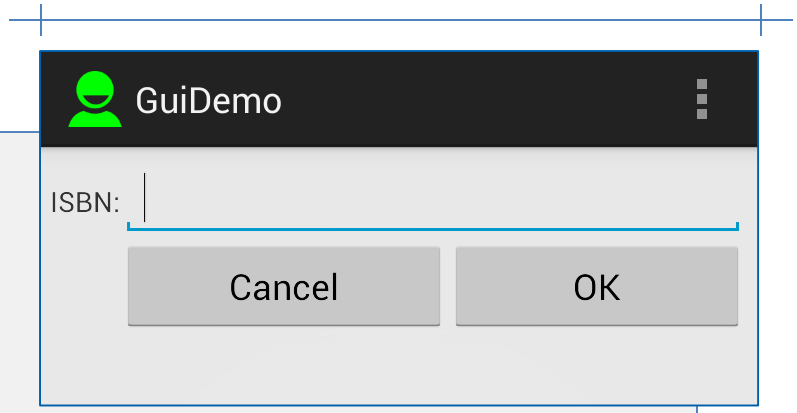
```
android:stretchColumns=“column(s)”
```

Jej wartość to indeks kolumny (bądź lista indeksów kolumn) która ma zostać rozciągnięta by zająć całą dostępną przestrzeń w ramach wiersza.

Table Layout

3. Przykład: Rozciąganie kolumn

W tym przykładzie kolumny 2 i 3 zajmują całą (wolną) dostępną przestrzeń w poziomie.



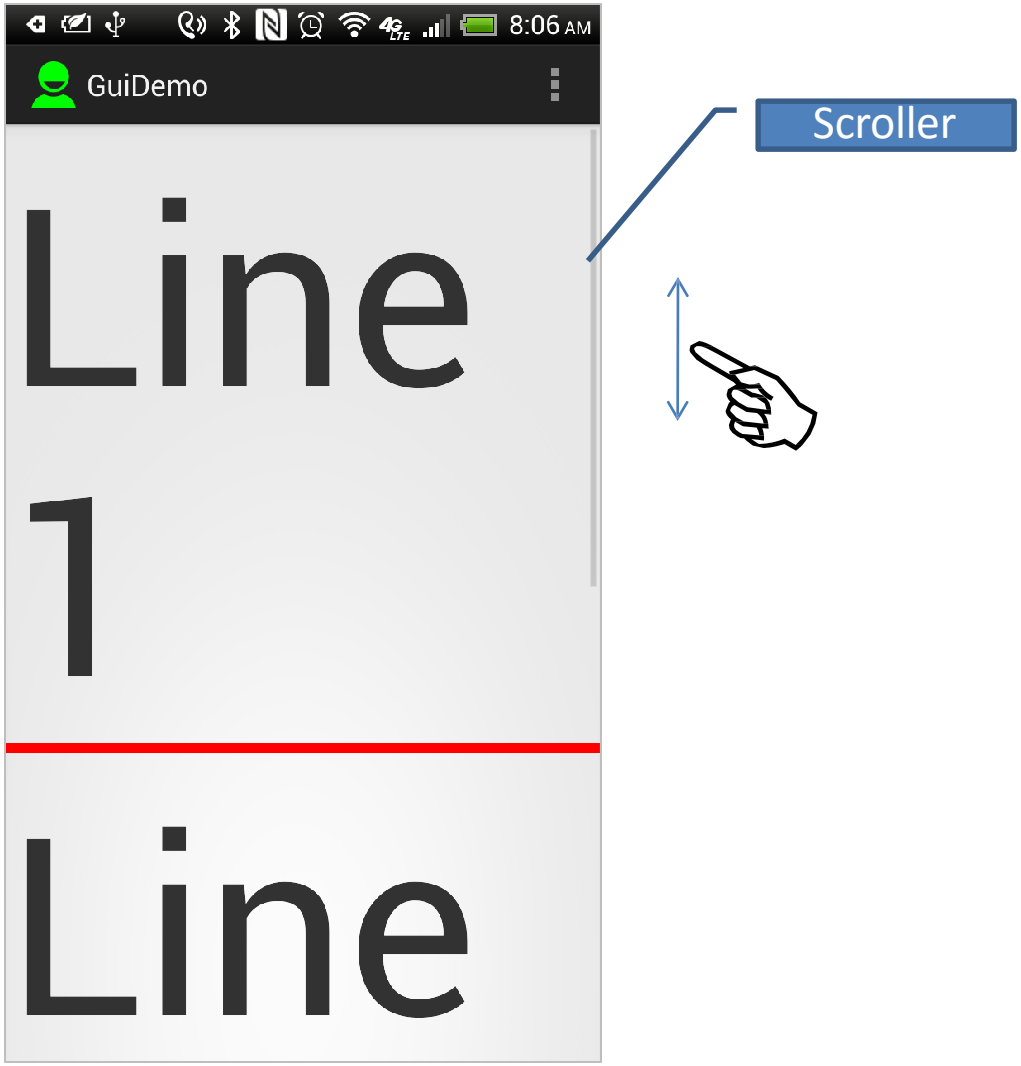

```
...
<TableLayout
  android:id="@+id/myTableLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:stretchColumns="2,3"
  xmlns:android="http://schemas.android.com/apk/res/android"
>

...
```

***Do zrobienia**: spróbuj rozciągania z innymi kolumnami.*

ScrollView

4. ScrollView Layout

- Komponent **ScrollView** jest przydatny w sytuacjach, gdy do wyświetlenia jest więcej danych niż można pomieścić na danym ekranie.
- ScrollView zapewnia dostęp do danych z użyciem pasków przewijania.
- Jedynie porcja danych jest wyświetlana naraz. Pozostała część jest ukryta.



ScrollView

4. Przykład: ScrollView Layout

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

<ScrollView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myScrollView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:id="@+id/myLinearLayoutVertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line1"
            android:textSize="150dp" />
        <View
            android:layout_width="match_parent"
            android:layout_height="6dp"
            android:background="#ffff0000" />
        <TextView
            android:id="@+id/textView2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line2"
            android:textSize="150dp" />

        <View
            android:layout_width="match_parent"
            android:layout_height="6dp"
            android:background="#ffff0000" />

        <TextView
            android:id="@+id/textView3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Line3"
            android:textSize="150dp" />
    </LinearLayout>

</ScrollView>
```

Przykład HorizontalScrollView Layout

```
<HorizontalScrollView

xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myHorizontalScrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <LinearLayout
        android:id="@+id/myLinearLayoutVertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Item1"
            android:textSize="75sp" />
         <View
            android:layout_width="6dp"
            android:layout_height="match_parent"
            android:background="#ffff0000" />
        <TextView
            android:id="@+id/textView2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Item2"
            android:textSize="75sp" />

        <View
          android:layout_width="6dp"
          android:layout_height="match_parent"
          android:background="#ffff0000" />

        <TextView
            android:id="@+id/textView3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Item3"
            android:textSize="75sp" />
    </LinearLayout>

</HorizontalScrollView>
```

Absolute Layout

5. Absolute Layout

- Układ pozwala podać dokładną pozycje (koordynaty x/y) jego potomków.
- Pozycjowanie absolutne *jest mniej elastyczne i trudniejsze w utrzymaniu* niż w przypadku pozostałych układów opartych o relację między komponentami.

Absolute Layout

5. Przykład Absolute Layout

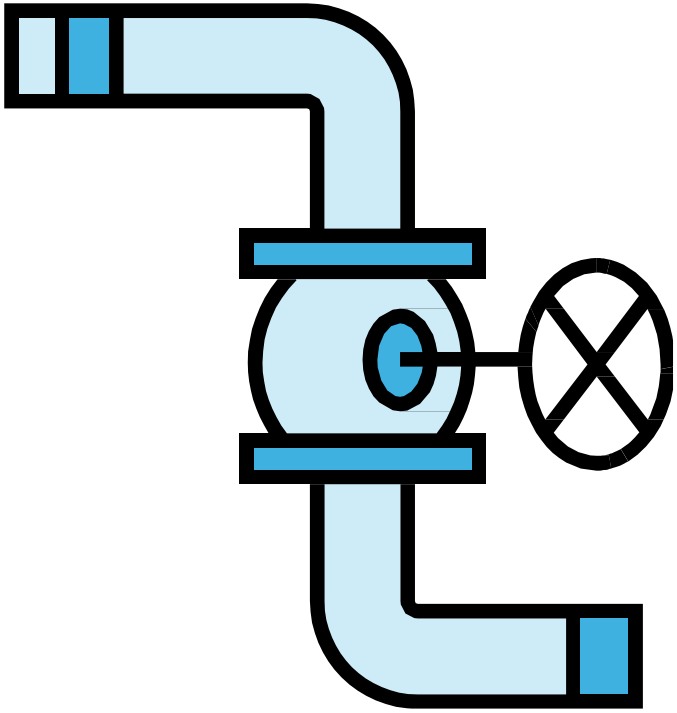
```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
android:id="@+id/myLinearLayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="#ff0033cc"
android:padding="4dp"
xmlns:android="http://schemas.android.com
/apk/res/android"
>

<TextView
android:id="@+id/tvUserName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textSize="16sp"
android:textStyle="bold"
android:textColor="#ff000000"
android:layout_x="0dp"
android:layout_y="10dp"
>
</TextView>
<EditText
android:id="@+id/etName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:layout_x="0dp"
android:layout_y="38dp"
>
</EditText>

<Button
android:layout_width="120dp"
android:text="Go"
android:layout_height="wrap_content"
android:textStyle="bold"
android:id="@+id/btnGo"
android:layout_x="100dp"
android:layout_y="170dp"  />
</AbsoluteLayout>
```


Button location

Powiązanie układu z kodem Java

Należy „połączyć” elementy XML i odpowiadające im obiekty w ramach instancji klasy aktywności napisanej w Javie.

XLM Layout
<xml….
. . .
. . .
</xml>



JAVA code
public class ….
{
. . .
. . .
}

Powiązanie układu z kodem Java

Załóżmy, że GUI zostało stworzone w *res/layout/main.xml*. Ten układ może zostać powiązany z daną aktywnością poprzez instrukcję:

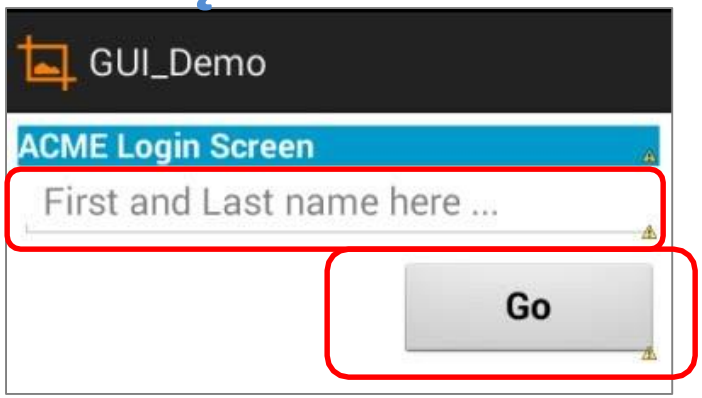
```
setContentView(R.layout.main);
```

Poszczególne widżety, jak np. *myButton* mogą zostać zaadresowane z pomocą polecenia `findViewByID(...)` tzn:

```
Button btn= (Button) findViewById(R.id.myButton);
```

gdzie **R** jest klasą automatycznie wygenerowaną by możliwe było śledzenie zasobów dostępnych dla danej aplikacji. W szczególności **R.id...** stanowi kolekcję widżetów zdefiniowanych w ramach interfejsu/ów XML.

Powiązanie układu z kodem Java




```
<!- XML LAYOUT -->
<LinearLayout
    android:id="@+id/myLinearLayout"
    ... >

    <TextView
        android:text="ACME Login Screen"
        ...  />

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

    <Button
        android:id="@+id/btnGo"
        ...  />

</LinearLayout>
```

Java code

```
package csu.matos.gui_demo;
import android…;

public class MainActivity extends Activity {

    EditText edtUserName;

    Button   btnGo;

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

    setContentView(R.layout.activity_main);

    edtUserName = (EditText) findViewById(R.id.edtUserName);

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

  }

 ...
}
```

Powiązanie układu z kodem Java

Dodawanie nasłuchiwaczy do widżetów

W tym momencie przycisk btn może zostać wykorzystany np. poprzez zdefiniowanie dla niego reakcji na kliknięcie:

```
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        updateTime();
    }
});

private void updateTime() {
    btn.setText(new Date().toString());
}
```

Czym jest kontekst?

Dodatek

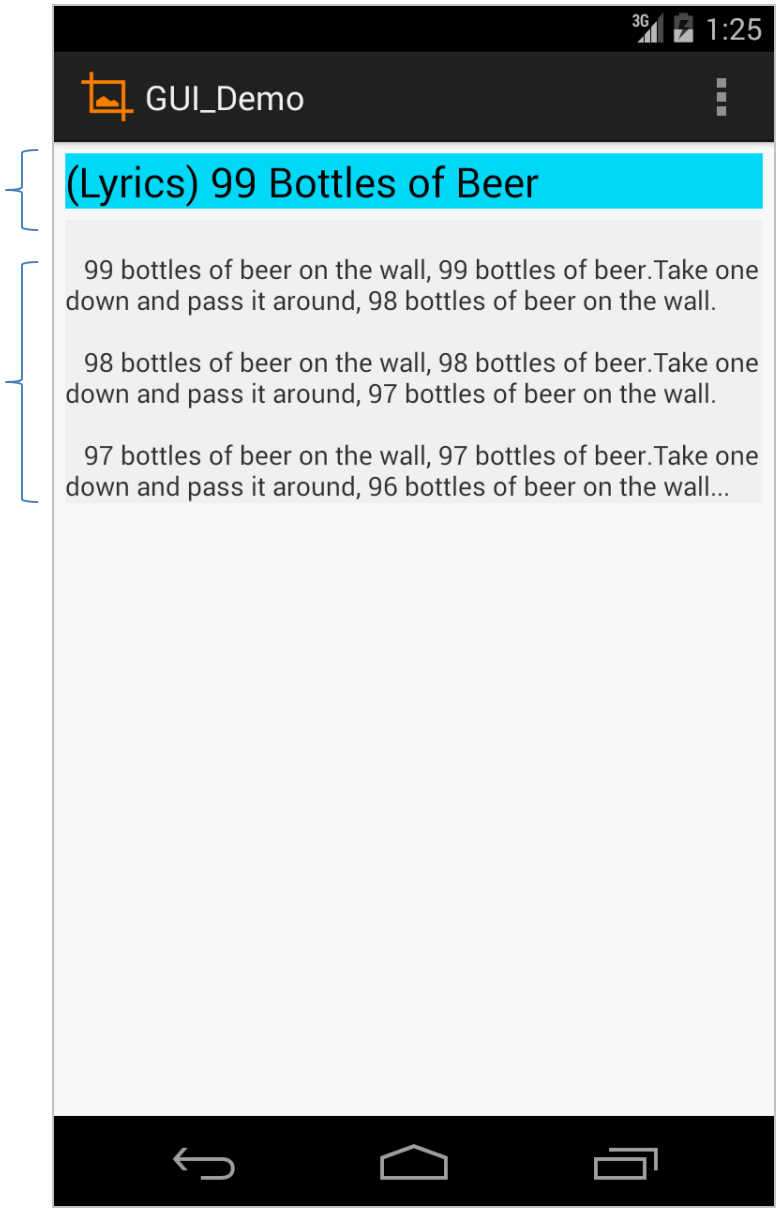
Dla Androida **Context** definiuje tzw. **środowisko** na którym aplikacja może tworzyć i używać zasobów.

- Kiedy widżet jest tworzony, jest mu przyporządkowany określony kontekst. Poprzez afiliację do tego środowiska, ma on wówczas dostęp do innych składowych hierarchii do której został przyporządkowany.

- Dla prostej aplikacji składającej się wyłącznie z jednej aktywności np. MainActivity metoda **getApplicationContext()** oraz referencja **MainActivity.this** zwracają ten sam rezultat.

- Aplikacja ma jednak zwykle **wiele aktywności**. W takiej sytuacji dostępny jest jeden globalny kontekst aplikacji oraz po jednym kontekście dla każdej z aktywności, każdy umożliwiający dostęp do zasobów zdefiniowanych w ramach tego kontekstu.

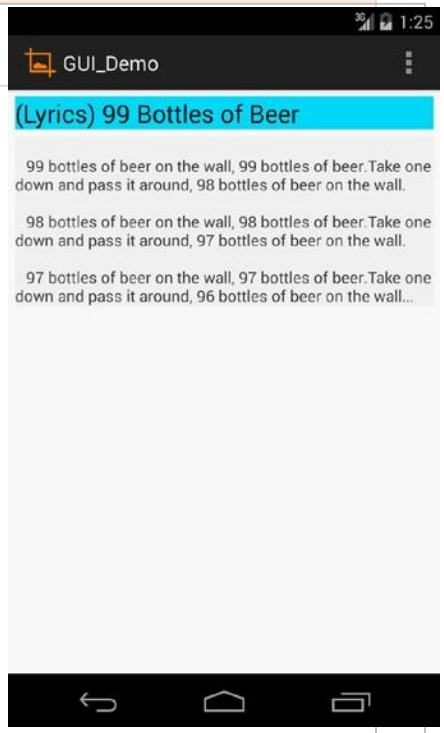
Etykiety



- Etykieta w kontekście Androida nazywana jest **TextView**.
- Zwykle wykorzystywana jest do prezentacji danych tekstowych bądź opisywania komponentów aplikacji.
- Etykiety nie umożliwiają edycji tekstu przez użytkownika.
- Tekst może zawierać pewne znaki specjalne jak np. **\n** (nowa linia)
- Można również wykorzystać formatowanie HTML:

```
Html.fromHtml("<b>bold</b> string")
```

Etykiety - przykład

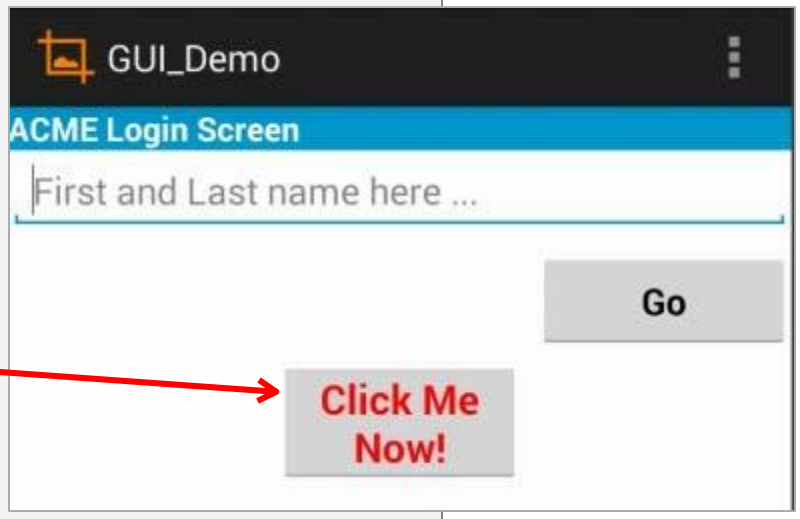



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/holo_blue_bright"
        android:text="(Lyrics) 99 Bottles of Beer"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="6dp"
        android:background="@color/gray_light"
        android:text="\n\t99 bottles of beer on the wall, 99 bottles of beer.Take one down and
pass it around, 98 bottles of beer on the wall.\n\n\t98 bottles of beer on the wall, 98 bottles
of beer.Take one down and pass it around, 97 bottles of beer on the wall. \n\n\t97 bottles of
beer on the wall, 97 bottles of beer.Take one down and pass it around, 96 bottles of beer on
the wall... "
        android:textSize="14sp" />

</LinearLayout>
```

Przyciski

- Widżet **Button** jest graficzną reprezentacją standardowego przycisku.
- **Button** jest podklasą **TextView**. Dlatego formatowanie przycisku przebiega podobnie jak formatowanie **TextView**.
- Można zmienić domyślny wygląd przycisku poprzez realizację własnej specyfikacji *drawable.xml (podanej jako „tło”)*. Można wówczas wskazać kształt, kolor, obramowanie, wierzchołki, gradient oraz wygląd poszczególnych stanów (naciśnięty, posiada focus).

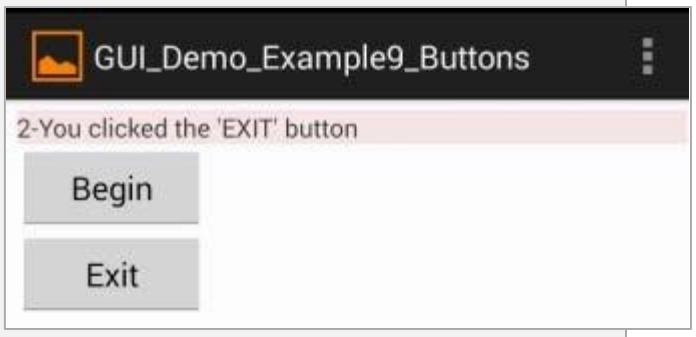
```
<Button
        android:id="@+id/btnClickMeNow"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="5dp“
        android:gravity="center"
        android:padding="5dp"
        android:text="Click Me Now!"
        android:textColor="#ffff0000"
        android:textSize="20sp"
        android:textStyle="bold" />
```



Podłączanie wielu przycisków

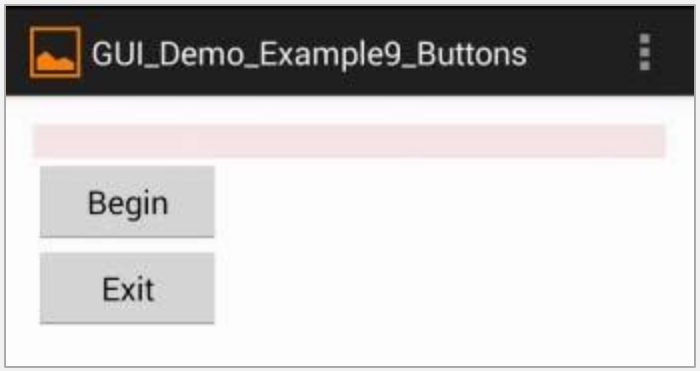
```
public class MainActivity extends Activity implements OnClickListener {
    TextView txtMsg;
    Button btnBegin;
    Button btnExit;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main );

        txtMsg = (TextView) findViewById(R.id.txtMsg);
        btnBegin = (Button) findViewById(R.id.btnBegin);
        btnExit = (Button) findViewById(R.id.btnExit);

        btnBegin.setOnClickListener(this);
        btnExit.setOnClickListener(this);
    }//onCreate


    @Override
    public void onClick(View v) {
        if (v.getId() == btnBegin.getId()) {
            txtMsg.setText("1-You clicked the 'BEGIN' button");
        }
        if (v.getId() == btnExit.getId()) {
            txtMsg.setText("2-You clicked the 'EXIT' button");
        }

    }//onClick
}
```

Proszę zwrócić uwagę na implementację interfejsu **OnClickListener**, jako alternatywną wersje obsługi wielu przycisków. Metoda **onClick** sprawdza, który z przycisków jest źródłem zdarzenia i wykonuje zadaną akcję.



Podłączanie wielu przycisków

```
<LinearLayout 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:orientation="vertical"
    android:padding="6dp" >

    <TextView
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#88eed0d0" />

    <Button
        android:id="@+id/btnBegin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="5"
        android:text="Begin" />

    <Button
        android:id="@+id/btnExit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="5"
        android:text="Exit" />

</LinearLayout>
```



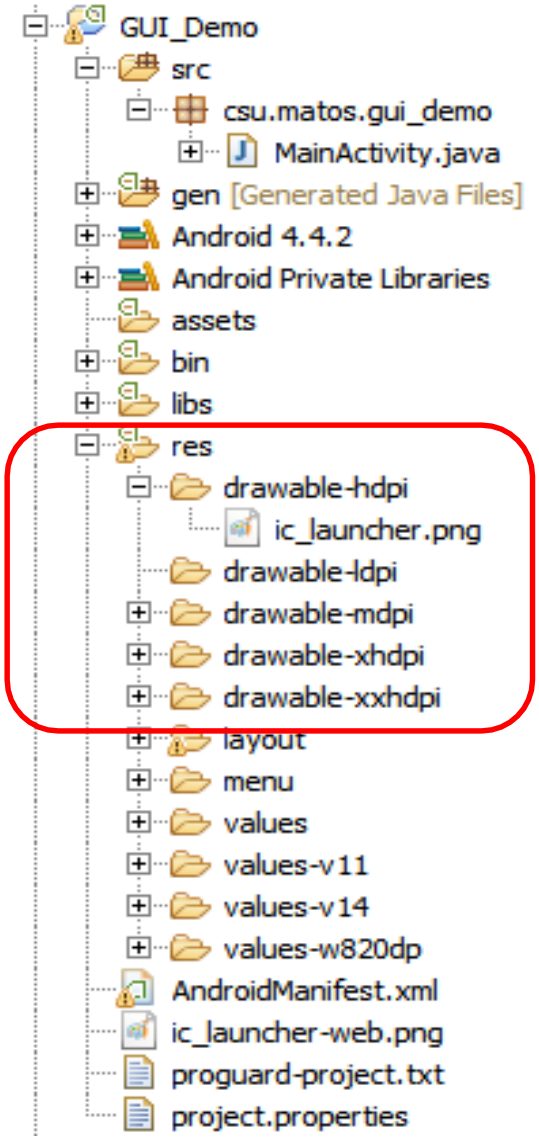
ImageView i ImageButton

- **ImageView** oraz **ImageButton** pozwalają na umieszczenie grafik (gif, jpg, png, etc).

- Stanowią analogię dla komponentów odpowiednio *TextView* oraz *Button*

- Każdy taki widżet posiada atrybut
  `android:src` lub
  `android:background`
  (w pliku XML) pozwalający określić używany obrazek.

- Obrazki przechowywane są w folderze **res/drawable** (opcjonalnie można dodać różne wersje tego samego obrazka w katalogach z przedrostkami *medium, high, x-high, xx-high i xxx-high*). Szczegóły dostępne są na: http://developer.android.com/design/style/iconography.html



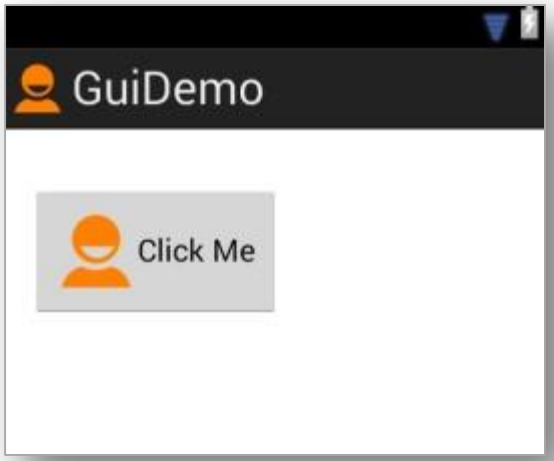
ImageView i ImageButton

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="6dp"
    android:orientation="vertical" >

    <ImageButton
        android:id="@+id/imgButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" >
    </ImageButton>

    <ImageView
        android:id="@+id/imgView1"
        android:layout_width="200dp"
        android:layout_height="150dp"
        android:scaleType="fitXY"
        android:src="@drawable/flowers1" >
    </ImageView>

</LinearLayout>
```

ImageView i ImageButton

Zwykły przycisk typu **Button** też może wyświetlać grafikę, ale **ImageButton** ma więcej opcji

```
<LinearLayout
    . . .

 <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:drawableLeft="@drawable/ic_launcher"
        android:gravity="left|center_vertical"
        android:padding="15dp"
        android:text="Click me" />

</LinearLayout>
```



Jak Android wykorzystuje ikony?

Ikony – małe obrazki reprezentujące aplikację bądź jej część. Mogą pojawiać się w wielu miejscach aplikacji takich jak:

- Ekran domowy
- Launcher
- Menu główne
- Action Bar
- Pasek stanu
- Zakładkach
- Dialogach
- Listach

Więcej informacji dostępnych jest pod:
http://developer.android.com/design/style/iconography.html

WSKAZÓWKA: Wiele witryn oferuje darmową konwersję między różnymi formatami graficznymi:
http://www.prodraw.net/favicon/index.php
http://converticon.com/

mdpi (761 bytes)
1x = 48 x 48 pixels
BaseLine

hdpi (1.15KB)
1.5x = 72 x 72 px

x-hdpi (1.52KB)
2x = 96 x 96 px

xx-hdpi (2.47KB)
3x = 144 x 144 px

Pola tekstowe

- Widżet EditText jest rozszerzeniem TextView umożliwiającym wprowadzenie tekstu przez użytkownika.

- Prócz zwykłego tekstu można używać podstawowych znaczników HTML by uzyskać pogrubienie, podkreślenie, kursywę. Umożliwia to metoda: **Html.fromHtml(html_text)**

- Dostęp do tekstu (pobieranie, modyfikacja) możliwy jest dzięki metodom:

```
txtBox.setText("tekst")
```

```
txtBox.getText().toString()
```

Pola tekstowe

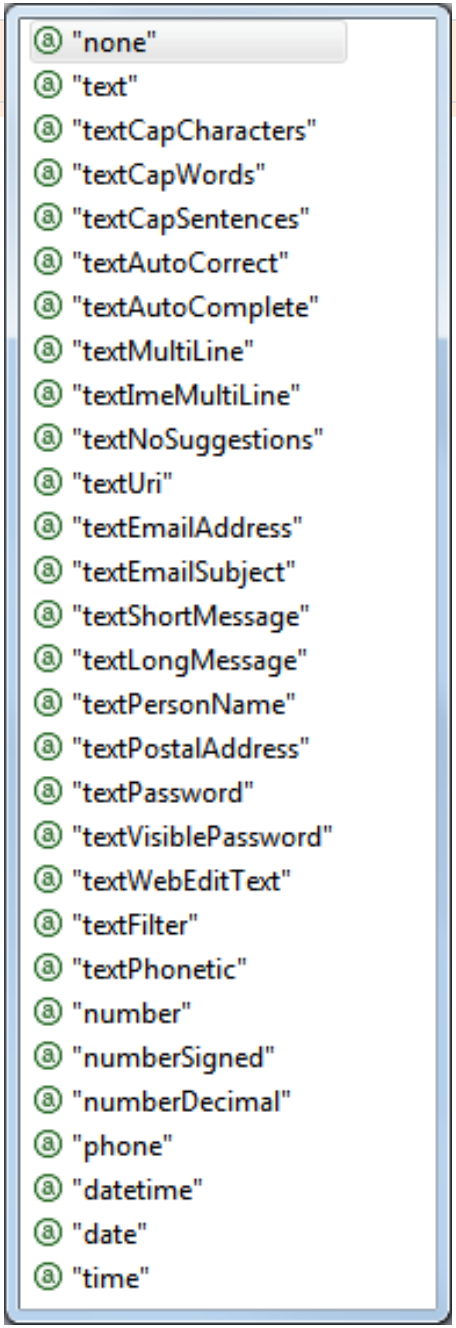
Format wprowadzania danych

Komponent EditText może być ustawiony by przyjmował tylko określone typu danych: numery (ze znakiem i częścią ułamkową bądź bez), numery telefonów, daty, czas, odnośniki itp.

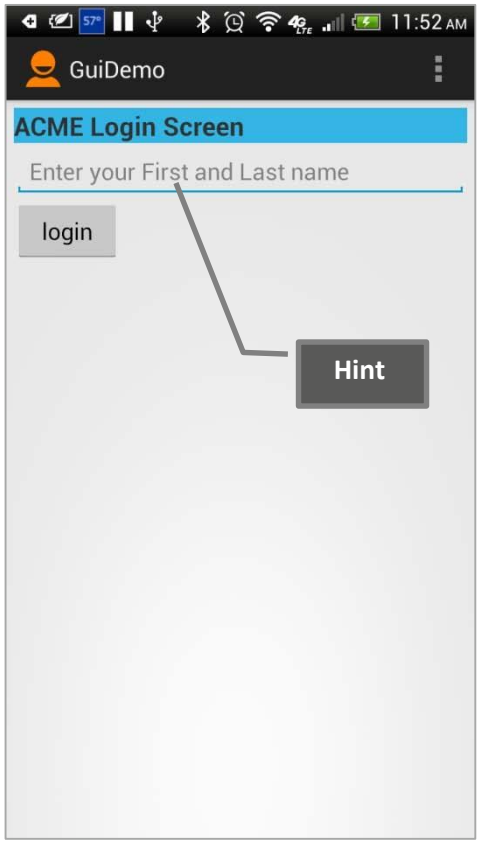
Ustawienie EditText by akceptował tylko wybrany typ danych umożliwia klauzula XML:

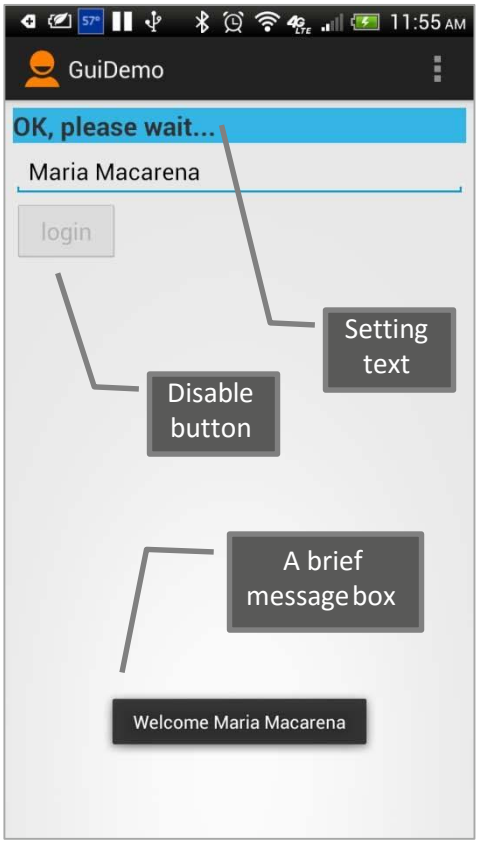
```
android:inputType="choices"
```

gdzie **choices** zawiera dowolne wartości pokazane na slajdzie. Wartości można łączyć, przykładowo zapis `textCapWords|textAutoCorrect` akceptuje słowa zaczynające się z wielkiej litery i umożliwia automatyczną kontrolę pisowni.



Ekran logowania - przykład





Zdjęcia z HTC-One

Ekran logowania - przykład

Układ 1 z 2

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="6dp" >

    <TextView
        android:id="@+id/txtLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/holo_blue_light"
        android:text="@string/ACME_Login_Screen"
        android:textSize="20sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/edtUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="2dp"
        android:hint="@string/Enter_your_First_and_Last_name"
        android:inputType="textCapWords|textAutoCorrect"
        android:textSize="18sp" >
        <requestFocus />
    </EditText>
```

Ekran logowania - przykład

Układ 2 z 2

```
    <Button
        android:id="@+id/btnLogin"
        android:layout_width="82dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="2dp"
        android:text="@string/login" />
</LinearLayout>
```

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- this is the res/values/strings.xml file -->
<resources>

    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>
    <string name="login">Login</string>
    <string name="ACME_Login_Screen">ACME Login Screen</string>
    <string name="Enter_your_First_and_Last_name">Enter your First and Last name</string>

</resources>
```

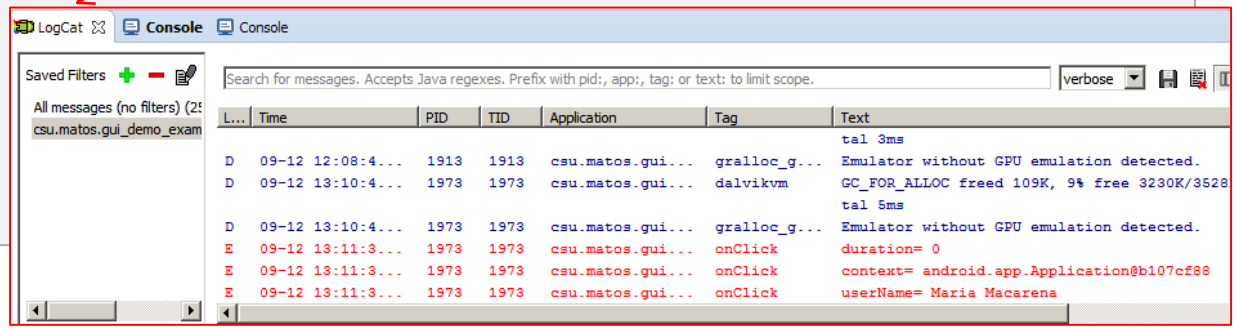
Ekran logowania - przykład

```
public class MainActivity extends ActionBarActivity {

 // class variables representing UI controls to be controlled from the Java program
 TextView txtLogin;
 EditText edtUserName;
 Button btnLogin;

 // variables used with the Toast message class
 private Context context;
 private int duration = Toast.LENGTH_SHORT;

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

  // show the login screen
  setContentView(R.layout.activity_main);
  context = getApplicationContext();

  // binding the UI's controls defined in "main.xml" to Java code
  txtLogin = (TextView) findViewById(R.id.txtLogin);
  edtUserName = (EditText) findViewById(R.id.edtUserName);
  btnLogin = (Button) findViewById(R.id.btnLogin);
```

Ekran logowania - przykład

```
// LISTENER: allowing the button widget to react to user interaction
btnLogin.setOnClickListener(new OnClickListener() {
 @Override
 public void onClick(View v) {

  String userName = edtUserName.getText().toString();

  Log.e("onClick ", "duration= " + duration);
  Log.e("onClick ", "context= " + context.toString());
  Log.e("onClick ", "userName= " + userName);

  if (userName.equals("Maria Macarena")) {
   txtLogin.setText("OK, please wait...");
   Toast.makeText(getApplicationContext(),
     "Welcome " + userName, duration).show();
   btnLogin.setEnabled(false);
  } else {
   Toast.makeText(context, userName + "  is not a valid USER",
     duration).show();
  }
 }
});// onClick

}// onCreate
```

Log.e jako debug – *usuń później*!!!



Ekran logowania - przykład

```
 @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;
 }

 @Override
 public boolean onOptionsItemSelected(MenuItem item) {
  // Handle action bar item clicks here. The action bar will
  // automatically handle clicks on the Home/Up button, so long
  // as you specify a parent activity in AndroidManifest.xml.
  int id = item.getItemId();
  if (id == R.id.action_settings) {
   return true;
  }
  return super.onOptionsItemSelected(item);
 }
}
```

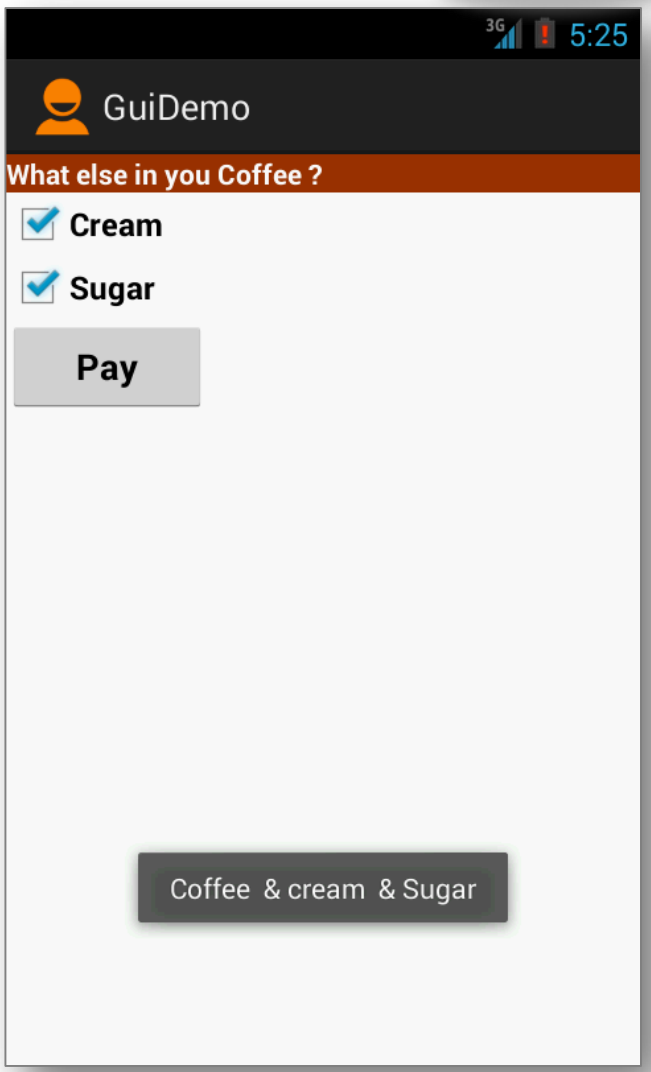
Pole typu CheckBox

Pole typu checkbox można interpretować jako **dwustanowy** przycisk który może być *zaznaczony* bądź *nie*.

Dany ekran może zawierać wiele niezależnie działających pól wyboru. W danym czasie więcej niż jeden checkbox może zostać zaznaczony.

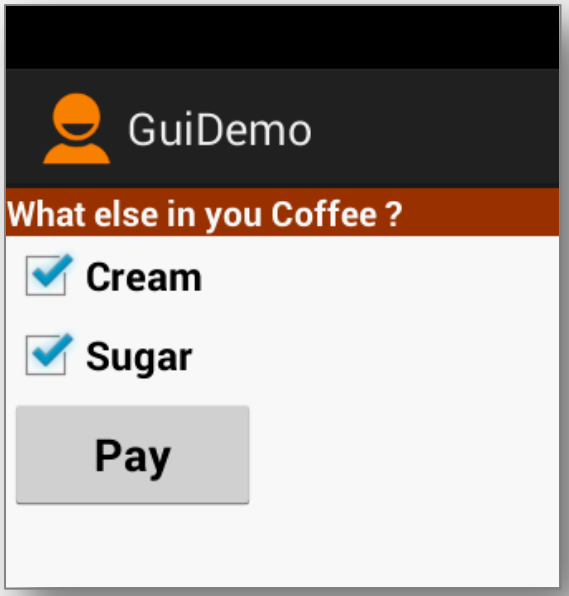
W przykładzie wykorzystano komponenty CheckBox do możliwości wyboru dodatków do kawy (cukier, śmietanka).

Po kliknięciu przycisku Pay użytkownikowi prezentowany jest komunikat zawierający wybraną kombinację zamówienia.



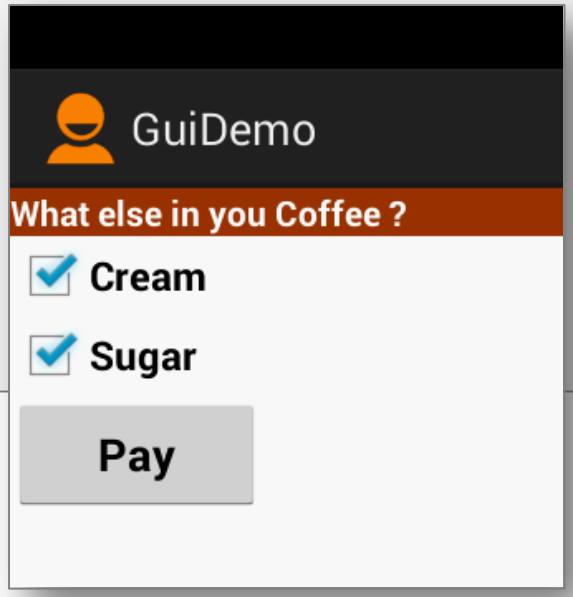
Checkbox - przykład

```
<?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:padding="6dp"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/labelCoffee"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ff993300"
        android:text="@string/coffee_addons"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

    <CheckBox
        android:id="@+id/chkCream"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cream"
        android:textStyle="bold" />
```



Checkbox - przykład

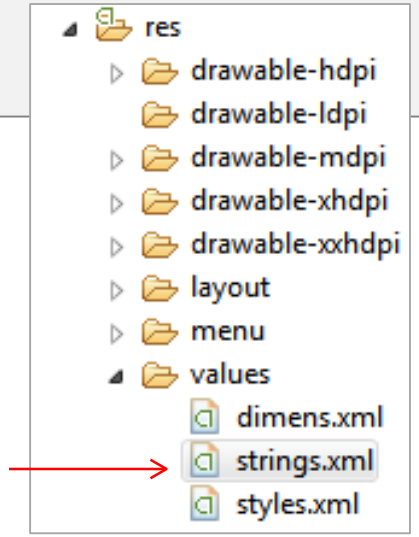
```
    <CheckBox
        android:id="@+id/chkSugar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/sugar"
        android:textStyle="bold" />

    <Button
        android:id="@+id/btnPay"
        android:layout_width="153dp"
        android:layout_height="wrap_content"
        android:text="@string/pay"
        android:textStyle="bold" />

</LinearLayout>
```



Checkbox – przykład [@string/…]

Zasoby: res/values/strings

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

    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>

    <string name="click_me">Click Me</string>
    <string name="sugar">Sugar</string>
    <string name="cream">Cream</string>
    <string name="coffee_addons">What else in your coffee?</string>
    <string name="pay">Pay</string>
</resources>
```



Checkbox - przykład

```
public class MainActivity extends Activity {

    CheckBox chkCream;
    CheckBox chkSugar;
    Button btnPay;

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //binding XMl controls with Java code
        chkCream = (CheckBox)findViewById(R.id.chkCream);
        chkSugar = (CheckBox)findViewById(R.id.chkSugar);
        btnPay = (Button) findViewById(R.id.btnPay);
```

Checkbox - przykład

```
 //LISTENER: wiring button-events-&-code
        btnPay.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
   String msg = "Coffee ";
   if (chkCream.isChecked()) {
      msg += " & cream ";
   }
   if (chkSugar.isChecked()){
      msg += " & Sugar";
   }
   Toast.makeText(getApplicationContext(),
                  msg, Toast.LENGTH_SHORT).show();
   //go now and compute cost...

   }//onClick
   });

  }//onCreate

}//class
```

Pole typu RadioButton

- Pole typu **RadioButton** (jak CheckBox) jest dwu-stanowym przyciskiem, który może być *zaznaczony* bądź *nie*.

- Zwykle pola tego typu są agregowane w kontener o nazwie **RadioGroup**. Zastosowanie tego kontenera powoduje, że pola wyboru zachowują się jako **wzajemnie wykluczające się selektory.** Oznacza to, że zmiana jednego z przycisków powoduje odznaczenie pozostałych.

- Właściwości dotyczących kroju, stylu czy koloru czcionki zarządzane są podobnie jak w przypadku etykiety TextView.

- Można wywołać metodę ***isChecked()*** by sprawdzić czy poszczególny przycisk jest zaznaczony, natomiast metoda **toggle()** odpowiada za zmianę jego stanu.

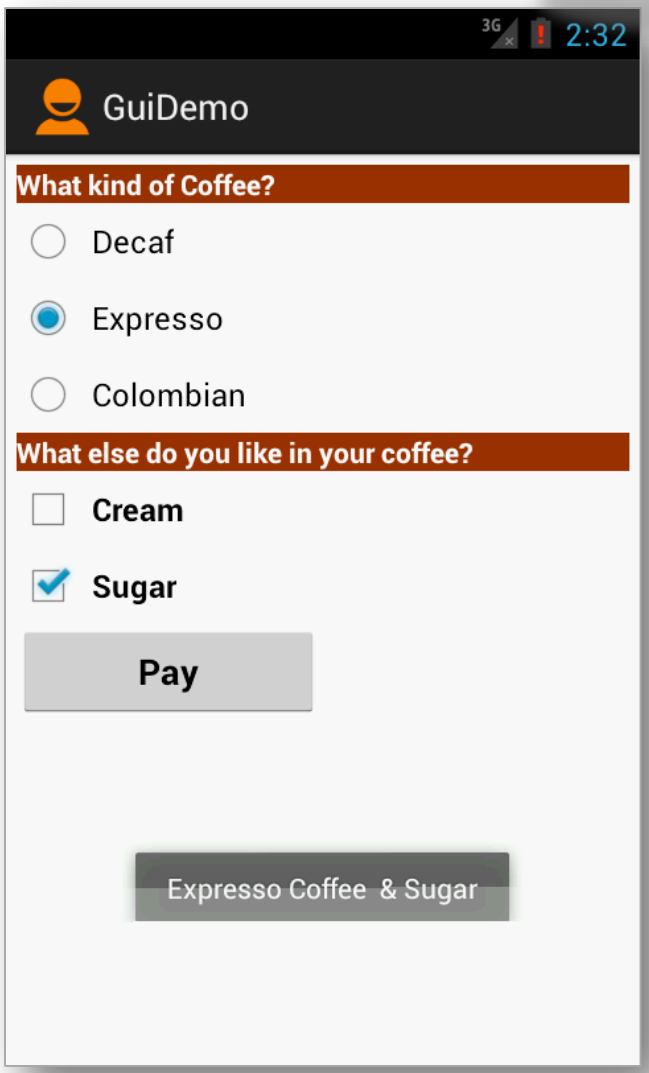
RadioButton - przykład

Przykład

Rozszerzono poprzednią aplikację dodając komponent ***RadioGroup*** *by umożliwić wybór konkretnego typu kawy.*

RadioGroup

Podsumowanie



RadioButton - przykład

Na podstawie poprzedniego przykładu – tylko nowe rzeczy są prezentowane

```
<TextView
      android:id="@+id/textView1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:background="#ff993300"
      android:text="@string/kind_of_coffee"
      android:textColor="#ffffff"
      android:textStyle="bold" />
```


```
<RadioGroup
      android:id="@+id/radioGroupCoffeeType"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <RadioButton
          android:id="@+id/radDecaf"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content
          "  android:text="@string/decaf" />

      <RadioButton
          android:id="@+id/radExpresso"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content
          "  android:text="@string/expresso"
          />

      <RadioButton
          android:id="@+id/radColombian"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content
          "  android:checked="true"
          android:text="@string/colombian" />
  </RadioGroup>
```

RadioButton - przykład

```
public class MainActivity extends Activity {

   CheckBox chkCream;
   CheckBox chkSugar;
   Button btnPay;

   RadioGroup radCoffeeType;
     RadioButton radDecaf;
     RadioButton radExpresso;
     RadioButton radColombian;

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

      chkCream = (CheckBox) findViewById(R.id.chkCream);
      chkSugar = (CheckBox) findViewById(R.id.chkSugar);
      btnPay = (Button) findViewById(R.id.btnPay);

      radCoffeeType = (RadioGroup) findViewById(R.id.radioGroupCoffeeType);

      radDecaf = (RadioButton) findViewById(R.id.radDecaf);
      radExpresso = (RadioButton) findViewById(R.id.radExpresso);
      radColombian = (RadioButton)
      findViewById(R.id.radColombian);
```

RadioButton - przykład

```
        // LISTENER: wiring button-events-&-code
        btnPay.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String msg = "Coffee ";
                if (chkCream.isChecked())
                    msg += " & cream ";
                if (chkSugar.isChecked())
                    msg += " & Sugar";

                // get selected radio button ID number
                int radioId = radCoffeeType.getCheckedRadioButtonId();

                // compare selected's Id with individual RadioButtons ID
                if (radColombian.getId() == radioId)
                    msg = "Colombian " + msg;
                // similarly you may use .isChecked() on each RadioButton
                if (radExpresso.isChecked())
                    msg = "Expresso " + msg;
                // similarly you may use .isChecked() on each RadioButton
                if (radDecaf.isChecked())
                    msg = "Decaf " + msg;

                Toast.makeText(getApplicationContext(), msg, 1).show();
                // go now and compute cost...
            }// onClick
        });
    }// onCreate

}// class
```

RadioButton - przykład

Wskazówka

```
radGroupradioId = (RadioGroup)findViewById(R.id.radioGroup1);

int radioId  = radGroupradioId.getCheckedRadioButtonId();

switch (radioId) {
  case R.id.radColombian: msg += " Colombian "; break;
  case R.id.radExpresso: msg += " Expresso "; break;
  case R.id.radDecaf: msg += " Decaf "; break;
}
```

Alternatywny przykład zarządzania **RadioGroup** – nie wymaga badania stanu każdego z przycisków z osobna.

Przydatne atrybuty XML i metody Java

By kontrolować focus (XML):

```
android:visibility      true/false – widoczność
android:background      color, image, drawable
<requestFocus />        ustaw domyślny focus
```

Metody Java

```
myButton.requestFocus()
myTextBox.isFocused()
myWidget.setEnabled()
myWidget.isEnabled()
```

GUI

Obraz wygenerowano przy pomocy
Android Asset Studio

http://romannurik.github.io/AndroidAssetStudio/

Dodatek A. Wykorzystanie dyrektywy @string

Dobry programista Android **NIE** wprowadza bezpośrednio literałów znakowych w kodzie aplikacji.

Przykładowo, definiując komponent **TextView** by pokazać np. lokalizację przedsiębiorstwa, nie powinno się stosować zapisu `android:text="Cleveland"` (powoduje to też wyświetlenie ostrzeżenia **Warning** *[I18N] Hardcoded string "Cleveland", should use @string resource*)

Zamiast tego powinno stosować się 2-etapową procedurę:

1. Dodać literał np.. *headquarter* do **res/values/string.xml.**

   ```
   <string name="headquarter">Cleveland</string>
   ```

2. Aby odwołać się do stworzonego literału należy wykorzystać zapis *@string/headquarter*. W przypadku etykiety wystarczy zapis

   ```
   android:text="@string/headquarter"
   ```

Dlaczego?

Dodatek B.

Android Asset Studio

LINK: http://romannurik.github.io/AndroidAssetStudio/

Wspomniane narzędzie oferuje możliwość tworzenia różnych rodzajów ikon oraz innych elementów graficznych

Generator ikon	Inne generatory	Pozostałe narzędzia
Launcher icons Action bar and tab icons Notification icons Navigation drawer indicator Generic icons	Device frame generator Simple nine-patch gen.	Android Action Bar Style Generator Android Holo Colors Generator

Dodatek C. Wielkość elementów graficznych

P. Co to **dpi** (znane jako **dp, ppi**) ?

Skrót od *dots per inch*. Można go wyliczyć korzystając z następującego wzoru:

$$dpi = \sqrt{szerokośćPiksele^2 + wysokośćPiksele^2} / przekątnaCale$$

G1 (320x480)	155.92 dpi	(3.7 in)
Nexus (480x800)	252.15 dpi	
HTC One (1080x1920)	468 dpi	(4.7 in)
Samsung S4 (1080x1920)	441 dpi	(5.5 in)

Q. Jaka jest różnica między **dp, dip** oraz **sp**?

dp *Density-independent Pixels* – Abstrakcyjna metryka oparta na faktycznej gęstości upakowania pikseli ekranu. Należy wykorzystywać do każdego zasobu prócz czcionek.

sp *Scale-independent Pixels* – podobnie jak w przypadku dp, ale wykorzystywany do ustawienia wielkości **czcionki**.

Dodatek D. Rozdzielczość ekranu

Rozdzielczość ekranu – jak Android sobie z tym radzi?

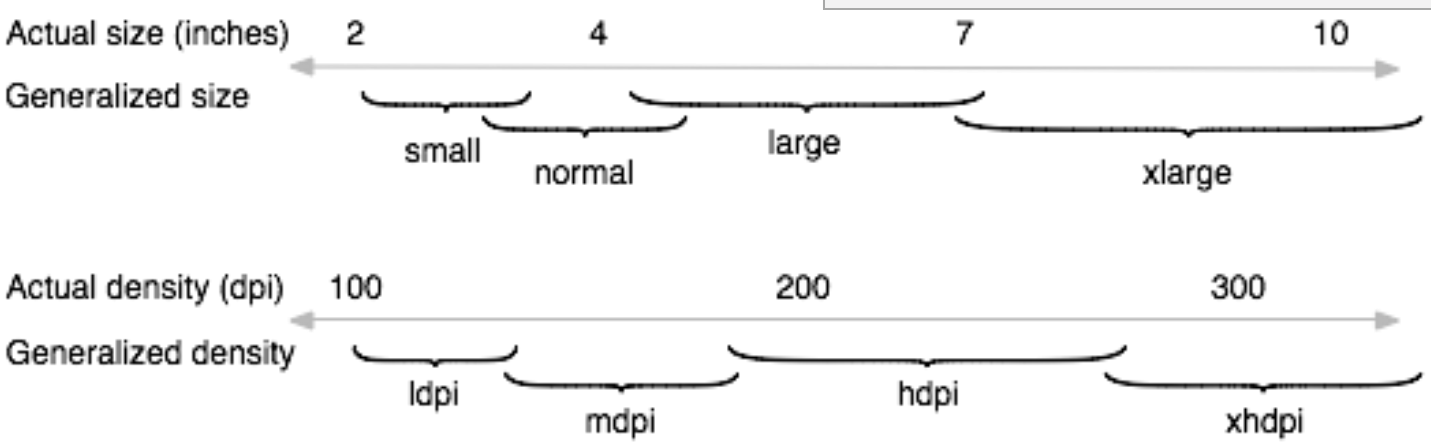
Poniżej znajdują się zapisy jak Android radzi sobie z klasyfikowaniem różnych rozdzielczości ekranu.

Zbiór czterech klas wielkości ekranu

xlarge ekran ma minimum 960dp x 720dp
large ekran ma minimum 640dp x 480dp
normal ekran ma minimum 470dp x 320dp
small ekran ma minimum 426dp x 320dp

Ogólne klasy gęstości:

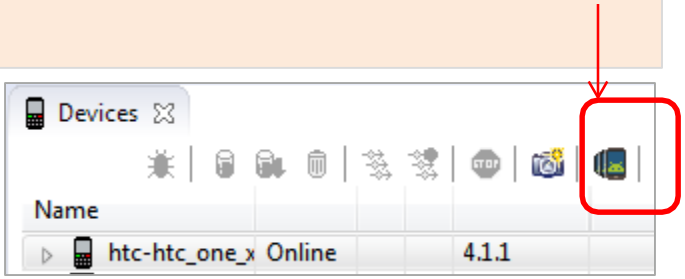
ldpi ~120dpi (low)
mdpi ~160dpi (medium)
hdpi ~240dpi (high)
xhdpi ~320dpi (extra-high)
xxhdpi ~480dpi (extra-extra-high)
Xxxhdpi ~640dpi (extra-extra-extra-high)

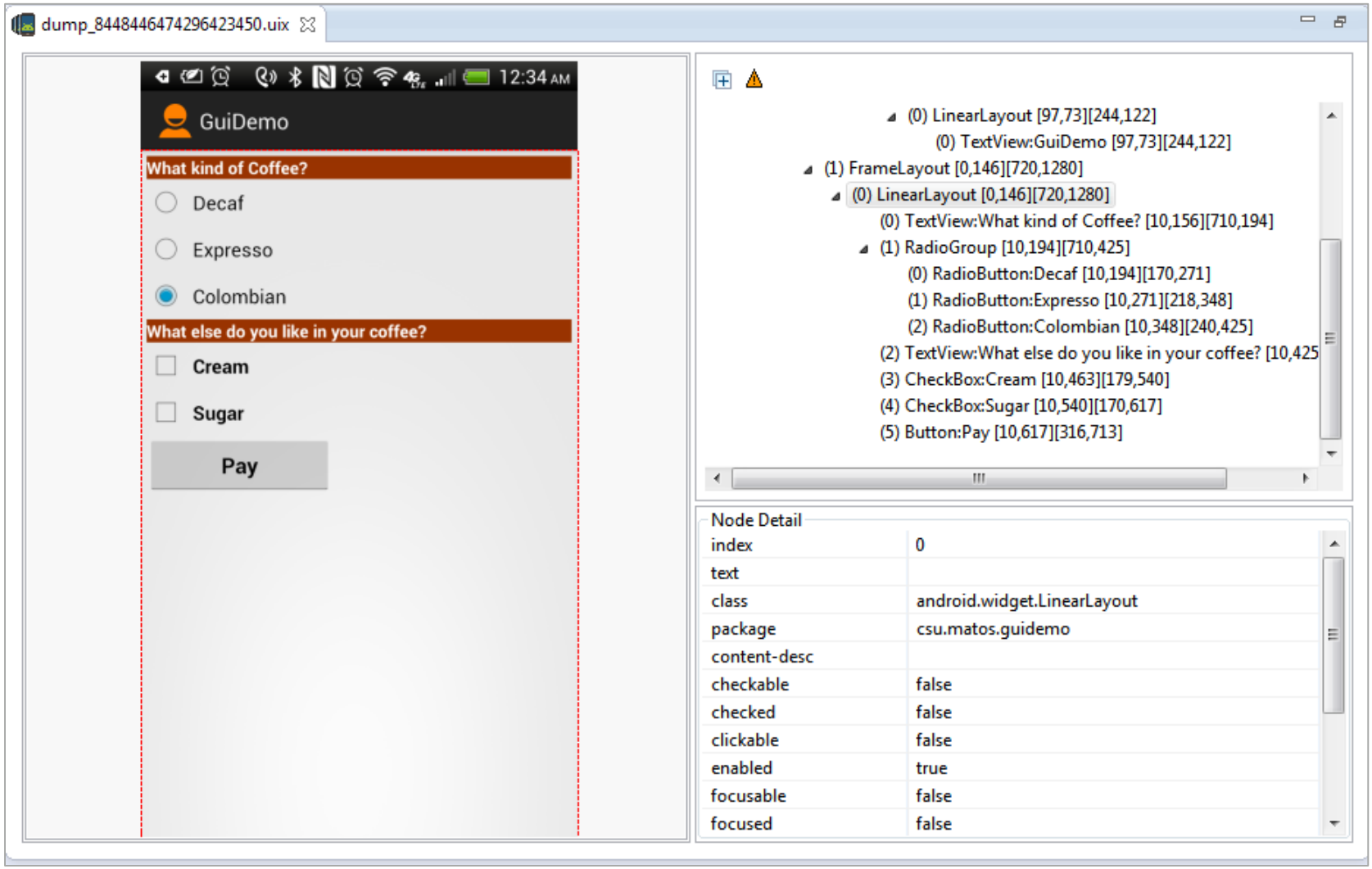


Źródło: http://developer.android.com/guide/practices/screens_support.html

Dodatek E. Hierarchy Viewer

Narzędzie HierarchyViewer umożliwia przejrzenie faktycznej hierarchii komponentów na interfejsie graficznym.





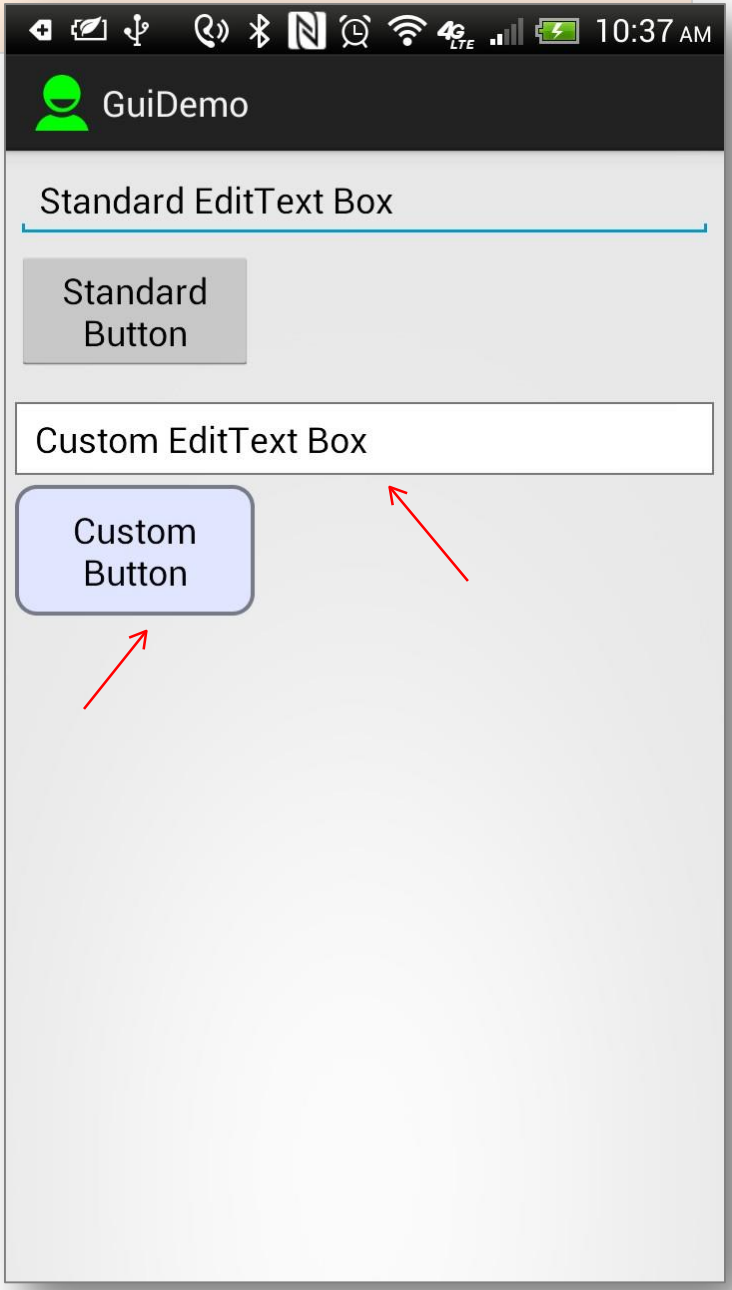
Dodatek F. Zmiana wyglądu widżetu

1. Zmiana wyglądu widżetu jest możliwa w bardzo prosty sposób. Można zmienić kształt, obramowanie, kolor, marginesy i inne właściwości.

2. Podstawowe figury to: prostokąt, elipsa, linia, pierścień.

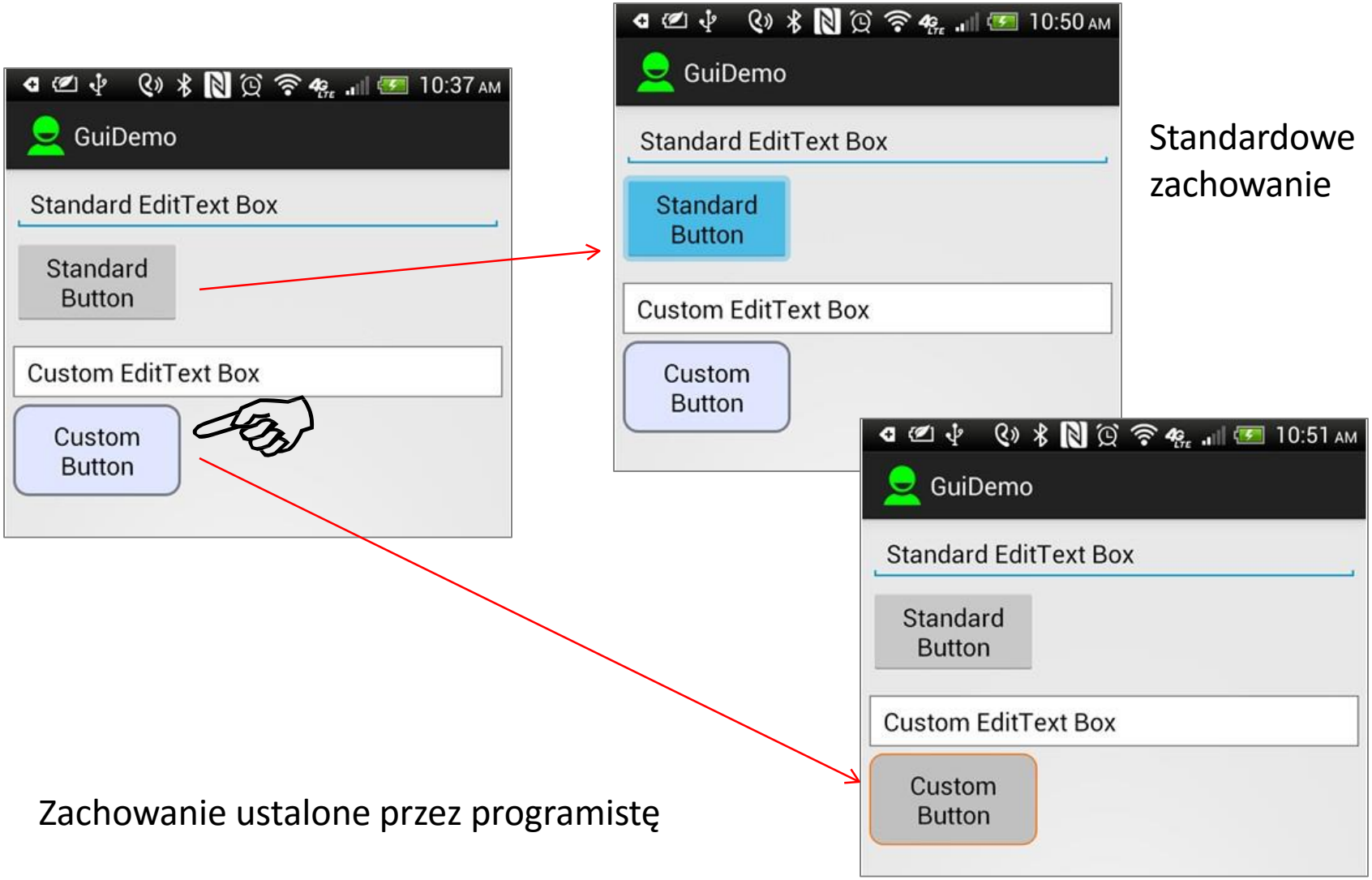
3. Prócz zmian typowo wizualnych w domyślnym stanie, można wprowadzić zmiany do stanów jak: kliknięty, posiada focus itp.

4. Rysunek przedstawia EditText i Button prezentowane w sposób standardowy na urządzeniu z androidem 4.4. Dolna część przedstawia te komponenty po odpowiednich zmianach.



Dodatek F. Zmiana wyglądu widżetu

Zmiana wyglądu z podziałem na stany przycisku

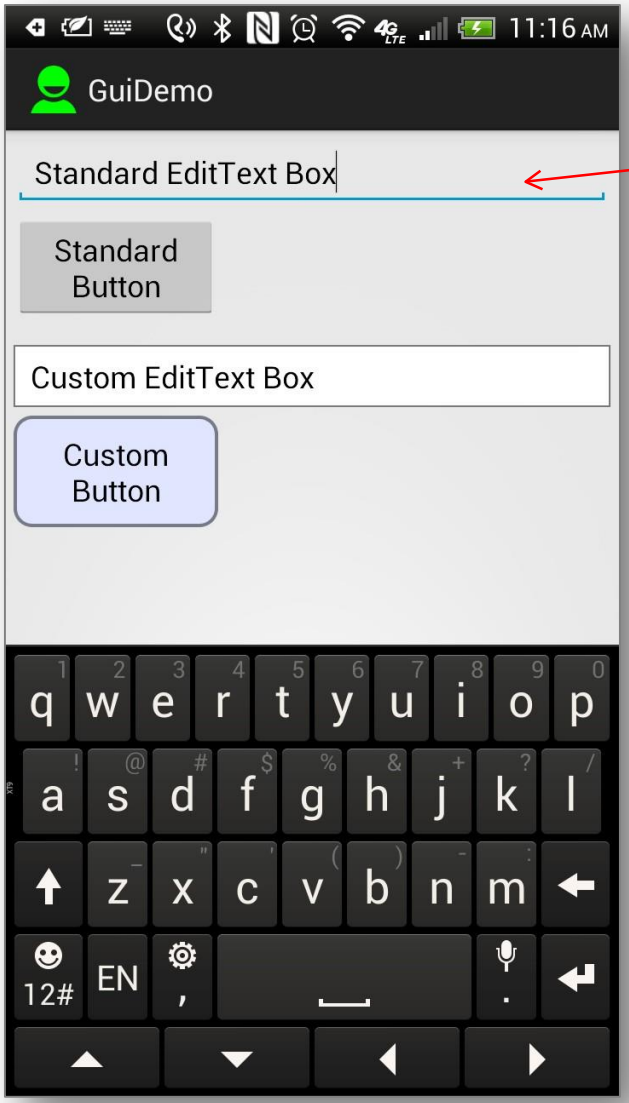


Standardowe zachowanie

Zachowanie ustalone przez programistę

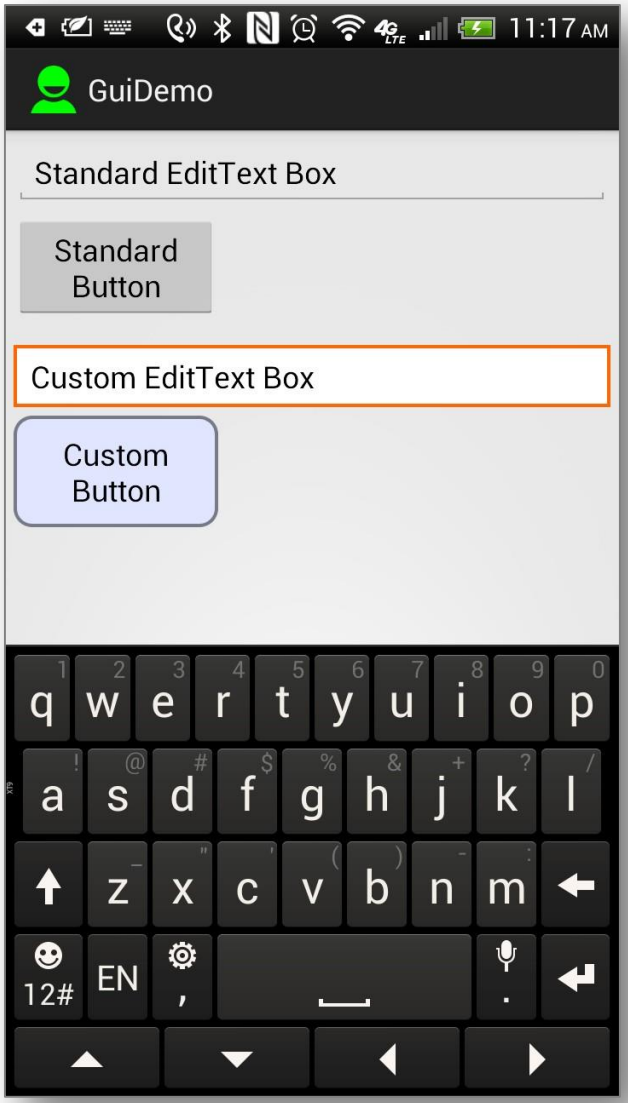
Dodatek F. Zmiana wyglądu widżetu

Zmiana wyglądu, gdy pole tekstowe posiada focus



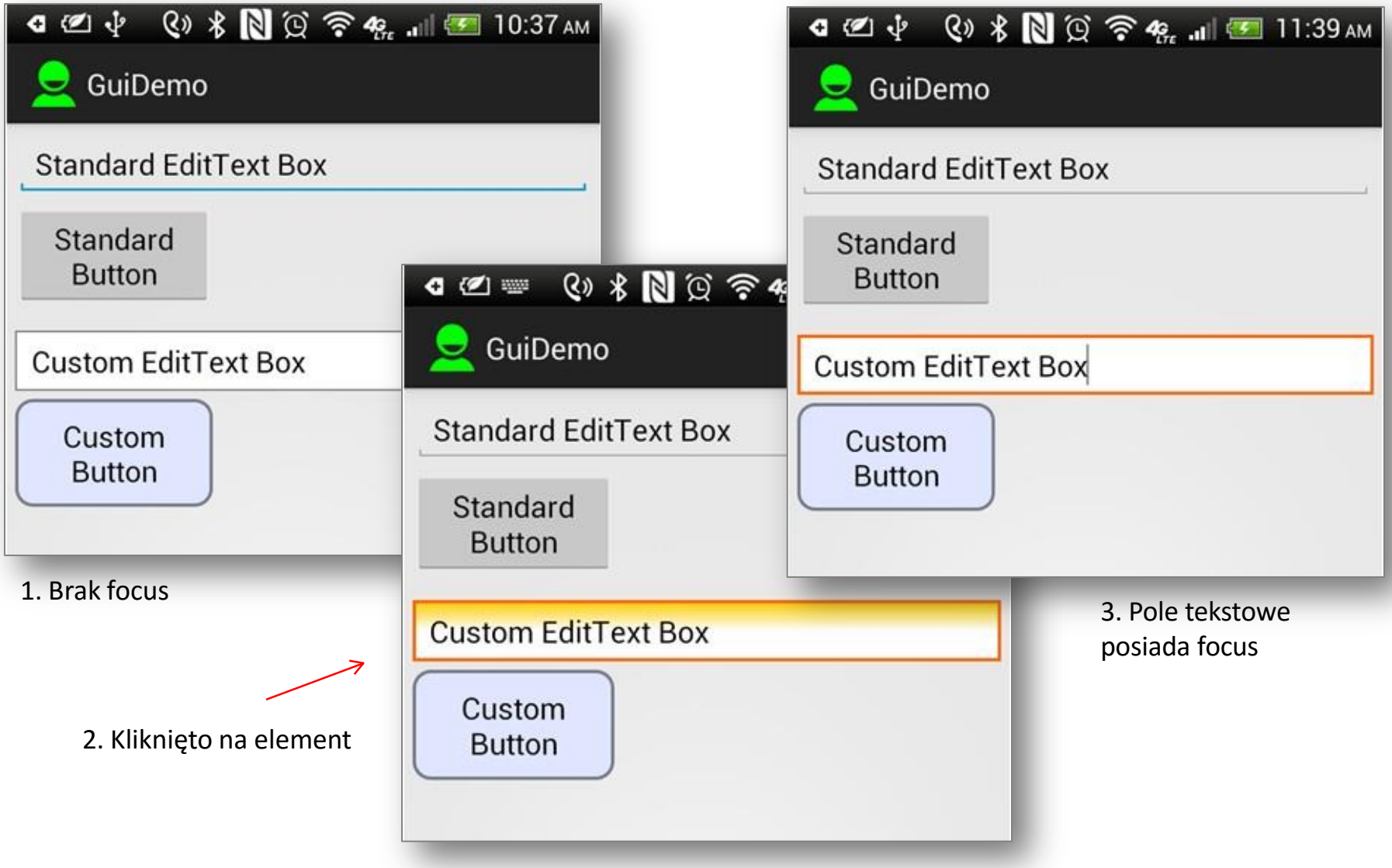
Standardowe zachowanie

Zdefiniowane przez programistę



Dodatek F. Zmiana wyglądu widżetu

Zmiana wyglądu, gdy użytkownik kliknął w pole tekstowe – użycie gradientu.

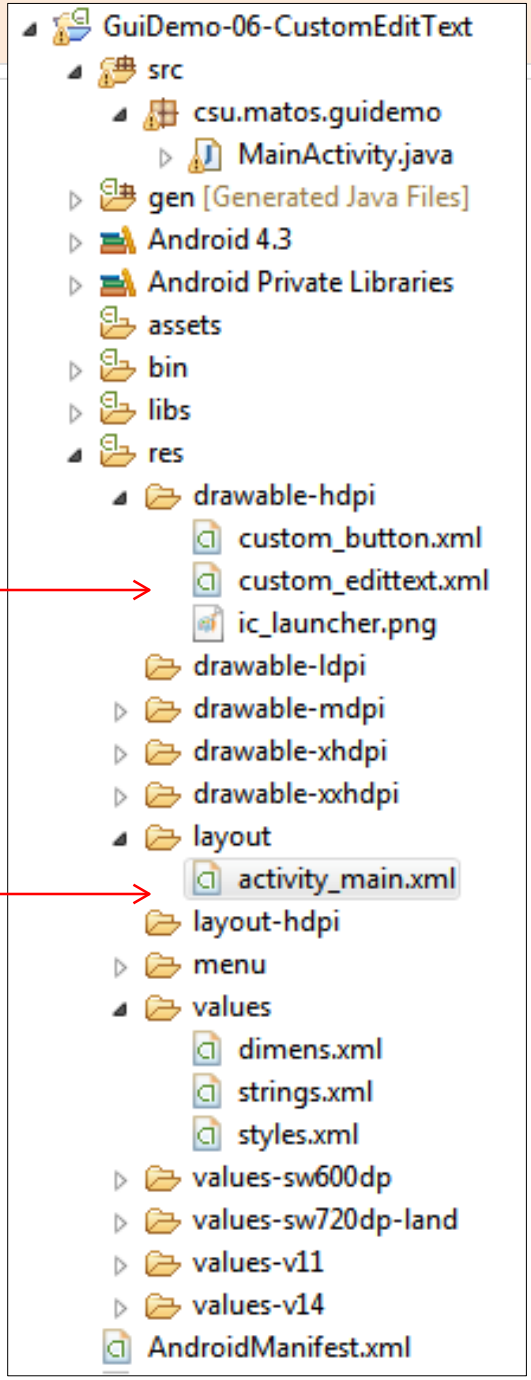


Dodatek F. Zmiana wyglądu widżetu

Organizing the application

Własne szablony wyglądu komponentów

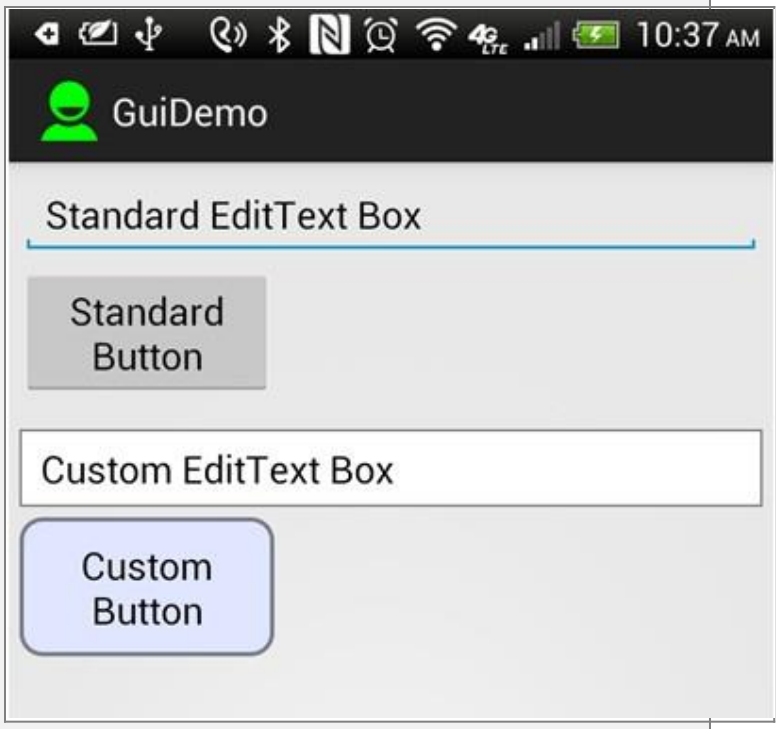
Główny wygląd aktywności



Dodatek F. Zmiana wyglądu widżetu

Układ

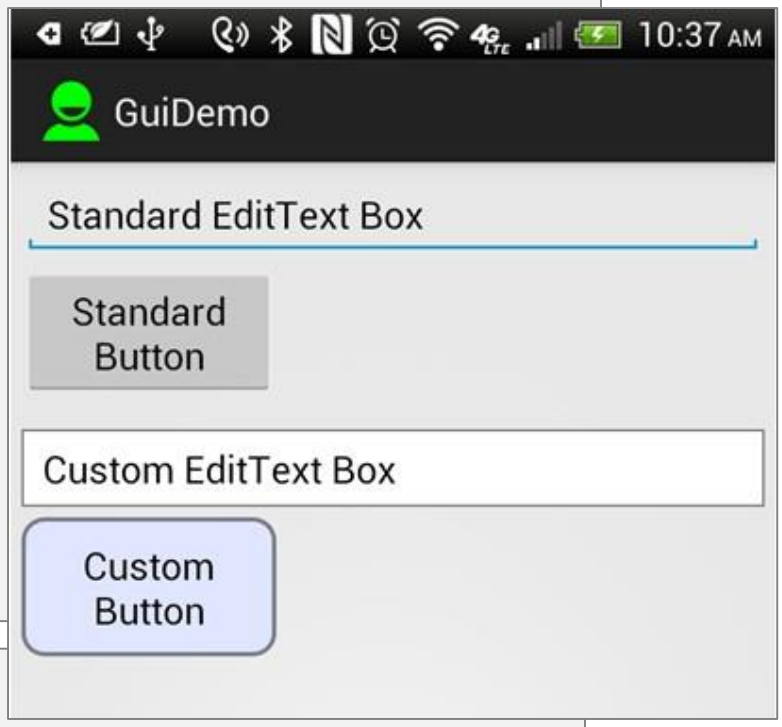
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:ems="10"
        android:inputType="text"
        android:text="@string/standard_edittext" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="15dp"
        android:text="@string/standard_button" />
```



Dodatek F. Zmiana wyglądu widżetu

Układ oraz Resource: res/values/strings

```
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:background="@drawable/custom_edittext"
        android:ems="10"
        android:inputType="text"
        android:text="@string/custom_edittext" />

    <Button
        android:id="@+id/button2"
        android:layout_width="120dp"
        android:layout_height="wrap_content"
        android:background="@drawable/custom_button"
        android:text="@string/custom_button" />

</LinearLayout>
```




```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">GuiDemo</string>
    <string name="action_settings">Settings</string>
    <string name="standard_button">Standard Button</string>
    <string name="standard_edittext">Standard EditText Box</string>
    <string name="custom_button">Custom Button</string>
    <string name="custom_edittext">Custom EditText Box</string>
</resources>
```

Dodatek F. Zmiana wyglądu widżetu

Zasób: res/drawable/custom_button.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:state_pressed="true">
        <shape android:shape="rectangle">
            <corners  android:radius="10dp"/>
            <solid    android:color="#ffc0c0c0" />
            <padding  android:left="10dp"
                      android:top="10dp"
                      android:right="10dp"
                      android:bottom="10dp"/>
            <stroke   android:width="1dp" android:color="#ffFF6600"/>
        </shape>
    </item>
    <item android:state_pressed="false">
        <shape android:shape="rectangle">
            <corners  android:radius="10dp"/>
            <solid    android:color="#ffE0E6FF"/>
            <padding  android:left="10dp"
                      android:top="10dp"
                      android:right="10dp"
                      android:bottom="10dp"/>
            <stroke   android:width="2dp" android:color="#ff777B88"/>
        </shape>
    </item>
</selector>
```

Dodatek F. Zmiana wyglądu widżetu

Zasób: res/drawable/custom_button.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector  xmlns:android="http://schemas.android.com/apk/res/android">

<item android:state_pressed="true">
    <shape android:shape="rectangle">
          <gradient
               android:angle="90"
               android:centerColor="#FFffffff
               "  android:endColor="#FFffcc00"
               android:startColor="#FFffffff"
               android:type="linear" />

        <stroke   android:width="2dp"
                  android:color="#FFff6600" />
        <corners  android:radius="0dp" />
        <padding  android:left="10dp"
                  android:top="6dp"
                  android:right="10dp"
                  android:bottom="6dp" />
    </shape>
</item>
```

Custom EditText Box

Dodatek F. Zmiana wyglądu widżetu

Zasób: res/drawable/custom_button.xml

```
<item android:state_focused="true">
    <shape>
        <solid    android:color="#FFffffff" />
        <stroke   android:width="2dp" android:color="#FFff6600" />
        <corners  android:radius="0dp" />
        <padding  android:left="10dp"
                  android:top="6dp"
                  android:right="10dp"
                  android:bottom="6dp" />
    </shape>
</item>

<item>
    <!-- state: "normal" not-pressed & not-focused  -->
    <shape>
        <stroke   android:width="1dp" android:color="#ff777777" />
        <solid    android:color="#ffffffff" />
        <corners  android:radius="0dp" />
        <padding  android:left="10dp"
                  android:top="6dp"
                  android:right="10dp"
                  android:bottom="6dp" />
    </shape>
</item>
</selector>
```

Custom EditText Box

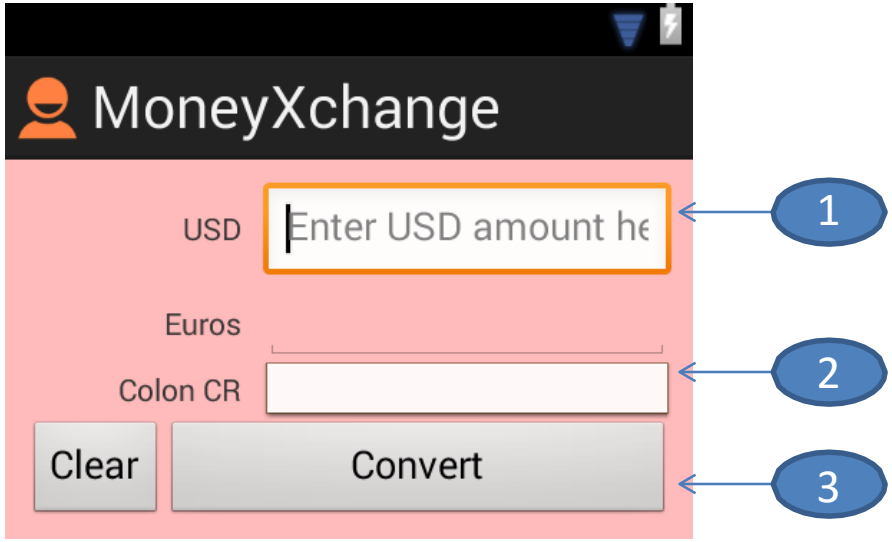
Custom EditText Box

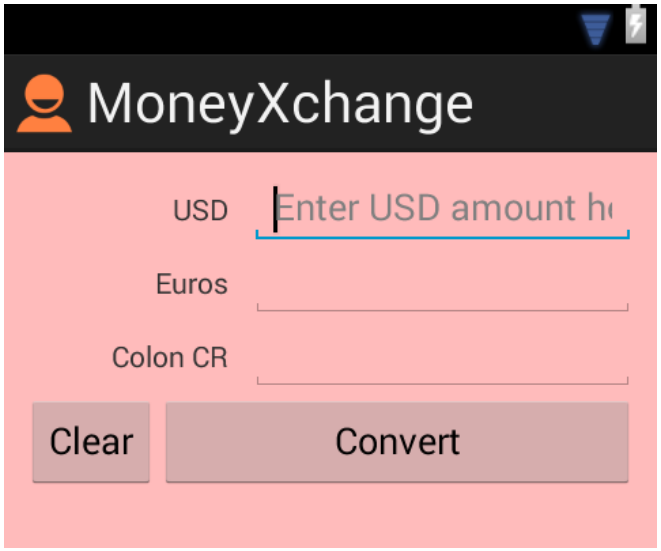
Dodatek G. Problemy z tłem

Zmiana koloru tła może ograniczyć się do zmiany odpowiedniej właściwości w pliku XML `android:background="#44ff0000"` (pół-przeźroczysty czerwony).

W zależności od skórki (theme) urządzenia może zdarzyć się problem z wypełnieniem komponentów – niektóre z nich przejmują globalny kolor tła.

Rozwiązanie jest bardzo proste – nadanie tym komponentom innego koloru tła poprzez wykorzystanie atrybutu `android:background`.






```
1. android:background="@android:drawable/edit_text"
2. android:background="@android:drawable/editbox_dropdown_light_frame"
3. android:background="@android:drawable/btn_default"
```