Programowanie aplikacji mobilnych

dr Przemysław Juszczuk

Katedra Inżynierii Wiedzy

Wykład 2

Szybka powtórka z Javy

- typy proste;
- typy referencyje;
- deklaracja zmiennych;
- instrukcje sterujące;
- rzutowanie;
- metody i zwracane typy;
- zasięg zmiennych;
- obiekty i dziedziczenie;
- interfejsy.

Java vs C#

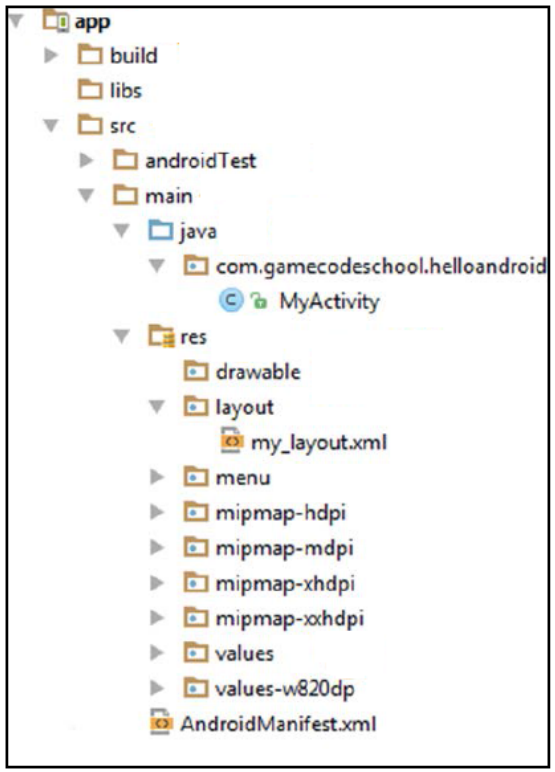
- Java to aplikacje przenośne;
- C# wydajniejsze;
- C# bardziej rozbudowane środowisko i sporo narzędzi ułatwiających pracę;
- W C# typy proste to też obiekty, natomiast w Javie trzeba je najpierw opakować (np. klasa Double, lub Integer);
- W Javie nie można w prosty sposób definiować typów prostych (w C# mamy struct);
- trzeba uważać na obsługę tablic w Java – odwołujemy się do pierwszego wymiaru, a potem do drugiego.

Co dobrze będzie znać?

- podstawy klas i obiektów;
- dziedziczenie (większość komponentów dziedziczy po innych elementach);
- podstawy liczb losowych (pseudolosowych);
- tablice statyczne (oraz dynamiczne);

Gradle

- Narzędzie do budowy projektów;
- alternatywa dla Ant'a i Maven'a w Android Studio;
- w przypadku Android mamy pewną domyślną strukturę projektu;



Rysunek: Gradle

```java
package com.example.scorpu.testapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Rysunek: Plik aktywności

```
package com.example.scorpu.testapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Rysunek: Plik aktywności – paczki jako części składowe projektów

```
package com.example.scorpu.testapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Rysunek: Plik aktywności – dodanie do projektu bibliotek dodatkowych takich, jak obsługa plików

```java
package com.example.scorpu.testapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Rysunek: Plik aktywności – wszystko jest klasą, podobnie jak w Java

```java
package com.example.scorpu.testapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Rysunek: Plik aktywności – każde okno to osobna aktywność

```
package com.example.scorpu.testapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Rysunek: Plik aktywności – extends, czyli dziedziczenie po klasie z prawej

```
package com.example.scorpu.testapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Rysunek: Plik aktywności – onCreate, czyli główna metoda, z której uruchamiamy daną aktywność

Nowe Aktywności

- nie mamy żadnej funkcji main umożliwiającej start całej aplikacji;
- każda nowa aktywność ma metodę onCreate(...);
- nowa aktywność tworzona jest w domyślnej paczce, w której mamy pozostałe elementy kodu (na początku tylko MainActivity);
- każda nowa aktywność wpisywana jest do pliku manifestu;

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

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

</manifest>
```

Rysunek: Plik manifestu

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:onClick="Test"
    android:id="@+id/textView" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:onClick="Test"
    android:layout_below="@+id/textView"
    android:layout_alignParentStart="true"
    android:layout_marginTop="27dp" />
```

Rysunek: Przypisujemy akcję do naciśnięcia pola tekstowego

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:onClick="Test"
    android:id="@+id/textView" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:onClick="Test"
    android:layout_below="@+id/textView"
    android:layout_alignParentStart="true"
    android:layout_marginTop="27dp" />
```

Rysunek: Przypisujemy akcję do naciśnięcia pola tekstowego – widget

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:onClick="Test"
    android:id="@+id/textView" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:onClick="Test"
    android:layout_below="@+id/textView"
    android:layout_alignParentStart="true"
    android:layout_marginTop="27dp" />
```

Rysunek: Przypisujemy akcję do naciśnięcia pola tekstowego – różne właściwości

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:onClick="Test"
    android:id="@+id/textView" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:onClick="Test"
    android:layout_below="@+id/textView"
    android:layout_alignParentStart="true"
    android:layout_marginTop="27dp" />
```

Rysunek: Przypisujemy akcję do naciśnięcia pola tekstowego – onClick, czyli co się stanie po naciśnięciu tego widgetu

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:onClick="Test"
    android:id="@+id/textView" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:onClick="Test"
    android:layout_below="@+id/textView"
    android:layout_alignParentStart="true"
    android:layout_marginTop="27dp" />
```

Rysunek: Przypisujemy akcję do naciśnięcia pola tekstowego – jaka metoda zostanie wywołana

```java
public class MainActivity extends AppCompatActivity {

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

    public void Test(View v)
    {
        Toast.makeText(this, "Wyświetlenie po naciśnięciu pola tekstowego", Toast.LENGTH_SHORT).show();
    }
    public void UruchomEkran(View v)
    {
        Intent i = new Intent(this, OtherActivity.class);
        startActivity(i);
    }
}
```

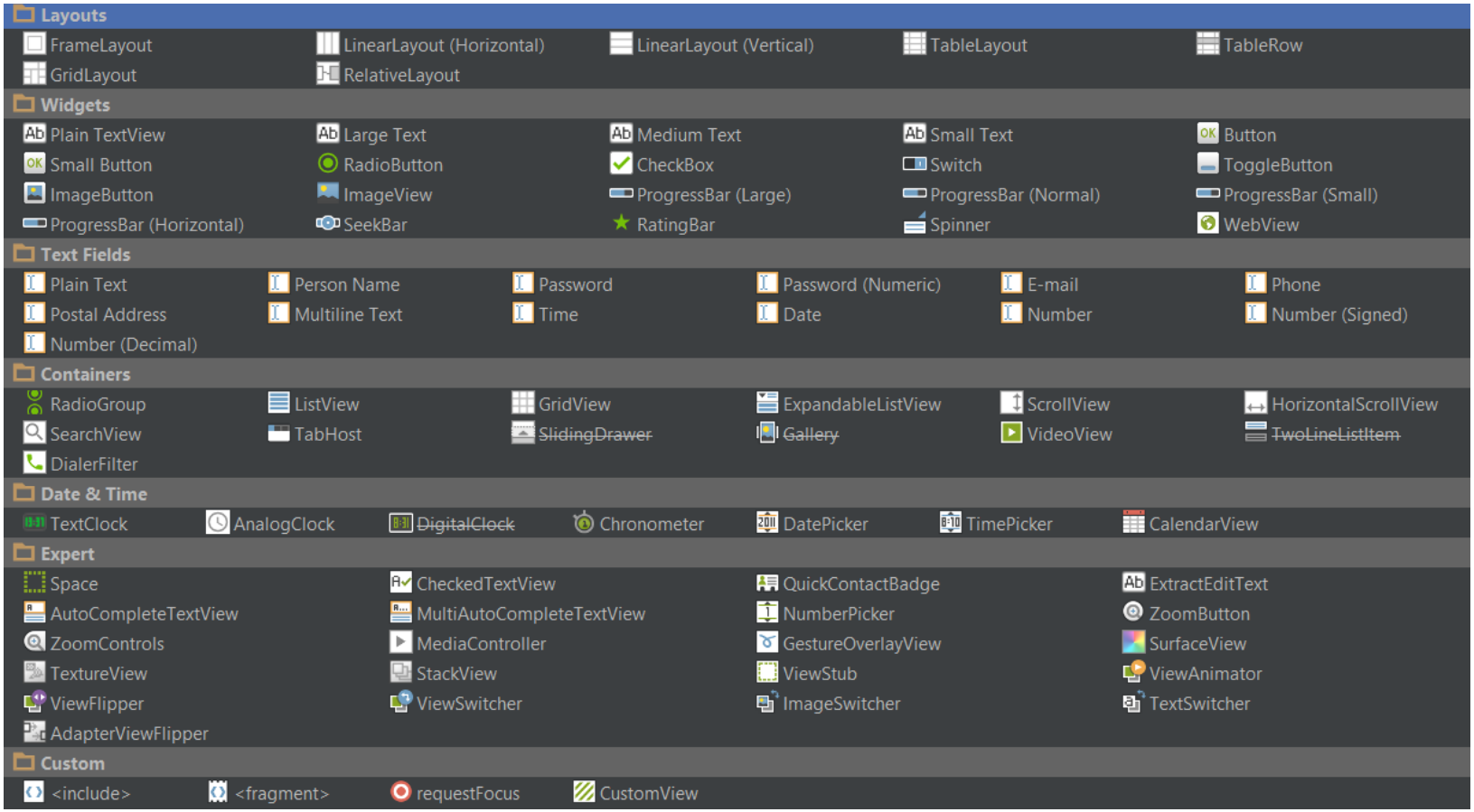
Rysunek: Wywołujemy nową intencję

```java
public class MainActivity extends AppCompatActivity {

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

    public void Test(View v)
    {
        Toast.makeText(this, "Wyświetlenie po naciśnięciu pola tekstowego", Toast.LENGTH_SHORT).show();
    }
    public void UruchomEkran(View v)
    {
        Intent i = new Intent(this, OtherActivity.class);
        startActivity(i);
    }
}
```

Rysunek: Pamiętamy o tym, że View i Intent to klasy, które trzeba zaimportować



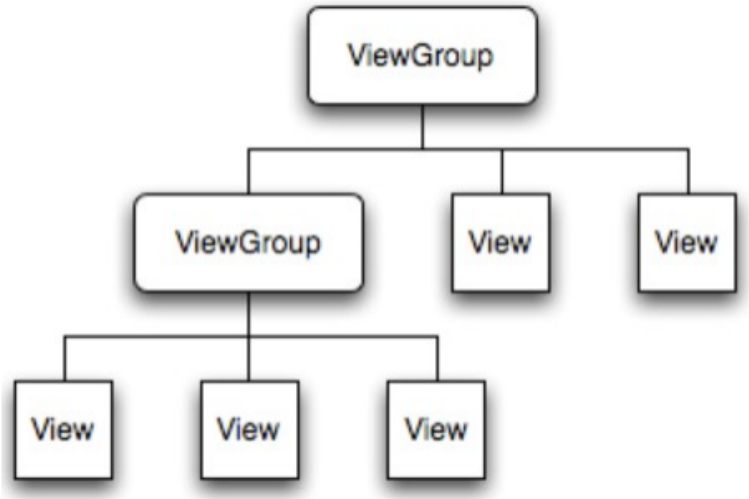
Rysunek: Widgety w środowisku Android

Właściwości Widgetów

- wysokość, szerokość, czcionka;
- wielkość elementu - match (dopasowanie do elementu rodzica);
- padding - margines wewnętrzny między ramką a obiektem (dlatego najpierw umieszczamy elementy wewnętrz pojemnika np. layout, a dopiero potem padding);
- margin - margines zewnętrzny pomiędzy ramką a sąsiednimi obiektami.

Podstawy GUI

- View - prosty komponent;
- View group - layout (grupua).



Rysunek: GUI - ekran Activity

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

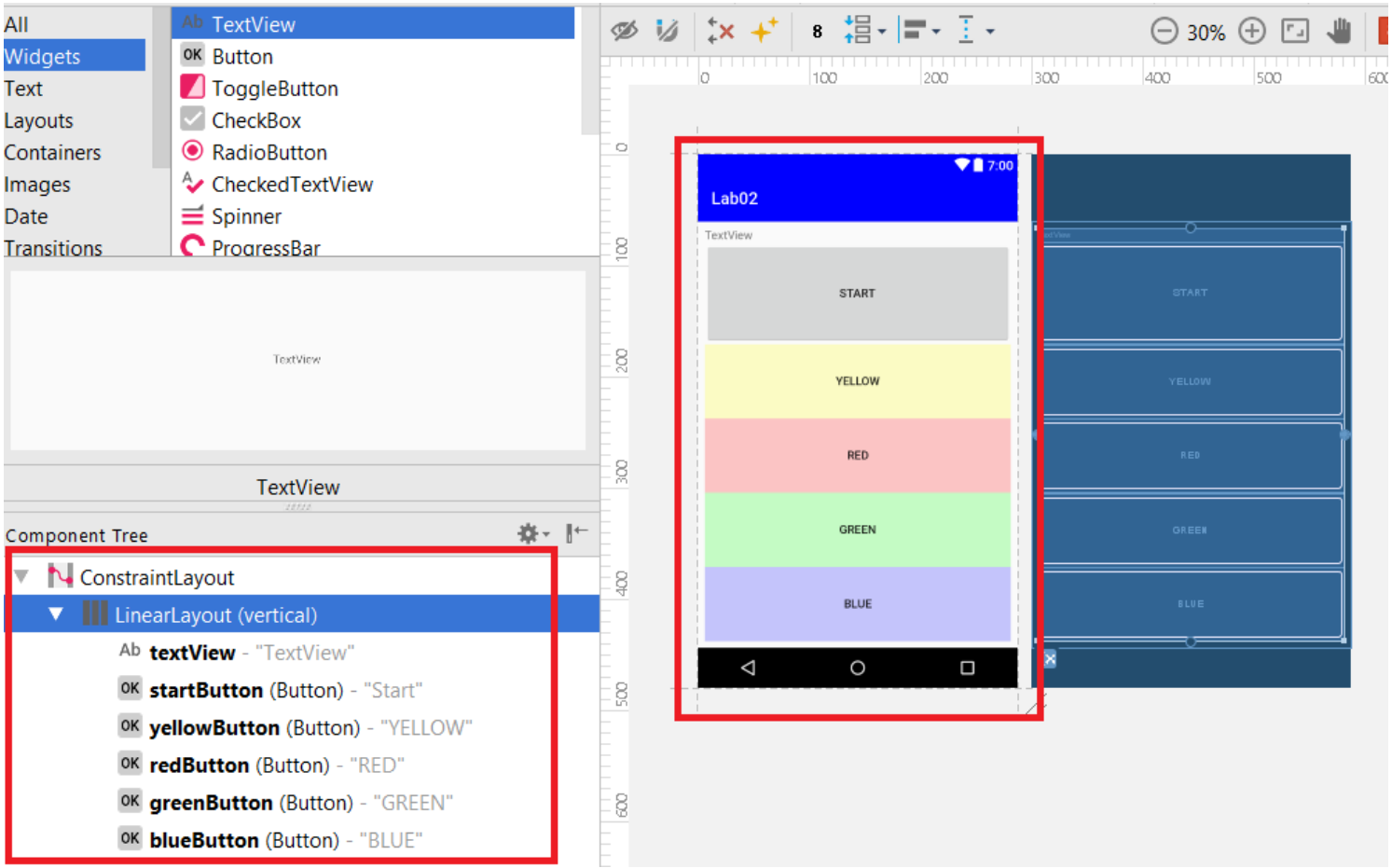
Rysunek: GUI - plik XML

View groups - LinearLayout i RelativeLayout

- układ w poziomie;
- układ w pionie;
- układ ustalany na podstawie parametru: android:orientation.

View groups - RelativeLayout

- układ w odniesieniu do rodzica - np. Container;
- odniesienie: Bottom Parent, Top Parent, (left, right, center);
- położenie elementów w odniesieniu do innych elementów;



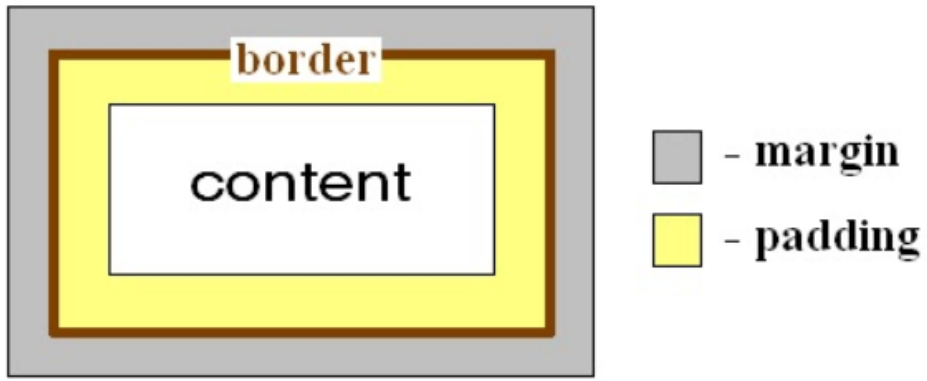
Rysunek: Przykład użycia LinearLayout

View groups - waga

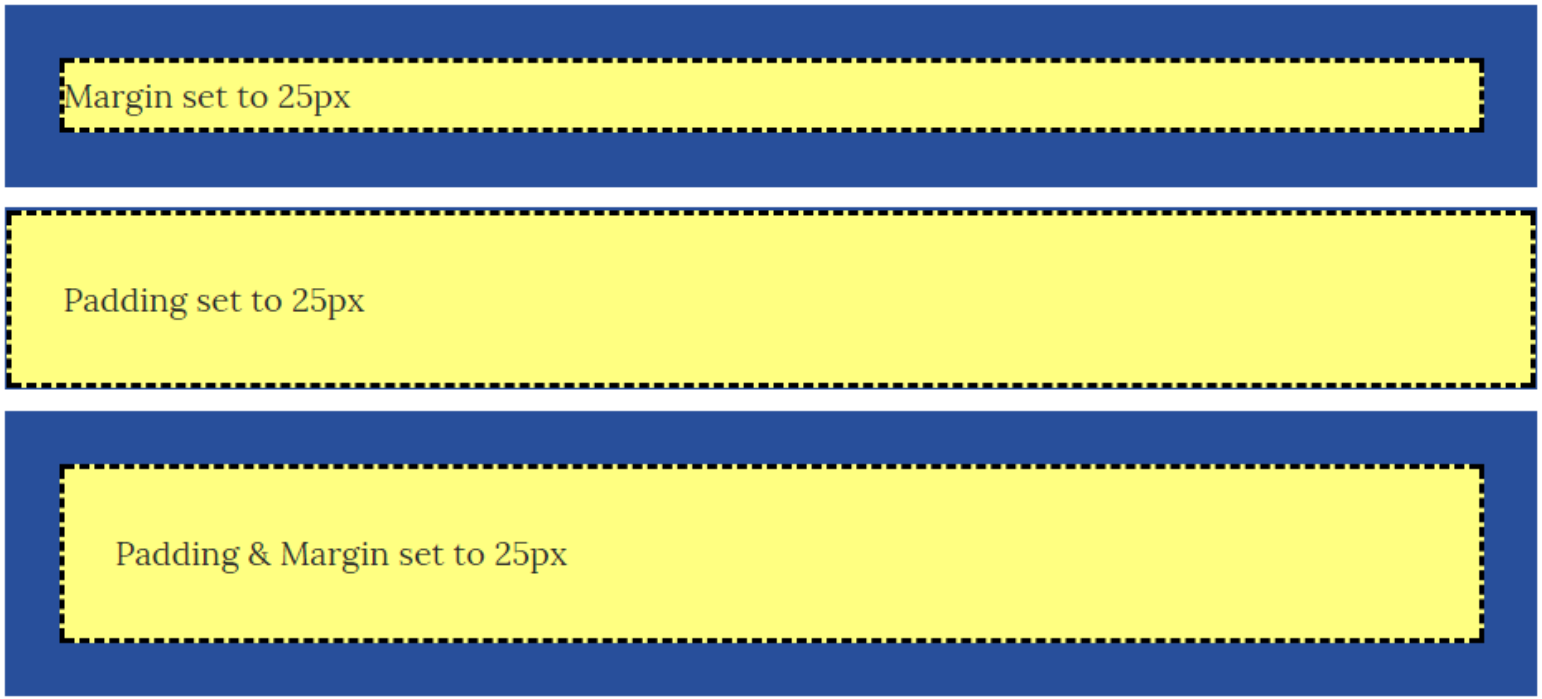
- ustalając LinearLayout i zmieniając wielkość danej kontrolki View na wartość 0 możemy przypisać jej wagę;
- waga oznacza, jaką część widoku będzie zajmować dany View;
- Przykładowo: 3 elementy, każdy z wagą 1 oznaczają, iż poszczególne elementy będą rozmieszczone na ekranie równomiernie.

Layouts – atrybuty ogólne

- android:id – unikalny numer; symbol @ oznacza, że odwołujemy się do zasobu;
- android:layout width i height – wrap dopasowuje się tak, żeby wymiar odpowiadał zawartości – np. żeby cały tekst przycisku był widoczny. Natomiast match parrent oznacza powiększenie obszaru kontrolki tak, żeby wypełniał cały dostępny obszar;
- padding – przestrzeń wewnątrz kontrolki pomiędzy brzegiem a zawartością;
- margin – przestrzeń poza kontrolką – na zewnątrz brzegu.



Rysunek: Margin vs padding Źródło: stackoverflow.com



Rysunek: Margin vs padding Źródło: stackoverflow.com

Jak zapewnić sobie wsparcie dla różnych urządzeń?

- rozmiary w postaci dip ($px = dp \cdot (\frac{dip}{160})$);
- dla tekstów używamy sp;
- $< supports - screens >$ - wskazujemy, jakie rozmiary ekranu są wspierane;
- dostarczamy różne elementy resources (dla różnych ekranów);
- stosujemy różne layout.

dpi, dip, dp, dps, sp?

- px - rzeczywiste piksele na ekranie;
- in - inches – cale;
- pt - points $\frac{1}{72}$ wielkości cala bazując na rzeczywistym rozmiarze ekranu;
- dpi - dots per inch (kropki na cal) - 160 (w przypadku ekranu o niższej rozdzielczości), lub 240 (w przypadku ekranu o większej rozdzielczości);
- pixels = dpi razy szerokość (1.5 cala), a więc odpowiednio 240 lub 360 pikseli;
- gęstość - 1.0 - dla ekranów o niższej rozdzielczości oraz 1.5 dla większej rozdzielczości;
- Denisity-independent pixels (dip,dp, dps) - 240;
- scale-independent pixels - zależy od ustawień dotyczących czcionki.

Density Bucket	Screen Density	Physical Size	Pixel Size
ldpi	120 dpi	0.5 x 0.5 in	0.5 in * 120 dpi = 60x60 px
mdpi	160 dpi	0.5 x 0.5 in	0.5 in * 160 dpi = 80x80 px
hdpi	240 dpi	0.5 x 0.5 in	0.5 in * 240 dpi = 120x120 px
xhdpi	320 dpi	0.5 x 0.5 in	0.5 in * 320 dpi = 160x160 px
xxhdpi	480 dpi	0.5 x 0.5 in	0.5 in * 480 dpi = 240x240 px
xxxhdpi	640 dpi	0.5 x 0.5 in	0.5 in * 640 dpi = 320x320 px

Unit	Description	Units Per Physical Inch	Density Independent	Same Physical Size On Every Screen
px	Pixels	Varies	No	No
in	Inches	1	Yes	Yes
mm	Millimeters	25.4	Yes	Yes
pt	Points	72	Yes	Yes
dp	Density Independent Pixels	~160	Yes	No
sp	Scale Independent Pixels	~160	Yes	No

Rysunek: Więcej szczegółów. Źródło: stackoverflow.com

Komponenty a intencje

- czynności, usługi oraz odbiorcy rozgłoszeń tworzeni są za pomocą intencji (intent);
- konieczne jest wcześniejsze utworzenie obiektu Intent, który będzie zawierał dane przekazywane wskazanemu komponentowi;
- intencja wiąże komponenty (niezależnie od tego, czy inny komponent należy do uruchamiającej aplikacji, czy innej).

Jeszcze intencje

- Czynność – za pomocą startActivity() lub startActivityForResult();
- Usługa – za pomocą startService() (lub jeżeli chcemy powiązać się z usługą za pomocą bindService());
- Rozgłoszenie – za pomocą sendBroadcast(), sendOrderedBroadcast() lub sendStickyBroadcast();
- Dostawca zawartości – można go odpytywać za pomocą metody query() na obiekcie ContentResolver.

Cechy intencji

- akcja - ogólna akcja, która ma zostać wykonana (ACTION VIEW, ACTION EDIT);
- dane - informacje, które mają zostać przekazane komponentowi (np. dane adresowe);
- kategoria - informacje dodatkowe;
- typ MIME danych (identyfikator formatu plików) składający się z typu oraz podtypu (przetwarzania wiadomości e-mail);
- komponent;
- dodatki.

The Draw 9-patch tool - tworzenie elementów GUI

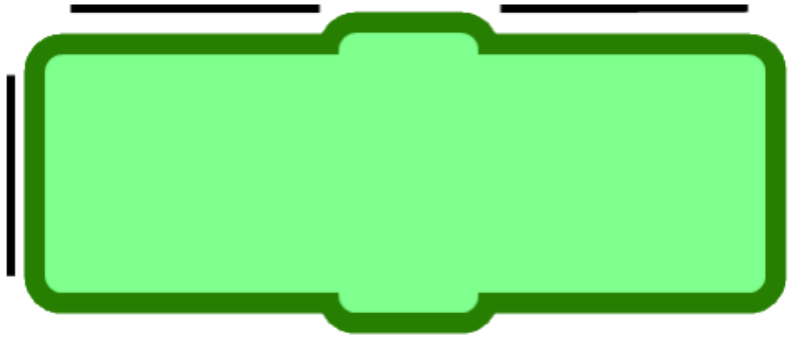
- obrazy (np. przyciski) ze specjalnym rodzajem skalowania;
- określenie wyglądu obramowań;
- używane do android:background;
- cały obrazek podzielony jest na 9 obszarów.

9 części

- 4 narożniki - statyczne;
- 4 obszary pomiędzy narożnikami - powielane w jednym wymiarze (na szerokość, lub na wysokość);
- środek - skalowalny.

Bardziej złożone struktury

- więcej elementów niż 9;
- możliwe skalowanie tylko poziomo, lub tylko pionowo;
- wybrane fragmenty ramki mogą być stale.



Rysunek: Przykład ramki

Rysunek: Stany przycisków

Interaktywność kontrolek

- interaktywność kontrolek;
- osobne grafiki dla poszczególnych stanów przycisku;
- nie mamy Listenera do przycisku.

Stany kontrolek

- android:state enabled – czy element jest aktywny;
- android:state focused – czy element ma fokus;
- android:state pressed – czy element jest wciśnięty;
- android:state checked – czy element jest wybrany.

Na koniec o powiadomieniach

- Toast - interaktywność (brak) - Krótko widoczne informacje wyświetlane domyślnie pośrodku dolnej części ekranu.
- Pasek stanu - interaktywność (mała) - Zdarzenia oczekujące akcji użytkownika (np. powiadomienie o sms, albo o e-mailu);
- Okno dialogowe - interaktywność (duża).

Typy intencji

- Intencja bezwarunkowa (Implicit) - nie ma znaczenia, który komponent obsługuje intencję - np. aplikacja, która umożliwia zrobienie zdjęcia. Nie jest istotne, która to będzie aplikacja, o ile jest w stanie zrobić zdjęcie.
- Intencja warunkowa (Explicit) - dokładnie wskazujemy, który komponent obsłuży tę intencję.

Typy intencji - przykład

- Intencja bezwarunkowa (Explicit) - aplikacja zbudowana na podstawie kilku intencji - wywołanie różnych okien aplikacji, ale mamy zdefiniowane, który przycisk wywołuje konkretną intencję.
- Intencja warunkowa (Implicit) - aplikacja, w której możliwe jest wysłanie e-maila, po naciśnięciu przycisku - nie ma znaczenia, jaka aplikacja do wysyłania maili obsłuży te intencję.
- Itencja typu Implicit (przykład 2) - uruchomienie strony internetowej po naciśnięciu przycisku - dowolna przeglądarka internetowa.

Jeszcze intencje

- Implicit - wymagana jest akcja oraz data URI (Uniform Resource Identifier). Opcjonalnie (kategoria, komponenty);
- Explicit - Kontekst i komponent (przeważnie klasa/ aktywność). Kontekst (Context) jest bazową klasą dla klas Activity, Service, Application - czyli ogólnie globalna informacja umożliwiająca dostęp do zasobów, wywoływania intencji. Zatem nasza przykładowa intencja była typu Explicit.

Event Listener

- akcja - kliknięcie, długie kliknięcie, przeciągnięcie po ekranie;
- nasłuchiwanie zdarzenia;
- właściwość onClick danego elementu powiązana z konkretną metodą;
- metoda umożliwiająca wykonanie określonej czynności (wyświetlenie tekstu, zmiana koloru tła, wypisanie tekstu na ekran, uruchomienie nowej aktywności);

Metody Callback

- onClick();
- onLongClick();
- onFocusChange();

Dziękuję za uwagę