

ANDROID platforma principi funkcionisanja aplikacija

Mobilni i distribuirani informacijski sistemi

Mr Bratislav Predić

2012. godina



Android aplikacije

- Android aplikacija se može sastojati od više komponenti:
 - Activity
Jedan grafički korisnički interfejs (ekran) na kome korisnik obavlja jedan tip aktivnosti. Jedan activity se prikazuje po startovanju aplikacije i može da aktivira ostale.
 - Service
Servis nema GUI već u pozadini obavlja neku aktivnost. Definiše interfejs za komunikaciju sa ostalim komponentama
 - Broadcast receiver
Prima i obrađuje obaveštenja. Obaveštenja šalje sistem ili druge aplikacije. Obično startuju activity.
 - Content provider
Omogućava drugim aplikacijama pristup podacima naše aplikacije.



Android aplikacije

- Svaka Android aplikacija se izvršava u svom procesu u svojoj instanci Dalvik virtuelne mašine
- Životni ciklus aplikacije
 - Aplikacija ne kontroliše direktno svoj životni ciklus
 - Životni ciklus svake aplikacije kontroliše sistem
 - Sistem kontroliše životni ciklus aplikacije na osnovu startovanih komponenti aplikacije, prioriteta aplikacije i raspoložive memorije


Početak

Životni ciklus aplikacije:
Aktivna/neaktivna
Vidljiva/nevidljiva

Kraj

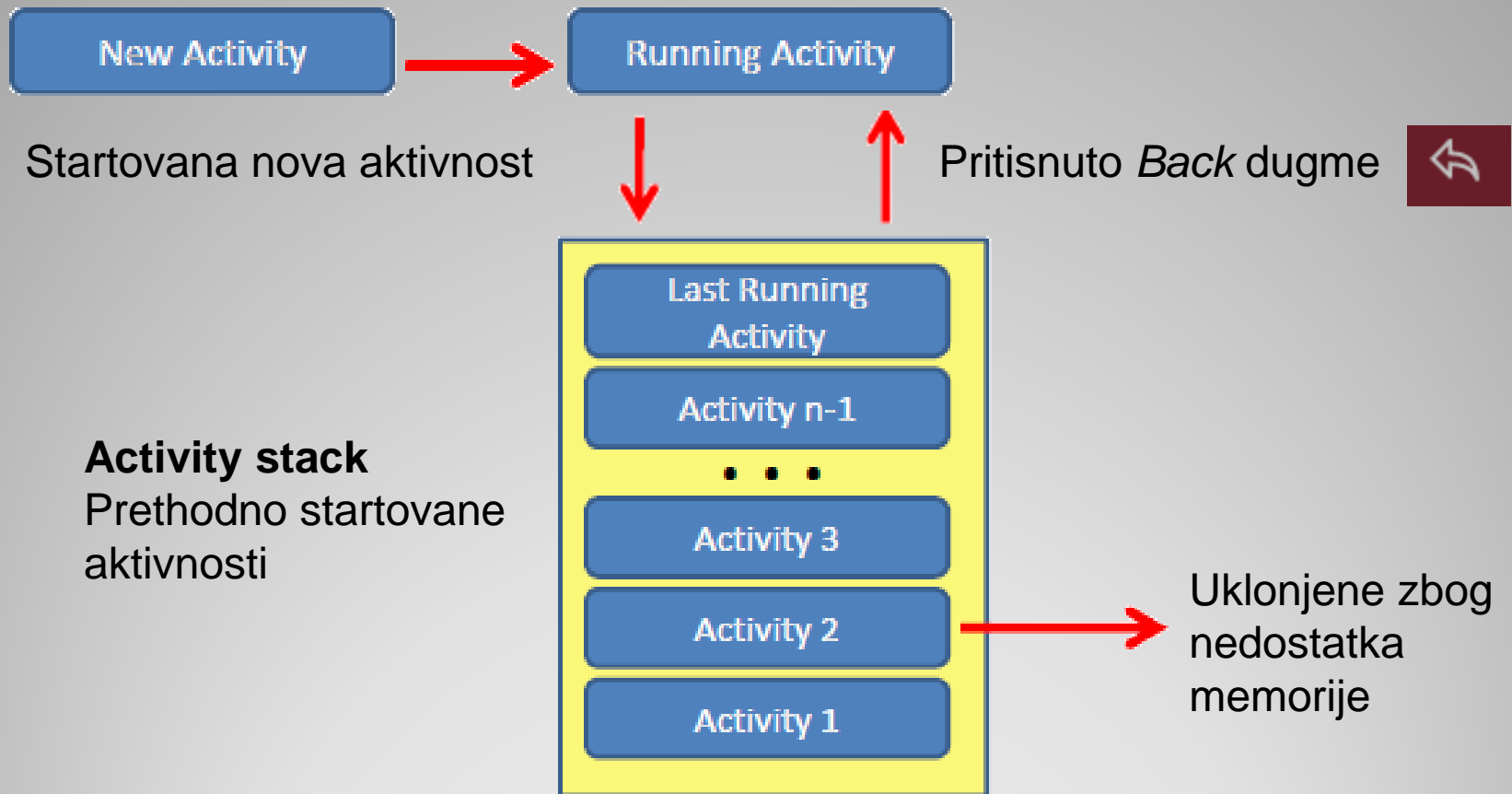


Activity stack

- Activity stack vodi evidenciju o startovanim aktivnostima
- Kada se startuje nova aktivnost smešta se na vrh stack-a i ona je aktivna
- Prethodno aktivna aktivnost ostaje ispod na stack-u
- Kada pritisnemo *Back* dugme  aktivnost ispod na stack-u se pomera na vrh i postaje aktivna

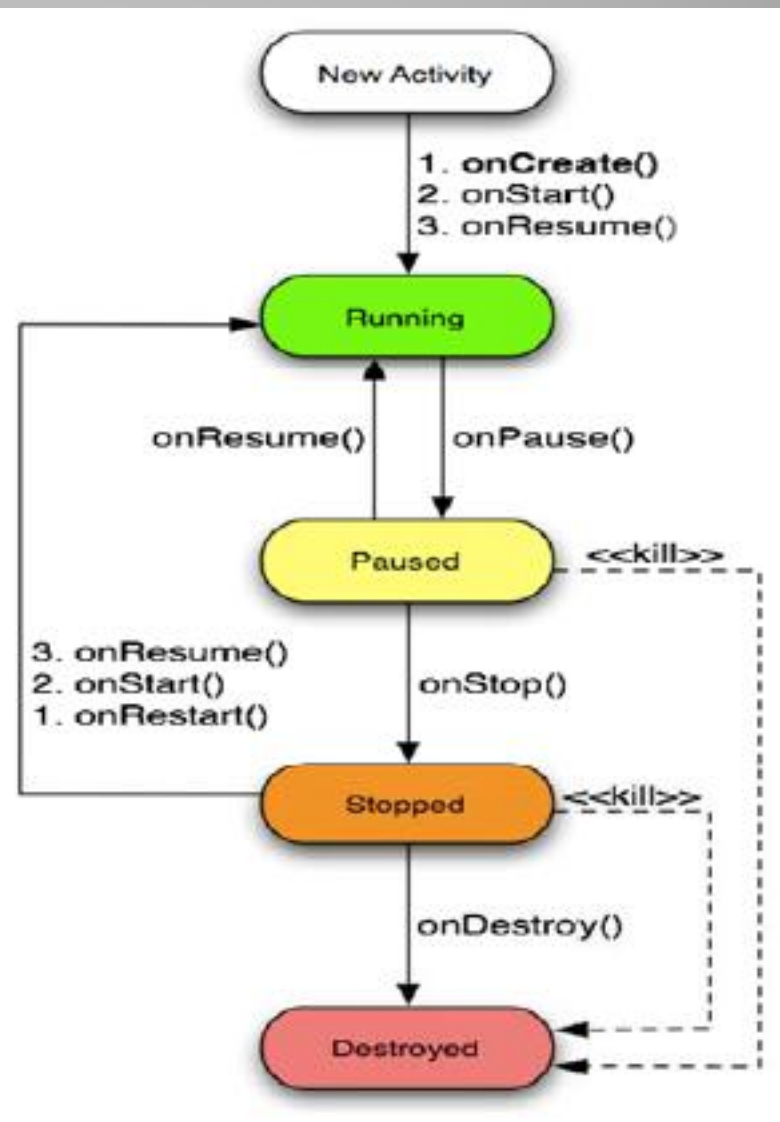


Activity stack



Stanja životnog ciklusa

- Aktivnost može biti u tri stanja
 - Aktivna (izvršava se)
 - Pauzirana
 - Zaustavljena

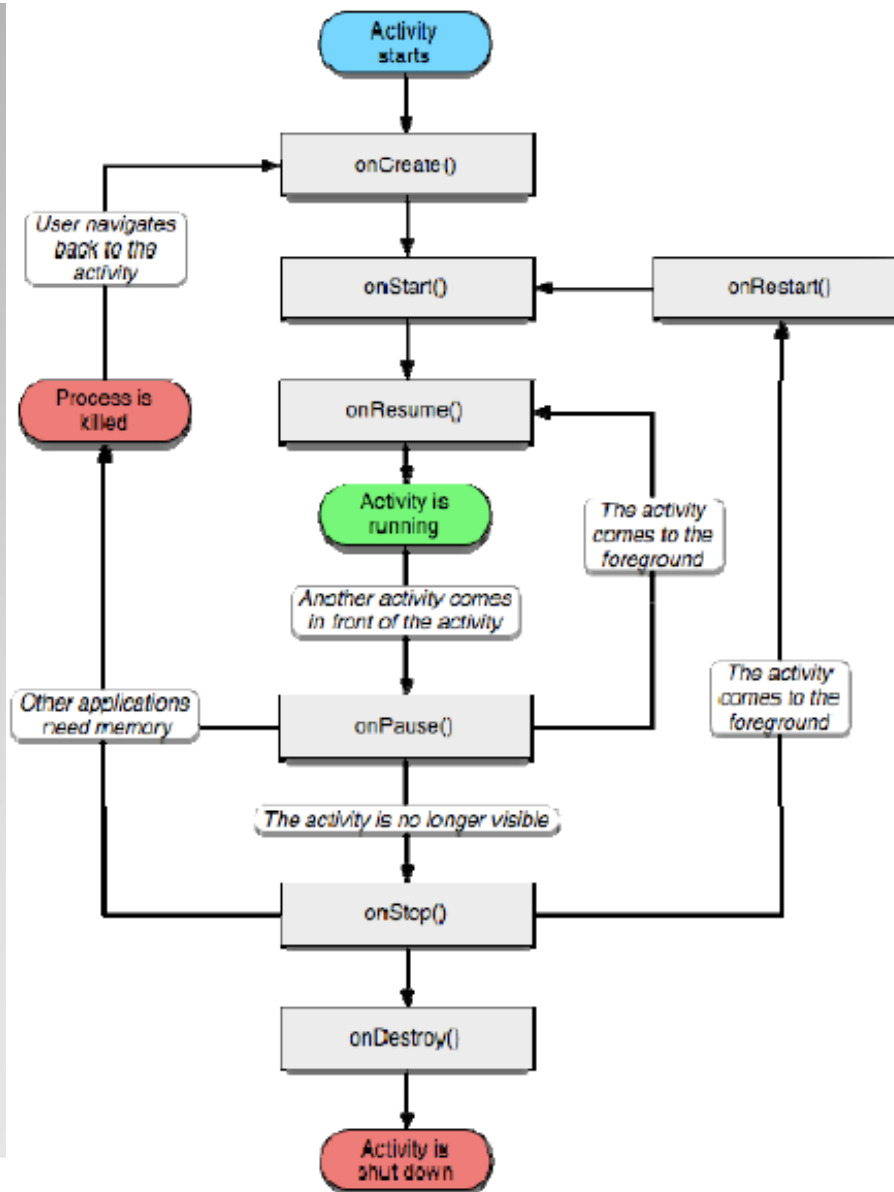


Stanja životnog ciklusa

- Stanje aktivan (izvršava se)
active (running)
 - Aktivnost je na vrhu activity stack-a za trenutni task i ima fokus korisničke interakcije
- Stanje pauziran (paused)
 - Aktivnost nema više fokus ali je i dalje vidljiva korisniku (potpuno ili delimično)
 - Aktivnost čuva sva stanja i vrednosti članica klase
 - Sistem će ovakvu aktivnost uništiti samo pri ekstremnim uslovima nedostatka memorije
- Stanje zaustavljen (stopped)
 - Aktivnost više nije vidljiva korisniku
 - I dalje čuva sva stanja i članice klase



Metode životnog ciklusa



Životni ciklus - eksperiment

- Kreiramo Android aplikaciju *LifeCycle*
- Layout *main.xml* treba da sadrži
 - Button (text: Finish, id: btnFinish)
 - EditText (text: "", id: txtMsg)
- U metodi onCreate() povezati kontrole sa promenljivama i prikazati poruku (toast)
 - `Toast.makeText(this, "onCreate", 1).show();`
- Click metoda samo poziva *finish()* koja zatvara aplikaciju
- Dodajmo po jednu poruku svim bitnim callback metodama životnog ciklusa
 - *onStart, onResume, onPause, onStop, onDestroy, onRestart*



Životni ciklus - eksperiment

- Eksperimentisati sa aplikacijom
- Zapaziti redosled Toast poruka
- Pritisnuti Finish dugme
- Ponovo startovati aplikaciju
- Dok je aplikacija aktivna pritisnuti HOME dugme
- Iz Launch pad-a klikom na ikonu aplikacije vratiti se u aplikaciju
- Klikom na zelenu slušalicu da li se aplikacija stopira ili pauzira?
- BACK dugmetom se vratiti u aplikaciju
- Šta se dešava kada dugo zadržimo HANG-UP dugme?

Životni ciklus - eksperiment

- Startovati drugi emulator
- Iz drugog emulatora uputiti glasovni poziv na prvi emulator
- Iz drugog emulatora poslati SMS prvom emulatoru
- U EditText upisati neki tekst i proveriti u kojim slučajevima se tekst pamti
- Moramo nekako da obezbedimo pamćenje unetog teksta
- U metodi *onPause()* pamtimo, a u metodi *onResume()* vraćamo zapamćeni tekst



Životni ciklus - eksperiment

- Modifikacija metoda *onPause()* i *onResume()*
- **onPause()**

```
SharedPreferences myFile1 = getSharedPreferences("myFile1",  
        Activity.MODE_PRIVATE);  
SharedPreferences.Editor myEditor = myFile1.edit();  
String temp = txtMsg.getText().toString();  
myEditor.putString("mydata", temp);  
myEditor.commit();
```

- **onResume()**

```
SharedPreferences myFile = getSharedPreferences("myFile1",  
        Activity.MODE_PRIVATE);  
if ( (myFile != null) && (myFile.contains("mydata")) ) {  
    String temp = myFile.getString("mydata", "***");  
    txtMsg.setText(temp);  
}
```



Životni ciklus - sumarno

- Ako je aktivnost pauzirana ili zaustavljena sistem je može prekinuti:
 - Tako što traži od aktivnosti da se zaustavi, pozivom *finish()* metode ili
 - Može samo da ubije proces (bez poziva *finish()* metode)
- Aktivnost se ponovo kreira ukoliko treba da se prikaže korisniku
- Prilikom prelaza između stanja sistem obaveštava aktivnost pozivom callback metoda

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onRestart()
void onResume()
```

```
void onPause()
void onStop()
void onDestroy()
```



Događaji životnog ciklusa

- Događaji životnog ciklusa su hook metode koje redefinišete kako bi proširili funkcionalnost aplikacije
- Obavezno
 - *onCreate()* – inicijalni setup kada se objekat instancira
- Preporučeno
 - *onPause()* – perzistencija promena podataka i priprema za prekid interakcije sa korisnikom
- **Kompletan životni vek** aplikacije se dešava između poziva
 - *onCreate()* i
 - *onDestroy()*



Događaji životnog ciklusa

- **Vidljiv životni vek** aplikacije je između metoda
 - *onStart()* i
 - *onStop()*
- Tokom ovog dela životnog veka aplikacija je vidljiva (ne mora da znači i da ima fokus)
- Ovaj par metoda se mogu pozivati više puta tokom životnog veka aplikacije
- **Aktivan životni vek** (aplikacija interaguje sa korisnikom) se dešava između poziva metoda
 - *onPause()* – kada se uređaj uspava ili startuje nova aktivnost i
 - *onResume()* – isporučen je novi intent



Događaji životnog ciklusa

- **onCreate()**

- Kada se aktivnost inicijalno kreira
- U ovoj metodi ide kreiranje view-a, povezivanje podataka sa listama...
- Ovoj metodi se prosleđuje Bundle koji sadrži prethodno snimljeno stanje aplikacije
- Uvek posle ove metode sledi *onStart()*

- **onRestart()**

- Neposredno pre ponovnog startovanja aktivnosti
- Uvek posle ove metode sledi *onStart()*

- **onStart()**

- Neposredno pre nego se aktivnost prikaže korisniku
- Sledi ili *onResume()* ili *onStop()*



Događaji životnog ciklusa

- **onResume()**

- Neposredno pre nego aktivnost počne interakciju sa korisnikom
- Aktivnost je na vrhu steka aktivnosti

- **onPause()**

- Poziva se kada sistem planira da startuje drugu aktivnost
- U ovoj metodi se tipično snimaju perzistentni podaci
- Metoda treba da bude kratka zato što blokira sledeću aktivnost da se pojavi
- Prate je pozivi *onResume()* ili *onStop()*
- U ovom stanju sistem može da uništi aktivnost



Događaji životnog ciklusa

- **onStop()**

- Poziva se kada aktivnost više nije vidljiva korisniku
- Prati je poziv *onRestart()* ili *onDestroy()*
- U ovom stanju sistem može da uništi aktivnost

- **onDestroy()**

- Poziva se pre nego se aktivnost uništi
- Ili je pozvana *Finish()* metoda ili sistem uništava aktivnost
- *isFinishing()* metoda razlikuje ova dva slučaja

- Samo **onPause()** metoda se garantovano poziva, sistem može da uništi aktivnost bez poziva *onStop()* i *onDestroy()*



Preferences

- Preference su mehanizam koji omogućava pamćenje ključ-vrednost parova primitivnih tipova podataka
- Preference mogu da budu deljene između komponenti iste aplikacije
 - `Context.getSharedPreferences()` ili
- privatne za aktivnost
 - `Activity.getPreferences()`
- Deljene preference imaju imena
- Preference se ne mogu deliti između aplikacija (jedino content provider-om)



Primer

- Primer demonstrira tranzicije između stanja i perzistenciju podataka
- Dodajmo u aktivnost ključ i mod rada preferenci

```
public static final String MYPREFSID = "MyPrefs001";  
public static final int actMode = Activity.MODE_PRIVATE;
```

- Dodajmo i listener za TextEdit

```
txtMsg.addTextChangedListener(new TextWatcher() {  
    public void onTextChanged(CharSequence s, int start, int before, int count)  
        {}  
    public void beforeTextChanged(CharSequence s, int start, int count, int  
after) {}  
    public void afterTextChanged(Editable s) {}  
});
```



Primer

- Stanje aktivnosti snimamo u *onPause()*

```
@Override
protected void onPause() {
    super.onPause();
    saveDataFromCurrentState();
    Toast.makeText(this, "onPause", 1).show();
}
```

- Učitavamo snimljeno stanje u *onStart()*

```
@Override
protected void onStart() {
    super.onStart();
    updateFromSavedState();
    Toast.makeText(this, "onStart", 1).show();
}
```



Metode za snimanje stanja

```
protected void saveDataFromCurrentState() {
    SharedPreferences myPrefs = getSharedPreferences(MYPREFSID, actMode);
    SharedPreferences.Editor myEditor = myPrefs.edit();
    myEditor.putString("txt", txtMsg.getText().toString());
    myEditor.commit();
} // saveDataFromCurrentState

protected void updateFromSavedState() {
    SharedPreferences myPrefs = getSharedPreferences(MYPREFSID, actMode);
    if ((myPrefs != null) && (myPrefs.contains("txt"))) {
        String text = myPrefs.getString("txt", "");
        txtMsg.setText(text);
    }
} // UpdateFromSavedState

protected void clearMyPreferences() {
    SharedPreferences myPrefs = getSharedPreferences(MYPREFSID, actMode);
    SharedPreferences.Editor myEditor = myPrefs.edit();
    myEditor.clear();
    myEditor.commit();
}
```



Automatsko snimanje stanja

- Metoda *onRestoreInstanceState()* se poziva posle *onStart()* kada aktivnost ima prethodno snimljeno stanje

```
/*  
protected void onRestoreInstanceState(Bundle savedInstanceState)  
This method is called after onStart() when the activity is being  
re-initialized  
from a previously saved state.  
The default implementation of this method performs a restore of any view  
state  
that had previously been frozen by onSaveInstanceState(Bundle).  
*/  
@Override  
protected void onRestoreInstanceState(Bundle savedInstanceState) {  
    super.onRestoreInstanceState(savedInstanceState);  
    Toast.makeText(getBaseContext(),  
        "onRestoreInstanceState ...BUNDLING", Toast.LENGTH_LONG).show();  
}
```



Automatsko snimanje stanja

- Metoda *onSaveInstanceState()* se poziva kada aktivnost može da bude uništena

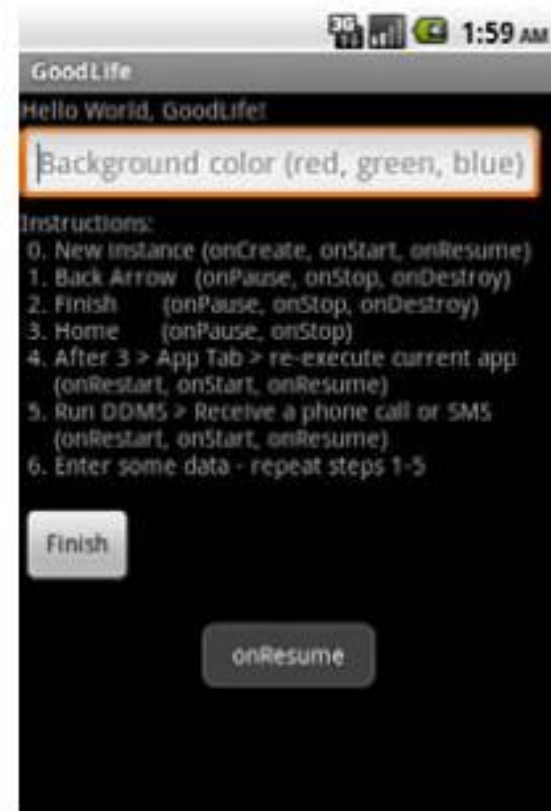
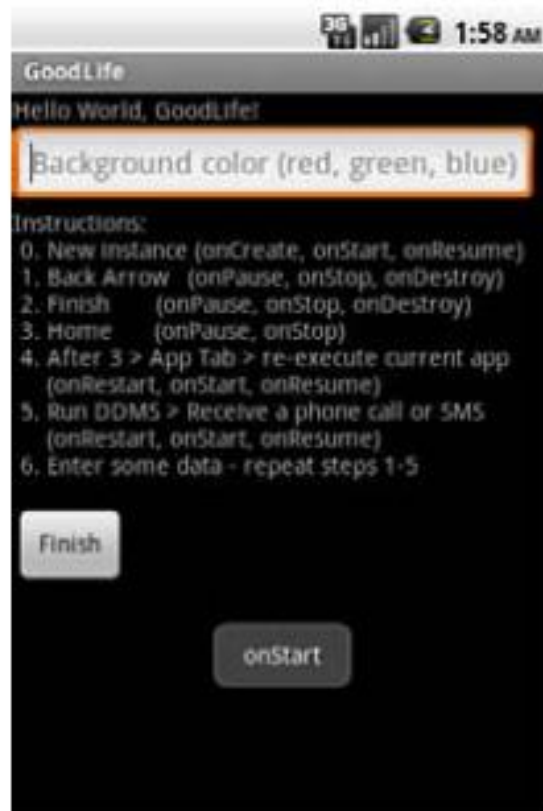
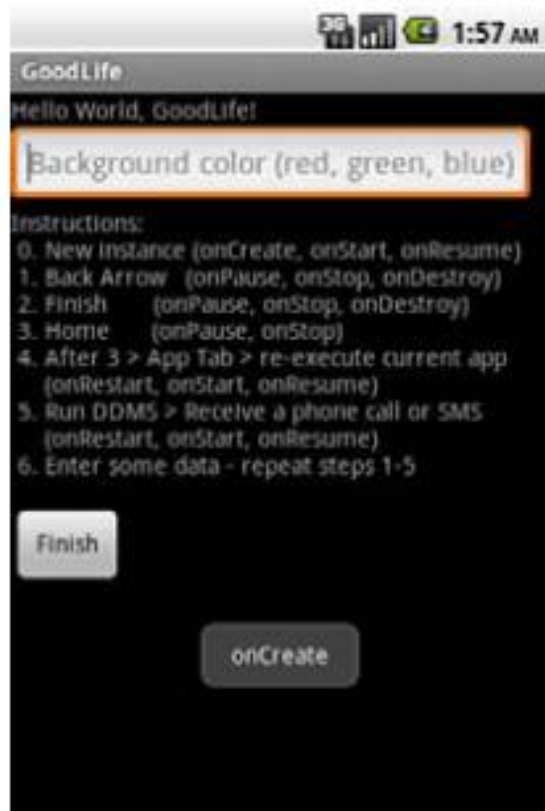
```
/*
protected void onSaveInstanceState(Bundle outState)
Called to retrieve per-instance state from an activity before being
killed so that the state can be restored in onCreate(Bundle) or
onRestoreInstanceState(Bundle) (the Bundle populated by this method
will be passed to both).
This method is called before an activity may be killed so that when it
comes back some time in the future it can restore its state. For example,
if activity B is launched in front of activity A, and at some point
activity A is killed to reclaim resources, activity A will have a chance
to save the current state of its user interface via this method so that
when the user returns to activity A, the state of the user interface can
be restored via: onCreate(Bundle) or onRestoreInstanceState(Bundle).
*/
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    Toast.makeText(getBaseContext(), "onSaveInstanceState ...BUNDLING",
        Toast.LENGTH_LONG).show();
} // onSaveInstanceState
```


Tranzicija stanja - startovanje

onCreate...

onStart...

onResume...

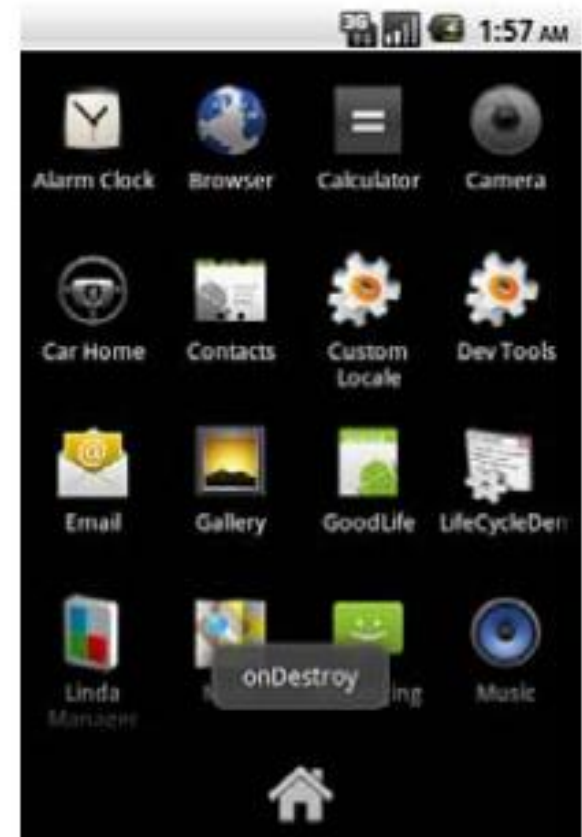


Tranzicija stanja - Back

onPause...

onStop...

onDestroy...



After pressing "Back Arrow"

Tranzicija stanja

Posle pritiska HOME tastera	Po ponovnom startovanju aplikacije	Posle pritiska BACK tastera ili Finish dugmeta
onSavedInstanceState onPause onStop	onRestart onStart onResume	onPause onStop onDestroy

Perzistencija podataka

1. Uneti neki tekst
2. Pritisnuti HOME dugme
3. onSavedInstanceState > onPause > onStop
4. Koristiti druge aplikacije
5. Ponovo startovati našu aplikaciju
6. onRestart > onStart > onResume
7. Vidimo tekst koji smo uneli u koraku 1



Automatsko snimanje stanja

- Metode *onRestoreInstanceState()* i *onSaveInstanceState()* se pozivaju kada sistem automatski uništi aplikaciju
 - Na primer prilikom promene orijentacije ekrana
- Korišćenje Bundle-a prilikom kreiranja aktivnosti

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ... somevalue = savedInstanceState.getString(SOME_KEY);
    ...
}

...
@Override protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putString(SOME_KEY, "blah blah blah");
}
```

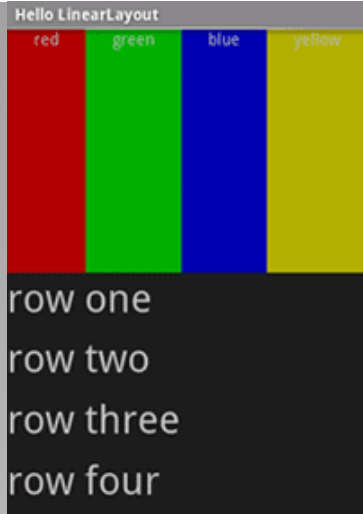


Korisnički interfejs

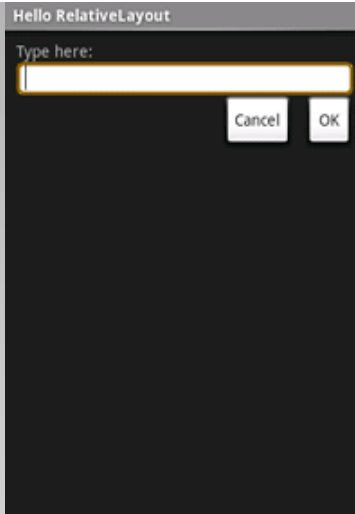
- Klasa *View* predstavlja osnovu UI-a
- *View* zauzima pravougaonu oblast na ekranu, zadužen je za iscrtavanje i obradu događaja
- *View* klasa je osnova za *Widget-e* koji predstavljaju UI kontrole
- *ViewGroup* klase je osnova za layout-e
 - Nevidljivi kontejneri koji sadrže ostale *View* ili *ViewGroup* klase
- Nad *View* instancama se obično
 - Postavljaju properties, postavlja fokus, postavljaju listener-i, postavlja vidljivost



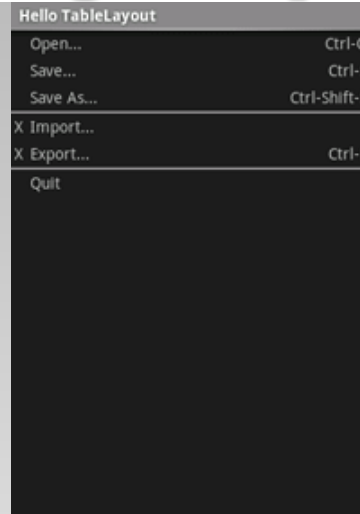
Korisnički interfejs - primeri



LinearLayout



RelativeLayout

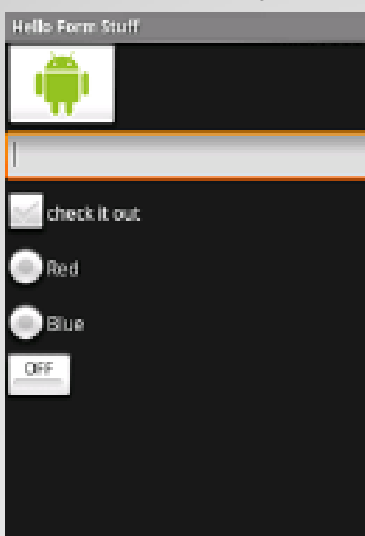


TableLayout

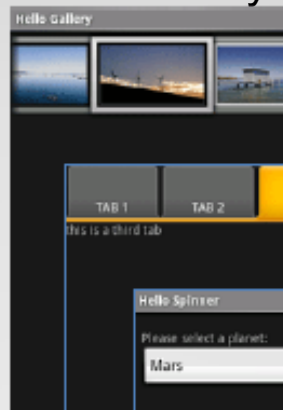
Layouts



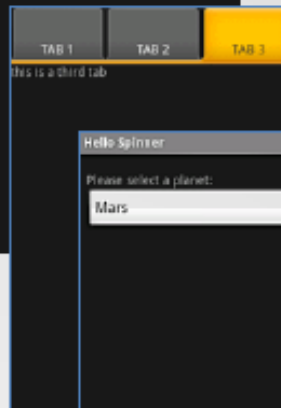
DatePicker



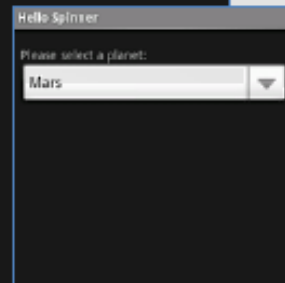
FormControls



GalleryView



TabWidget



Spinner

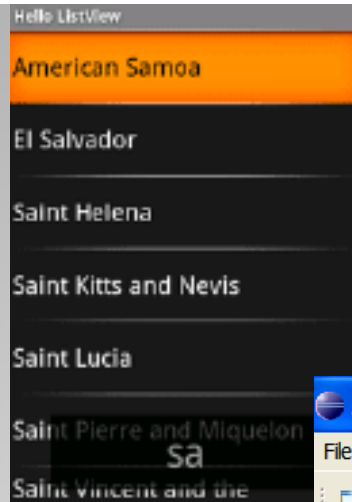
Widgets

Korisnički interfejs - primeri

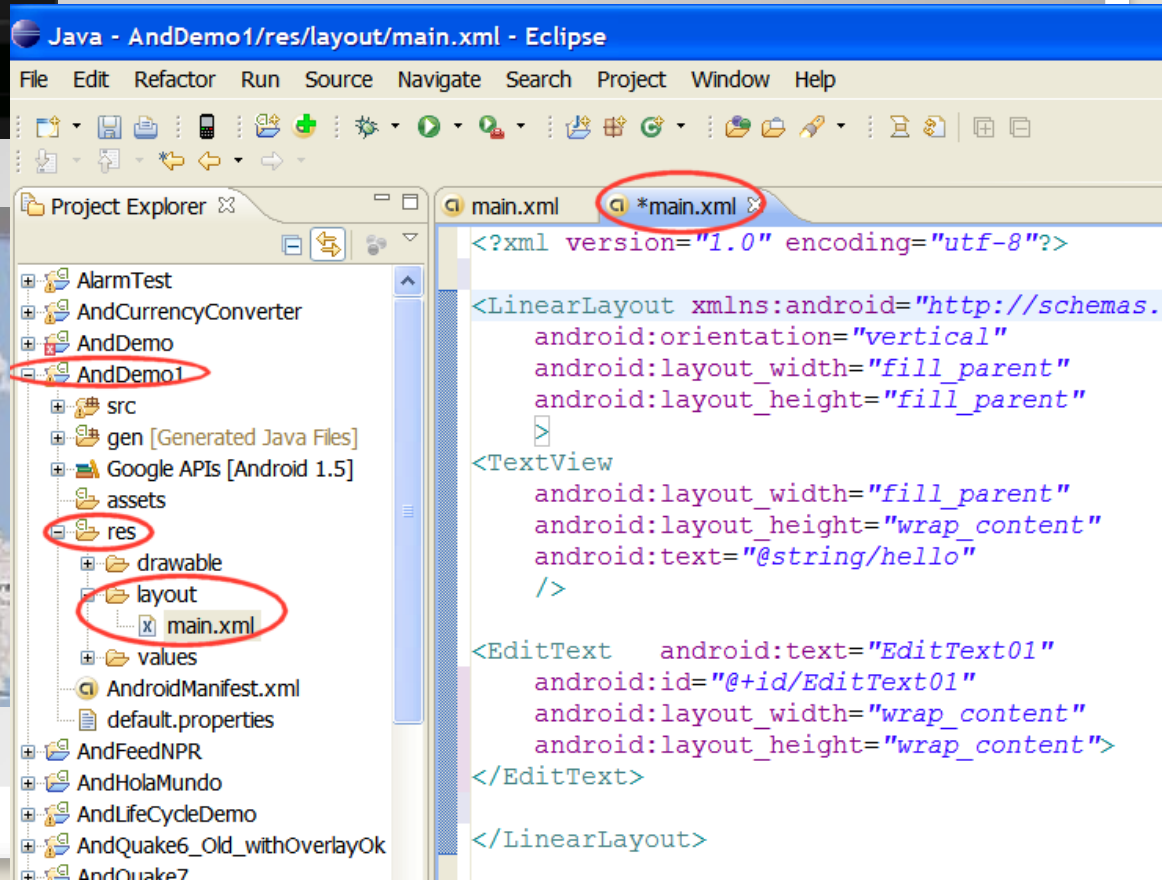
- Layout se specificira XML dokumentom
- Layout je resurs



AutoComplete
TextView



ListView



WebView



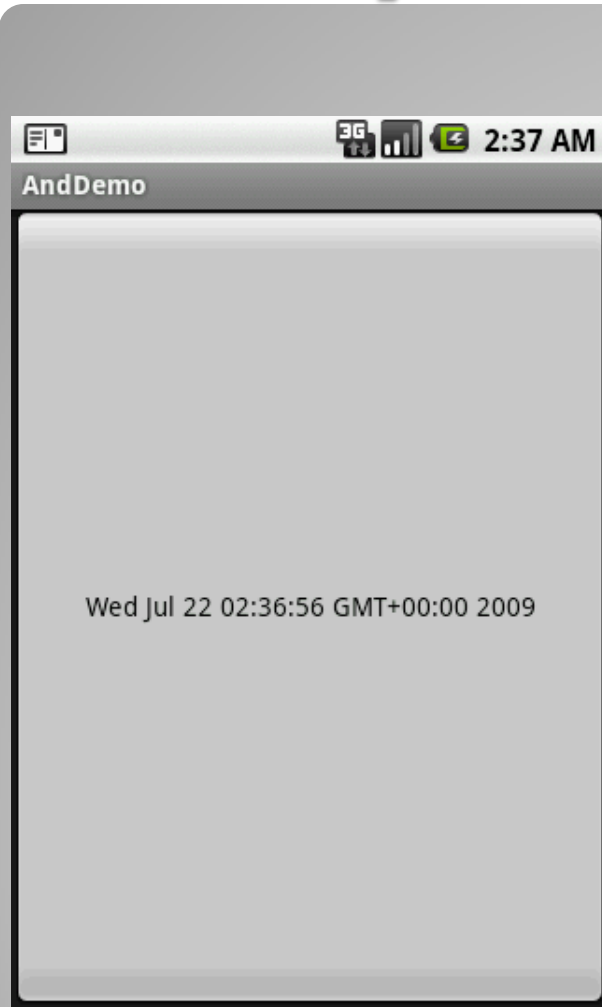
MapView

XML layout

- XML layout predstavlja stablo UI elemenata
- Atributi XML elemenata su properties i određuju kako izgledaju i kako se ponašaju UI elementi
- Svaki element UI-a u XML layout-u kome želimo da pristupama iz koda mora da ima dodeljen ID atribut
 - `android:id`
- Želimo da prikažemo tekst na dugmetu
 - `android:text`
- Dimenzije dugmeta (kako popunjava roditeljski kontejner)
 - `android:layout_width` i `android:layout_height`



XML layout



```
import java.util.Date;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

Public class AndDemo extends Activity {
    Button btn;
    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.main);
        btn= (Button) findViewById(R.id.myButton);
        btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                updateTime();
            }
        });
    } // onCreate
    private void updateTime() {
        btn.setText(new Date().toString());
    }
}
```

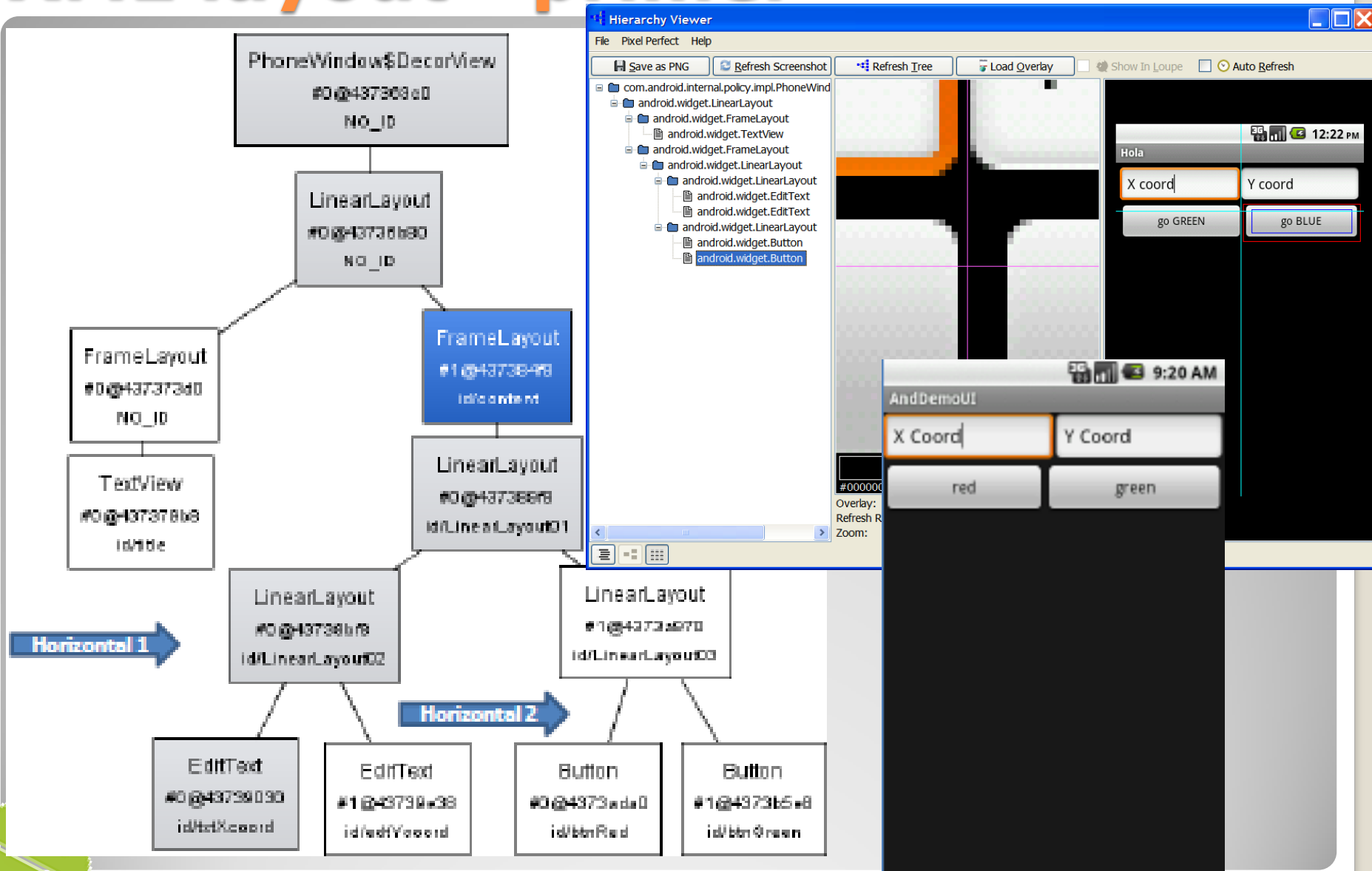
XML layout

```
<?xmlversion="1.0"encoding="utf-8"?>
<Buttonxmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/myButton"
android:text=""
android:layout_width="fill_parent"
android:layout_height="fill_parent"
/>
```

- Alat *HierarchyViewer* (iz Tools podfoldera u SDK) prikazuje UI stablo trenutno aktivne aplikacije (na emulatoru ili povezanom uređaju)
- Svaki element u XML-u je ili View ili ViewGroup
- Android UI framework obilazi stablo pre-order i svaki čvor se iscrtava



XML layout - primer



XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
  android:layout_width="fill_parent" android:layout_height="fill_parent"
  android:orientation="vertical" xmlns:android="http://schemas.android.com/apk/res/android">
  <LinearLayout android:id="@+id/LinearLayout02"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <EditText android:id="@+id/txtXcoord" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="X Coord"
      android:layout_weight="1">
    </EditText>
    <EditText android:id="@+id/edtYcoord" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="Y Coord"
      android:layout_weight="1">
    </EditText>
  </LinearLayout>
  <LinearLayout android:id="@+id/LinearLayout03"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <Button android:id="@+id/btnRed" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="red"
      android:layout_weight="1">
    </Button>
    <Button android:id="@+id/btnGreen" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="green"
      android:layout_weight="1">
    </Button>
  </LinearLayout>
</LinearLayout>
```



XML layout kontejneri

- LinearLayout je najčešće korišćen
- Može se ugnježdavati
- Najčešće korišćeni layout-i:

