

# Aktivnosti, namere, fragmenti i prava pristupa

## Mobilne aplikacije

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti
- 3 Zadaci i povratni stek
- 4 Namere i filteri namera
- 5 Startovanje aktivnosti
- 6 Fragmenti
- 7 Prava pristupa

# Komponente Android aplikacije

- Aktivnosti (activities)
- Servisi (services)
- Dobavljači sadržaja (content providers)
- Prijemnici poruka (broadcast receivers)

# Šta je aktivnost?

- Aktivnost je pojedinačna fokusirana stvar koju korisnik može da uradi
- Aktivnost je pojedinačan ekran Android aplikacije

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti**
- 3 Zadaci i povratni stek
- 4 Namere i filteri namera
- 5 Startovanje aktivnosti
- 6 Fragmenti
- 7 Prava pristupa

# Pravljenje aktivnosti

- Definirati klasu koja nasljeđuje Activity klasu (ili neku od njenih nasljednica)
- Dodati activity element u `AndroidManifest.xml`

# Pravljenje aktivnosti

```
1 package com.example.project ;
2 import android.app.Activity ;
3
4 public class ExampleActivity extends Activity {
5
6     @Override
7     public void onCreate (...) {
8         // ...
9     }
10
11     @Override
12     public void onStart () {
13         // ...
14     }
15
16     @Override
17     public void onRestart () {
18         // ...
19     }
20 }
```

# Pravljenje aktivnosti

```

    @Override
2   public void onResume() {
        // ...
4   }

    @Override
6   public void onPause() {
        // ...
8   }

    @Override
10  public void onStop() {
        // ...
14  }

    @Override
16  public void onDestroy() {
        // ...
18  }
20 }
```



# Pravljenje aktivnosti

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3      <application ... >
4          <activity android:name=".ExampleActivity" ... >
5              <intent-filter>
6                  <action android:name="android.intent.action.MAIN" />
7                  <category android:name="android.intent.category.LAUNCHER" />
8              </intent-filter>
9          </activity>
10 </application>
11 </manifest>
12
```

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti**
- 3 Zadaci i povratni stek
- 4 Namere i filteri namera
- 5 Startovanje aktivnosti
- 6 Fragmenti
- 7 Prava pristupa

# Životni ciklus aktivnosti

Aktivnost može da se nalazi u jednom od tri stanja:

- resumed (aktivnost se izvršava)
- paused (aktivnost je pauzirana)
- stopped (aktivnost je zaustavljena)

# Životni ciklus aktivnosti

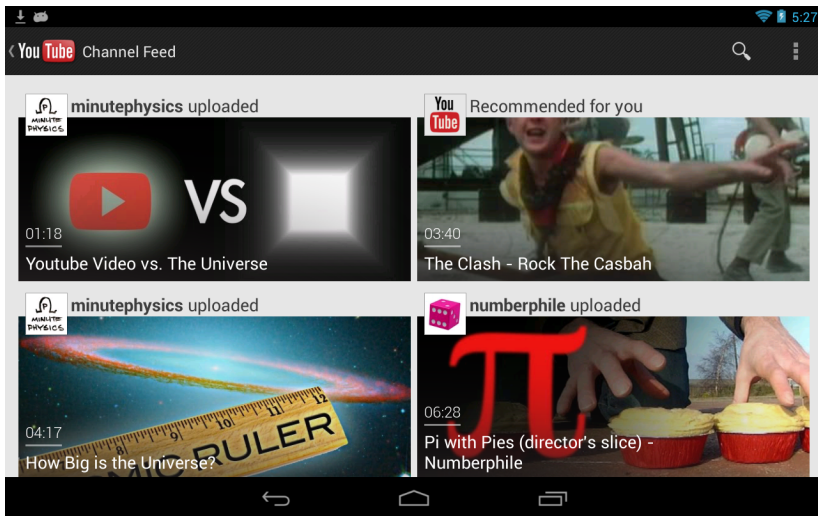


Figure 1: Aktivnost se izvršava (resumed).

# Životni ciklus aktivnosti

- Aktivnost se izvršava ako se nalazi u prvom planu i ima fokus.

# Životni ciklus aktivnosti

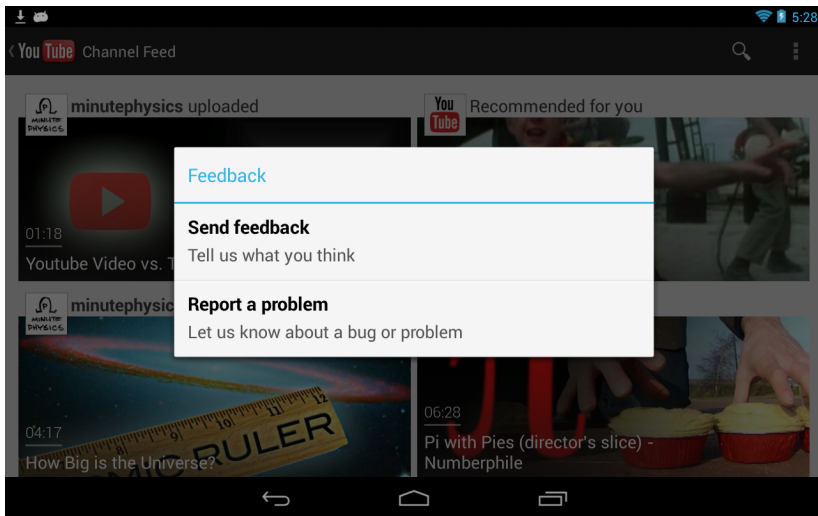


Figure 2: Aktivnost je pauzirana (paused).

# Životni ciklus aktivnosti

- Aktivnost je pauzirana ako se druga aktivnost nalazi u prvom planu i ima fokus, ali je prva aktivnost još uvek vidljiva (zato što je druga aktivnost transparentna ili ne pokriva ceo ekran).
- Pauzirana aktivnost je "živa" (instancija klase je zadržana u memoriji i povezana je sa rukovaocem prozora), ali može biti "ubijena" ako sistem ima jako malo slobodne memorije.

# Životni ciklus aktivnosti

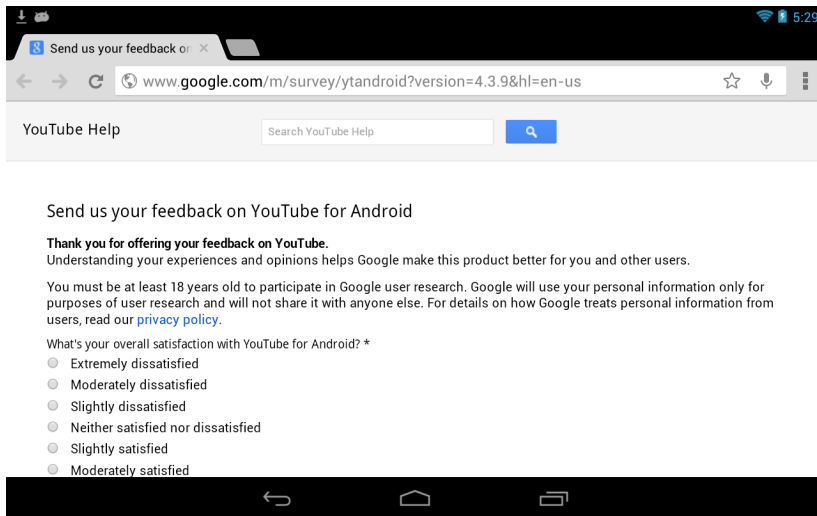


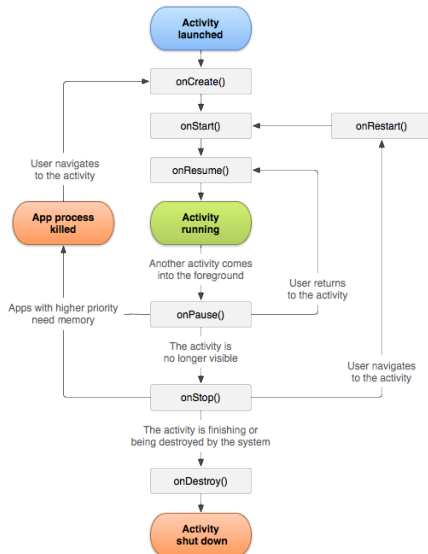
Figure 3: Aktivnost je zaustavljena (stopped).



# Životni ciklus aktivnosti

- Aktivnost je zaustavljena ako se nalazi u pozadini (potpuno je prekrivena drugom aktivnošću).
- Zaustavljena aktivnost je "živa" (instanca klase je zadržana u memoriji, ali nije povezana sa rukovaocem prozora), ali može biti "ubijena" ako sistem ima malo slobodne memorije.

# Životni ciklus aktivnosti



# onCreate

- Sistem poziva `onCreate` metodu kada startuje aktivnost.
- Ova metoda treba da zauzme resurse i inicijalizuje komponente neophodne za pravilno funkcionisanje aktivnosti.
- Pozivom `setContentView` metode iscrtava se korisnički interfejs.

# onCreate

```
@Override
2 protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
4    setContentView(R.layout.main);
    ...
6 }
```

# onStart

- Sistem poziva `onStart` metodu neposredno pre nego što aktivnost postane vidljiva korisniku.

# onStart

```
@Override
2 protected void onStart() {
    super.onStart();
4     ...
    }
6
```

# onResume

- `onResume` metoda se poziva neposredno pre nego što aktivnost počne interakciju sa korisnikom. U ovom trenutku aktivnost se nalazi na vrhu steka aktivnosti.

# onResume

```
@Override
2 protected void onResume() {
    super.onResume();
4     ...
    }
6
```



# onPause

- Sistem poziva onPause metodu neposredno pre nego što pauzira izvršavanje aktivnosti.
- Ova metoda se obično koristi za snimanje perzistentnih podataka i zaustavljanje procesa koji zauzimaju procesor.
- Mora biti vrlo brza zato što sledeća aktivnost ne može da počne da se izvršava sve dok se ova metoda ne završi.

# onPause

```
@Override
2 protected void onPause() {
    super.onPause();
4     ...
    }
6
```

# onStop

- Poziva se kada aktivnost više nije vidljiva korisniku.

# onStop

```
@Override
2 protected void onStop() {
    super.onStop();
4     ...
    }
6
```

# onRestart

- `onRestart` metoda se poziva nakon što je aktivnost zaustavljena, a pre nego što je ponovo startovana.

# onRestart

```
@Override
2 protected void onRestart() {
    super.onRestart();
4     ...
    }
6
```

# onDestroy

- Poslednja metoda koja se poziva pre nego što se aktivnost uništi.
- Ova metoda oslobađa zauzete resurse pre nego što se aktivnost uništi.

# onDestroy

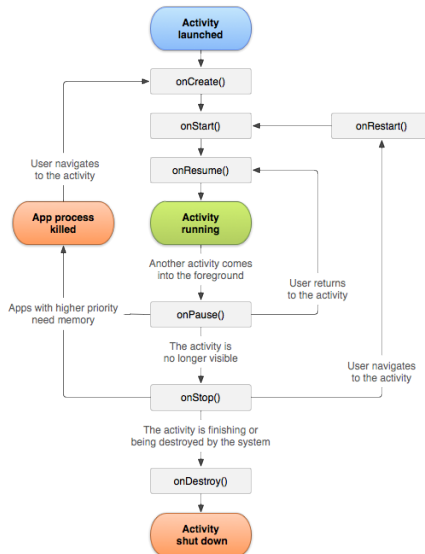
```
@Override
2 protected void onDestroy() {
    super.onDestroy();
4     ...
    }
6
```



# Životni vek aktivnost

- Ceo životni vek
- Životni vek u kome je vidljiva
- Životni vek u kome je u prvom planu

# Životni vek aktivnosti



# Snimanje stanja aktivnosti

- Kada se aktivnost pauzira ili zaustavi, njeno stanje je sačuvano u memoriji.
- Međutim, da bi se sačuvalo stanje aktivnosti ako se ona uništi, potrebno je implementirati dodatnu metodu.
- Oprez: Android može u bilo kom trenutku "ubiti" aktivnost koja se ne nalazi u prvom planu!!!

# Snimanje stanja aktivnosti

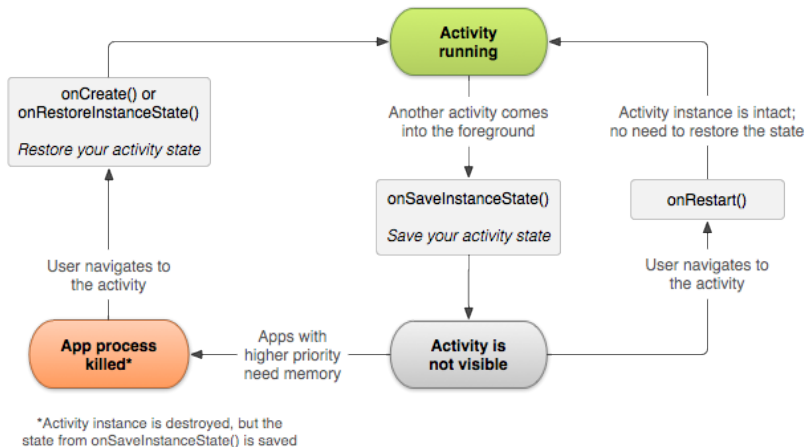


Figure 4: Snimanje stanja aktivnosti.

# onSaveInstanceState

- Poziva se pre nego što se aktivnost uništi da bi se snimilo njeno stanje koje se ponovo inicijalizuje u onCreate ili onRestoreInstanceState metodi.

# onSaveInstanceState

```
@Override
2 protected void onSaveInstanceState(Bundle bundle)() {
    super.onSaveInstanceState(bundle);
4    bundle.putInt("counter", counter);
    ...
6 }
```

## onRestoreInstanceState

- Poziva se posle onStart metode da bi se aktivnost ponovo inicijalizovala iz prethodno snimljenog stanja.

# onRestoreInstanceState

```
@Override
2 protected void onRestoreInstanceState(Bundle bundle) {
    super.onRestoreInstanceState(bundle);
4    counter = bundle.getInt("counter");
    ...
6 }
```



# Snimanje stanja aktivnosti

Klasa Bundle sadrži metode oblika:

- `T getT(String key);`
- `void putT(String key, T value);`

# Snimanje stanja aktivnosti

- Podrazumevana implementacija pomenutih metoda poziva `onSaveInstanceState` metodu nad svakim elementom korisničkog interfejsa što za rezultat ima činjenicu da se stanje korisničkog interfejsa automatski snima.

# Rukovanje promenom konfiguracije

- Ako se konfiguracija uređaja promeni (orijentacija ekrana, jezik, itd.), korisnički interfejs se mora osvežiti da bi odgovarao konfiguraciji.
- Promena konfiguracije prouzrokuje uništenje i ponovno stvaranje aktivnosti.

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti
- 3 Zadaci i povratni stek**
- 4 Namere i filteri namera
- 5 Startovanje aktivnosti
- 6 Fragmenti
- 7 Prava pristupa

# Zadatak

- Aplikacija se obično sastoji iz više aktivnosti.
- Zadatak (task) je skup aktivnosti sa kojima korisnik intereaguje da bi izvršio određen posao.

# Povratni stek

- Aktivnosti su uređene u povratni stek (back stack) u redosledu u kome su startovane.
- Kada se aktivnost startuje, stavlja se na vrh steka, prelazi u prvi plan i dobija fokus.
- Pritiskom na Back dugme, tekuća aktivnost se skida sa vrha steka i uništava.

# Povratni stek

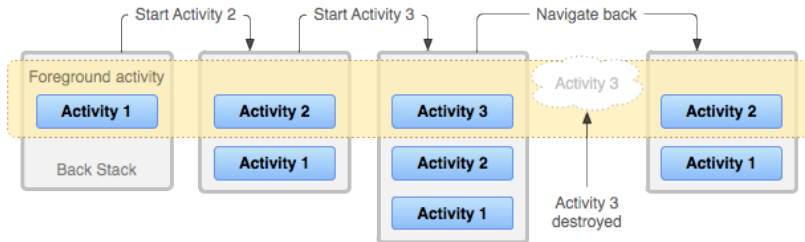


Figure 5: Povratni stek.

# Povratni stek

- Svakom zadatku odgovara jedan povratni stek.
- Samo jedan zadatak se može nalaziti u prvom planu i imati fokus u datom trenutku.
- Zadatak koji je u pozadini ima očuvan svoj povratni stek.



# Upravljanje zadacima

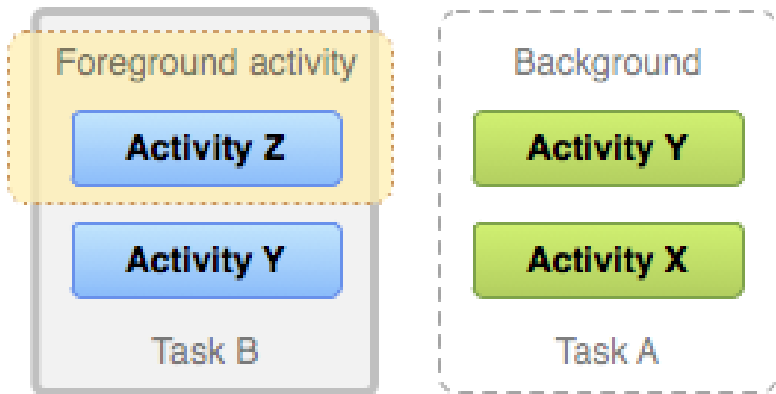


Figure 6: Upravljanje zadacima.

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti
- 3 Zadaci i povratni stek
- 4 Namere i filteri namera**
- 5 Startovanje aktivnosti
- 6 Fragmenti
- 7 Prava pristupa

# Namera

- Namera (intent) je apstraktni opis akcije koja treba da se izvrši.
- Služi za povezivanje komponenti aplikacije.
- Sadrži svojstva potrebna komponenti koja obrađuje nameru (akcija, podaci, dodatne informacije) i sistemu (komponenta, kategorije i oznake).

# Eksplicitne i implicitne namere

- Eksplicitne namere eksplicitno opisuju komponentu koja treba da izvrši akciju.
- Implicitne namere implicitno opisuju akciju koja treba da se izvrši.

# Komponenta

- Opisuje komponentu koja treba da obradi nameru.
- Postavlja se u konstruktoru ili Intent `setClassName(String packageName, String className)` metodom.
- Ukoliko je ovo svojstvo postavljeno, namera je eksplicitna.

# Akcija

- Svojestvo akcija (action) opisuje akciju koja treba da se izvrši.
- Akcija u najvećoj meri određuje kako je strukturiran ostatak namere (podaci i dodatne informacije).
- Postavlja se u konstruktoru ili Intent `setAction(String action)` metodom.
- Preporučuje se korišćenje predefinisanih akcija.

# Akcija

Constant	Meaning
ACTION_MAIN	Start up as the initial activity of a task.
ACTION_CALL	Initiate a phone call.
ACTION_EDIT	Display data for the user to edit.
ACTION_SYNC	Synchronize data on a server with data on the mobile device.

Table 1: Akcije.

# Podaci i tip

- Svojstva podaci (data) i tip (type) opisuju podatke koji treba da se obrade i MIME tip tih podataka.
- Postavljaju se u konstruktoru ili `setData(Uri data)`, `setType(String type)` i `setDataAndType(Uri data, String type)` metodama.
- Zavise od akcije koja treba da se izvrši.



## Dodatne informacije

- Dodatne informacije (extra) potrebne komponenti koja obrađuje nameru opisane su uređenim parovima (ključ, vrednost).
- Postavljaju se metodama oblika `putExtra(String key, T value)`.

# Dodatne informacije

Constant	Meaning
EXTRA_PHONE_NUMBER	A String holding the phone number to call.
EXTRA_EMAIL	A String array holding e-mail addresses that should be delivered to.
EXTRA_TEXT	A String used to supply the literal data to be sent.

Table 2: Dodatne informacije

# Kategorije

- Svojstvo kategorije (`categories`) opisuje vrstu komponente koja obrađuje nameru.
- Postavlja se `addCategory(String category)` metodom.
- Jedna namera može sadržati više kategorija.

# Kategorije

Constant	Meaning
CATEGORY_DEFAULT	Set if the activity should be an option for the default action to perform on a piece of data.
CATEGORY_LAUNCHER	The activity can be the initial activity of a task and is listed in the top-level application launcher.
CATEGORY_GADGET	The activity can be embedded inside of another activity that hosts gadgets.
CATEGORY_PREFERENCE	The target activity is a preference panel.

Table 3: Kategorije

# Oznake

- Oznake (flags) sugerišu sistemu kako da startuje aktivnost (npr. kom zadatku treba da pripada) i kako da je tretira nakon što je startuje (npr. da li treba da se prikaže u spisku nedavnih aktivnosti).
- Postavljaju se metodom `setFlags(int flags)`.
- Jedna namera može sadržati više oznaka (onda se oznake postavljaju disjunkcijom predefinisanih vrednosti).

# Oznake

Constant	Meaning
NEW_TASK	If set, this activity will become the start of a new task on this history stack.
EXCLUDE_FROM_RECENTS	If set, the new activity is not kept in the list of recently launched activities.

Table 4: Oznake

# Namera na čekanju

- Namera na čekanju (pending intent) je namera koja omogućava jednoj komponenti da izvrši operaciju koristeći identitet i prava pristupa druge komponente.

# Filter namera

- Filter namera (intent filter) opisuje mogućnost komponente (namere koje komponenta može da obradi)
- Sadrži polja koja odgovaraju svojstvima namere (akcija, podaci, i kategorija)



# Filter namera

```
<intent-filter ... >  
2   <action android:name="android.intent.action.PICK" />  
    ...  
4 </intent-filter>
```

# Filter namera

```
<intent-filter ... >
2   <data
      android:scheme="http"
4     android:host="example.com"
      android:port="80"
6     android:path="movies"
      android:mimeType="video/mpeg" />
8   ...
  </intent-filter>
10
```

# Filter namera

```
<intent-filter ... >  
2   <category android:name="android.intent.category.DEFAULT" />  
    ...  
4 </intent-filter>
```

## Filter namera

Kada primi implicitnu nameru da startuje aktivnost, sistem pronalazi odgovarajuće aktivnosti tako što poredi nameru i filtere namera na osnovu:

- akcije (akcija specificirana u nameri mora da odgovara jednoj od akcija specificiranih u filteru),
- podataka (URI i MIME tip specificirani u nameri moraju da odgovaraju jednom URI-u i MIME tipu specificiranim u filteru),
- kategorije (svaka kategorija specificirana u nameri mora da odgovara jednoj od kategorija specificiranih u filteru; ne mora da važi obrnuto).

Namera mora proći sva tri testa da bi bila prosleđena komponenti. Jedna komponenta može sadržati više filtera.

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti
- 3 Zadaci i povratni stek
- 4 Namere i filteri namera
- 5 Startovanje aktivnosti**
- 6 Fragmenti
- 7 Prava pristupa

# Startovanje aktivnosti

- Aktivnost se startuje pozivom `startActivity` ili `startActivityForResult` metode.
- Ove metode omogućavaju startovanje navedene aktivnosti (prosleđivanjem eksplicitne namere) ili neke aktivnosti koja je opisana određenim svojstvima (prosleđivanjem implicitne namere).

# Eksplicitna namera

```
Intent intent = new Intent();  
2 intent.setClassName("com.example", "ExampleActivity");  
startActivity(intent);
```

4

# Implicitna namera

```
Intent intent = new Intent();  
2 intent.setAction(Intent.ACTION_SEND);  
  intent.putExtra(Intent.EXTRA_EMAIL, recipients);  
4 intent.putExtra(Intent.EXTRA_TEXT, text);  
  startActivity(intent);  
6
```



# Povratna vrednost

```
protected void onClick() {  
2    Intent intent = new Intent();  
    intent.setAction(Intent.ACTION_PICK);  
4    intent.setData(Contacts.CONTENT_URI);  
    startActivityForResult(intent, PICK_CONTACT_REQUEST);  
6 }  
  
8 @Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
10     if (requestCode == PICK_CONTACT_REQUEST && resultCode == Activity.RESULT_OK  
        ) {  
        ...  
12     }  
    }  
14 }
```

## Zaustavljanje aktivnosti

- Aktivnost se može zaustaviti pozivom `finish()` metode, međutim zaustavljanje aktivnosti treba prepustiti sistemu.

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti
- 3 Zadaci i povratni stek
- 4 Namere i filteri namera
- 5 Startovanje aktivnosti
- 6 Fragmenti**
- 7 Prava pristupa

# Fragmenti

- Fragmenti predstavljaju deo ponašanja ili GUI-a aktivnosti (mogu se posmatrati kao podaktivnosti).
- Jedna aktivnost može da sadrži više fragmenata i jedan fragment može da bude sadržan u više aktivnosti (ali ne ista instanca fragmenta).
- Fragmenti imaju životni ciklus (koji zavisi od životnog ciklusa aktivnosti u kojoj se nalaze) i mogu da obrađuju događaje koje stvara GUI.
- U toku izvršavanja aplikacije se mogu izvršavati transakcije nad fragmentima (mogu se dodavati, uklanjati, zamenjivati, itd.).

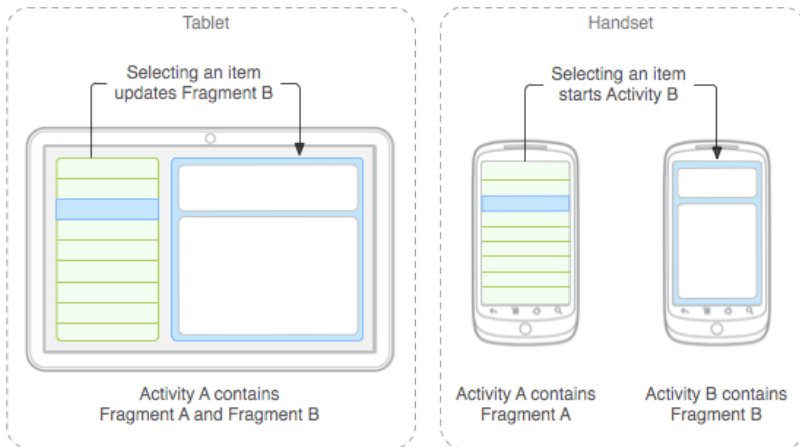


Figure 7: Fragmenti.

# Pravljenje fragmenta

- Napisati klasu koja nasleđuje `Fragment` klasu
- Dodati `fragment` element u XML datoteku koja deklariše korisnički interfejs.

# Pravljenje fragmenta

```
1 package com.example.project;
2 import android.app.Fragment;

4 public static class ExampleFragment extends Fragment {
    @Override
6     public View onCreateView (...) {
        // Inflate the layout for this fragment
8         return inflater.inflate(R.layout.example_fragment, container,
            false);
    }
10 }
```

# Stvaranje fragmenta

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout ... >
   <fragment android:name="com.example.project.ExampleFragment" ... />
4 </LinearLayout>
```



# Životni ciklus fragmenta



# Životni ciklus fragmenta

Životni ciklus fragmenta je sličan životnom ciklusu aktivnosti, ali oni sadrže dodatne metode koji omogućavaju interakciju sa aktivnošću koja ih sadrži:

- `onAttach` (poziva se kada se fragment povezuje sa aktivnošću)
- `onCreateView` (poziva se da bi se iscrtao korisnički interfejs fragmenta)
- `onActivityCreated` (poziva se kada se `onCreate` metoda aktivnosti izvrši)
- `onDestroyView` (poziva se da bi se uništio korisnički interfejs fragmenta)
- `onDetach` (poziva se kada se fragment odvezuje od aktivnosti)

# Transakcije nad fragmentima

```
// Create new fragment and transaction
2 ExampleFragment fragment = new ExampleFragment();
  FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
4
  // Replace whatever is in the fragment_container view with this fragment,
6 // and add the transaction to the back stack
  transaction.replace(R.id.fragment_container, fragment);
8 transaction.addToBackStack(null);

10 // Commit the transaction
  transaction.commit();
12
```

# Transakcije nad fragmentima

U okviru transakcije je moguće izvršiti sledeće operacije:

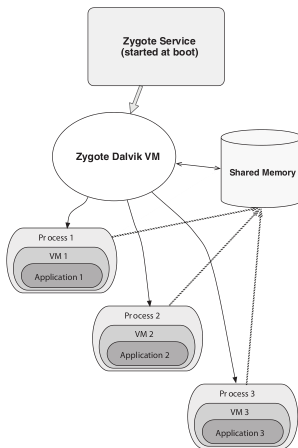
- `add` (dodavanje fragmenta u aktivnost)
- `remove` (uklanjanje fragmenta iz aktivnosti)
- `replace` (zamena jednog fragmenta drugim fragmentom)
- `hide` (skrivanje prikazanog fragmenta)
- `show` (prikazivanje skrivenog fragmenta)
- `detach` (odvajanje fragmenta od GUI)
- `attach` (spajanje fragmenta nakon što je odvojen od GUI)

# Pregled sadržaja

- 1 Uvod
- 2 Aktivnosti
- 3 Zadaci i povratni stek
- 4 Namere i filteri namera
- 5 Startovanje aktivnosti
- 6 Fragmenti
- 7 Prava pristupa**

# Prava pristupa

Operativni sistem izoluje aplikacije (kako aplikacije međusobno tako i operativni sistem od aplikacija).



# Prava pristupa

- Dodatne funkcije bezbednosti su implementirane mehanizmom prava pristupa
- Aplikacija ne može da izvrši ni jednu operaciju koja može da negativno utiče na druge aplikacije, operativni sistem ili korisnike ukoliko joj to nije dozvoljeno
- Implementacija mehanizma prava pristupa razlikuje se do Androida 5.1 (statička prava pristupa) i od Androida 6.0 (dinamička prava pristupa)

# Prava pristupa

Constant	Meaning
CALL_PHONE	Allows an application to initiate a phone call.
SEND_SMS	Allows an application to send SMS messages.
RECORD_AUDIO	Allows an application to record audio.
CAMERA	Required to be able to access the camera device.
VIBRATE	Allows access to the vibrator.

Table 5: Prava pristupa.



# Prava pristupa

Constant	Meaning
ACCESS_COARSE_LOCATION	Allows an app to access approximate location derived from network location sources such as cell towers and Wi-Fi.
ACCESS_FINE_LOCATION	Allows an app to access precise location from location sources such as GPS, cell towers, and Wi-Fi.
INTERNET	Allows applications to open network sockets.
BLUETOOTH	Allows applications to connect to paired bluetooth devices.

Table 6: Prava pristupa.

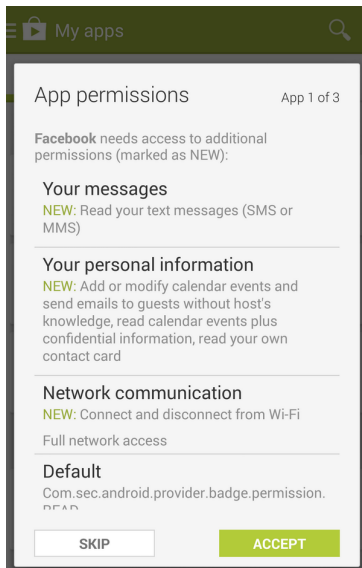
# Statička prava pristupa

- Do Androida 5.1 prava pristupa koja su potrebna za izvršavanje aplikacije statički se deklarišu u `AndroidManifest.xml`.
- Korisnik može da aplikaciji prilikom instalacije dodeli prava pristupa koja traži ili da odustane od instalacije aplikacije
- Svaki pokušaj da aplikacija izvrši nedozvoljene operacije biće sprečen

# Prava pristupa

```
1 <manifest ... >
2   ...
3   <uses-permission android:name="android.permission.INTERNET" />
4   ...
5 </manifest>
6
```

# Statička prava pristupa



# Dinamička prava pristupa

- Od Androida 6.0 aplikacija dinamički traži prava pristupa koja su joj potrebna
- To znači da aplikacija mora da svaki put pre nego što izvrši operaciju koja zahteva pravo pristupa proveriti da li ima to pravo pristupa
- Android može automatski odobriti aplikaciji pravo pristupa ili može zatražiti od korisnika da joj odobri pravo pristupa (u zavisnosti od osetljivosti operacije i resursa)
- Korisnik ima mogućnost da aplikaciji u svakom trenutku oduzme pravo pristupa

# Dinamička prava pristupa

```
// Assume thisActivity is the current activity
2 int permissionCheck = ContextCompat.checkSelfPermission(
    this, Manifest.permission.WRITE_CALENDAR);
4
```

# Permisije i nivoi rizika

- U zavisnosti od toga kojim podacima i akcijama ograničavaju pristup, razlikuju se permisije niskog nivoa rizika i permisije visokog nivoa rizika.
- Permisije niskog nivoa rizika se automatski dozvoljavaju. Dovoljno ih je zatražiti u Manifest fajlu.
- Neke od permisija niskog nivoa rizika su INTERNET, BLUETOOTH, VIBRATE
- Permisije visokog nivoa rizika se moraju dinamički zatražiti tj. korisnik ih mora eksplicitno dozvoliti.
- Primeri permisija visokog nivoa su  
ACCESS\_FINE\_LOCATION,  
ACCESS\_COARSE\_LOCATION, CALL\_PHONE,  
SEND\_SMS, ...

# Dinamička prava pristupa

```
1 private static final int MY_PERMISSIONS_REQUEST_READ_CONTACTS = 0;
2
3 ...
4
5 if (ContextCompat.checkSelfPermission(this, Manifest.permission.
6     READ_CONTACTS)
7     != PackageManager.PERMISSION_GRANTED) {
8
9     ActivityCompat.requestPermissions(
10         this,
11         new String[]{ Manifest.permission.READ_CONTACTS },
12         MY_PERMISSIONS_REQUEST_READ_CONTACTS);
13
14     // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
15     // app-defined int constant. The callback method gets both,
16     the
17     // result of the request paired and this constant.
18 }
```



# Dinamička prava pristupa

```
@Override
2 public void onRequestPermissionsResult(int requestCode, String permissions[],
    int[] grantResults) {

4     switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
8             // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0 && grantResults[0] == PackageManager.
                PERMISSION_GRANTED) {

10                 // permission was granted, yay! Do the
                // contacts-related task you need to do.

12             } else {

14                 // permission denied, boo! Disable the
                // functionality that depends on this permission.

16             }

18             return;
20         }

22         // other 'case' lines to check for other
        // permissions this app might request

24     }

26 }
```