VIII – Meniji kod Android OS

S A D R Ž A J

8.1 – Formiranje menija

- Meniji su sastavni deo korisničkog interfejsa u većini aplikacija.
- Stariji Android uređaji (pre verzije 3.0) su zahtevali hardverski taster za pokretanje menija koji se danas retko nalazi na uređajima
- **Tri vrste** programskih menija je moguće kreirati u aplikacijama:
  1. **Meniji za opcije** (***Menu options***, ***Action bar***) – standardni meni koji se prikazuje u aktivnosti i služi za globalne opcije aplikacije
  2. **Kontekstualni meni** (***Contextual Action Mode***) – lista stavki menija koja se otvara kada korisnik duže drži pritisnuti taster
  3. **Podmeniji** ili padajući meniji (***owerflow***) – lista stavki koja se prikazuje kada korisnik izabere jednu od stavki iz menija
- Meniji su slični drugim komponentama Android korisničkog interfejsa
- Za definisanje svih tipova menija Android koristi standardni XML kod
- Meni se ne definiše u kodu aktivnosti već se sa svim stavkama definiše u vidu XML datoteke u folderu ***res/menu/*** a zatim uključuje u aktivnost
- Korišćenjem menija kao resursa omogućeno je razdvajanje menija od koda kao i kreiranje alternativnih menija za različite Android verzije,

8.1 – Formiranje menija

- Za prezentovanje korisničkih akcija i drugih opcija u aktivnostima potrebno je koristiti **android.view.Menu** klasu.
- Da bi definisali meni potrebno je u okviru projekta kreirati XML dokument u ***res/menu*** direktorijumu.
- Primer menija moj_meni.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

8.1 – Struktura menija

- Obavezni delovi menija su:

1. **<menu>** - ovaj element je koreni element sadrži jedan ili više <item> i <group> elementa.

2. **<item>** - kreira stavku menija koja može sadržati ugnježden <menu> element i na taj način se formira struktura podmenija.

3. **<group>**-opcioni, nevidljivi kontejner za<item>element. Omogućava kategorizaciju stavki menija u cilju deljenja zajedničkih osobina

- **<item>** element ima podršku za nekoliko atributa koji se koriste za definisanje ponašanja i načina pojavljivanja stavki menija.
- Stavke menija iz prethodnog primera sadrže i atribut **android:showAsAction** koji obezbeđuje da se stavka menija pojavi u **action bar**-u samo ako ima slobodnog mesta.

8.1 – Struktura menija

- Ukoliko nema mesta stavka menija se pojavljuje u ***overflow***(podmeni)
- U svaki meni moguće je dodati stavku koja predstavlja novi meni
- Da bi se meni koristio u aktivnosti potrebno je ubaciti meni kao resurs korišćenjem **MenuInflater.inflate().**

```
http://schemas.android.com/apk/res/android"><?xml version="1.0"
encoding="utf-8"?>
<menu xmlns:android="
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
    </item>
</menu>
        </menu>
```

8.1 – Primena pomoćnih metoda

- Pre nego što programer kreira meni sa opcijama, ili kontekstni meni, prinuđen je da kreira dve pomoćne metode.
- Zadatak ovih metoda je prikazivanje liste stavki i upravljanje događajima nakon izbora stavke menija.
- Ove metode su deo klase aktivnosti projekta i u konkretnom slučaju biće nazvane ***CreateMenu()*** i ***MenuChoice()***.
- Kreiranje pomoćnih metoda predstavlja polazni osnov za kreiranje aplikacije za rad sa menijima u Android OS.
- ***CreateMenu()*** metoda preuzima argument tipa ***Menu*** i dodaje niz stavki menija.
- Za dodavanje stavke u meni, potrebno je kreirati objekat klase ***MenuItem*** i izvršiti metodu ***add()*** objekta ***Menu***

8.1 – Primena pomoćnih metoda

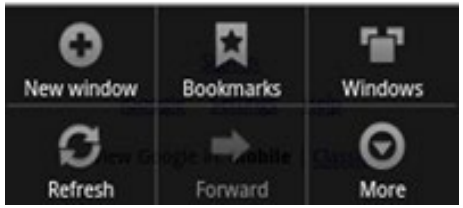
- Metoda ***add()*** poseduje četiri argumenta:
  1. ***groupID*** – identifikator grupe kojoj stavka menija pripada;
  2. ***itemId*** – jedinstveni identifikator stavke grupe;
  3. ***order*** – redosled u kome stavka treba da bude prikazana;
  4. ***title*** – tekst koji se prikazuje za određenu stavku menija.
- Metoda ***setAlphabeticShortcut()*** je iskorišćena za podešavanje prečice na tastaturi za izbor konkretne stavke manija.
- Metodom ***setIcon()*** definiše se slika koja se prikazuje kao određena stavka menija.
- Metoda ***MenuChoice()*** preuzima ***MenuItem*** argument, proverava njegov ***ID*** i ispituje koja je stavka liste izabrana.
- U primeru je iskorišćena ***Toast*** poruka za prikazivanje rezultata izbora stavke na ekranu mobilnog uređaja

8.1 – Primena pomoćnih metoda

```
private void CreateMenu(Menu menu){
    menu.setQwertyMode(true);
    MenuItem mnu1 = menu.add(0, 0, 0, "Stavka 1");{

        mnu1.setAlphabeticShortcut('a');
        mnu1.setIcon(R.drawable.ic_launcher);
    }
    MenuItem mnu2 = menu.add(0, 1, 1, "Stavka 2");{
        mnu2.setAlphabeticShortcut('b');
        mnu2.setIcon(R.drawable.ic_launcher);
    }
    MenuItem mnu3 = menu.add(0, 2, 2, "Stavka 3");{
        mnu3.setAlphabeticShortcut('c');
        mnu3.setIcon(R.drawable.ic_launcher);
    }
    MenuItem mnu4 = menu.add(0, 3, 3, "Stavka 4");{
        mnu4.setAlphabeticShortcut('d');
    }
    menu.add(0, 4, 4, "Stavka 5");
    menu.add(0, 5, 5, "Stavka 6");
    menu.add(0, 6, 6, "Stavka 7");
}
```

```
private boolean MenuChoice(MenuItem item){

    switch (item.getItemId()) {
    case 0:
        Toast.makeText(this, "Kliknuli ste na stavku  1",
            Toast.LENGTH_LONG).show();
        return true;
    case 1:
        Toast.makeText(this, "Kliknuli ste na stavku  2",
            Toast.LENGTH_LONG).show();
        return true;
    case 2:
        Toast.makeText(this, "Kliknuli ste na stavku  3",
            Toast.LENGTH_LONG).show();
        return true;
    case 3:
        Toast.makeText(this, "Kliknuli ste na stavku  4",
            Toast.LENGTH_LONG).show();
        return true;
    case 4:
        Toast.makeText(this, "Kliknuli ste na stavku  5",
            Toast.LENGTH_LONG).show();
        return true;
    case 5:
        Toast.makeText(this, "Kliknuli ste na stavku  6",
            Toast.LENGTH_LONG).show();
        return true;
    case 6:
        Toast.makeText(this, "Kliknuli ste na stavku  7",
            Toast.LENGTH_LONG).show();
        return true;
    }
    return false;
}
```

8.2 – Meniji sa opcijama



- Ovo je osnovna kolekcija stavki menija za aktivnost
- Korisnici vide ovaj meni klikom na dugme ***Menu***.
- Ukoliko se aplikacija razvija za verziju Android 2.3 ili nižu sadržaj menija opcija se pojavljuje na dnu ekrana nakon klika na dugme ***Menu***.
- Ovaj meni sadrži do 6 stavki menija i ako postoji više stavki Android ostale stavke smešta u ***overflow*** meni koji se otvara tasterom ***More***.
- Da bi se naveo meni sa opcijama za neku aktivnost potrebno je koristiti metodu ***onCreateOptionsMenu()***
- U ovoj metodi je moguće ubaciti naš meni u vidu resursa (definisan u XML fajlu) u ***Menu***

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.moj_meni, menu);
    return true;
}
```

8.2 – Meniji sa opcijama

- Kada korisnik selektuje stavku menija iz ponuđenih opcija, sistem poziva ***onOptionsItemSelected()*** metod unutar naše aktivnosti.
- Ovaj metod prosleđuje selektovanu stavku menija.
- Identifikacija stavke je moguća pozivom ***getItemId()***, koja vraća jedinstveni ID stavke menija koji je definisan u ***android:id*** atributu
- Ispitivanjem ovog atributa moguće je izvršavati različite akcije:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

- Ukoliko je stavka menija uspešno obrađena odgovor je **TRUE** u suprotnom je potrebno pozvati ***onOptionsItemSelected()***.

8.2 – Meniji sa opcijama

- Nakon što sistem pozove metodu ***onCreateOptionsMenu()***, zadržava instancu napunjenog menija i više ne poziva ***onCreateOptionsMenu()*** metodu osim ukoliko meni iz nekog razloga postane nevalidan.
- Zato se metoda ***onCreateOptionsMenu()*** koristi samo za kreiranje inicijalnog menija opcija a ne za izmene tokom životnog ciklusa aktivnosti.
- Za izmene menija opcija koje se baziraju na događajima koji se dešavaju tokom životnog ciklusa aktivnosti koristi se metod ***onPrepareOptionsMenu()***.
- Ovaj metod prosleđuje ***Menu*** objekat u trenutnom stanju pa je moguće vršiti izmene kao što su dodavanje, brisanje ili onemogućavanje stavki.
- U Android 2.3.x i nižim verzijama operativnog sistem, metoda ***onPrepareOptionsMenu()*** se poziva svaki put kada korisnik otvori meni opcija tj. pritisne ***Menu*** dugme.

8.2 – Meniji sa opcijama

- Da bi prikazali meni sa opcijama u nekoj aktivnosti, treba primeniti metode ***onCreateOptionsMenu()*** i ***onOptionsItemSelected()***
- Prva metoda se izvršava kada korisnik klikne na taster **MENU**.
- U konkretnom primeru, pomoćna metoda ***onCreateMenu()*** izvršava se za prikazivanje menija sa opcijama.
- Izborom stavke iz menija izvršava se metoda ***onOptionsItemSelected()*** koja implementira metodu ***menuChoice()*** za prikazivanje izabrane stavke i sprovođenje predviđene akcije.

- Za kreiranje aplikacija sa menijima, neophodno je i uključivanje izvesnog broja paketa sa odgovarajućim klasama koje su prikazane na slici desno.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
```

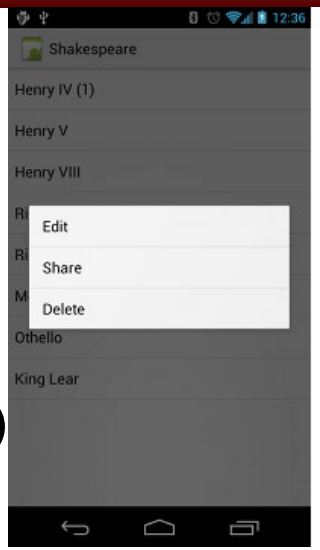
8.2 – Meniji sa opcijama

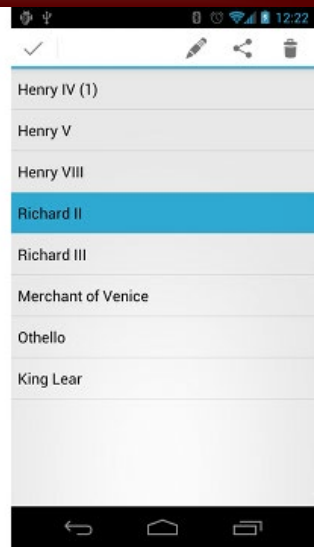
- Program.kod metoda za kreiranje i upravljanje menijem sa opcijama:

```
package net.learn2develop.Menus;
import android.app.Activity;
public class MenusActivity extends Activity {
    /** Poziva se kada se aktivnost kreira. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button btn = (Button) findViewById(R.id.button1);
        btn.setOnCreateContextMenuListener(this);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        CreateMenu(menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item){
        return MenuChoice(item);
    }
    private void CreateMenu(Menu menu){
       // kod metode je priložen
    }
    private boolean MenuChoice(MenuItem item){
        // kod metode je priložen
        }
    }
```

8.3 – Kontekstualni meni

- Kontekstualni meni je kretajući meni koji se pojavljuje kada korisnik izvršava produženi klik na elementu.
- Omogućava akciju nad selektovanom stavkom ili okvirom u kojem se kontekst pojavljuje.
- Prikaz kretajućeg kontekstualnog menija (levo) i kontekstualnog ***action bar*** (desno)





- Kontekstualni meni je moguće obezbediti za bilo koji pogled ali se najčešće koristi za elemente u **ListView** i **GridView**-u.
- Postoje dva načina za obezbeđivanje kontekstualnih akcija:

1. Kretajući kontekstualni meni - meni se pojavljuje kao pokretna lista stavki menija nad kojim korisnik može da izvrši kontekstualnu akciju
2. Kontekstualni ***action mode*** - kontekstualni ***action bar*** se pojavljuje na vrhu ekrana sa elementima koji utiču na selektovani element. Ovde korisnici mogu da izvršavaju akcije nad više elemenata istovremeno (ukoliko to aplikacija dozvoljava). Ova opcija je omogućena samo na verzijama Android 3.x i višim.

8.3-Kreiranje kretajućeg kontekstualnog menija

1. Registruje se pogled (*View*) sa kojim će kontekstualni meni biti povezan pozivom ***registerForContextMenu()*** i prosleđivanjem odgovarajućeg pogleda. Ukoliko aktivnost koristi ***ListView*** ili ***GridView*** a potrebno je da svaka stavka ima isti kontekstualni meni dovoljno je proslediti ***ListView*** ili ***GridView*** ka ***registerForContextMenu()***
2. Potrebno je implementirati metodu ***onCreateContextMenu()*** u konkretnoj aktivnosti. Kada registrovani pogled dobije događaj dugog klika, sistem poziva ***onCreateContextMenu()*** metodu. Tamo su definisane stavke menija obično ubacivanjem u resurse menija.

Na primer:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

8.3-Kreiranje kretajućeg kontekstualnog menija

- ✓ ***MenuInflater*** omogućava da se kontekstualni meni ubaci iz resursa menija.
- ✓ Parametri ***callback*** metode uključuju pogled koji je korisnik selektovao a objekat ***ContextMenu.ContextMenuInfo*** obezbeđuje dodatne informacije o selektovanom elementu.
- ✓ Ukoliko aktivnost ima nekoliko pogleda koje obezbeđuju različite kontekstualne menije ovi parametri se mogu koristiti kako bi se donela odluka koji kontekstualni meni je potrebno ubaciti.

3. Implementirati ***onContextItemSelected()***.

8.3-Kreiranje kretajućeg kontekstualnog menija

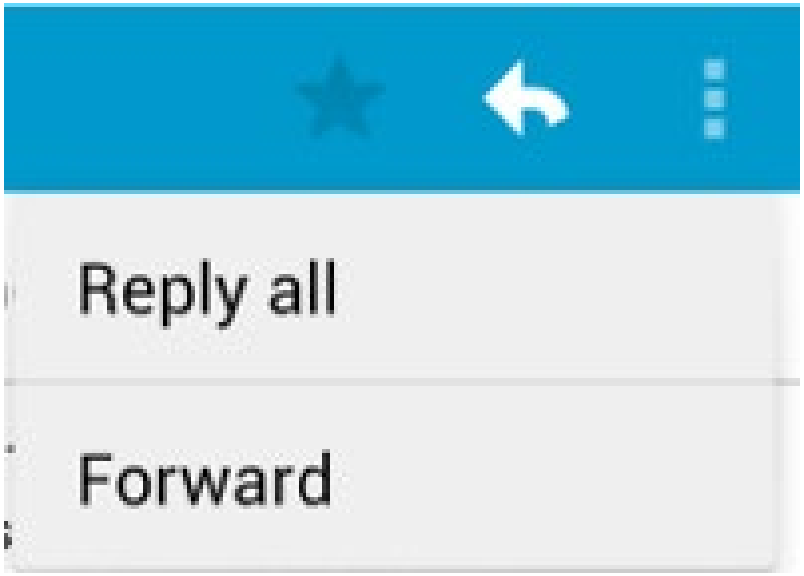
- Kada korisnik selektuje stavku menija sistem poziva ovaj metod kako bi se uzvršile određene akcije. Na primer:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

- Metod ***getItemId()*** ispituje ID kod selektovane stavke menija koji se dodeljuje svakoj stavci u XML dokumentu, tačnije u android:id atributu
- Ako se stavka menija neuspešno obradi potrebno ju je proslediti superklasi

8.4 – Iskačući meniji

- **Popup meni** prikazuje listu stavki u vertikalnoj listi koja je povezana sa pogledom koji poziva meni.
- Akcije u **iskačućem meniju** ne bi trebalo da utiču na odgovarajući sadržaj jer su za to zadužene kontekstualne akcije.
- **Iskačući meniji** se koriste za dodatne akcije koje se odnose na delove sadržaja unutar akrivnosti.
- **Iskačući meni** je dostupan od verzije Android 3.x i više.



8.5 – Kreiranje meni grupa

- **Meni grupa** je skup stavki menija koje imaju sledeće osobine:
  - ✓ Prikazivanje ili sakrivanje svih stavki sa ***setGroupVisible()***
  - ✓ Omogućavanje/onemogućavanje svih stavki sa ***setGroupEnabled()***
  - ✓ Definisati da li moguće čekirati stavke pomoću ***setGroupCheckable()***
- Grupu je moguće kreirati gnježđenjem **<item>** elemenata unutar **<group>** elementa u resursima menija ili definisanjem grupnog ID elementa korišćenjem ***add()*** metode.

```
<?xml version="1.0" encoding="utf-8"?>
   <menu xmlns:android="http://schemas.android.com/apk/res/android">
      <item android:id="@+id/menu_save"
           android:icon="@drawable/menu_save"
           android:title="@string/menu_save" />
      <!-- menu group -->
      <group android:id="@+id/group_delete">
         <item android:id="@+id/menu_archive"
              android:title="@string/menu_archive" />
         <item android:id="@+id/menu_delete"
              android:title="@string/menu_delete" />
      </group>
   </menu>
```

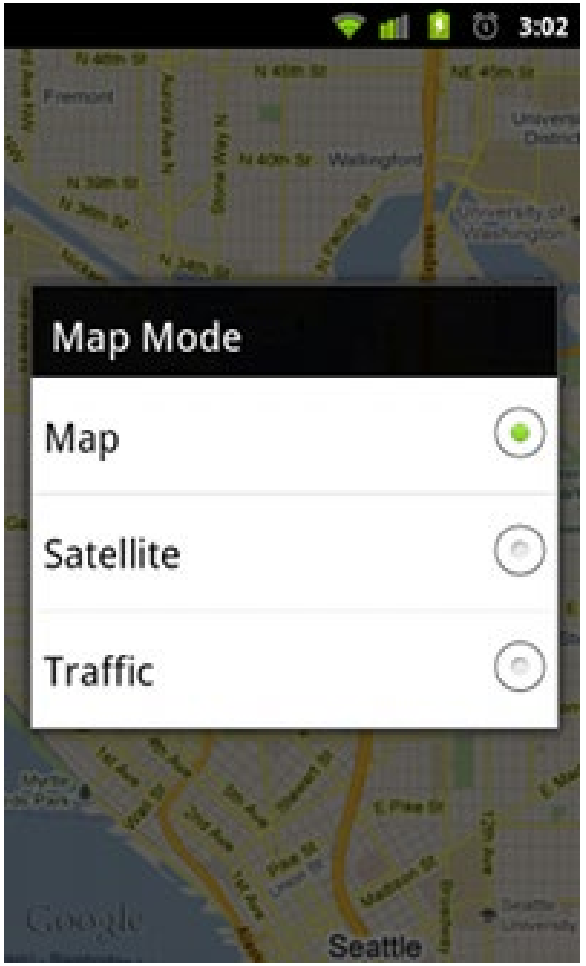
8.5 – Kreiranje meni grupa

- Stavke koje se u grupi pojavljuju na istom nivou su braća i sestre.
- Moguće je promeniti osobine dve stavke iz grupe referenciranjem grupnog ID-a i korišćenjem prethodno opisanih metoda ali sistem neće nikada razdvojiti grupisane stavke.
- Na primer ukoliko se deklariše ***android:showAsAction="ifRoom"*** za svaku stavku, one će se ili obe pojaviti u **action bar**-u ili u **action overflow**-u.
- Ovaj meni je veoma koristan kada je u pitanju interfejs pomoću kojeg se neka opcija uključuje ili isključuje ili za grupu međusobno isključivih opcija.
- Moguće je definisati da neka stavka menija ima čekirajuću osobinu korišćenjem ***android:checkable atributa*** u **<item>** elementu, ili da cela grupa ima tu osobinu korišćenjem ***android:checkableBehavio***r atributa u **<group>** elementu.
- Na sledećem primeru je prikazano da su sve stavke iz grupe u formi radio dugmeta koje se može čekirati.

8.5 - Kreiranje meni grupa

- Prikaz podmenija koji sadrži stavke koje se čekiraju

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item android:id="@+id/red"
            android:title="@string/red" />
        <item android:id="@+id/blue"
            android:title="@string/blue" />

</menu>
```



8.5 – Kreiranje meni grupa

- ✓ **android:checkableBehavior** atribut prihvata samo sledeće osobine:
  - **Single** - samo jedna stavka iz grupe može biti selektovana (radio dugme)
  - **all** - sve stavke mogu biti čekirane (checkboxes)
  - **none** - nijednu stavku nije moguće čekirati
- ➢ Moguće je primeniti podrazumevano stanje na stavku korišćenjem **android:checked** atributa u **<item>** elementu i promene u kodu korišćenjem ***setChecked()*** metode.
- ➢ Kada je stavka koju je moguće čekirati selektovana sistem poziva odgovarajuću (za tu stavku) ***callback*** metodu (kao što je ***onOptionsItemSelected()***).
- ➢ Obavezno je da se izvrši postavka stanja ***checkbox***-a – iz razlga što ***checkbox*** ili ***radio dugme*** ne menjaju automatski svoje stanje.
- ➢ Potrebno je proveriti stanje stavke (isto kao pre selektovanja) pomoću ***isChecked()*** metode i onda postaviti stanje pomoću ***setChecked()*** metode.

8.5 – Kreiranje meni grupa

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.vibrate:
        case R.id.dont_vibrate:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

- Ako se stanje ne postavi na ovaj način onda se vidljivo stanje stavke neće promeniti kada ga korisnik selektuje.
- Sa druge stanje kada se stanje postavi na prethodno opisan način aktivnost čuva čekirano stanje i prilikom ponovnog otvaranja menija novočekirano stanje je vidljivo korisniku.
- Stavke menija koje se čekiraju se koriste samo po sesiji i neće biti sačuvane kada se sesija uništi.

8.6-Događaji u okviru korisničkog interfejsa

- Obrada događaja u okviru korisničkog interfejsa se zasniva na hvatanju događaja u okviru specifičnog pogleda (***View***) sa kojim korisnik trenutno interaguje.
- U okviru svake od podklasa klase ***View*** nalazi se nekoliko javnih metoda koje mogu biti korisne za UI događaje.
- Android ***framework*** poziva ove metode kada god se desi neka akcija nad definisanom klasom, odnosno objektom.
- Na primer, kada korisnik dodirne određeni ***View*** (na primer **Button**), poziva se ***onTouchEvent()*** metoda nad tim objektom.
- Da bi se ovaj događaj obradio, neophodno je proširiti odgovarajuću klasu i realizovati ***override*** metode.
- Međutim, proširivanje svakog pojedinačnog tipa ***View***-a da bi se obradio događaj, nije najbolje rešenje.
- S tim u vezi, klasa ***View*** sadrži kolekciju ugneždenih interfejsa sa ***callbacks*** koji se jednostavnije definišu.
- Ovi interfejsi se zovu ***event listeners***.

8.6 – Event listener

- ***Event listener*** je interfejs u okviru ***View*** klase koji sadrži jednu **callback** metodu.
- Ove metode se pozivaju od strane Androida kada korisnik preko UI interaguje sa ***View*** objektom za koji je ***listener*** registrovan.
- Event listener interfejsi obuhvataju sledeće **callback** metode:

1. **onClick()** - iz ***View.OnClickListener***. Poziva se kada korisnik: dodirne elemenat, fokusira se na elemenat, pritisne enter ili pritisne pokazivač.
2. **onLongClick() -** iz ***View.OnLongClickListener***. Obuhvata događaje koji se odnose na zadržavanje ulaznog dodira/fokusa i sl. nad određenim elementom.
3. **onFocusChange()** - iz ***View.OnFocusChangeListener***. Odnosi se na situaciju kada korisnik postavlja ili uklanja fokus sa nekog elementa.
4. **onKey()** - iz ***View.OnKeyListener***. Slučaj kada je korinsik postavio fokus na neki element i pritiska ili pušta dugme na telefonu.
5. **onTouch()** - iz ***View.OnTouchListener***. Bilo koja aktivnost na ekranu kao dodir, obuhvatajući pritisak, puštanje ili pokret preko ekrana.
6. **onCreateContextMenu()** - iz ***View.OnCreateContextMenuListener***. Poziva se kada se kreira kontekstualni meni.

8.6 – Event listener

- Sve ove metode su jedini članovi svojih interfejsa.
- Da bi se ove metode definisale potrebno je implementirati ugneždene interfejse u okviru Aktivnosti ili ih definisati kao anonimne klase.
- Nakon toga, šalje se instanca implementiranog interfejsa ka odgovarajućem ***View. Set…Listener*** (na primer poziva se ***setOnClickListener()*** a njemu se prosleđuje ***OnClickListener***).
- Prikazano je kako se registruje **on-click listener** za **Button view**:

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};
protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

8.6 – Event listener

- **On-click listener** može se implementirati i u okviru Aktivnosti:

```
public class ExampleActivity extends Activity implements
OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewByld(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

- U primeru **onclick()** nema povratnu vrednost, ali neki drugi **listener events** moraju vratiti **boolean** vrednost, što zavisi od konkretnog događaja.

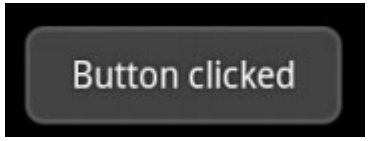
8.6 - Event listener

- Umesto korišćenja ***OnClickListener***-a za dugme u okviru aktivnosti, moguće je dodeliti odgovarajući metod dugmetu u okviru XML fajla

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/tekstd"
    android:onClick="mojametoda" />
```

- Kada korisnik klikne na dugme Android će automatski pozvati metodu ***mojametoda***. **View** koji se prosleđuje predstavlja referencu ka vidžetu koji je kliknut.

```
public mojamteoda (View v)
{
Toast.makeText(Nazivaktivnosti.this, "Button clicked",
Toast.LENGTH_LONG).show();
}
```



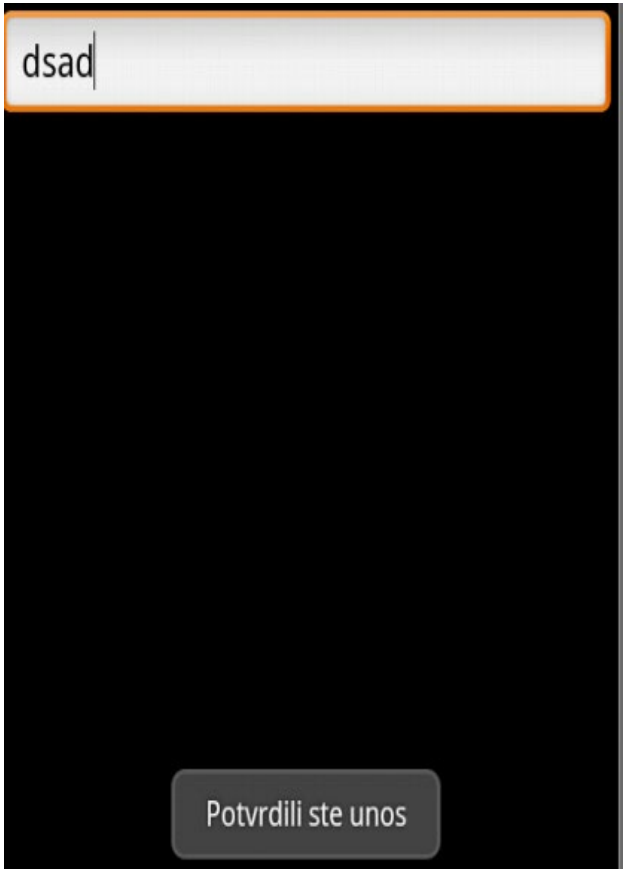
- U ovom primeru *mojametoda* ispisuje kratku poruku u vidu **Toast view** notifikacije da je dugme kliknuto. **Toast view** se prikazuje kao kretajući natpis iznad aplikacije i nikad ne dobija fokus. Ideja je da se korisnik ne uznemirava dok nešto radi, ali da mu se ipak prikaže neka informacija

8.6-Događaji u okviru korisničkog interfejsa

- Sledeći primer ilustruje obradu događaja nad poljem za unos teksta:

```
<EditText
        android:id="@+id/txt"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    />
```

- Neophodno je implementirati *OnKeyListener*.
- U okviru metode *onKey* koja prihvata 3 parametra, u primeru se prate pritisci na dugmad i kada se pritisne dugme ENTER ispisuje se poruka na ekranu.
- Da bi se iščitao tekst koji je korsnik uneo u polje, potrebno je pozvati metodu *getText()* (*edittext.getText()*)



8.6-Događaji u okviru korisničkog interfejsa

```
public class Nazivaktivnosti extends Activity implements OnKeyListener
....
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.nov);
        EditText edittext = (EditText) findViewByld(R.id.txt);
        edittext.setOnKeyListener(this);
    }
....
public boolean onKey(View v, int keyCode, KeyEvent event) {
                    if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
                   (keyCode == KeyEvent.KEYCODE_ENTER)) {
                  // Perform action on key press
                  Toast.makeText(Nazivaktivnosti.this, "Potvrdili ste unos",
Toast.LENGTH_SHORT).show();
                   return true;
                 }
                 return false;
        }
```

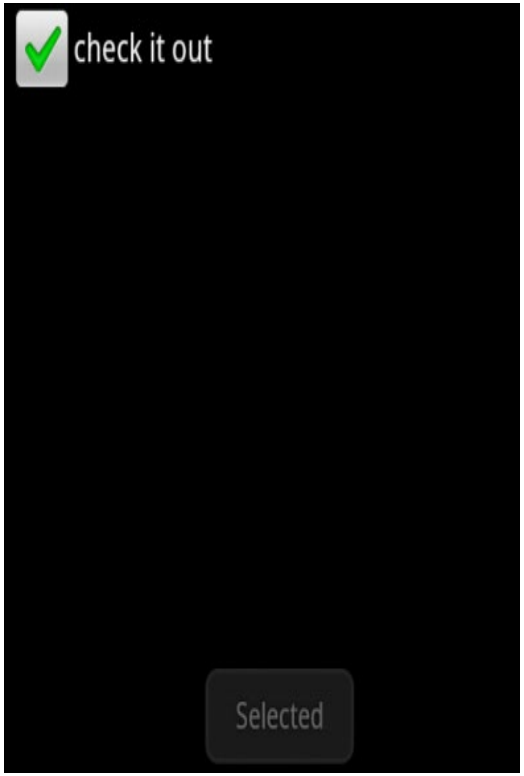
8.6-Događaji u okviru korisničkog interfejsa

- Sledeći primer se odnosi na obradu selektovanja ***checkbox*** elementa:

```
<CheckBox android:id="@+id/checkbox"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="check it out"
      android:onClick="onCheckboxClicked"/>
```

- U okviru Aktivnosti treba implementirati odgovarajuće metode:

```
public void onCheckboxClicked(View v) {
            // Perform action on clicks, depending on
               whether it's now checked
            if (((CheckBox) v).isChecked()) {
                Toast.makeText(SubotaActivity.this,
"Selected", Toast.LENGTH_SHORT).show();
            } else {
                Toast.makeText(SubotaActivity.this, "Not
selected", Toast.LENGTH_SHORT).show();
            }
        }
```



8.6-Događaji u okviru korisničkog interfejsa

- Sledeći primer prikazuje obradu selektovanja *radiobutton* elementa:

```
<RadioGroup
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical">
   <RadioButton android:id="@+id/radio_red"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Red"
      android:onClick="onRadioButtonClicked"/>
   <RadioButton android:id="@+id/radio_blue"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Blue"
      android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

8.6-Događaji u okviru korisničkog interfejsa

- U okviru Aktivnosti potrebno je ubaciti implementaciju odgovarajuće metode:

```
public void onRadioButtonClicked(View v) {
    // Perform action on clicks
    RadioButton rb = (RadioButton) v;
    Toast.makeText(HelloFormStuff.this, rb.getText(),
Toast.LENGTH_SHORT).show();
}
```

8.6-Događaji u okviru korisničkog interfejsa

- Android okruženje omogućava praćenje fokusa nad elementima, odnosno **View** objektima.
- Fokus se menja tako što **View** nestaje, ili se sakriva u pozadini ili se pojavljuje novi **View** objekat.
- Svaki **View** objekat može podesiti svoju dostupnost za fokus pomoću metode ***setFocusable()***, odnosno ispitati fokus pomoću metode ***isFocusable()***.
- Pomeranje fokusa se zasniva na **algoritmu najbližeg suseda** (objekta **View** na ekranu) u datom pravcu.
- U slučaju kada je potrebno izmeniti default pomeranje fokusa, mogu se iskoristit sledeći XML atributi u okviru **layout** fajla:

1. ***nextFocusDown***,
2. ***nextFocusLeft***,
3. ***nextFocusRight***,
4. ***nextFocusUp***.

8.6-Događaji u okviru korisničkog interfejsa

```
<LinearLayout
  android:orientation="vertical"
  ... >
 <Button android:id="@+id/top"
       android:nextFocusUp="@+id/bottom"
       ... />
 <Button android:id="@+id/bottom"
       android:nextFocusDown="@+id/top"
       ... />
</LinearLayout>
```

- Bez dodatnih atributa, navigacija na gore od prvog dugmeta i navigacija na dole od drugog dugmeta ne bi vodila nikud.
- Kada se ubace dodatni atributi, omogućena je navigacija u oba smera.
- Ukoliko je potrebno definisati neki **View** kao ***focusable***, potrebno je dodati ***android:focusable*** XML attribut u **View** u okviru ***layout*** deklaracije i podesiti ga na **true**.
- Da bi se postavio fokus na određeni **View** poziva se funkcija ***requestFocus()***.

8.6-Događaji u okviru korisničkog interfejsa

- Postoje situacije kada je potrebno obavestiti korisnika da se desio neki događaj u okviru aplikacije, iako aplikacija može da bude u pozadini u tom trenutku.

1. Kada se u pozadini preuzima neki fajl sa Interneta u momentu završetka ***downloada***, trebalo bi porukom obavestiti korisnika da je proces završen
2. Ako aplikacija radi u pozadini, ali je u nekom momentu potrebno da korisnik definiše dalje korake, javlja se notifikacija o tome
3. Ukoliko aplikacija nešto radi dok korisnik čeka, potrebno je obavestiti korisnika o napretku ili vremenu čekanja
4. E-mail aplikacija može da obavesti korisnika da je stigla nova poruka;
5. Aplikacije za vremensku prognozu, mogu da obaveste korisnika o promenama vremena;
6. Aplikacije za berzu, mogu da obaveste korisnika ako je nastala neka promena cena nekih akcija;

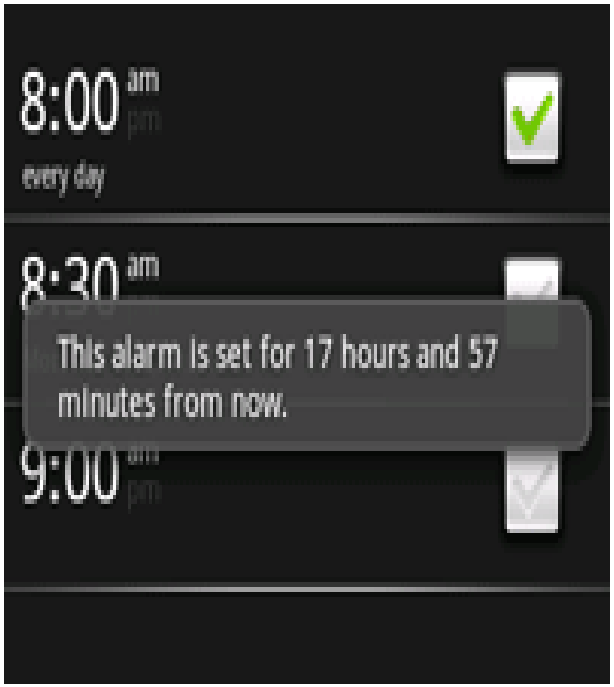
8.6-Događaji u okviru korisničkog interfejsa

- Notifikacije se mogu realizovati na tri načina:
  1. ***Toast notifikacija*** – kratka poruka koja dolazi iz pozadine
  2. **Status bar notifikacija** – perzistentni reminder koji dolazi iz pozadine i zahteva odgovor korisnika
  3. **Dialog notifikacija** – notifikacije vezane za aktivnosti
- Korisnici znaju da cene ova obaveštenja, jer dolaze do potrebnih informacija bez da pokrenu aplikaciju.
- Međutim, postoji granica između dobrog obaveštajnog sistema i onog koji preteruje sa notifikacijama, ne treba korisnika stalno uznemiravati.
- Svako obaveštenje treba da bude urađeno tako da na korisnika deluje samo pozitivno.
- Android platforma nudi nekoliko načina kako se mogu obaveštavati korisnici.
- Obaveštenje se često prikazuje u statusnoj liniji u gornjem delu ekrana.
- Obaveštenje može da podrazumeva: tekstualne informacije, grafički ili zvučni prikaz, vibracija uređaja ili kontrola nad svetlosnim indikatorom.

8.6 - Toast notifikacija

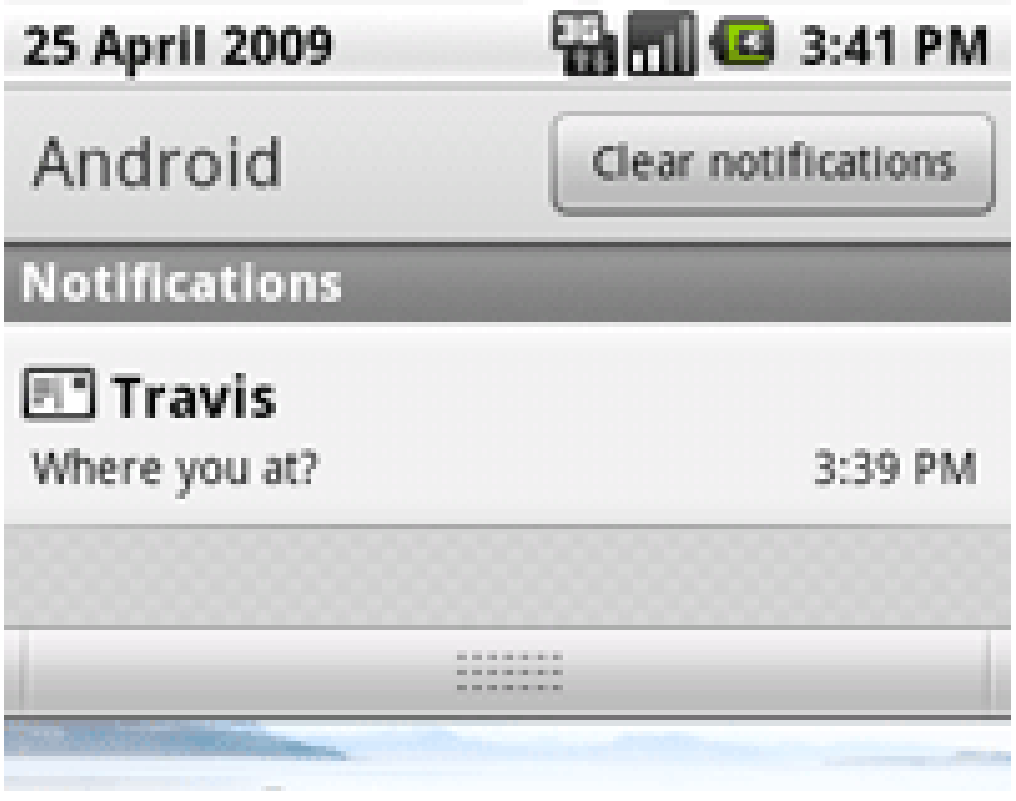
- Ova notifikacije predstavlja poruku na površini prozora.
- Na kratko popunjava jedan deo ekrana, a korisnikova aktivnost ostaje aktivna i vidljiva sve vreme.
- Notifikacija automatski isčezava sa ekrana i ne prihvata događaje.
- Može se kreirati i prikazati u okviru aktivnosti i servisa.

```
Context context =
getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context,
text, duration);

toast.setGravity(Gravity.TOP|Gravity.L
EFT, 0, 0);


toast.show();
```



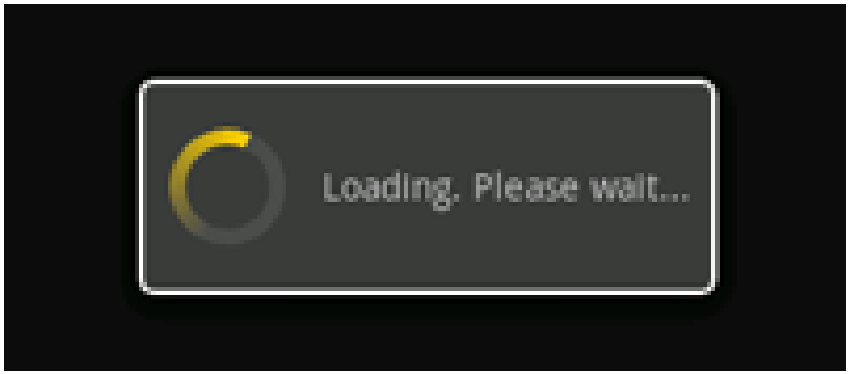
8.6 – Status bar notifikacija

- Status bar notifikacija dodaje ikonu na sistemski status bar i poruku u "Notification prozoru".
- Kada korisnik selektuje poruku iz notifikacije, Android poziva intent kojim se aktivira aktivnost kojoj notifikacija pripada.
- Ovaj tip notifikacije može da upozori korisnika i putem zvuka, vibracije ili svetla na mobilnom uređaju.
- Najčešće se koristi kada akitvnost ili servis nešto radi u pozadini i ima potrebu da komunicira sa korisnikom u nekom momentu.



8.6 – Dijalog notifikacija

- ***Dijalog notifikacija*** se najčešće javlja u okviru trenutno aktivne aktivnosti.
- Dijalog od aktivnosti preuzima interakciju sa korisnikom.



8.6 – Dijalozi

- Android omogućava prikazivanje interaktivnih dijaloga koji u sebi mogu da sadrže bilo koje elemente korisničkog interfejsa.
- Neke vrste dijaloga dolaze sa predefinisanim izgledom i namenom, a moguće je definisati i spostveni layout za dijalog ili čak napraviti sopstvenu klasu na osnovu klase ***Dialog*** koja predstavlja zajedničku nadklasu za sve vrste dijaloga.
- Bez obzira na klasu dijaloga, potrebno je obratiti pažnju na način njihovog kreiranja.
- Najjednostavniji način je da se dijalog kreira u samom kodu, tamo gde je potrebno i zatim prikaže pozivanjem metode ***show()*** nad objektom dijaloga.
- Ovaj pristup je problematičan, jer se o takvom dijalogu moramo sami starati u slučaju promene konfiguracije telefona (tipično: u slučaju rotacije iz Portrait u Landscape i obrnuto).

```
Dialog mojDijalog = new MojaKlasaDijaloga()

mojDijalog.show();
```

8.6 – Dijalozi

- Bolja varijanta je da staranje o dijalogu prepustimo našoj aktivnosti u kojoj će dijalog biti prikazan.
- Da bi to postigli, prvo moramo da implementiramo metodu ***onCreateDialog(int)*** ili ***onCreateDialog(int, bundle)***.
- **Int** parametar u oba slučaja predstavlja neki naš ID dijaloga kojeg treba prikazati, a opcioni **bundle** parametar predstavlja paket podataka koji se mogu proslediti pri kreiranju dijaloga.

```
@Override
protected Dialog onCreateDialog(int id, Bundle args) {
        Dialog rezultat;
        switch(id) {
        case 1:
                rezultat = new MojaKlasaDijaloga(); break;
        case 2:
                rezultat = new NekiDrugiDijalog(); break;
        }
        return rezultat;
}
```

8.6 – Dijalozi

- U ovoj metodi proveravamo koji dijalog se traži (npr. dijalog sa ID 1 je upozorenje korisniku da se greška desila, a dijalog sa ID 2 mu nudi izbor OK/CANCEL), pa zavisno od toga kreiramo odgovarajući dijalog i vraćamo ga kao rezultat.
- Ova metoda će automatski biti pozvana kada korisnik prvi put pozove metodu **showDialog(int)** ili ***showDialog(int, bundle)***.
- Pri sledećim pozivanjima će biti upotrebljen postojeći, već kreirani dijalog.
- Ako dođe do promene konfiguracije, sama aktivnost će se postarati za ponovno iscrtavanje dijaloga.

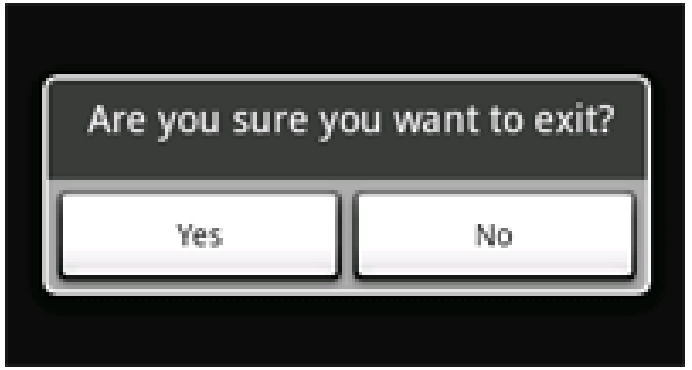
showDialog(1, null); // hoćemo prikaz greške (ID 1), ne šaljemo Bundle

- Ako hoćemo da napravimo neke izmene na dijalogu pri svakom prikazivanju (svaki put želimo drugačiji tekst greške), onda možemo pored ***onCreateDialog*** da implementiramo i metodu ***onPrepareDialog*** koja se zove svaki put po pozivu metodu ***showDialog***.

8.6 – Kreiranje Alert dijaloga

- Vrši se pomoću klase ***AlertDialog.Builder***.
- Može se odrediti tekst poruke koja će biti prikazana metodom ***setMessage***, kao i broj dugmića (1/2/3), tekst na njima i listeneri koji će reagovati na klik.
- Dugmići i njihovi listeneri se dodaju metodama ***setNeutralButton***, ***setPositiveButton*** i ***setNegativeButton***. Sve navedene metode vraćaju referencu na samog Builder-a pa je omogućeno pozivanje više metoda u jednom redu - ***builder.metoda1().metoda2().metoda3();***

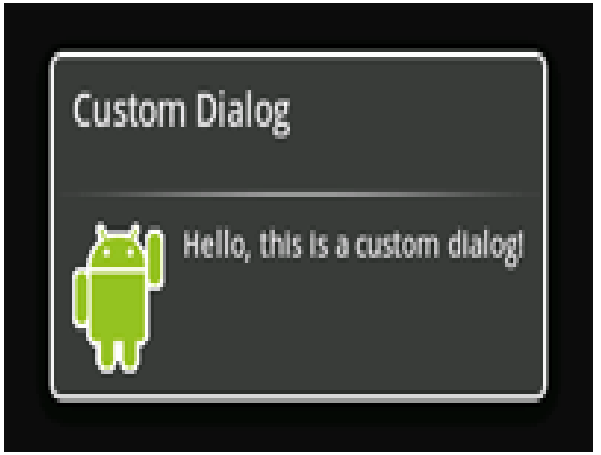
```
AlertDialog.Builder builder = new
AlertDialog.Builder(roditelj);
Dialog dijalog = builder.setMessage(idTeksta)
.setNeutralButton(R.string.standardOK, null)
.create();
```



8.6-Kreiranje dijaloga sa sopstvenim layout-om

- U ovoj varijanti moramo prvo da kreiramo svoj layout fajl u **res/layout**, a zatim, pri kreiranju dijaloga prosledimo ID layouta koji želimo da se koristi.
- Dalja procedura je ista kao pri radu sa layout-om aktivnosti - vadimo reference na elemente layouta i dodeljujemo im vrednosti/listenere po želji.

```
Dialog dialog = new Dialog(this);
dialog.setContentView(R.layout.moj_lejaut);
dialog.setTitle(R.string.KvizKraj);
TextView polje1 = (TextView)
dialog.findViewById(R.id.dijalog_polje1);
polje1.setText("tekst polja 1");
TextView polje2 = (TextView)
dialog.findViewById(R.id.dijalog_polje2);
polje2.setText("tekst polja 2");
```



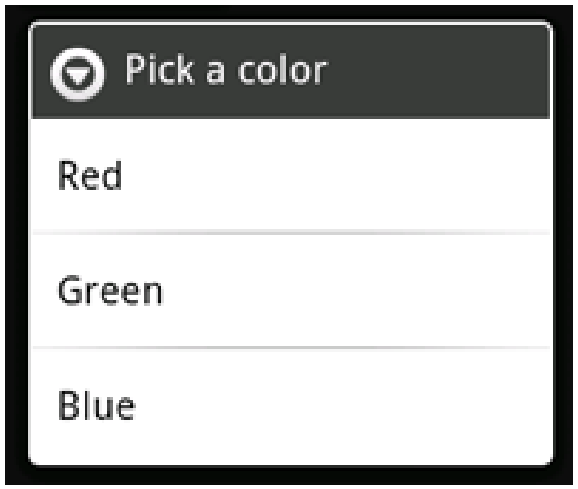
8.6 - Dijalozi

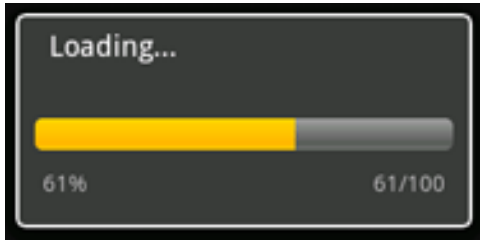
- Korišćen layout (moj_layout.xml):

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

      <TextView android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/dijalog_polje1" />
      <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/dijalog_polje2" />

</LinearLayout>
```

8.6 – Dijalozi

- Druge, specifične vrste dijaloga je uglavnom dovoljno samo kreirati uz odgovarajuće parametre pri konstrukciji pošto im je izgled predefinisan.
- Primeri drugih vrsta dijaloga:





Hvala na pažnji !!!

Pitanja

? ? ?