MOBILNE APLIKACIJE

Vežbe 3

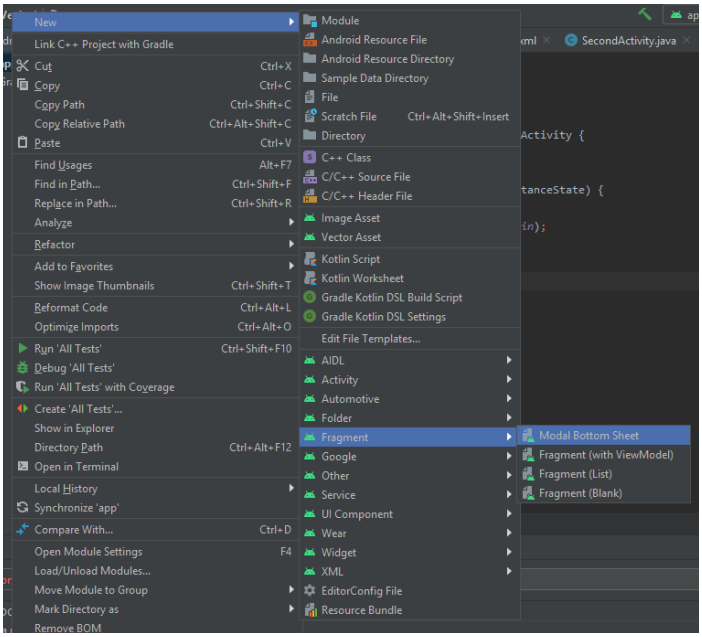
Sadržaj

1. Fragmenti

Fragment možemo posmatrati kao podaktivnost. To je deo ponašanja ili GUI-a aktivnosti. Jedna aktivnost može da sadrži više fragmenata i jedan fragment može da bude sadržan u više aktivnosti.

1.1 Kreiranje fragmenta

Novi fragment se kreira odabirom stavke iz menija: File > New > Fragment (slika 1) ili jednostavnim kreiranjem Java klase koja nasleđuje klasu *Fragment.* U ovom primeru odabrali smo *Fragment (Blank).*



Slika 1. Kreiranje novog fragmenta

Za razliku od aktivnosti, fragmente ne moramo da specificiramo unutar *Manifest* datoteke.

Na slikama 2 i 3 se vide klasa, *BlankFragment.java*, i izgled, *fragment_blank.xml*, koji su kreirani za nas.

```java
/**
 * A simple {@link Fragment} subclass.
 * Use the {@link BlankFragment#newInstance} factory method to
 * create an instance of this fragment.
 */
public class BlankFragment extends Fragment {
    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    private static final String ARG_PARAM1 = "param1";
    private static final String ARG_PARAM2 = "param2";

    // TODO: Rename and change types of parameters
    private String mParam1;
    private String mParam2;

    public BlankFragment() {
        // Required empty public constructor
    }

    /**
```

Slika 2. Primer dela klase predefinisanog fragmenta *BlankFragment*

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".BlankFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello blank fragment" />

</FrameLayout>
```

Slika 3. *BlankFragment layout*

Isto kao i aktivnost, fragment ima specifičnu metodu za postavljanje layout-a fragmenta (slika 4). Takođe, na istoj slici se može videti i kako se preuzimaju podaci koji su poslati u fragment.

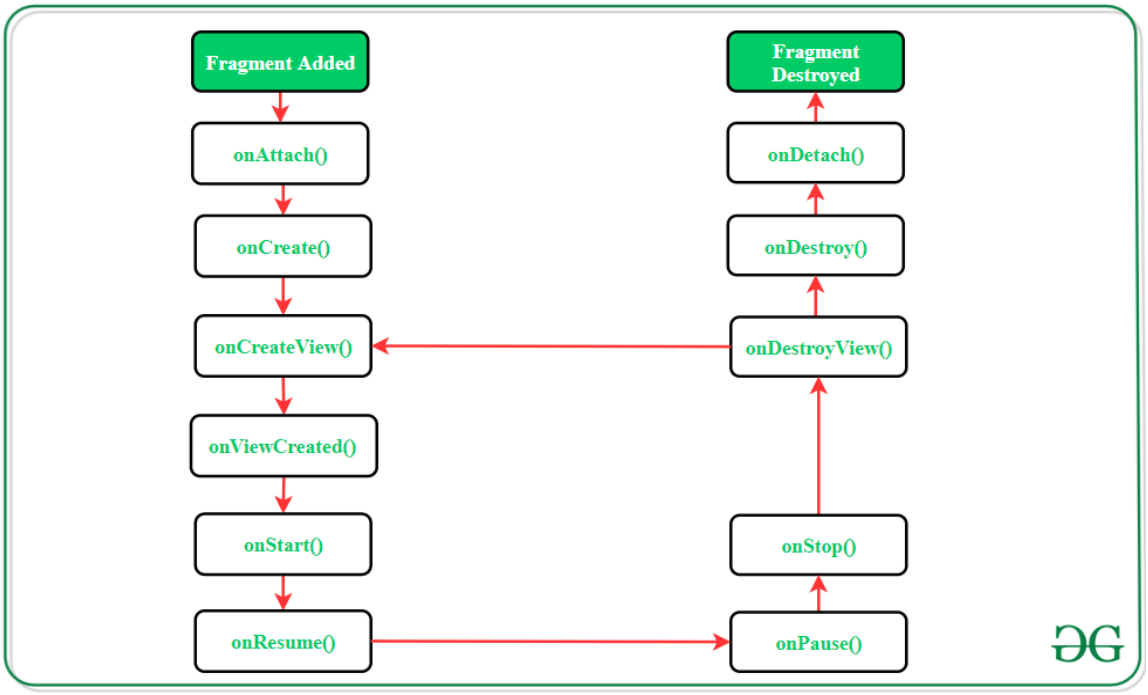
Kako se fragment postavlja na aktivnost možete pogledati u primeru za vežbe 3 na GitLab-u.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup vg, Bundle data) {
    View view = inflater.inflate(R.layout.mysecondfragment_layout, vg,  attachToRoot: false);

    Bundle bundle = getArguments();
    if (bundle != null){
        String param = bundle.getString( key: "SOME_PARAM_KEY",  defaultValue: "DEFAULT");
        TextView textView = view.findViewById(R.id.textView2);
        textView.setText(param);
    }

    return view;
}
```

Slika 4. Povezivanje fragmenta i njegovog layout-a

1.2 Životni ciklus fragmenta

Fragmenti takođe imaju životni ciklus i on zavisi od životnog ciklusa aktivnosti u kojoj se nalaze. Životni ciklus fragmenta je sličan kao i životni ciklus aktivnosti (slika 5), samo što je proširen sa dodatnim metodama koje omogućavaju interakciju sa aktivnošću u kojoj se nalaze.



Slika 5. Životni ciklus fragmenta [1]

onAttach

Metoda *onAttach* se poziva kada se fragment povezuje sa aktivnošću (slika 6).

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    Toast.makeText(getActivity(), text: "onAttach()", Toast.LENGTH_SHORT).show();
}
```

Slika 6. Primer *onAttach* metode

onCreate

Ova metoda se poziva kada se *onCreate* metoda aktivnosti izvrši (slika 7).

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Toast.makeText(getActivity(), text: "onCreate()", Toast.LENGTH_SHORT).show();
}
```

Slika 7. Primer *onCreate* metode

onCreateView

Metoda *onCreateView* se poziva da bi se iscrtao korisnički interfejs fragmenta (slika 8).

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup vg, Bundle data) {
    return inflater.inflate(R.layout.myfragment_layout, vg, attachToRoot: false);
}
```

Slika 8. Primer *onCreateView* metode

onDestroyView

Da bi se uništio korisnički interfejs fragmenta poziva se metoda *onDestroyView* (slika 9).

```
@Override
public void onDestroyView() {
    super.onDestroyView();
    Toast.makeText(getActivity(), text: "onDestroyView()", Toast.LENGTH_SHORT).show();
}
```

Slika 9. Primer *onDestoryView* metode

onStop

Kada fragment više nije vidljiv, poziva se metoda *onStop* (slika 10).

```
@Override
public void onStop() {
    super.onStop();
    Toast.makeText(getActivity(),  text: "onStop()", Toast.LENGTH_SHORT).show();
}
```

Slika 10. Primer *onStop* metode

onDetach

Na samom kraju se poziva *onDetach* kada se fragment odvezuje od aktivnosti (slika 11).

```
@Override
public void onDetach() {
    super.onDetach();
    Toast.makeText(getActivity(),  text: "onDeatach()", Toast.LENGTH_SHORT).show();
}
```

Slika 11. Primer *onDetach* metode

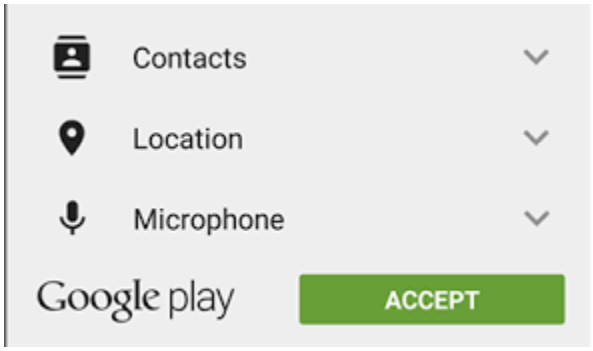
2. Permisije

Permisije omogućavaju zaštitu privatnosti korisnika štiteći pristup ograničenim podacima i akcijama. Mnoge funkcionalnosti je moguće implementirati bez potrebe za permisijama, pa je poželjno proveriti da li je upotreba permisija zaista neophodna.

Permisije (prava pristupa) možemo podeliti na statičke i dinamičke.

2.1 Statičke permisije

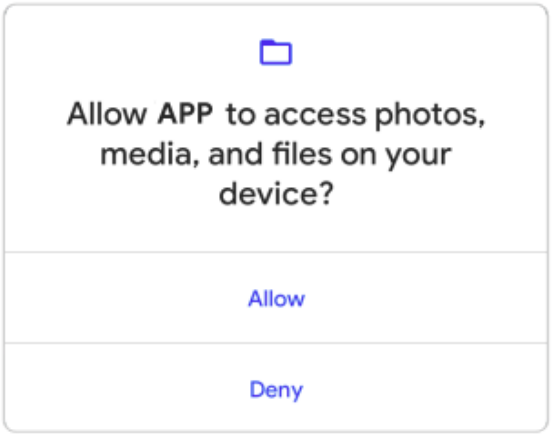
Do Androida 6 (API 23), prava pristupa su tražena prilikom instalacije same aplikacije. Ukoliko bi korisnik odbio prava pristupa, otkazao bi instalaciju. Prikaz za odobrenje ovog tipa prava pristupa dat je na slici 12.



Slika 12. Prikaz dijaloga za statička prava pristupa

2.2 Dinamičke permisije

Od Androida 6, od korisnika se traži dozvola za pravo pristupa u toku rada same aplikacije, neposredno pre pristupa ograničenim podacima ili pre izvršenja ograničenih akcija. Na slici 13 dat je primer sistemskog prompta kojim se traži odobrenje prava pristupa.



Slika 13. Prompt za odobrenje dinamičke permisije

Permisije mogu biti grupisane u setove logički povezanih permisija.

Sistem može automatski da odobrava sva prava pristupa, ako je korisnik prihvatio jedno iz te grupe. Ovim se smanjuje broj dijaloga prezentovanih korisniku kada aplikacija traži pristup blisko povezanim pravima pristupa. Ako tražimo od korisnika da odobri npr. READ_CONTACTS i on prihvati, sledi da kada aplikacija bude tražila da se odobri WRITE_CONTACTS neće se zahtevati interakcija sa korisnikom, već će se zahtev automatski odobriti, jer je korisnik već prihvatio READ_CONTACT, koji je u istoj grupi.

2.3 Primer

Kako bi dobili pravo pristupa, navodimo željenu permisiju u *AndroidManifest.xml* fajlu. Na slici 14 navedene su tražene permisije za pristup internetu, promeni statusa WiFi-ja i kameri.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CAMERA" />
```

Slika 14. Statički deklarisane permisije

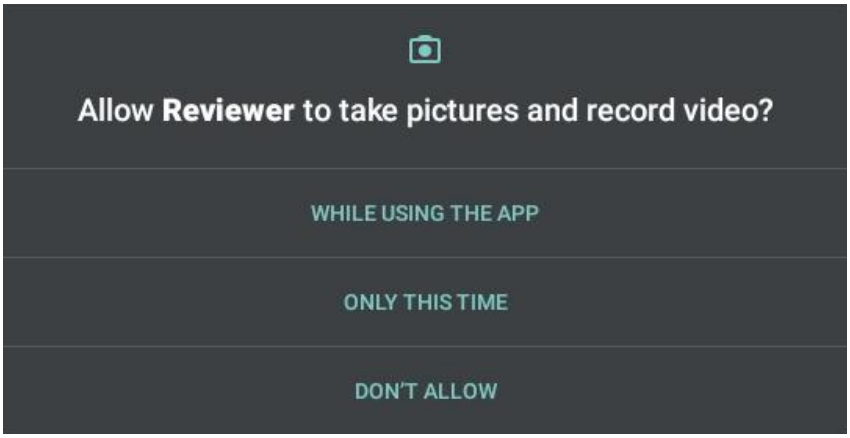
Određena prava pristupa se automatski odobravaju bez ikakve interakcije sa korisnikom i za njih je dovoljno navesti ih unutar *AndroidManifest.xml* fajla. U ovakve permisije ubrajamo pristup internetu i promenu statusa WiFi-ja.

Pre korišćenja podataka ili pokretanje akcije za koju je potrebna permisija, neophodno je proveriti da li je već dozvoljena. Na slici 15 vrši se provera permisije za kameru. Ukoliko ne postoji dozvola za upotrebu kamere, korisniku se ispisuje poruka i traži pristup prikazom prompta putem *ActivityCompat.requestPermissions()*.

```
if(ContextCompat.checkSelfPermission( context: MainActivity.this, Manifest.permission.CAMERA)
        != PackageManager.PERMISSION_GRANTED)
{
    Snackbar.make(v, text: "CAMERA NOT PERMITTED", Snackbar.LENGTH_LONG)
            .setAction( text: "Action", listener: null).show();
    ActivityCompat.requestPermissions( activity: MainActivity.this,
            new String[]{Manifest.permission.CAMERA, Manifest.permission.CAMERA},
            CAMERA_PERMISSION_CODE);

} else {

    Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse( uriString: "http://www.google.com"));
    startActivity(i);
}
```

Slika 15. Provera permisije

Na slici 16 prikazan je prompt gde je korisniku omogućeno da prihvati ili odbije davanje dozvole.



Slika 16. Prompt za davanje ili odbijanje prava pristupa

Nakon izbora korisnika, pokreće se metoda *onRequestPermissionResult()* kojoj se prosleđuje izbor korisnika, permisije i kod prava pristupa (sliika 17).

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults)
{
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (requestCode == CAMERA_PERMISSION_CODE) {

        // Checking whether user granted the permission or not.
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

            // Showing the toast message
            Toast.makeText( context: MainActivity.this,  text: "Camera Permission Granted", Toast.LENGTH_SHORT).show();
        }
        else {
            Toast.makeText( context: MainActivity.this,  text: "Camera Permission Denied", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Slika 17. Metoda pozvana nakon prompta za dozvolu permisija

3. Domaći

Domaći se nalazi na *Canvas-u (canvas.ftn.uns.ac.rs)* na putanji *Vežbe/03 Zadatak.pdf*.

Primer možete preuzeti na sledećem linku:
https://gitlab.com/antesevicceca/mobilne-aplikacije-sit

Za dodatna pitanja možete se obratiti asistentima:

- Svetlana Antešević (svetlanaantesevic@uns.ac.rs)
- Jelena Matković (matkovic.jelena@uns.ac.rs)