Koriscenje **Preference AndroidX** biblioteke za kreiranje korisnickih podesavanja

Sadrzaj

[**Pregled** 2](#_Toc122521514)

[**Uvod u Prefference biblioteku** 2](#_Toc122521515)

[**Kreiranje hijerarhije** 2](#_Toc122521516)

[**Ukljucivanje hijerarhije** 3](#_Toc122521517)

[**Organizacija korisnickih podesavanja** 3](#_Toc122521518)

[**Kategorizacija Preference elemenata** 3](#_Toc122521519)

[**Podela hijerarhije na vise ekrana** 4](#_Toc122521520)

[**Prilagodjavanje korisnickih podesavanja** 6](#_Toc122521521)

[**Pronalazak Preference elemenata** 6](#_Toc122521522)

[**Kontrolisanje vidljivosti Preference elementa** 6](#_Toc122521523)

[**Dinamicki update summary-ja** 7](#_Toc122521524)

[**Koriscenje SimpleSummaryProvider** 7](#_Toc122521525)

[**Prilagodjavanje EditTextPreference dialoga** 8](#_Toc122521526)

[**Akcije Preference-a** 9](#_Toc122521527)

[**Setovanje Intent-a** 9](#_Toc122521528)

[**OnPreferenceClickListener** 9](#_Toc122521529)

[**Koriscenje sacuvanih vrednosti** 10](#_Toc122521530)

[**Preference data storage** 10](#_Toc122521531)

[**SharedPreferences** 10](#_Toc122521532)

[**PreferenceDataStore** 10](#_Toc122521533)

[**Citanje vrednosti iz Preference elementa** 10](#_Toc122521534)

[**Cekanje na promenu vrednosti Preference element-a** 11](#_Toc122521535)

[**OnPreferenceChangeListener** 11](#_Toc122521536)

[**OnSharedPreferenceChangeListener** 11](#_Toc122521537)

[**Koriscenje custom data store-a** 12](#_Toc122521538)

[**Kreiranje hijerarhije u kodu** 13](#_Toc122521539)

**Ovaj vodic nam objasnjava kako da koristimo AndroidX Prefference biblioteku.**

# **Pregled**

Podesavanja (eng. Settings) omogucavaju korisniku da promeni odredjenu funkcionalnost I ponasanje date aplikacije. Podesavanja mogu uticati na ponasanje pozadinskog procesiranja (background behaviour), poput ucestalosti sinhronizacije aplikacije sa cloudom.

Preporuka za integraciju korisnickih podesavanja u aplikaciju je koriscenje ***AndroidX Prefference*** biblioteke. Ova biblioteka upravlja koriscnikim UI-jem I komunicira sa storage-om pa programer moze definisati samo individualna podesavanja koje ce korisnik konfigurisati u daljem radu sa aplikacijom. Ova biblioteka dolazi sa ***Material Theme*** koja nam omogucava konzistentan user experience na svim uredjajima I verzijama operativnog Sistema.

## **Uvod u Prefference biblioteku**

***Prefference*** je osnovna gradivna jedinica Prefference biblioteke. Ekran za korisnicka podesavanja (eng. *Settings Screen)* se sastoji od hijerarhije Prefference elemanata. Hijerarhija se moze definisati unutar XML resursa ili se taj hijerarhija moze napraviti u kodu.

### **Kreiranje hijerarhije**

|  |
| --- |
| <PreferenceScreen     xmlns:app="http://schemas.android.com/apk/res-auto">      <SwitchPreferenceCompat         app:key="notifications"         app:title="Enable message notifications"/>      <Preference         app:key="feedback"         app:title="Send feedback"         app:summary="Report technical issues or suggest new features"/>  </PreferenceScreen> |

*Pre svega, mali osvrt na snippet. Root element u xml mora biti <PreferenceScreen> I ovaj xml resurs mora biti smesten u res/xml/ direktorijumu.*

Ova hijerarhija prikaza u snippetu gore sadrzi dva individualna Preference elementa: ***SwitchPreferenceCompat*** koji nam omogucava da korisnik toogle-uje podesavanje na on/off I osnovni Preference bez odredjenog widgeta. Prilikom kreiranja hijerarhije, posebno obratiti paznju da svaki element ima jedinstveni kljuc.

### **Ukljucivanje hijerarhije**

Da bi inflate-ovali hijerarhiju iz xml-a, potrebno je kreirati klasu koja ce naslediti ***PreferenceFragmentCompat*** I potrebno je override-ovati ***onCreatePreferences***() I obezbediti xml resurs koji cemo inflate-ovati kao sto je prikazano u snippetu ispod:

|  |
| --- |
| class MySettingsFragment : PreferenceFragmentCompat() {     override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {         setPreferencesFromResource(R.xml.preferences, rootKey)     } } |

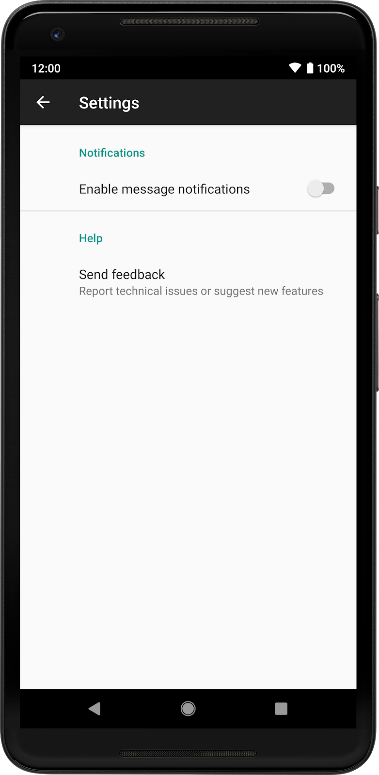
# **Organizacija korisnickih podesavanja**

## **Kategorizacija Preference elemenata**

Ukoliko imamo nekoliko povezanih Preference elemenata na jednom ekranu, oni mogu biti grupisani uz pomoc ***PreferenceCategory*** elementa. Ovaj element prikazuje naslov kategorije I vizualno odvaja kategoriju na ekranu.

Da bi se definisala kategorija na XML, potrebno je obuhvatiti Preference elemente sa PreferenceCategory elementom kao sto je prikazano na sledecem snippetu:

|  |
| --- |
| <PreferenceScreen     xmlns:app="http://schemas.android.com/apk/res-auto">  **<PreferenceCategory         app:key="notifications\_category"         app:title="Notifications">**          <SwitchPreferenceCompat             app:key="notifications"             app:title="Enable message notifications"/>      **</PreferenceCategory>**  **<PreferenceCategory         app:key="help\_category"         app:title="Help">**          <Preference             app:key="feedback"             app:summary="Report technical issues or suggest new features"             app:title="Send feedback"/>  **</PreferenceCategory>**  </PreferenceScreen> |



## **Podela hijerarhije na vise ekrana**

Ukoliko imamo veliki broj Preference elemenata ili bas razlicite kategorije, kompletan ekran mozemo reorganizovati tako sto cemo elemente prikazivati na razlicitim ekranima. Svaki ekran treba da bude PreferenceFragmentCompat sa svojom sopstvenom hijerarhijom. Preference elementi na inicijalnom ekranu se mogu iskoristiti kao veza (eng. *Link*) ka separatisanim ekranima koji takodje sadrze preference elemente.

Graphical user interface, application, Teams

Description automatically generated

A picture containing text, screenshot, monitor, electronics

Description automatically generated

Da bi se povezivanje ekrana sa odredjenim Preference elementom radilo, potrebno je dekarisati ***app:fragment*** u XML fajlu, ili se moze koristiti ***Preference.setFragment()*** metoda. Potrebno je definisati kompletan package name PreferenceFragmentCompat-a koji ce biti pokrenut kada je Preference aktiviran kao sto je prikazano u snipprtu dole:

|  |
| --- |
| <Preference         app:fragment="com.example.SyncFragment"         .../> |

Kada korisnik pritisne dati Preference element sa asociranim fragmentom, interfejs metoda ***PreferenceFragmentCompat.onPreferenceStartFragmentCallback.onPreferenceStartFragment()*** se poziva. Ova metoda je mesto gde treba implementirati prikazivanje novog ekrana I treba da bude implementirana u activity klasi. Ukoliko se ova metoda ne implementira, fallback implementacija ce biti koriscena. U vecini slucajeva ovo radi ali preporuka je implementacija ove metode da bi se tranzicija izmedju framenata bila pravilno konfigurisana I promena naslova bila pravilno prikazana.

|  |
| --- |
| class MyActivity : AppCompatActivity(),     ***PreferenceFragmentCompat.OnPreferenceStartFragmentCallback {     override fun onPreferenceStartFragment(caller: PreferenceFragmentCompat, pref: Preference): Boolean*** {         // Instantiate the new Fragment         val args = pref.extras         val fragment = supportFragmentManager.fragmentFactory.instantiate(                 classLoader,                 pref.fragment)         fragment.arguments = args         fragment.setTargetFragment(caller, 0)         // Replace the existing Fragment with the new Fragment         supportFragmentManager.beginTransaction()                 .replace(R.id.settings\_container, fragment)                 .addToBackStack(null)                 .commit()         return true     } } |

Kao alternativu, ukoliko imamo potrebe za ozbilnim prilagodjavanjem svako ekrana, ili imamo potrebe za potpunim activity tranzicijama izmedju ekrana, mozemo koristiti odvojen Activity za svaki PreferenceFragmentCompat. Ovaj pristup se generalno ne preporucuje za vecinu aplikacija ali je moguc za odredjene slucajeve u praksi.

# **Prilagodjavanje korisnickih podesavanja**

## **Pronalazak Preference elemenata**

Da bismo pristupili individualnom Preference, poput preuzimanja ili setovanja Preference vrednosti, potrebno je koristiti ***PreferenceFragmentCompat.findPreference()***. Ova metoda pretrazuje hijerarhiju I trazi odredjeni Preference element sa prosledjenim kljucem.

|  |
| --- |
| <EditTextPreference         app:key="signature"         app:title="Your signature"/>  override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {     setPreferencesFromResource(R.xml.preferences, rootKey)     val signaturePreference: EditTextPreference? = findPreference("signature")     // do something with this preference } |

## **Kontrolisanje vidljivosti Preference elementa**

Kada korisnik prikaze Settings ekran, aplikacija moze da kontrolise koji Preference elementi mogu biti vidljivi koirnisku. Npr. odredjeni Preference ima smisla samo ako je odredjeni feature enable-ovan I samim tim taj Preference treba da bude vidljiv. U suprotnom, treba ga sakriti sa ekrana.

Da bi se prikazao odredjeni Preference samo kada se steknu uslovi za njegovo prikazivanje, potrebno je prvo podesiti njegov visibility na false vrednost u xmlu.

|  |
| --- |
| <EditTextPreference         app:key="signature"         app:title="Your signature"         app:isPreferenceVisible="false"/> |

Nakon toga, u onCreatePreference() je potrebno implementirati logiku za njegovo prikazivanje kada se ispostuju uslovi za isto. To se moze implementirati na sledeci nacin:

|  |
| --- |
| override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {     setPreferencesFromResource(R.xml.preferences, rootKey)     if(/\*some feature\*/) {         val signaturePreference: EditTextPreference? = findPreference("signature")         signaturePreference?.isVisible = true     } } |

## **Dinamicki update summary-ja**

Preference element koji cuva podatke treba da prikaze trenutnu vrednost u svom summary u cilju unapredjenja korisnickom razumevanja trenutnog stanja Preference elementa. Npr. EditTextPreference treba da prikaze trenutno sacuvanu tekstualnu vrednost, ili da ListPreference prikaze trenutno selektovan element liste. Jedan od nacin za postizanje ovog cilja jeste ***SummaryProvider.***

### **Koriscenje SimpleSummaryProvider**

ListPreference i EditTextPreference ukljucuje jednostavnu SummaryProvider implementaciju koja automatski prikazuje sacuvane vrednosti Preference u svom summary. Ukoliko vrednost nije sacuvana, ova komponenta prikazuje *„Not Set“.*

Ova implementacija se moze ukljuciti u xml uz pomoc sledeceg atributa ***app:useSimpleSummaryProvider=“true“***

Alternativno, u kodu se moze koristiti *ListPreference.SimpleSummaryProvider.getInstance()* i *EditTextPreference.SimpleSummaryProvider.getInstance()* za dobijanje jednostavne SummaryProvider instance i setovati je odredjenom Preference elementu, kao sto je prikazano u sledecem primeru:

|  |
| --- |
| listPreference.summaryProvider = ListPreference.SimpleSummaryProvider.getInstance() editTextPreference.summaryProvider = EditTextPreference.SimpleSummaryProvider.getInstance() |

#### **Koriscenje custom SummaryProvider-a**

Android nudi opciju da mozemo kreirati sopstveni SummaryProvider i da reimplementiramo provideSummary() metodu koja ce prilagoditi summary kad god je to zatrazeno od strane Preference elementa. Npr, EditTextPreference moze prikazivati duzinu sacuvane tekstualne vrednosti kao sopstveni summary.

Uzecemo sledeci primer:

|  |
| --- |
| <EditTextPreference         app:key="counting"         app:title="Counting preference"/> |

U onCreatePreference(), potrebno je kreirati novi SummaryProvider i reimplementirati provideSummary() koji ce vratiti summary koji ce biti prikazani:

|  |
| --- |
| val countingPreference: EditTextPreference? = findPreference("counting")  countingPreference?.summaryProvider = SummaryProvider<EditTextPreference> { preference ->     val text = preference.text     if (TextUtils.isEmpty(text)) {         "Not set"     } else {         "Length of saved value: " + text.length     } } |

Sada Preference summary prikazuje duzinu sacuvane vrednosti ili „Not Set“ ukoliko ta vrednost ne postoji.

## **Prilagodjavanje EditTextPreference dialoga**

Unutar EditTextPreference dialoga se moze prilagoditi ponasanje tekstualnog polja time sto cemo attach-ovati **OnBindEditTextListener**. Ovaj listener se poziva kada je dialog prikazan korisniku na ekranu.

Kao primer, prilagodavanje dialoga se moze odraditi tako da dialog prima samo brojeve. To se moze postici sledecom implementacijom:

|  |
| --- |
| <EditTextPreference         app:key="number"         app:title="Numbers only preference"/> |

Nakon toga, u onCreatePreferences() je potrebno kreirati novi OnBindEditTextListener i reimplementirati onBindEditText() u cilju prilagodavanja EditText kada bude prikazan korisniku:

|  |
| --- |
| val numberPreference: EditTextPreference? = findPreference("number")  numberPreference?.setOnBindEditTextListener { editText ->     editText.inputType = InputType.TYPE\_CLASS\_NUMBER } |

Sada kada dialog postane vidljiv korisniku, keyboard ce se otvarati samo u numeric modu, pa korisnik moze uneti samo numericke vrednosti unutar EditText-a.

## **Akcije Preference-a**

Preference element moze imati specificnu akciju kada korisnik tapne element. Npr. Preference se moze ponasati kao link ka odvojenoj odredjenoj komponenti aplikacije. Da bi se dodala akcija Preference elementu, mozemo ili setovati Intent unutar Preference-a ili mozemo setovati onPreferenceClickListener za jos specificniju logiku.

### **Setovanje Intent-a**

Za pokretanje novog Fragmenta ili Activity-ja prilikom tap na odredjeni Preference element, mozemo setovati Intent na taj Preference. To se postize na sledeci nacin:

|  |
| --- |
| <Preference         app:key=”activity”         app:title="Launch activity"> ***<intent             android:targetPackage="com.example"             android:targetClass="com.example.ExampleActivity">         <extra                 android:name="example\_key"                 android:value="example\_value"/>     </intent>*** </Preference>  Ili u kodu:  val intent = Intent(context, ExampleActivity::class.java) activityPreference.setIntent(intent) |

### **OnPreferenceClickListener**

OnPreferenceClickListener se moze dodeliti odredjenom Preference elementu, sto predstavlja callback metodu za **onPreferenceClick()** koji se poziva kada je Preference tapnut. Npr. ovaj listener mozemo koristiti kada zelimo da predjemo na drugi Fragment ili Activity ukoliko ta navigacija ima kompleksnu logiku. Da bi se ovaj listener setovao, mozemo iskoristiti sledeci kod:

|  |
| --- |
| onClickPreference.setOnPreferenceClickListener({     // do something     true }) |

# **Koriscenje sacuvanih vrednosti**

Ova sekcija opisuje kako da store-ujemo i koristimo vrednosti Preference elementa uz pomoc Prefrence biblioteke.

## **Preference data storage**

### **SharedPreferences**

Inicijalno, Preference koristi **SharedPreferences** komponentu za cuvanje vrednosti. SharedPreferences API omogucava citanje i pisanje jednostavnih key-value parova iz fajla koji je dostupan kroz celu aplikaciju i sesijama koje se javljaju prilikom rada aplikacije. Preference Library koristi privatnu SharedPreference instancu tako da samo aplikacija koju implementiramo moze da je koristi.

|  |
| --- |
| <SwitchPreferenceCompat         app:key="notifications"         app:title="Enable message notifications"/> |

Kada korisnik promeni stanje ovog switch-a na „On“, SharedPreferences fajl je update-ovan sa key value parom sa vrednoscu „notifications“:“true“.

### **PreferenceDataStore**

Dok Preference biblioteka cuva podatke unutar SharedPreferences po defaultu, SharedPreferences nisu uvek idealna solucija za koriscenje. Npr. ako aplikacija zahteva od korisnika da se signin-uje, zelecemo da sacuvamo aplikaciona podesavanja na cloud-u pa samim tim postizemo da se ta podesavanja refklektuju na svim uredjajima i platformama. Jednako, ukoliko aplikacija ima konfiguracione opcije koje su specificne za odredjeni device, svaki korisnik bi trebao imati odvojena podesavanja, sto samim tim pravi SharedPreferences soluciju kao ne idealnu.

PreferenceDataStore nam omogucava da koristimo custom storage za cuvanje Preference vrednosti.

## **Citanje vrednosti iz Preference elementa**

Da bismo dobili SharedPreferences objekar koji se koristi, potrebno je pozvati metodu ***PreferenceManager.getDefaultSharedPreferences().*** Ova metoda je dostupna u celoj aplikaciji i koristi se na sledeci nacin:

|  |
| --- |
| val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this /\* Activity context \*/) val name = sharedPreferences.getString("signature", "") |

## **Cekanje na promenu vrednosti Preference element-a**

Da bismo cekali i slusali promene Preference vrednosti, mozemo birati izmedju dva interfejsa:

* Preference.OnPreferenceChangeListener
* SharedPreference.OnSharedPreferenceChangeListener

Tabela prikazuje razlike izmejdu ova dva interfejsa:

|  |  |
| --- | --- |
| **OnPreferenceChangeListener** | **OnSharedPreferenceChangeListener** |
| Setuje se na individualni Preference | Dodeljuje se svim Preference elementima |
| Poziva se kada Preference promeni svoju sacuvanu vrednost. Ovo vazi ako se nova vrednost ne podudara sa sacuvanom vrednosti | Poziva se samo ako je sacuvana vrednost Preference promenjena |
| Poziva se samo kroz Preference biblioteku. Odredjeni odvojeni deo aplikacije moze promeniti sacuvanu vrednost | Poziva se kad god je sacuvana vrednost promenjena, cak i od strane neke komponente iz drugog dela aplikacije |
| Pozvana je pre nego sto dodje do cuvanja nove vrednosti | Pozvana nakon cuvanja nove vrednosti |
| Pozvana prilikom koriscenja SharedPreferences ili PreferenceDataStore | Koristi se samo kada koristimo SharedPreferences |

### **OnPreferenceChangeListener**

Implementiranje OnPreferenceChangeListener-a nam omogucava da cekamo promenu vrednosti pre nego sto se ona dogodi. U tom slucaju, mozemo validirati da li se ova promena moze dogoditi i da li je vrednost koja ce se sacuvati validna.

|  |
| --- |
| override fun onPreferenceChange(preference: Preference, newValue: Any): Boolean {     Log.e("preference", "Pending Preference value is: $newValue")     return true } |

U sledecem koraku, potrebno je setovati ovaj listener direktno uz pomoc ***setOnPreferenceChangeListener()*** metode.

|  |
| --- |
| preference.onPreferenceChangeListener = ... |

### **OnSharedPreferenceChangeListener**

Kada cuvamo vrednost iz Preference elemenata koristeci SharedPreferences, takodje mozemo koristiti SharedPreferences.OnSharedPreferencesChangeListener za slusanje promena. Ovo nam omogucava da slusamo kada dodje do promene vrednosti Preference poput sinhronizacije podesavanja sa serverom.

|  |
| --- |
| override fun onSharedPreferenceChanged(sharedPreferences: SharedPreferences, key: String) {     if (key == "signature") {         Log.i(TAG, "Preference value was updated to: " + sharedPreferences.getString(key, ""))     } }  preferenceManager.sharedPreferences.registerOnSharedPreferenceChangeListener(...)  val listener: SharedPreferences.OnSharedPreferenceChangeListener =         SharedPreferences.OnSharedPreferenceChangeListener {...}  override fun onResume() {     super.onResume()     preferenceManager.sharedPreferences.registerOnSharedPreferenceChangeListener(this) }  override fun onPause() {     super.onPause()     preferenceManager.sharedPreferences.unregisterOnSharedPreferenceChangeListener(this) } |

## **Koriscenje custom data store-a**

Obzirom da je cuvanje Preference elemenata uz pomoc SharedPreferences preporucljiva, ipak mozemo koristiti i custom data store. Custom data store moze biti koristan ukoliko aplikacija cuva podatke u bazi podataka ili su vrednost device specific.

Da bi se implementirao custom data store, prvo treba kreirati klasu koja nasledjuje PreferenceDataStore kao sto je prikazano u sledecim snippetu:

|  |
| --- |
| class DataStore : PreferenceDataStore() {     override fun putString(key: String, value: String?) {         // Save the value somewhere     }      override fun getString(key: String, defValue: String?): String? {         // Retrieve the value     } } |

Treba obratiti paznju da zahtevne operacije izmestimo sa main thread-a u cilju izbegavanja blokiranja korisnickog interfejsa.

Nakon sto implementiramo custom data store, moramo setovati taj data store u onCreatePreferences() da bi Preference objekat cuvao podatke unutar data store umesto default pristupa sa SharedPreferences. Data store moze biti enable-ovan za svaki Preference ili za celokupnu hijerarhiju.

Da bi se enable-ovao custom data store za specificni Preference, potrebno je pozvati setPreferenceDataStore() nad Preference elementom kao sto je prikazano u primeru dole:

|  |
| --- |
| val preference: Preference? = findPreference("key") preference?.preferenceDataStore = dataStore  ili za celu hijerahiju:  val preferenceManager = preferenceManager preferenceManager.preferenceDataStore = dataStore |

# **Kreiranje hijerarhije u kodu**

|  |
| --- |
| override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {     val context = preferenceManager.context     val screen = preferenceManager.createPreferenceScreen(context)      val notificationPreference = SwitchPreferenceCompat(context)     notificationPreference.key = "notifications"     notificationPreference.title = "Enable message notifications"      val feedbackPreference = Preference(context)     feedbackPreference.key = "feedback"     feedbackPreference.title = "Send feedback"     feedbackPreference.summary = "Report technical issues or suggest new features"      screen.addPreference(notificationPreference)     screen.addPreference(feedbackPreference)      preferenceScreen = screen } |

Dodavanje PreferenceCategory je slicno:

|  |
| --- |
| override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {     val context = preferenceManager.context     val screen = preferenceManager.createPreferenceScreen(context)      val notificationPreference = SwitchPreferenceCompat(context)     notificationPreference.key = "notifications"     notificationPreference.title = "Enable message notifications"      val notificationCategory = PreferenceCategory(context)     notificationCategory.key = "notifications\_category"     notificationCategory.title = "Notifications"     screen.addPreference(notificationCategory)     notificationCategory.addPreference(notificationPreference)      val feedbackPreference = Preference(context)     feedbackPreference.key = "feedback"     feedbackPreference.title = "Send feedback"     feedbackPreference.summary = "Report technical issues or suggest new features"      val helpCategory = PreferenceCategory(context)     helpCategory.key = "help"     helpCategory.title = "Help"     screen.addPreference(helpCategory)     helpCategory.addPreference(feedbackPreference)      preferenceScreen = screen } |