



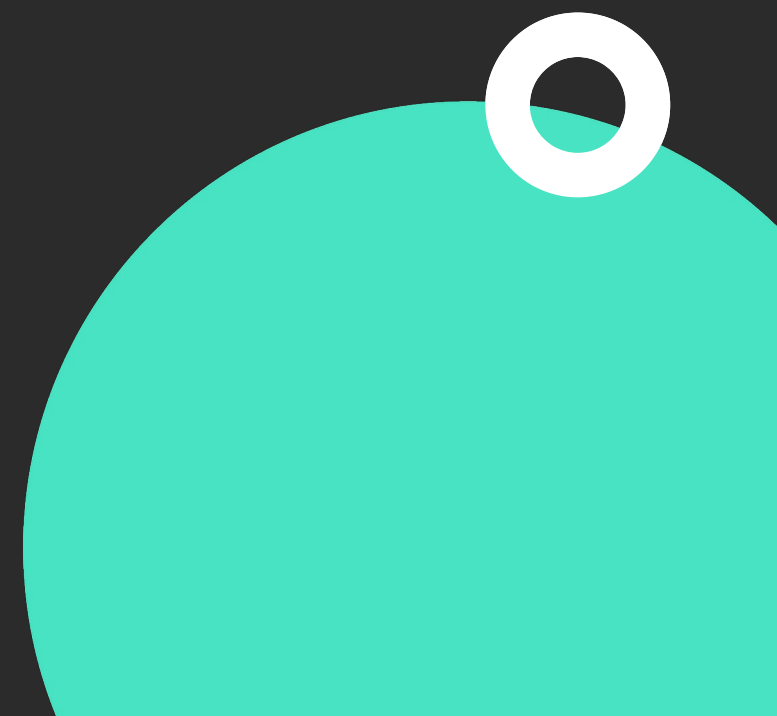
# Exploring the Preferences DataStore



Mehdi Haghgoo

 @IAmMehdiHaghgoo

 in/MehdiHaghgoo



# Section Overview

- How to create a Preferences DataStore
- Read from and write to Preferences DataStore
- Exception handling
- Code challenge: New sort order option



# Creating a Preferences DataStore Instance

- Create an instance of a Preferences Datastore
  - Using the `preferencesDataStore` property delegate with a `Context` Receiver
    - To provide your datastore instance manually
  - Using the `PreferenceDataStoreFactory` API
    - Used to provide DataStore instance via dependency injection frameworks such as Hilt



# Using property delegate

- Construct Preferences Datastore instance using `preferencesDataStore` property delegate

```
// Preferences DataStore
private const val USER_PREFERENCES_NAME =
    "user_preferences"

private val Context.dataStore by
    preferencesDataStore(name = USER_PREFERENCES_NAME)
```



# Using PreferenceDataStoreFactory

- Construct Preferences Datastore instance using PreferenceDataStoreFactory
- Params:
  - `corruptionHandler`
  - `scope`
  - `produceFile`

```
PreferenceDataStoreFactory.create(  
    corruptionHandler = ReplaceFileCorruptionHandler (  
        produceNewData = { emptyPreferences() }  
    ),  
    scope = CoroutineScope (  
        Dispatchers.IO + SupervisorJob () ),  
    produceFile = { appContext.preferencesDataStoreFile  
        ( USER_PREFERENCES) }  
)
```



# Reading from Preferences DataStore

- Done in background
- `DataStore<Preferences>.data` property
- Read data exposed via `Flow<Preference>`
- Returned Flow will emit a value or throw exception

```
fun getUserPreferences(): Flow<UserPreference> {  
    return datastore.data.map {preferences->  
        preferences.toUserPreferences()  
    }  
}
```



# Persist Data (Write) to Preferences DataStore

- Done in background
- `DataStore<Preferences>.edit(transform: suspend (MutablePreferences) -> Unit)`
- The `transform` block applies changes to the persisted data

```
val SORT_ORDER_KEY = stringPreferencesKey("sort_order")
suspend fun disableSorting() {
    datastore.edit { preferences ->
        preferences[SORT_ORDER_KEY] = SortOrder.NONE.name
    }
}
```



# Handling Preferences DataStore Read Exceptions

```
fun getUserPreferences(): Flow<UserPreference> {  
    return datastore.data.catch {exception->  
        if (exception is IOException){  
  
            Log.e(LogTag,"The following error occurred while reading data from data preferences $exception")  
            emit(emptyPreferences())  
  
        }else throw exception  
    }.map {preferences-> preferences.toUserPreferences() }  
}
```





# Handle Preferences DataStore Write Exceptions

```
suspend fun disableSorting( ) {  
    try {  
        datastore.edit {preferences->  
            preferences[SORT\_ORDER\_KEY] = SortOrder.NONE.name  
        }  
    } catch (e: IOException) {  
        // Handle error  
    }  
}
```



# Code Challenge

- Add a new sort order option
  - New sort order must enable the user to sort the comics based on names
- Implement read and write functionality of the new sort order option in the **UserPreferencesRepository.kt**.



# Section Summary

- How to create a Preferences DataStore instance
- How to read from and write to the Preferences DataStore
- How to handle read and write exceptions thrown by Preferences DataStore





# Preferences DataStore in Action

Up Next

