



PROGRAMOWANIE URZĄDZEŃ MOBILNYCH 2

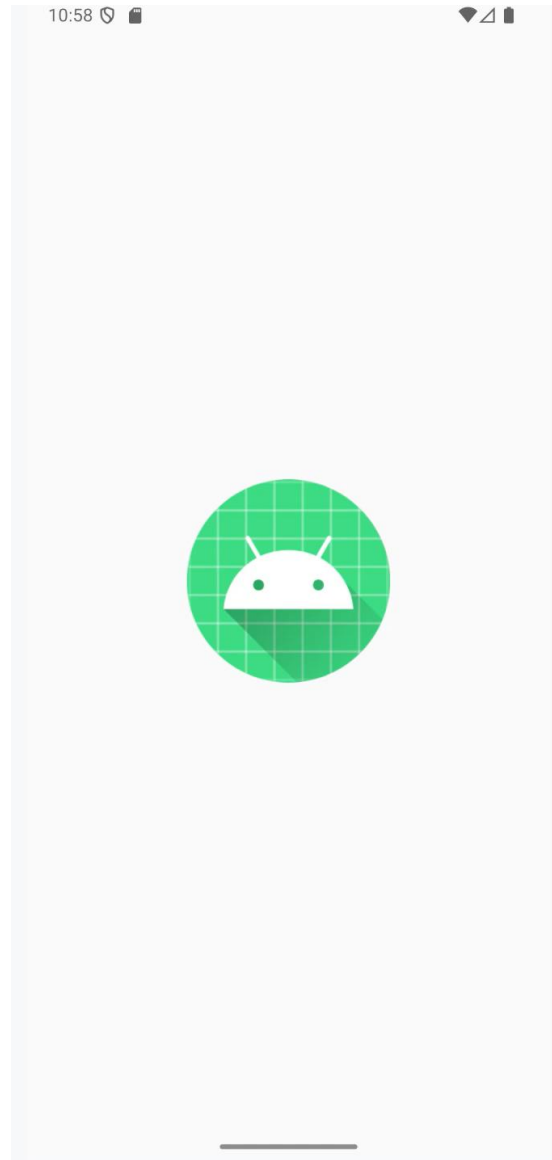
WYKŁAD 8

Zapis Danych do Plików:

- SharedPreferences,
- DataStore

Dwa mechanizmy:

- **SharedPreferences** - to wbudowany w Androida mechanizm do przechowywania **prostych danych** w formie **klucz-wartość** (np. nazwa_ustawienia -> wartość). Aplikacja może zapisywać i odczytywać **krótkie informacje**.
- **DataStore** - nowsza biblioteka, która występuje w dwóch wariantach, **Preferences DataStore** i **Proto DataStore**. Wszystkie operacje są asynchroniczne i bezpieczne do wywołania z wątku UI. Przechowuje dane w formie **klucz-wartość** (Preferences) lub **całe obiekty** (Proto)



SharedPreferences

Klasa przyjmuje **Context** jako parametr, ponieważ jest on **niezbędny** do uzyskania **dostępu** do **SharedPreferences** specyficznych dla tej aplikacji.

```
class SettingsManager(context: Context) {
```

2 Usages

```
private val prefs = context.getSharedPreferences(  
    p0 = "app_settings",  
    p1 = Context.MODE_PRIVATE)
```

2 Usages

```
companion object {
```

2 Usages

```
private const val NOTIFICATIONS_KEY = "notifications_enabled"
```

```
}
```

1 Usage

```
fun saveNotificationsSetting(isEnabled: Boolean) {  
    prefs.edit { putBoolean(  
        p0 = NOTIFICATIONS_KEY,  
        p1 = isEnabled) }  
}
```

1 Usage

```
fun getNotificationsSetting(): Boolean {  
    return prefs.getBoolean(  
        p0 = NOTIFICATIONS_KEY,  
        p1 = true)
```

```
}
```

```
}
```

SharedPreferences

Klasa przyjmuje **Context** jako parametr, ponieważ jest on **niezbędny** do uzyskania **dostępu** do **SharedPreferences** specyficznych dla tej aplikacji.

Tworzy lub **otwiera** plik XML o nazwie app_settings.xml. Context.MODE_PRIVATE zapewnia, że **tylko ta aplikacja** może odczytać ten plik.

```
class SettingsManager(context: Context) {  
    2 Usages  
    private val prefs = context.getSharedPreferences(  
        p0 = "app_settings",  
        p1 = Context.MODE_PRIVATE)  
    2 Usages  
    companion object {  
        2 Usages  
        private const val NOTIFICATIONS_KEY = "notifications_enabled"  
    }  
    1 Usage  
    fun saveNotificationsSetting(isEnabled: Boolean) {  
        prefs.edit { putBoolean(  
            p0 = NOTIFICATIONS_KEY,  
            p1 = isEnabled) }  
    }  
    1 Usage  
    fun getNotificationsSetting(): Boolean {  
        return prefs.getBoolean(  
            p0 = NOTIFICATIONS_KEY,  
            p1 = true)  
    }  
}
```

SharedPreferences

Klasa przyjmuje **Context** jako parametr, ponieważ jest on **niezbędny** do uzyskania **dostępu** do **SharedPreferences** specyficznych dla tej aplikacji.

Tworzy lub **otwiera** plik XML o nazwie `app_settings.xml`. `Context.MODE_PRIVATE` zapewnia, że **tylko ta aplikacja** może odczytać ten plik.

```
class SettingsManager(context: Context) {
```

2 Usages

```
private val prefs = context.getSharedPreferences(
```

```
    p0 = "app_settings",
```

```
    p1 = Context.MODE_PRIVATE)
```

2 Usages

```
companion object {
```

2 Usages

```
private const val NOTIFICATIONS_KEY = "notifications_enabled"
```

```
}
```

1 Usage

```
fun saveNotificationsSetting(isEnabled: Boolean) {
```

```
    prefs.edit { putBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = isEnabled) }
```

```
}
```

1 Usage

```
fun getNotificationsSetting(): Boolean {
```

```
    return prefs.getBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = true)
```

```
}
```

```
}
```

Singleton zwracający jedyną instancję `SharedPreferences`.

SharedPreferences

Klasa przyjmuje **Context** jako parametr, ponieważ jest on **niezbędny** do uzyskania **dostępu** do **SharedPreferences** specyficznych dla tej aplikacji.

Tworzy lub **otwiera** plik XML o nazwie `app_settings.xml`. `Context.MODE_PRIVATE` zapewnia, że **tylko ta aplikacja** może odczytać ten plik.

- `MODE_PRIVATE` – dostęp do pliku **tylko z poziomu aplikacji**
- `MODE_WORLD_READABLE` – zezwala **innym aplikacjom** na **odczyt**
- `MODE_WORLD_WRITABLE` – zezwala **innym aplikacjom** na **zapis**

```
class SettingsManager(context: Context) {
```

2 Usages

```
private val prefs = context.getSharedPreferences(
```

```
    p0 = "app_settings",
```

```
    p1 = Context.MODE_PRIVATE)
```

2 Usages

```
companion object {
```

2 Usages

```
private const val NOTIFICATIONS_KEY = "notifications_enabled"
```

```
}
```

1 Usage

```
fun saveNotificationsSetting(isEnabled: Boolean) {
```

```
    prefs.edit { putBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = isEnabled) }
```

```
}
```

1 Usage

```
fun getNotificationsSetting(): Boolean {
```

```
    return prefs.getBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = true)
```

```
}
```

```
}
```

Singleton zwracający jedyną instancję `SharedPreferences`.

SharedPreferences

Klasa przyjmuje **Context** jako parametr, ponieważ jest on **niezbędny** do uzyskania **dostępu** do **SharedPreferences** specyficznych dla tej aplikacji.

Tworzy lub **otwiera** plik XML o nazwie `app_settings.xml`. `Context.MODE_PRIVATE` zapewnia, że **tylko ta aplikacja** może odczytać ten plik. Zwraca obiekt `SharedPreferences`

Zmienna `prefs` przechowuje obiekt typu `SharedPreferences`. Nie jest to surowa zawartość pliku, ale **interfejs**. Dzięki temu obiektowi można w bezpieczny sposób wykonywać operacje **dodania, usunięcia, edycji danych**.

```
class SettingsManager(context: Context) {
```

2 Usages

```
private val prefs = context.getSharedPreferences(
```

```
    p0 = "app_settings",
```

```
    p1 = Context.MODE_PRIVATE)
```

2 Usages

```
companion object {
```

2 Usages

```
private const val NOTIFICATIONS_KEY = "notifications_enabled"
```

```
}
```

1 Usage

```
fun saveNotificationsSetting(isEnabled: Boolean) {
```

```
    prefs.edit { putBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = isEnabled) }
```

```
}
```

1 Usage

```
fun getNotificationsSetting(): Boolean {
```

```
    return prefs.getBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = true)
```

```
}
```

```
}
```

Singleton zwracający jedyną instancję `SharedPreferences`.

SharedPreferences

Klasa przyjmuje **Context** jako parametr, ponieważ jest on **niezbędny** do uzyskania **dostępu** do **SharedPreferences** specyficznych dla tej aplikacji.

Tworzy lub **otwiera** plik XML o nazwie `app_settings.xml`. `Context.MODE_PRIVATE` zapewnia, że **tylko ta aplikacja** może odczytać ten plik. Zwraca obiekt `SharedPreferences`

Zmienna `prefs` przechowuje obiekt typu `SharedPreferences`. Nie jest to surowa zawartość pliku, ale **interfejs**. Dzięki temu obiektowi można w bezpieczny sposób wykonywać operacje **dodania, usunięcia, edycji danych**.

```
class SettingsManager(context: Context) {
```

2 Usages

```
private val prefs = context.getSharedPreferences(
```

```
    p0 = "app_settings",
```

```
    p1 = Context.MODE_PRIVATE)
```

2 Usages

```
companion object {
```

2 Usages

```
private const val NOTIFICATIONS_KEY = "notifications_enabled"
```

```
}
```

1 Usage

```
fun saveNotificationsSetting(isEnabled: Boolean) {
```

```
    prefs.edit { putBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = isEnabled) }
```

```
}
```

1 Usage

```
fun getNotificationsSetting(): Boolean {
```

```
    return prefs.getBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = true)
```

```
}
```

```
}
```

Singleton zwracający jedyną instancję `SharedPreferences`.

Klucz jednoznacznie identyfikujący wartość

SharedPreferences

Klasa przyjmuje **Context** jako parametr, ponieważ jest on **niezbędny** do uzyskania **dostępu** do **SharedPreferences** specyficznych dla tej aplikacji.

Tworzy lub **otwiera** plik XML o nazwie `app_settings.xml`. `Context.MODE_PRIVATE` zapewnia, że **tylko ta aplikacja** może odczytać ten plik. Zwraca obiekt `SharedPreferences`

Zmienna `prefs` przechowuje obiekt typu `SharedPreferences`. Nie jest to surowa zawartość pliku, ale **interfejs**. Dzięki temu obiektowi można w bezpieczny sposób wykonywać operacje **dodania, usunięcia, edycji danych**.

```
class SettingsManager(context: Context) {
```

2 Usages

```
private val prefs = context.getSharedPreferences(
```

```
    p0 = "app_settings",
```

```
    p1 = Context.MODE_PRIVATE)
```

2 Usages

```
companion object {
```

2 Usages

```
private const val NOTIFICATIONS_KEY = "notifications_enabled"
```

```
}
```

1 Usage

```
fun saveNotificationsSetting(isEnabled: Boolean) {
```

```
    prefs.edit { putBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = isEnabled)
```

```
    }
```

1 Usage

```
fun getNotificationsSetting(): Boolean {
```

```
    return prefs.getBoolean(
```

```
        p0 = NOTIFICATIONS_KEY,
```

```
        p1 = true)
```

```
}
```

```
}
```

Singleton zwracający jedyną instancję `SharedPreferences`.

Klucz jednoznacznie identyfikujący wartość

Edycja wartości

Odczyt wartości

SharedPreferences

Pobiera **Context** aplikacji, który jest **niezbędny** do **uzyskania dostępu** do SharedPreferences

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun SettingsScreen() {
    val context = LocalContext.current
    val settingsManager = remember { SettingsManager(context) }

    var isNotificationsEnabled by remember {
        mutableStateOf( value = settingsManager.getNotificationsSetting())
    }

    Scaffold(
        topBar = {...}
    ) { padding ->
        Column(...) {
            Row(...) {
                Text(...)
                Switch(
                    checked = isNotificationsEnabled,
                    onCheckedChange = { newCheckedState ->
                        isNotificationsEnabled = newCheckedState
                        settingsManager.saveNotificationsSetting(
                            isEnabled = newCheckedState
                        )
                    }
                )
            }
        }
    }
}
```

SharedPreferences

Pobiera **Context aplikacji**, który jest **niezbędny do uzyskania dostępu** do SharedPreferences

Context jest tutaj niezbędny ponieważ:

Dostęp do prywatnej przestrzeni: Android **przydziela** każdej aplikacji jej własny, **odizolowany folder** na dane. (/data/data/com.example.twojaaplikacja/) Bez niego, funkcja `getSharedPreferences` nie wiedziałaby, gdzie na dysku ma szukać lub stworzyć plik.

Uprawnienia: System operacyjny zarządza dostępem do plików. Kontekst **jednoznacznie określa** która aplikacja plików, a nie próbuje odczytać danych innej aplikacji.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun SettingsScreen() {
    val context = LocalContext.current
    val settingsManager = remember { SettingsManager(context) }

    var isNotificationsEnabled by remember {
        mutableStateOf( value = settingsManager.getNotificationsSetting())
    }

    Scaffold(
        topBar = {...}
    ) { padding ->
        Column(...) {
            Row(...) {
                Text(...)
                Switch(
                    checked = isNotificationsEnabled,
                    onCheckedChange = { newCheckedState ->
                        isNotificationsEnabled = newCheckedState
                        settingsManager.saveNotificationsSetting(
                            isEnabled = newCheckedState
                        )
                    }
                )
            }
        }
    }
}
```

SharedPreferences

Pobiera **Context** aplikacji, który jest **niezbędny** do **uzyskania dostępu** do SharedPreferences

Użycie **remember** **zapewnia**, że obiekt ten jest tworzony **tylko raz**, a nie **przy każdej rekompozycji**.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun SettingsScreen() {
    val context = LocalContext.current
    val settingsManager = remember { SettingsManager(context) }

    var isNotificationsEnabled by remember {
        mutableStateOf( value = settingsManager.getNotificationsSetting())
    }

    Scaffold(
        topBar = {...}
    ) { padding ->
        Column(...) {
            Row(...) {
                Text(...)
                Switch(
                    checked = isNotificationsEnabled,
                    onCheckedChange = { newCheckedState ->
                        isNotificationsEnabled = newCheckedState
                        settingsManager.saveNotificationsSetting(
                            isEnabled = newCheckedState
                        )
                    }
                )
            }
        }
    }
}
```

SharedPreferences

Pobiera **Context** aplikacji, który jest **niezbędny** do **uzyskania dostępu** do SharedPreferences

Użycie **remember** **zapewnia**, że obiekt ten jest tworzony **tylko raz**, a nie **przy każdej rekompozycji**.

funkcję wywołaną z managera, aby trwale zapisać nową wartość w SharedPreferences. Ta wartość zostanie odczytana przy następnym uruchomieniu ekranu. Operacja jest wykonana synchronicznie.

```
@OptIn( ...markerClass = ExperimentalMaterial3Api::class)
@Composable
fun SettingsScreen() {
    val context = LocalContext.current
    val settingsManager = remember { SettingsManager(context) }

    var isNotificationsEnabled by remember {
        mutableStateOf( value = settingsManager.getNotificationsSetting())
    }

    Scaffold(
        topBar = {...}
    ) { padding ->
        Column(...) {
            Row(...) {
                Text(...)
                Switch(
                    checked = isNotificationsEnabled,
                    onCheckedChange = { newCheckedState ->
                        isNotificationsEnabled = newCheckedState
                        settingsManager.saveNotificationsSetting(
                            isEnabled = newCheckedState
                        )
                    }
                )
            }
        }
    )
}
```

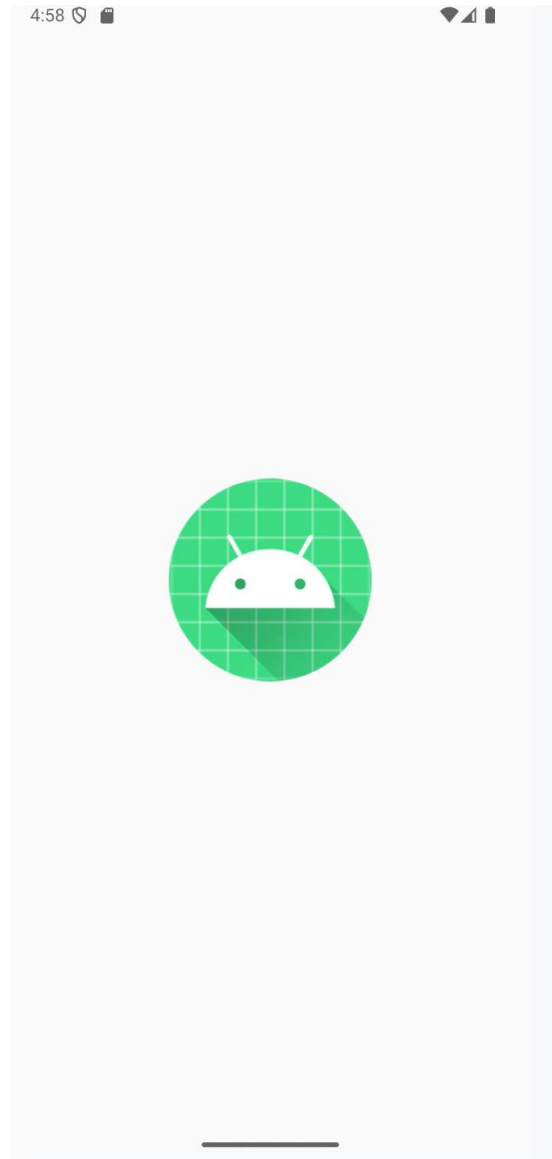
- **Asynchroniczność:** Wszystkie operacje (odczyt/zapis) są **asynchroniczne** i wykorzystują Korutyny oraz Flow, dzięki czemu nie blokują wątku UI.
- **Bezpieczeństwo:** Wbudowana obsługa błędów (przez wyjątki w Flow)
- **Reaktywność:** Odczyt danych odbywa się za pomocą Flow.

1. Preferences DataStore

- Proste dane w formie **klucz-wartość**, tak jak SharedPreferences.
- Zapisywanie niewielkich ilości informacji - ustawień aplikacji i prostych preferencji (np. tryb ciemny, status logowania).
- Prosty w użyciu, bezpośredni następca SharedPreferences.


2. Proto DataStore

- Całe, **silnie typowane obiekty** (instancje klas data class).
- Przechowywanie bardziej złożonych, ustrukturyzowanych danych, jak np. cały obiekt z ustawieniami użytkownika lub cache prostych danych.
- Gwarantuje pełne bezpieczeństwo typów, ale wymaga dodatkowej konfiguracji (pliki .proto).



Preferences DataStore

Tworzy to **właściwość rozszerzającą** dla klasy Context. Oznacza to, że od teraz **każdy** obiekt Context w aplikacji będzie miał **właściwość** o nazwie **dataStore**



```
private val Context.dataStore:
    DataStore<Preferences> by preferencesDataStore(name = "user_pref")
2 Usages
object UserPrefsKeys {
    2 Usages
    val USER_NAME = stringPreferencesKey( name = "user_name")
}
```

Preferences DataStore

Tworzy to **właściwość rozszerzającą** dla klasy Context. Oznacza to, że od teraz **każdy** obiekt Context w aplikacji będzie miał **właściwość** o nazwie **dataStore**

```
private val Context.dataStore:  
    DataStore<Preferences> by preferencesDataStore(name = "user_pref")  
2 Usages  
object UserPrefsKeys {  
    2 Usages  
    val USER_NAME = stringPreferencesKey( name = "user_name")
```

Definiuje klucz wykorzystywany
do **zapisu i odczytu** wartości

Preferences DataStore

Tworzy to **właściwość rozszerzającą** dla klasy Context. Oznacza to, że od teraz **każdy** obiekt Context w aplikacji będzie miał **właściwość** o nazwie **dataStore**

```
private val Context.dataStore:  
    DataStore<Preferences> by preferencesDataStore(name = "user_pref")  
2 Usages  
object UserPrefsKeys {  
    2 Usages  
    val USER_NAME = stringPreferencesKey( name = "user_name")
```

Definiuje klucz wykorzystywany
do zapisu i odczytu wartości

```
class UserPreferencesManager(private val context: Context) {  
    1 Usage  
    val userNameFlow: Flow<String> = context.dataStore.data  
        .map { preferences ->  
            preferences[UserPrefsKeys.USER_NAME] ?: ""  
        }  
    1 Usage  
    suspend fun saveUserName(name: String) {  
        context.dataStore.edit { preferences ->  
            preferences[UserPrefsKeys.USER_NAME] = name  
        }  
    }  
}
```

Klasa **pośrednicząca** pomiędzy
DataStore i ViewModel

Preferences DataStore

Tworzy to **właściwość rozszerzającą** dla klasy Context. Oznacza to, że od teraz **każdy** obiekt **Context** w aplikacji będzie miał **właściwość** o nazwie **dataStore**

```
private val Context.dataStore:
    DataStore<Preferences> by preferencesDataStore(name = "user_pref")
2 Usages
object UserPrefsKeys {
    2 Usages
    val USER_NAME = stringPreferencesKey( name = "user_name")
}

class UserPreferencesManager(private val context: Context) {
    1 Usage
    val userNameFlow: Flow<String> = context.dataStore.data
        .map { preferences ->
            preferences[UserPrefsKeys.USER_NAME] ?: ""
        }
    1 Usage
    suspend fun saveUserName(name: String) {
        context.dataStore.edit { preferences ->
            preferences[UserPrefsKeys.USER_NAME] = name
        }
    }
}
```

Definiuje klucz wykorzystywany
do **zapisu i odczytu** wartości

Klasa **pośrednicząca** pomiędzy
DataStore i ViewModel

Daje dostęp do Flow, który
emituje **pełen** zestaw
preferencji za każdym razem,
gdy **jakakolwiek wartość** w
DataStore się **zmieni**.

Preferences DataStore

Tworzy to **właściwość rozszerzającą** dla klasy Context. Oznacza to, że od teraz **każdy** obiekt Context w aplikacji będzie miał **właściwość** o nazwie **dataStore**

```
private val Context.dataStore:
    DataStore<Preferences> by preferencesDataStore(name = "user_pref")
2 Usages
object UserPrefsKeys {
    2 Usages
    val USER_NAME = stringPreferencesKey( name = "user_name")
}

class UserPreferencesManager(private val context: Context) {
    1 Usage
    val userNameFlow: Flow<String> = context.dataStore.data
        .map { preferences ->
            preferences[UserPrefsKeys.USER_NAME] ?: ""
        }
    1 Usage
    suspend fun saveUserName(name: String) {
        context.dataStore.edit { preferences ->
            preferences[UserPrefsKeys.USER_NAME] = name
        }
    }
}
```

Definiuje klucz wykorzystywany do zapisu i odczytu wartości

Klasa **pośrednicząca** pomiędzy DataStore i ViewModel

Daje dostęp do Flow, który emituje **pełen** zestaw **preferencji** za każdym razem, gdy **jakakolwiek wartość** w DataStore się **zmieni**.

Transformuje strumień Preferences w **prostszy** strumień Flow<String>.

Preferences DataStore

Tworzy to **właściwość rozszerzającą** dla klasy Context. Oznacza to, że od teraz **każdy** obiekt Context w aplikacji będzie miał **właściwość** o nazwie **dataStore**

```
private val Context.dataStore:  
    DataStore<Preferences> by preferencesDataStore(name = "user_prefs")  
2 Usages  
object UserPrefsKeys {  
    2 Usages  
    val USER_NAME = stringPreferencesKey( name = "user_name")
```

Definiuje klucz wykorzystywany do zapisu i odczytu wartości

Klasa **pośrednicząca** pomiędzy DataStore i ViewModel

Daje dostęp do Flow, który emituje **pełen** zestaw **preferencji** za każdym razem, gdy **jakakolwiek wartość** w DataStore się **zmieni**.

```
class UserPreferencesManager(private val context: Context) {  
    1 Usage  
    val userNameFlow: Flow<String> = context.dataStore.data  
        .map { preferences ->  
            preferences[UserPrefsKeys.USER_NAME] ?: ""  
        }  
    1 Usage  
    suspend fun saveUserName(name: String) {  
        context.dataStore.edit { preferences  
            preferences[UserPrefsKeys.USER_NAME] = name  
        }  
    }  
}
```

Transformuje strumień Preferences w **prostszy** strumień Flow<String>.

Operacja transakcyjna to sekwencja działań, która jest **traktowana jako jedna, niepodzielna całość**.

Uruchamia **transakcyjną** operację zapisu.

Preferences DataStore

```
class UserViewModel(private val prefsManager: UserPreferencesManager) : ViewModel() {  
    3 Usages  
    var textFieldValue by mutableStateOf( value = "")  
        private set  
    1 Usage  
    val savedUserName: StateFlow<String> = prefsManager.userNameFlow  
        .stateIn(  
        scope = viewModelScope,  
        started = SharingStarted.WhileSubscribed( stopTimeoutMillis = 5000),  
        initialValue = ""  
    )  
    1 Usage  
    fun onTextFieldValueChanged(newValue: String) {  
        textFieldValue = newValue  
    }  
    1 Usage  
    fun onSaveClicked() {  
        viewModelScope.launch {  
            prefsManager.saveUserName( name = textFieldValue)  
        }  
    }  
}
```

Konwersja na
gorący strumień

zwraca **zimny** Flow, który **emituje**
zapisaną **nazwę użytkownika** za
każdym razem, **gdy się ona zmieni**.