BCS 371 Mobile Application Development I

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today’s Lecture

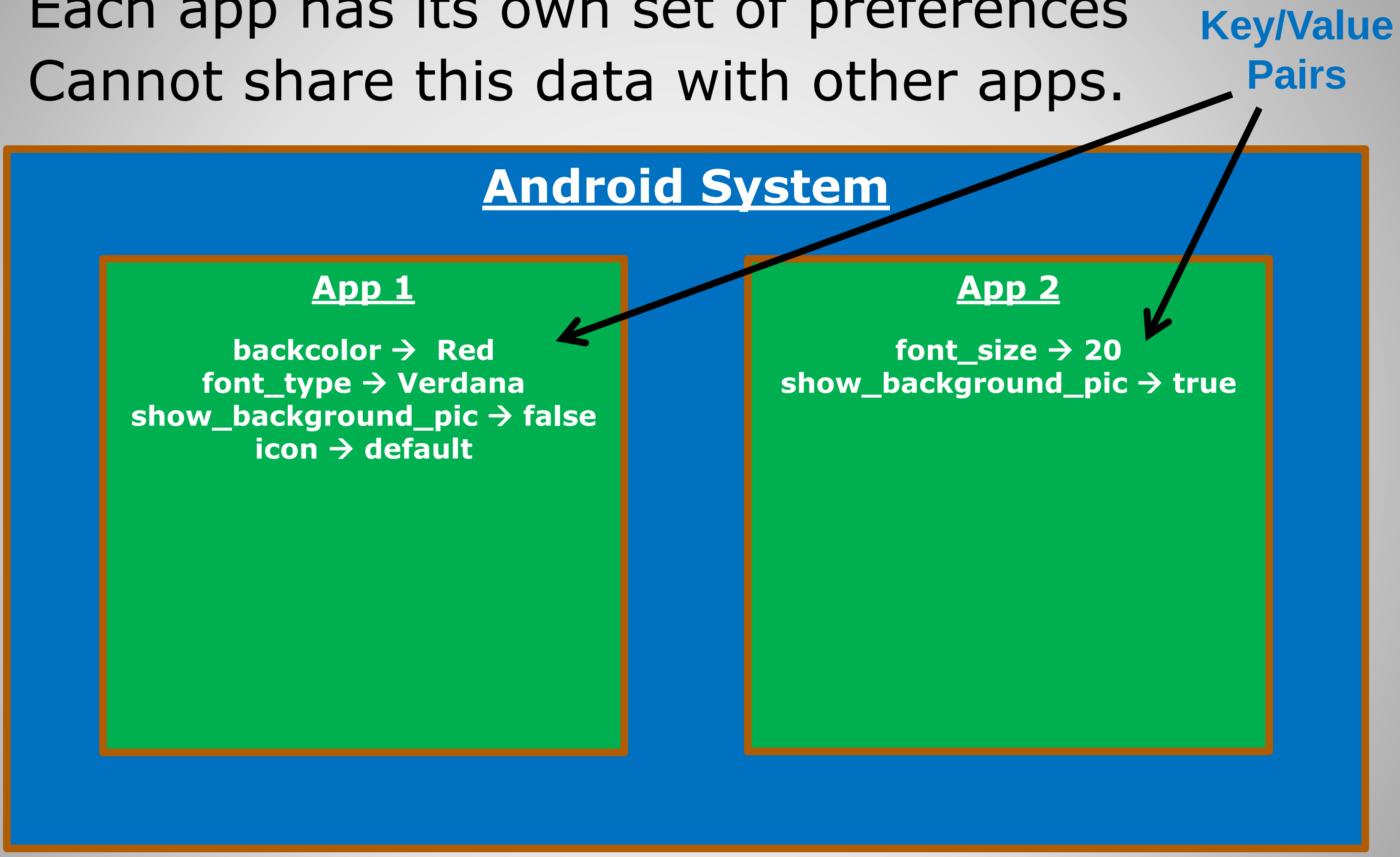
- Preferences DataStore

Preferences DataStore

- Stores **user settings** as **key/value pairs**.
- Can be **persisted across sessions**. Even if app is killed values will be there when you restart it.
- This data can be shared by application components within the **SAME** application (data is private to the application).
- Data is **NOT** available to other applications.
- For example…

Preferences DataStore

- Each app has its own set of preferences
- Cannot share this data with other apps.



Shared Preferences

<u>Gradle Depenency – Preferences DataStore</u>

- Include the following Gradle dependency:

```
implementation("androidx.datastore:datastore-preferences:1.0.0")
```

- **<u>MAKE SURE YOU USE THE ABOVE VERSION OF THE DEPENDENCY</u>** (ran into issues with the sample code on upcoming slides when using later versions of it).

Note: Make sure to sync the Gradle file after adding the dependency.

Gradle Dependency – Preferences DataStore

Setting Up Preference DataStore - Overview

1. Create a class to interact with Preferences DataStore
2. Setup StateFlow in ViewModel class.
3. Manipulate preferences in a screen composable function.

Setup Preferences DataStore - Overview

1. Create a Class to Interact with Preferences DataStore (Imports)

- Here are the imports to use for the MyPreferences class (MyPreferences class is defined on the next slide).

```
import android.content.Context
import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.MutablePreferences
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.core.booleanPreferencesKey
import androidx.datastore.preferences.core.edit
import androidx.datastore.preferences.preferencesDataStore
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map

import com.example.testpreferencesdatastore.MyPreferences.PreferenceKeys.showBackgroundPic
```

Important!!! Should replace com.example.testpreferencesdatastore with the name or your package here

1. Create a Class to Interact with Preferences DataStore (Imports)

1. Create a Class to Interact with Preferences DataStore

- This class will interact directly with the Preferences DataStore
- Add variables to PreferenceKeys for each item you want to store.
- Add a pair of update/watch methods for each preference key.

```
val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = "settings")
class MyPreferences (val context: Context) {
    private object PreferenceKeys {
        val showBackgroundPic : Preferences.Key<Boolean> = booleanPreferencesKey("showBackgroundPic")
    }

    suspend fun updateShowPic(newShowBackgroundPicValue: Boolean) =
        context.dataStore.edit { preferences: MutablePreferences ->
            preferences[showBackgroundPic] = newShowBackgroundPicValue
    }

    fun watchShowPic(): Flow<Boolean> = context.dataStore.data.map { preferences: Preferences ->
        return@map preferences[showBackgroundPic] ?: false
    }
}
```

Initialize Preferences DataStore

Set key to showBackgroundPic

This code updates the value for showBackgroundPic

watchShowPic returns a Flow ("cold" flow)

Default value is false

return@map tells it to return to map on the previous line

1. Create a Class to Interact with Preferences DataStore

2. Setup StateFlow in ViewModel

- Setup the StateFlow in the ViewModel.

```
class MainScreenViewModel(application:Application) : AndroidViewModel(application) {

    private val myPreferences: MyPreferences
    val showBackgroundPicStateFlow: StateFlow<Boolean>

    init {
        val context: Context = getApplication<Application>().applicationContext
        myPreferences = MyPreferences(context)
        showBackgroundPicStateFlow =
            myPreferences.watchShowPic().stateIn(viewModelScope, SharingStarted.Lazily, false)
    }

    fun toggleShowBackgroundPic() {
        viewModelScope.launch {
            val newShowPicValue: Boolean = !showBackgroundPicStateFlow.value
            myPreferences.updateShowPic(newShowPicValue)
        }
    }
}
```

Private variable to manipulate preferences inside the ViewModel

Asynchronous flow of data. This StateFlow will be observed by a composable function.

Set initial value in preference to false

stateIn converts a "cold" flow (Flow type) to a "hot" flow (StateFlow type)

toggleShowBackgroundPic will toggle the Boolean show background pic value and then save that new value to Preferences Datastore

2. Setup StateFlow in ViewModel

3. Manipulate Preferences in a Screen Composable Function

- Sample code to get/set preference values.

```
@Composable
fun MainScreen(modifier: Modifier) {
    val viewModel = MainScreenViewModel(LocalContext.current.applicationContext as Application)
    val showPicBackgroundPicLocal: Boolean by viewModel.showBackgroundPicStateFlow.collectAsState()

    Column(modifier) {
        Button(
            onClick = {
                viewModel.toggleShowBackgroundPic()
            }
        )
        {
            Text(text = "Toggle Show Background Pic")
        }

        Text("Show Background Pic Value = " + showPicBackgroundPicLocal.toString())
    }
}
```

Pass the Application instance into the ViewModel

collectAsState collects values from the StateFlow. Every time a new value is put in the StateFlow the showBackbroundPickLocal variable will be updated with the new value. The screen will then be recomposed with the new value.

3. Manipulate Preferences in a Screen Composable Function

- End of Slides

End of Slides