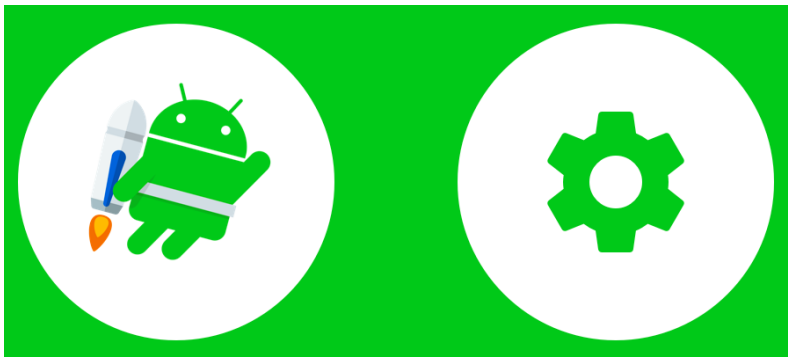
Mobile Applications
CSCI 448
Lecture 31

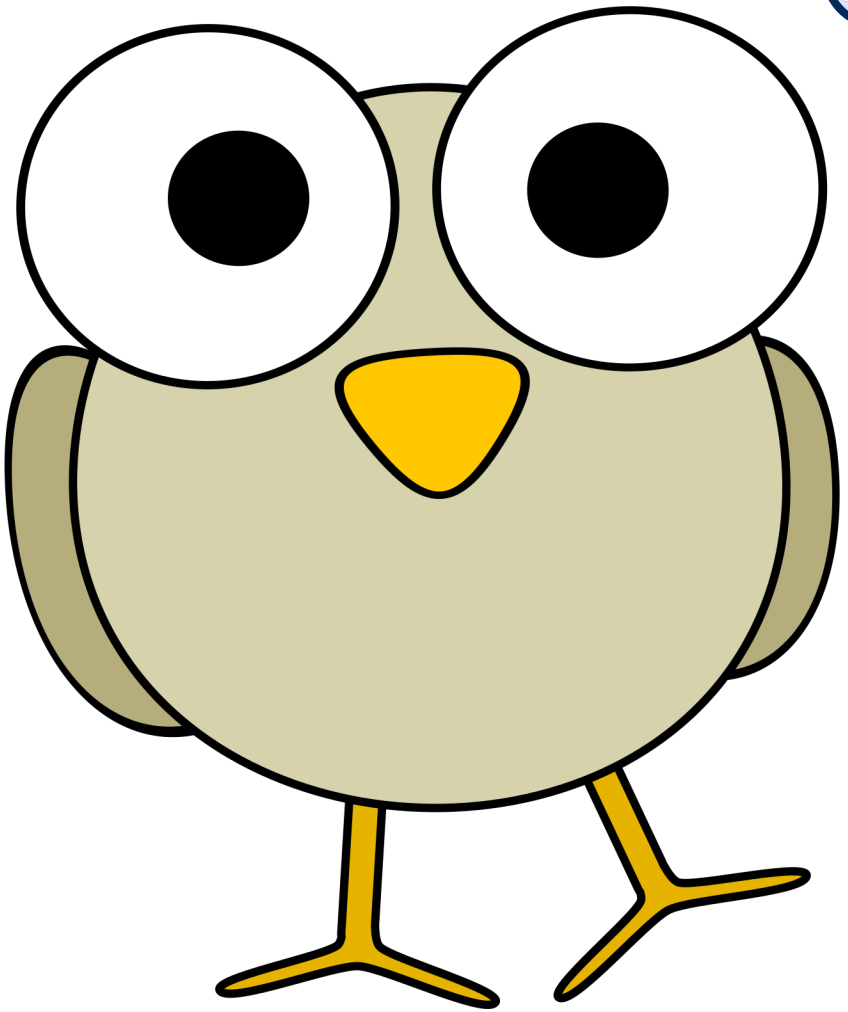
Datastore



Previously in CSCI 448

- Paging
  - Break large list into smaller sublists
  - Sublists sent along a Flow stream
  - When nearing end of current sublist, request next sublist in sequence

Questions?



Learning Outcomes For Today

- Discuss where application preferences should be stored
- Explain how to create the preferences option screen
- Implement an app that sets and retrieves preferences set by the user

On Tap For Today

- Datastore
  - Model
  - View
  - ViewModel
- Practice!

On Tap For Today

- Datastore
  - Model
  - View
  - ViewModel
- Practice!

Datastore

- Add dependency to build.gradle

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

Datastore

- Built upon coroutines and **`Flow`**

- Replaces previous SharedPreferences API built upon suspend functions and **`LiveData`**

On Tap For Today

- Datastore
  - Model
  - View
  - ViewModel
- Practice!

DataStore<Preferences>

- Map of key-value pairs
  - No predefined schema
- Create keys based on type of value

```
val KEY_INT_COUNTER = intPreferencesKey("my_counter_key")
val KEY_FLOAT_DISTANCE = floatPreferencesKey("distance_key")
// etc
```

- See https://developer.android.com/reference/kotlin/androidx/datastore/preferences/core/package-summary

DataStore<Preferences>

- Members
  - `data: Flow<Preferences>`
    - Returns contents of the preferences map as a flow
  - `edit(transform: (MutablePreferences) -> Unit)`
    - Allows values within the map to be modified
- Will see usage in ViewModel

On Tap For Today

- Datastore
  - Model
  - View
  - ViewModel
- Practice!

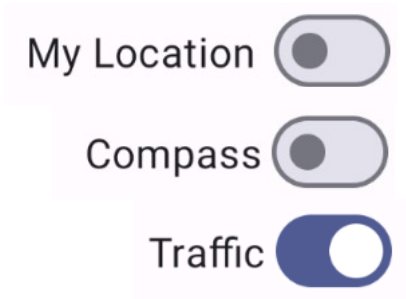
Datastore View

- Need to create composables to display current values and edit the values

Datastore Screen

```
@Composable
fun MyDatastoreScreen(dataPreferenceMap: Map<String, Pair<T, (T) -> Unit>>)
{
  Column {
    dataPreferenceMap.forEach { (dataName, data, dataEdit) ->
      DatastoreRow(dataName, data, dataEdit)
    }
  }
}

@Composable
fun DatastoreRow(dataName: String, data: T, dataEdit: (T) -> Unit) {
  Row {
       Text( text = dataName )
       Switch(
         checked = data,
         onCheckedChange = { dataEdit(it) }
       )
  }
}
```



On Tap For Today

- Datastore
  - Model
  - View
  - ViewModel
- Practice!

Getting the Datastore

- Create a top-level extension function using property delegate

```
val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = "settings")
```

- Access the datastore via a Context

```
fun task(context: Context) {
  context.dataStore ...
}
```

Retrieving a Value

```
val data: Int = 0
val dataKey = intPreferencesKey("data_key")
val dataFlow: Flow<Int> = context.dataStore.data
  .map { preferences ->
    // no type safety!
    preferences[dataKey] ?: 0
  }

// else where
coroutine.launch {
  dataFlow.collect { prefData ->
     data = prefData
  }
}
```

Setting a Value

```
val dataKey = intPreferencesKey("data_key")
context.dataStore.data.edit { preferences ->
  val currentValue = preferences[dataKey] ?: 0
  preferences[dataKey] = currentValue + 1
}
```

Wrap Inside Manager Class

```
class DataStoreManager(private val context: Context) {
  companion object {
    private const val DATA_STORE_NAME = "preferences"
    private val Context.dataStore by preferencesDataStore(
        name = DATA_STORE_NAME
    )
    private val DATA_KEY = intPreferencesKey("data_key")
  }

  val dataFlow: Flow<Int> = context.dataStore.data
    .map { preferences ->
      preferences[DATA_KEY] ?: 0
    }
  suspend fun setData(newValue: Int) {
    context.dataStore.edit { preferences ->
      preferences[DATA_KEY] = newValue
    }
  }
}
```

Use Manager

```
class MainActivity() {
  override fun onCreate() {
    ...
    val dataStoreManager = DataStoreManager(this)
    setContent {
      MyScreen(dataStoreManager)
    }
  }
}
```

Work With DataStore

```
@Composable
fun MyScreen(dataStoreManager: DataStoreManager) {
  val coroutineScope = rememberCoroutineScope()
  val dataState = dataStoreManager.dataFlow.collectAsStateWithLifecycle()
  Row {
    Text( text = "Data is ${dataState.value}" )
    Button(
      onClick = {
        coroutineScope.launch {
          dataStoreManager.setData( dataState.value + 1 )
        }
      }
    ) {
      Text( text = "Increment!" )
    }
  }
}
```

On Tap For Today

- Datastore
  - Model
  - View
  - ViewModel
- Practice!

Next Time

- Lab11A
  - Creating a DataStore to store Map settings