4.1 Saving Data

Objectives

- Saving key-value pairs of simple data types in a shared preferences file
- Saving arbitrary files in Android's file system
- Using databases managed by SQLite
- Explain app installation location options

Saving Data

Share Preferences	Data Files	SQLite
Private or Public	Private or Public	Private
Key-value pair	Data file; text, sound, images, and etc	Repeating and structured data

Shared Preferences

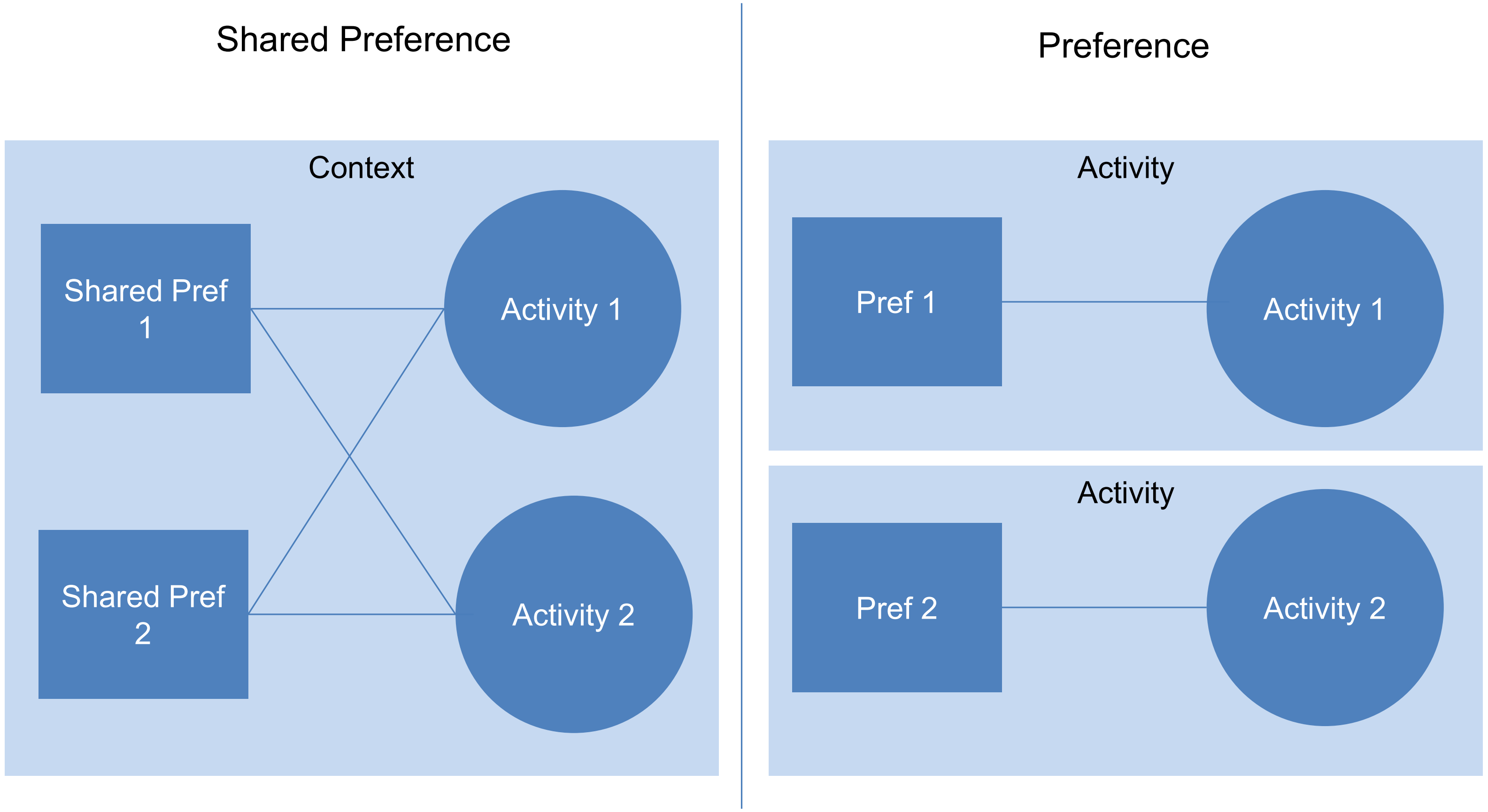
- Stores private primitive data in key-value pairs
- Primitive data: boolean, float, int, long, and string
- Data will persist across session, even if an app is killed
- PreferenceActivity class: provides framework to create user preferences

https://developer.android.com/training/data-storage/shared-preferences

Shared Preferences

Method	getSharedPreferences	getPreferences
Number of shared preference file	Multiple	Single
Access level	Context	Activity

Shared Preferences



Getting Shared Preferences

This code accesses the shared preferences file that's identified by the resource string R.string.preference_file_key and opens it using the private mode so the file is accessible by only your app

```
val sharedPref = activity?.getSharedPreferences(
                              getString(R.string.preference_file_key),
                              Context.MODE_PRIVATE)
```

When naming your shared preference files, you should use a name that's uniquely identifiable to your app. An easy way to do this is prefix the file name with your application ID.

For example: "`com.example.myapp.PREFERENCE_FILE_KEY`"

Getting Preferences

If you need just one shared preference file for your activity, you can use the getPreferences() method

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE)
```

Write to shared preferences

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return

with (sharedPref.edit()) {
      putInt(getString(R.string.saved_high_score_key), newHighScore)
      commit()
}
```

You can call apply() or commit() to save the changes. The apply() writes the updates to disk asynchronously. The commit() writes the data to disk synchronously, you should avoid calling it from your main thread because it could pause your UI rendering.

Read from shared preferences

- To read values call methods such as `getBoolean()`, `getInt()` and `getString()`
- You must provide a key for the value you want

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return

val defaultValue = resources.getInteger(R.integer.saved_high_score_default_key)

val highScore = sharedPref.getInt(getString(R.string.saved_high_score_key), defaultValue)
```

Question?

1. What is the main difference between the getPreferences and getSharedPreference methods?

2. Can a SharedPreferences file accessed by:
   a. multiple Activities?
   b. multiple apps?

Saving Files

- Android uses the File APIs for reading and writing large amount of data in a start-to-finish order without skipping around
- It is good for image files or anything exchanged over a network
- The exact location of the where your files can be saved might vary across devices

https://developer.android.com/training/data-storage/files

Choose internal or external storage

Characteristics	Internal	External
Removeable	No	Yes (Possible)
Availability	Always	Not always
Accessibility	Your app	World-readable

Save a file on internal storage

- An app's internal storage directory is specified by the app's package name
- No permission is required to perform read/write operations
- Use `getFreeSpace()` or `getTotalSpace()` to check available internal storage space to avoid IO Error

Create and write a file

```
// Create a new file
val file = File(context.filesDir, filename)

// Write to a file
file.bufferedWriter().use {
         out -> out.write(fileContent)
}
```

External Storage

- Removable storage media (such as an SD card) or an internal (non-removable) storage

- Files saved to the external storage are:
  - <u>world-readable</u>
  - can be modified by the user
  - can become unavailable if the user unmount or remove the storage (SD Card)

External Storage

- Request permission:

```
<manifest ...>
    <!-- If you need to modify files in external storage, request
         WRITE_EXTERNAL_STORAGE instead. -->
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
                     android:maxSdkVersion="18" />
</manifest>
```

- Two types of files:

Characteristics	**Public**	**Private**
Accessibility	All apps	Specific app*
State after uninstallation of app	Available	Deleted

*Files stored in an *app-specific directory*—accessed using Context.getExternalFilesDir().

Verify external storage

```
/* Checks if external storage is available to at least read */
fun isExternalStorageReadable(): Boolean {
     return Environment.getExternalStorageState() in
        setOf(Environment.MEDIA_MOUNTED, Environment.MEDIA_MOUNTED_READ_ONLY)
}
```

Saving Private File

```kotlin
fun getPrivateAlbumStorageDir(context: Context, albumName: String): File? {
    // Get the directory for the app's private pictures directory.
    val file = File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES), albumName)

    if (!file?.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created")
    }
    return file
}
```

Note: Other directories include DIRECTORY_MUSIC, DIRECTORY_PICTURES, DIRECTORY_RINGTONES and etc

Storage Guide (New)

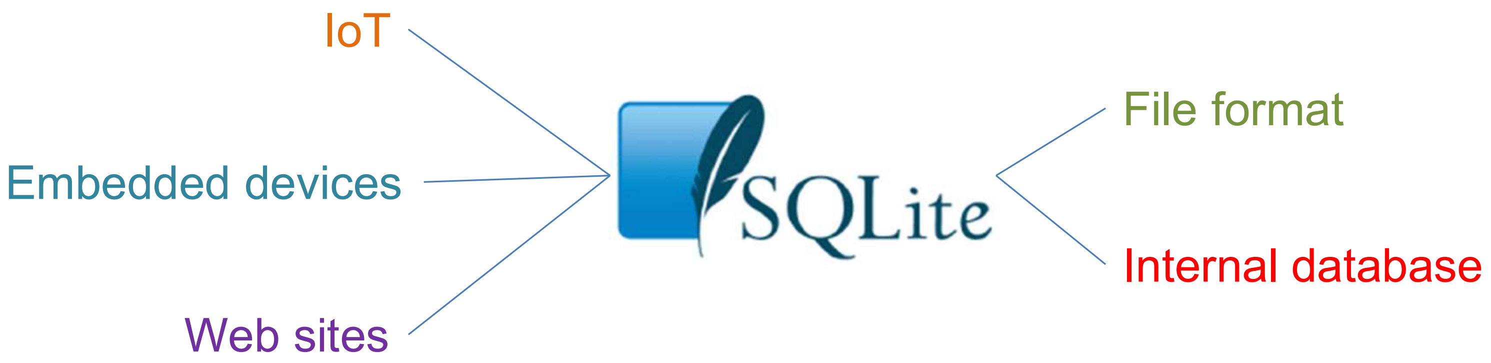
Type	Content	Other apps can access?	Files removed on app uninstall?
App-specific files	Files means for your app's use only	No	Yes
Media	Shareable media files (images, audio files, videos)	Yes (with permission)	No
Documents and other files	Other types of shareable content, including downloaded files	Yes	No
App preferences	Key-value pairs	No	Yes
Database	Structure data	No	Yes

Question?

1. Among internal and external storage, which one is suitable for files that you want to allow user to access with a computer?

2. All Android devices offered build-in non-volatile memory that is non-removeable but can be accessed with a computer. Is this part classified as internal storage?

What is SQLite?



Find out more from www.sqlite.org

SQLite Data Type

Type	Definition
NULL	The value is a NULL value.
INTEGER	The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
REAL	The value is a floating point value, stored as an 8-byte
TEXT	The value is a text string
BLOB	Binary large object. Suitable for images, audio or other media objects

SQLite Database

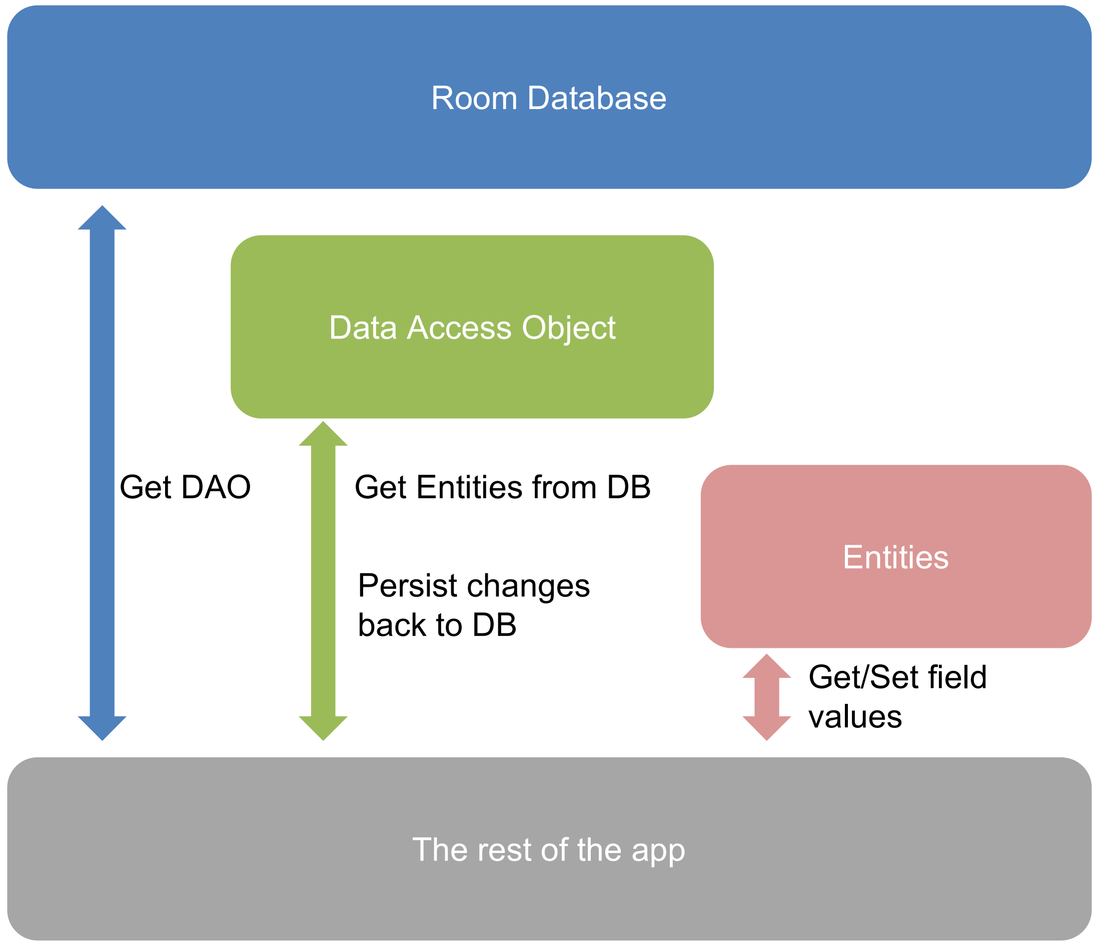
- SQLite can be accessed using the Room persistence library
- 3 components:
  - Database: access point to DB
  - Entity: table
  - Data access object (DAO): methods to access DB

https://developer.android.com/training/data-storage/room

Using the Room

Dependencies for Room should be included in the `build.grandle` file

```
dependencies {
  def room_version = "2.2.0"

  implementation "androidx.room:room-runtime:$room_version“

  // For Kotlin use kapt instead of annotationProcessor
  annotationProcessor "androidx.room:room-compiler:$room_version"
  // optional - Kotlin Extensions and Coroutines support for Room
  implementation "androidx.room:room-ktx:$room_version"

  // optional - RxJava support for Room
  implementation "androidx.room:room-rxjava2:$room_version"

  // optional - Guava support for Room, including Optional and ListenableFuture
  implementation "androidx.room:room-guava:$room_version"

  // Test helpers
  testImplementation "androidx.room:room-testing:$room_version"
}
```


Room Database
Data Access Object
Get DAO
Get Entities from DB
Persist changes back to DB
Entities
Get/Set field values
The rest of the app

Entity

Step 1: Create an entity class for each table

```
@Entity
data class User(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```

A nullable attribute

Entity

```
// An entity with a composite primary key
@Entity(tableName = "users" primaryKeys = arrayOf("firstName", "lastName"))



// Define foreign key constrains between entities
@Entity(foreignKeys = arrayOf(ForeignKey(
            entity = User::class,
            parentColumns = arrayOf("id"),
            childColumns = arrayOf("user_id"))
       )
)
```

https://developer.android.com/training/data-storage/room/relationships

DAO

Step 2: Create a DAO

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

Variable number of arguments

AppDatabase Abstract Class

Step 3: Create an AppDatabase abstract class

```
@Database(entities = arrayOf(User::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

Instance

Step 4: Create a database instance

```
val db = Room.databaseBuilder(
            applicationContext,
            AppDatabase::class.java, "database-name"
        ).build()
```

https://developer.android.com/training/data-storage/room/defining-data

Using Room

- Don’t use Room for database access on the UI thread. Asynchronously run the query on a background thread

- More information about background thread will be covered in Chapter 4.2

- Example of using Room:
  https://github.com/seekweeteck/Demo4.1

Question?

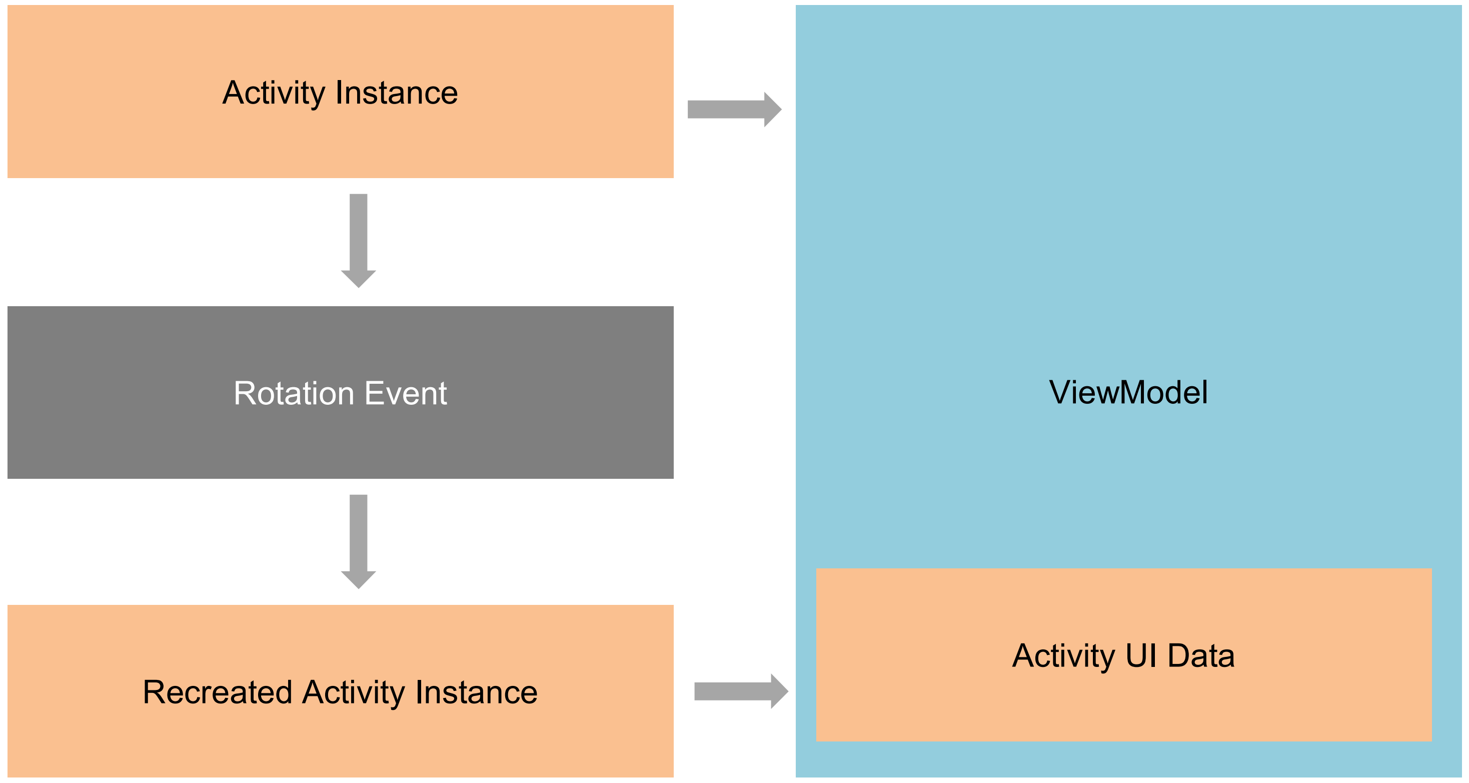
1. Explain the purpose of the following objects:
   a. Entity
   b. Data Access Object (DAO)

ViewModel and Repository

Data Binding Library

- A support library
- Allows you to bind UI components in layout to data source
- Implementation:
  - –ViewModel class
  - –It holds UI data

Traditional vs ViewModel



Traditional vs ViewModel

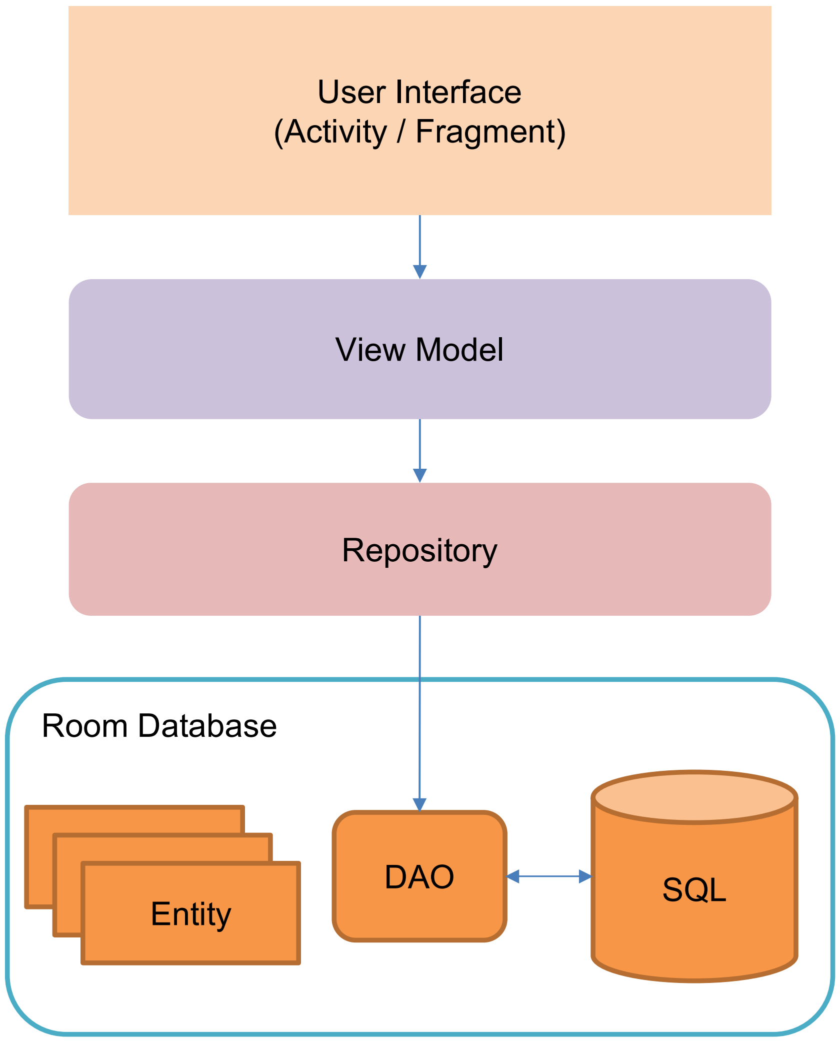
Traditional

```
// Program Code
TextView textView = findViewById(R.id.sample_text);
textView.setText(“Your text here”);
```

ViewModel

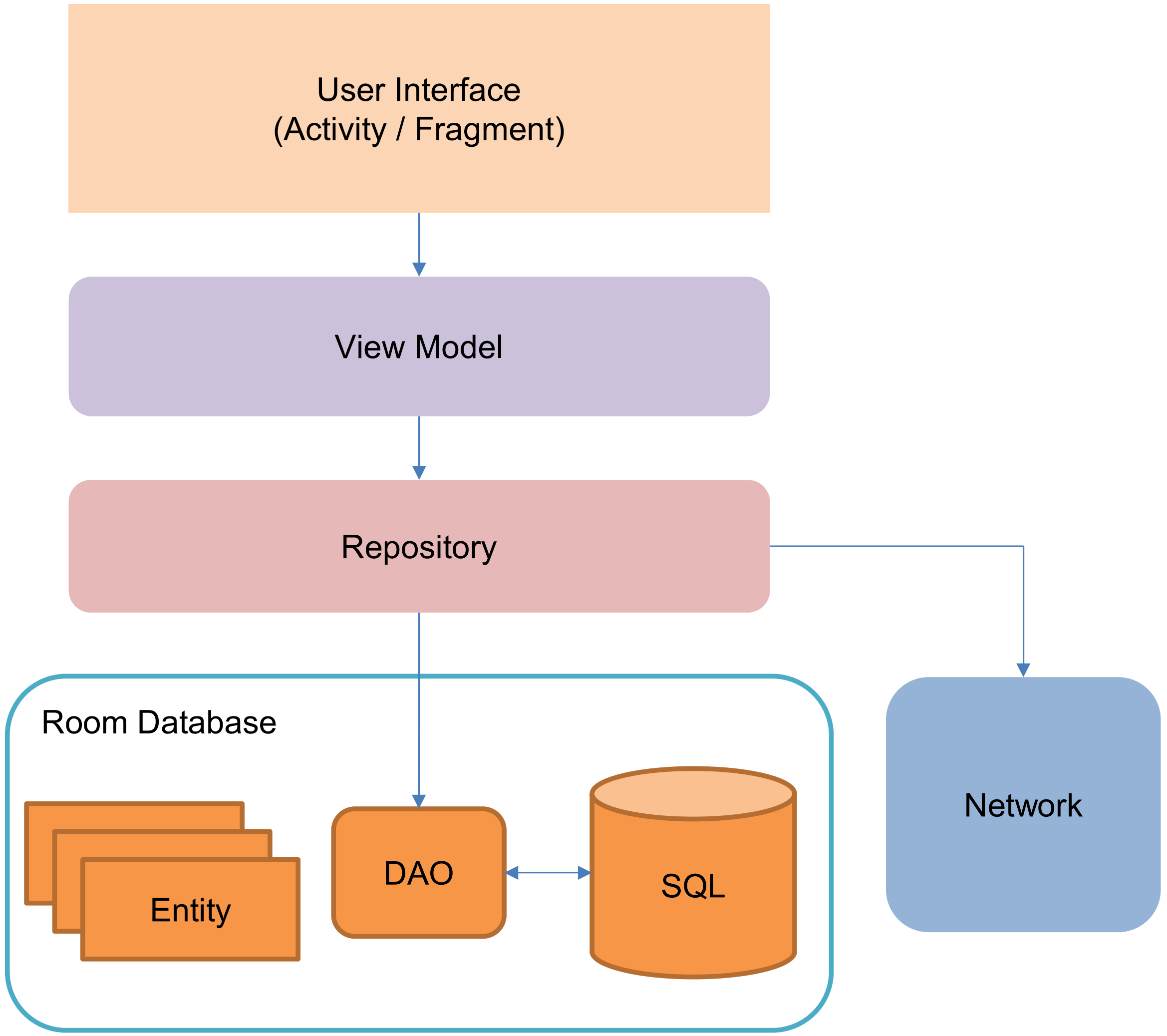
```
// Program Code
TextView textView = findViewById(R.id.sample_text);
textView.setText(viewModel.getUserName());
```

```
// Layout File
<TextView
    android:text="@{viewmodel.userName}" />
```


User Interface
(Activity / Fragment)
View Model
Repository
Room Database
Entity
DAO
SQL

Repository

- It provides access to multiple data sources
- Manages query threads and allows you to use multiple backends
- E.g. A Repository implements the logic for deciding whether to fetch data from a network or use results cached in a local database


User Interface
(Activity / Fragment)
View Model
Repository
Room Database
Entity
DAO
SQL
Network

Coroutine

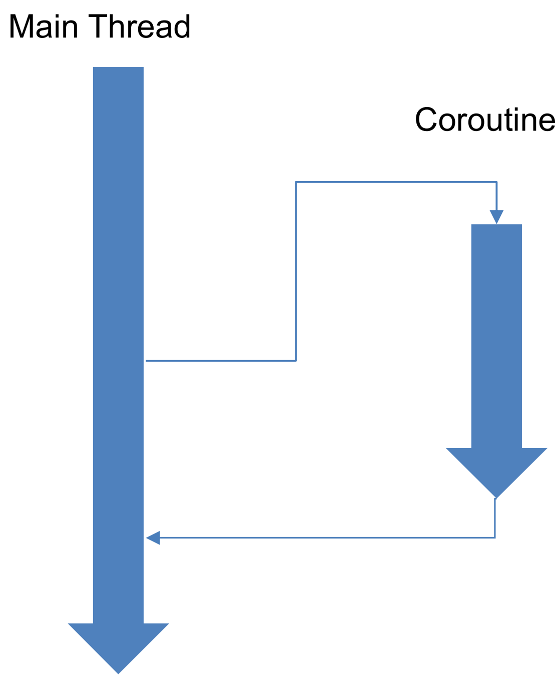
- Asynchronous programming

- Manage long-running tasks that might otherwise block the main thread and cause your app to freeze

- Providing *main-safety*, or safely calling network or disk operations from the main thread

https://kotlinlang.org/docs/reference/coroutines-overview.html

Coroutine



Coroutine

- Two ways to start coroutine:
  - **launch** starts a new coroutine and doesn't return the result to the caller. Any work that is considered "fire and forget" can be started using launch
  - **async** starts a new coroutine and allows you to return a result with a suspend function called await
- In Android, works well with the ViewModel

https://developer.android.com/kotlin/coroutines

Example

```
fun main() {
    GlobalScope.launch { // launch a new coroutine in background and continue
        delay(1000L)    // non-blocking delay for 1 second (default time unit is ms)
        println("World!")// print after delay
    }
    println("Hello,")    // main thread continues while coroutine is delayed
    Thread.sleep(2000L)  // block main thread for 2 seconds to keep JVM alive
}
```

Output:

```
Hello,
World!
```

Question?

An online property agency, myhome.com.my, is looking for a mobile solution to make its service accessible to more agents. The company would like to create a mobile app which allows its agents to upload media files, post comments, and record locations of properties. Explain TWO techniques to be used to implement the above-mentioned system.

App Install Location

- API Level 8 onward, you can install app on external storage

- Manifest attribute
  - android:installLocation=“preferExternal”
  - android:installLocation = “auto”

- Default location: internal

App Install Location

- App installed on external storage:
  - No effect on performance
  - Only APK file is saved on external storage. Private user data, database, native code, and etc are on internal device memory
  - Container in which an app is stored is encrypted; an app on SD card works for only one device
  - User can move an app to internal storage

App Install Location

- App that must be installed on internal storage:
  - Services
  - Alarm Services
  - Input Method Engines
  - Live Wallpapers
  - App Widgets
  - Account Managers
  - Sync Adapters
  - Device Administrators
  - Broadcast Receiver

Question?

Your team has been assigned to create a mobile app which allows a company to distribute the latest product information to all salesperson. The app shall check for any new products information from a server at regular interval and download them to the storage of mobile devices for offline access. Product information is in the form of PDF file, which could be a few megabytes in size. Among internal and external storage, which storage option is suitable for the above-mentioned system? Justify your selection.