Room and LiveData

What is Room?

Room is a **persistence library** provided by Android Jetpack. It provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.

Why use Room?

- Compile-time SQL query verification.
- Easy database migrations.
- Works seamlessly with LiveData and Kotlin Coroutines.
- Reduces boilerplate with annotations.

Key Components of Room:

1. Entity – Represents a table in the database.

```
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "user_table")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String
)
```

2. DAO (Data Access Object) – Contains methods to access the database.

```
import androidx.lifecycle.LiveData
import androidx.room.*

@Dao
interface UserDao {
    @Query("SELECT * FROM user_table")
    fun getAllUsers(): LiveData<List<User>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(user: User)
```

```
    @Delete
    suspend fun deleteUser(user: User)
}
```

3. Database – The database holder.

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [User::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao

    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "user_database"
                ).build().also { INSTANCE = it }
            }
        }
    }
}
```

What is LiveData?

LiveData is a lifecycle-aware data holder class. It is part of Android Architecture Components and is designed to allow data to be **observable** and **reactive**.

Why use LiveData?

- Automatically respects the lifecycle of app components (e.g. Activity, Fragment).
- Reduces memory leaks.
- Ensures UI is always up-to-date with the latest data.

LiveData Example in ViewModel:

```
import androidx.lifecycle.*

class UserViewModel(private val userDao: UserDao) : ViewModel() {
    val allUsers: LiveData<List<User>> = userDao.getAllUsers()

    fun addUser(name: String) {
        viewModelScope.launch {
            userDao.insertUser(User(name = name))
        }
    }

    fun deleteUser(user: User) {
        viewModelScope.launch {
            userDao.deleteUser(user)
        }
    }
}
```

Observe LiveData in UI (Compose Example):

```
@Composable
fun UserList(viewModel: UserViewModel) {
    val userList by viewModel.allUsers.observeAsState(emptyList())

    Column {
        userList.forEach { user ->
            Text(text = user.name)
        }
    }
}
```