MODUL 10
Android Architecture Components

THEME DESCRIPTION

In this module, students will learn about the key components of the Android Jetpack libraries and what benefits they bring to the standard Android framework. Students will also learn how to structure their code and give different responsibilities to your classes with the help of Jetpack components.

WEEKLY LEARNING OUTCOME (SUB-LEARNING OUTCOME)

Students will be able to create applications that handle the life cycles of activities and fragments with ease. They'll also know more about how to persist data on an Android device using Room and how to use ViewModels to separate the logic from the Views.

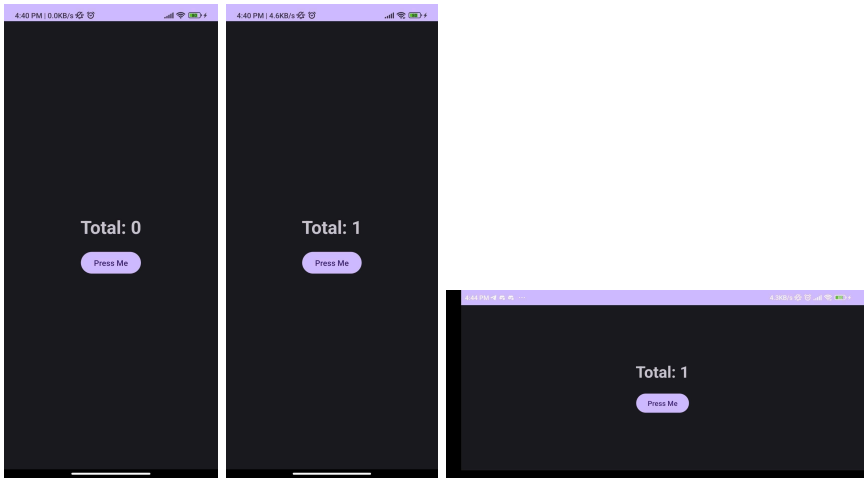
TOOLS/SOFTWARE USED

- Android Studio

PRACTICAL STEPS

Part 1 - Building App with ViewModel and LiveData

1. Open Android Studio and click **New Project.**
2. Choose the **Empty Views Activity** to start with.
3. Name your project **"LAB_WEEK_10"**.
4. Set the minimum SDK to **"API 24: Android 7.0 (Nougat)"**.
5. Click **Finish**, and let your android application build itself.
6. Here's what you will be building in this part. An activity that consists of a **TextView** and a **Button**. The **TextView** displays a count and the **Button** is used to increment that count. Normally when you rotate (which recreates the activity) the count will be reset back to 0, you will be using **ViewModel** to fix this issue. On top of that, you will also be using **LiveData** to update the UI more efficiently.



7. First, import the necessary **Dependencies** to your **build.gradle.kts (Module :app)** and don't forget to **Gradle Sync**.

```
implementation ("androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2")
```

8. Now let's make the activity layout. Update your **activity_main.xml** to the code below.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center"
   android:orientation="vertical"
   tools:context=".MainActivity">
   <TextView
       android:id="@+id/text_total"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/text_total"
       android:layout_marginBottom="20dp"
       android:textStyle="bold"
       android:textSize="32sp"/>
   <Button
       android:id="@+id/button_increment"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/button_increment" />
```

```
</LinearLayout>
```

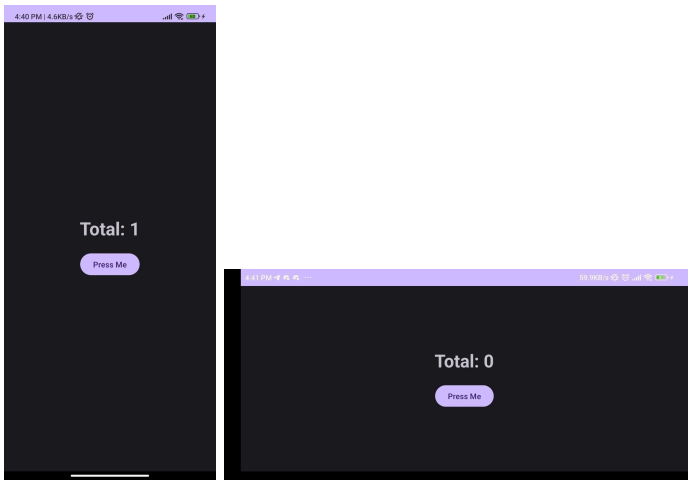
9. Your layout is done, now update your **Kotlin File**. Update your **Class** in **MainActivity.kt** to the code below.

```
class MainActivity : AppCompatActivity() {
    private var total: Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        total = 0
        updateText(total)
        findViewById<Button>(R.id.button_increment).setOnClickListener {
            incrementTotal()
        }
    }

    private fun incrementTotal() {
        total++
        updateText(total)
    }

    private fun updateText(total: Int) {
        findViewById<TextView>(R.id.text_total).text =
            getString(R.string.text_total, total)
    }
}
```

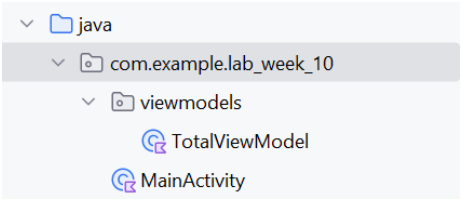
10. The code above basically binds the **TextView** to the **total** variable and sets the **onClickListener** for the **Button**.
11. Add below into your **res/values/strings.xml**

```
<resources>
    <string name="app_name">LAB_WEEK_10</string>

    <string name="text_total">Total: %d</string>
    <string name="button_increment">Press Me</string>

</resources>
```

12. Now **Run** your application, and the total should appear as **0** at first. Now, press the button and the total should increase to **1**. Then, **Rotate** your device. This will **Recreate** the activity, therefore resetting the total back to **0**.



13. You are now going to fix this issue by implementing **ViewModel** into the app.
14. First, create a new package called **viewmodels**, then inside that package create a new file called **TotalViewModel.kt**. Your new package should look like this now.



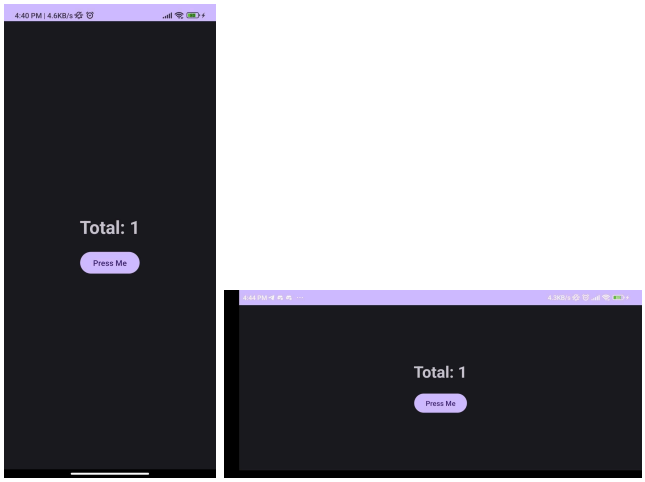
15. Inside your new **TotalViewModel.kt**, you can basically store any data inside it, and it'll be separate from any activity files. With this concept, the stored data won't be changed even though the activity is **Recreated** or **Destroyed**. You can also use this **ViewModel** for **Multiple Activities**. Update it to the code below.

```
class TotalViewModel: ViewModel() {
    var total: Int = 0

    fun incrementTotal(): Int {
        total++
        return total
    }
}
```

16. Now update your **MainActivity.kt** with the newly created **TotalViewModel**. Update it to the code below.

```
class MainActivity : AppCompatActivity() {
    private val viewModel by lazy {
        ViewModelProvider(this)[TotalViewModel::class.java]
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        prepareViewModel()
    }

    private fun updateText(total: Int) {
        findViewById<TextView>(R.id.text_total).text =
            getString(R.string.text_total, total)
    }

    private fun prepareViewModel(){
        viewModel.total.observe(this) { total ->
            updateText(total)
        }

        findViewById<Button>(R.id.button_increment).setOnClickListener {
            viewModel.incrementTotal()
        }
    }
}
```

17. Now **Run** your application, and after rotating your device, it should keep the previous total value.



COMMIT to GITHUB at this point. Commit Message: Commit No. 1: Implement ViewModel to Handle Rotation

18. The app still has 1 problem. To see this problem in action, let's make a **Fragment** and add it to the **MainActivity**. Create a new **Fragment** called **FirstFragment**. Update your **fragment_first.xml** to the code below.

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center"
   android:orientation="vertical"
   tools:context=".FirstFragment">
   <TextView
       android:id="@+id/text_total"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/text_total"
       android:layout_marginBottom="20dp"
       android:textStyle="bold"
       android:textSize="32sp"/>
</LinearLayout>
```

19. Next, update your **FirstFragment.kt** to the code below.

```
class FirstFragment : Fragment() {
```

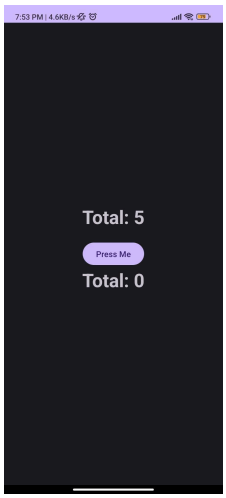
```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_first, container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        prepareViewModel()
    }

    private fun updateText(total: Int) {
        view?.findViewById<TextView>(R.id.text_total)?.text =
            getString(R.string.text_total, total)
    }

    private fun prepareViewModel(){
        val viewModel =
ViewModelProvider(requireActivity()).get(TotalViewModel::class.java)
         viewModel.total.observe(viewLifecycleOwner) { total ->
             updateText(total)
         }

    companion object {
        fun newInstance(param1: String, param2: String) =
            FirstFragment()
    }
}
```

20. And lastly, update your **activity_main.xml** to the code below.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/text_total"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_total"
        android:layout_marginBottom="20dp"
        android:textStyle="bold"
        android:textSize="32sp"/>
    <Button
        android:id="@+id/button_increment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_increment" />
    <androidx.fragment.app.FragmentContainerView
        android:layout_width="match_parent"
        android:layout_height="400dp"
        android:id="@+id/fragment_container"
        android:name="com.example.lab_week_10.FirstFragment"/>
</LinearLayout>
```

21. Now **Run** your application. After every increment, although the **Activity** and the **Fragment** are using the same **ViewModel**, the total value only updates for the **Activity**.



22. You will fix this by implementing a **DataStream** called **LiveData**. First, let's add the **LiveData Dependency** into your **build.gradle.kts (Module :app)**. Don't forget to sync after the change.

```
implementation ("androidx.lifecycle:lifecycle-livedata-ktx:2.6.2")
```

23. You will begin by changing the data type stored in the **ViewModel** to **LiveData**. Update your **TotalViewModel.kt** to the code below.

```
class TotalViewModel: ViewModel() {
    //Declare the LiveData object
    private val _total = MutableLiveData<Int>()
    val total: LiveData<Int> = _total

    //Initialize the LiveData object
    init {
        //postValue is used to set the value of the LiveData object
        //from a background thread or the main thread
        //While on the other hand setValue() is used
        //only if you're on the main thread
        _total.postValue(0)
    }

    //Increment the total value
    fun incrementTotal() {
        _total.postValue(_total.value?.plus(1))
    }
}
```

24. Next, update your **prepareViewModel** method in your **MainActivity.kt** to the code below.

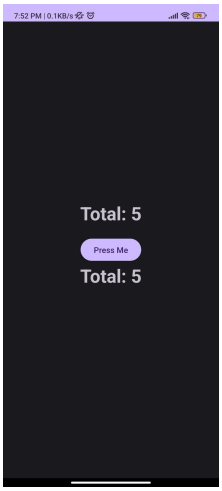
```
private fun prepareViewModel(){
    // Observe the LiveData object
    viewModel.total.observe(this, {
        // Whenever the value of the LiveData object changes
        // the updateText() is called, with the new value as the
parameter
        updateText(it)
    })

    findViewById<Button>(R.id.button_increment).setOnClickListener {
```

```
        viewModel.incrementTotal()
    }
}
```

25. Do the same with your **Fragment.** Update your **prepareViewModel** method in your **FirstFragment.kt** to the code below.

```
private fun prepareViewModel(){
    val viewModel =
ViewModelProvider(requireActivity()).get(TotalViewModel::class.java)
    // Observe the LiveData object
    viewModel.total.observe(viewLifecycleOwner, {
        // Whenever the value of the LiveData object changes
        // the updateText() is called, with the new value as the parameter
        updateText(it)
    })
}
```

26. Now if you **Run** the application. The total value also updates for the **Fragment**.



COMMIT to GITHUB at this point. Commit Message: Commit No. 2: Implement LiveData to Sync Activity and Fragment UI

Part 2 - Room Integration

1. There's still 1 more thing missing from the app. Currently, if you close the app then reopen it again, the value from the previous attempt of opening the app isn't being carried on. You will be implementing **2 Libraries** to achieve this. First, **SQLite** to store

the data locally on your device, and second, **Room** to implement the **CRUD (Create, Read, Update, Delete)** for the app. You don't need to worry about **SQLite** as **Room** already uses **SQlite** as its database storage engine.

2. Before you begin, add the **Room** library into your application. In your **build.gradle.kts (Module :app)** add `kotlin("kapt")` inside the **plugins** scope. **Kapt** (Kotlin Annotation Processing Tool) is used for the **Room Annotation** that you will be using later.

```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("org.jetbrains.kotlin.kapt")
}
```

3. Next, inside **build.gradle.kts (Project: LAB_WEEK_10)**, add the **Kapt Version**.

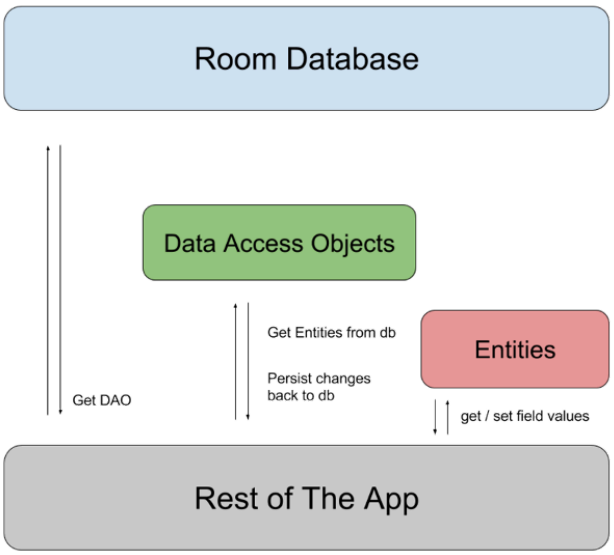
```
plugins {
    id("com.android.application") version "8.1.0" apply false
    id("org.jetbrains.kotlin.android") version "1.8.0" apply false
    id("org.jetbrains.kotlin.kapt") version "1.9.20"
}
```

4. Lastly, inside **build.gradle.kts (Module :app)**, add the **Room Dependency** inside the **dependencies** scope. Don't forget to **Gradle Sync** after this.

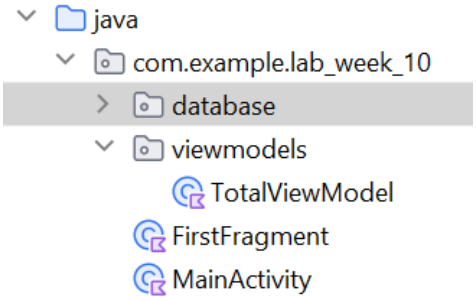
```
val roomVersion = "2.6.0"
implementation ("androidx.room:room-runtime:$roomVersion")
annotationProcessor("androidx.room:room-compiler:$roomVersion")
kapt ("androidx.room:room-compiler:$roomVersion")
```

5. Now, you can begin creating your **Room**. A **Room** consists of 3 important objects:

    - **Entities:** You can specify how you want your data to be stored and the relationships between your data.
    - **Data access object (DAO):** The operations that can be done on your data.
    - **Database:** You can specify the configurations that your database should have (the name of the database and migration scenarios).

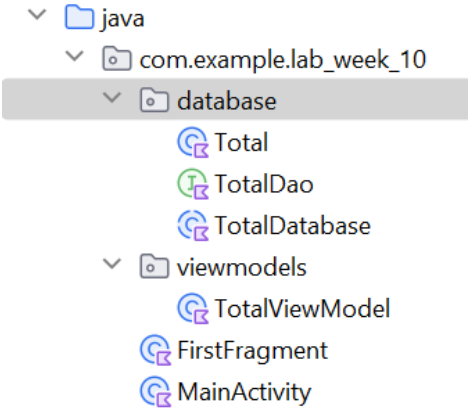


6. First, create a new **Package** called **database** to store all of your **Room Objects**.



7. Inside the new package, create 3 new files:

   - **Total.kt** - as the **Entity**
   - **TotalDao.kt** - as the **Dao**
   - **TotalDatabase.kt** - as the **RoomDatabase**



8. Start by creating your **Entity**. Update **Total.kt** to the code below.

```
// Create an Entity with a table name of "total"
// You can add ForeignKeys as a second parameter to set the foreign keys
// You can also add PrimaryKeys as a second parameter to set the primary
keys
@Entity(tableName = "total")
// Every variables declared inside this class counts
// as declaring a new column inside the table
data class Total(
    // @PrimaryKey sets a column into a primary key
    @PrimaryKey(autoGenerate = true)
    // @ColumnInfo sets the name of the column
    @ColumnInfo(name = "id")
    // The variable name can be different from the column name
    val id: Long = 0,

    // Here we set another column to store the total value
    @ColumnInfo(name = "total")
    val total: Int = 0,
)
```

9. Next, create the **Dao** for that **Entity**. Update your **TotalDao.kt** to the code below.

```
// Declare a Dao with the @Dao annotation
@Dao
// Dao is represented as an interface
// Inside the Dao, CRUD can be performed
interface TotalDao {
    // @Insert is used to insert a new row
    // OnConflictStrategy.REPLACE is used to replace existing row
    // if the id of the inserted row already exists in the database
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insert(total: Total)

    // @Update is used to update an existing row
    @Update
    fun update(total: Total)

    // @Delete is used to delete an existing row
    @Delete
```

```
    fun delete(total: Total)

    // @Query is used to define a custom query, usually to select
rows
    @Query("SELECT * FROM total WHERE id = :id")
    fun getTotal(id: Long): List<Total>
}
```

10. Lastly, create the **RoomDatabase** for the **Dao**. Update your **TotalDatabase.kt** to the code below.

```
// Create a database with the @Database annotation
// It has 2 parameters:
// entities: You can define which entities the database relies on.
// It can rely on multiple entities
// version: Used to define schema version when there's a change to the
schema.
// Update the version when you try to change the schema
@Database(entities = [Total::class], version = 1)
abstract class TotalDatabase : RoomDatabase() {
    // Declare the Dao
    abstract fun totalDao(): TotalDao
    // You can declare another Dao here for other Entities
}
```

11. Now, time to edit your **MainActivity.kt** and integrate your **Room** into your **Activity**. Add the code below at the top of your **MainActivity** class.

```
// Create an instance of the TotalDatabase
// by lazy is used to create the database only when it's needed
private val db by lazy { prepareDatabase() }
// Create an instance of the TotalViewModel
// by lazy is used to create the ViewModel only when it's needed
private val viewModel by lazy {
    ViewModelProvider(this)[TotalViewModel::class.java]
}
```

12. Next, update your **onCreate** method to the code below.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
```

```
    setContentView(R.layout.activity_main)

    // Initialize the value of the total from the database
    initializeValueFromDatabase()

    // Prepare the ViewModel
    prepareViewModel()
}
```

13. By now, you should have 2 errors in your IDE, and that is the missing **`prepareDatabase()`** and the missing **`initializeValueFromDatabase()`** declaration. Add the 2 functions anywhere inside your **MainActivity Class**.

```
// Create and build the TotalDatabase with the name 'total-database'
// allowMainThreadQueries() is used to allow queries to be run on the main thread
// This is not recommended, but for simplicity it's used here
private fun prepareDatabase(): TotalDatabase {
    return Room.databaseBuilder(
        applicationContext,
        TotalDatabase::class.java, "total-database"
    ).allowMainThreadQueries().build()
}

// Initialize the value of the total from the database
// If the database is empty, insert a new Total object with the value of 0
// If the database is not empty, get the value of the total from the database
private fun initializeValueFromDatabase() {
    val total = db.totalDao().getTotal(ID)
    if (total.isEmpty()) {
        db.totalDao().insert(Total(id = 1, total = 0))
    } else {
        viewModel.setTotal(total.first().total)
    }
}

// The ID of the Total object in the database
// For simplicity, we only have one Total object in the database
// So the ID is always 1
companion object {
    const val ID: Long = 1
```

```
}
```

14. Don't forget to also add **setTotal** to your **ViewModel**. In your **TotalViewModel.kt**, add the code below after the **incrementTotal** method.

```
//Set new total value
fun setTotal(newTotal: Int) {
    _total.postValue(newTotal)
}
```

15. Your application can now **Read** data from your local database (**SQLite**) and will use that value if it exists. Now what's left to do is to **Update** the database every time there's a change to the total value in your application. There are several approaches for this, one of them is to save the data whenever your application is **Paused**. Add the code below inside the **onPause Callback** inside **MainActivity**.

```
// Update the value of the total in the database
// whenever the activity is paused
// This is done to ensure that the value of the total is always up to date
// even if the app is closed
override fun onPause() {
    super.onPause()
    db.totalDao().update(Total(ID, viewModel.total.value!!))
}
```

16. Now **Run** your application, and your app should now save the previous value even if you **Kill** the app.

COMMIT to GITHUB at this point. Commit Message: Commit No. 3: Integrate Room Database for Data Persistence

ASSIGNMENT

Feel free to skip the assignment, but it will be considered as a **Bonus Point** if you manage to finish it.

[Bonus] Continue your **LAB_WEEK_10** project. Currently, your **TotalEntity** only has 2 columns, **ID** and **Total**. Update it so your **Total** can now hold **2 Values**, one for the

actual **total value** and the other one for the **date** containing the last time the value is updated. You can do this by using the `@Embedded` annotation.

Here's the new **Entity** that you will be using.

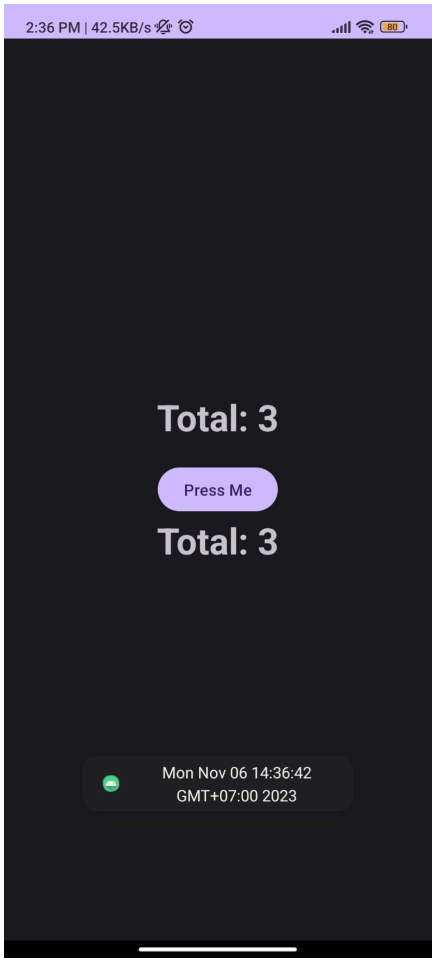
```
@Entity(tableName = "total")
// Every variables declared inside this class counts
// as declaring a new column inside the table
data class Total(
    // @PrimaryKey sets a column into a primary key
    @PrimaryKey(autoGenerate = true)
    // @ColumnInfo sets the name of the column
    @ColumnInfo(name = "id")
    // The variable name can be different from the column name
    val id: Long = 0,

    // Here we set a second column to store the total object
    // containing the value and the date
    // @Embedded is used to set column as an object
    @Embedded val total: TotalObject
)

// Here, we declare the class that's embedded into the column
data class TotalObject(
    @ColumnInfo(name = "value") val value: Int,
    @ColumnInfo(name = "date") val date: String
)
```

Now, the total column holds 2 values in it:

- **total.value**
- **total.date**

Now show **total.date** everytime the application is started. You can do this by updating **total.date** in the **onPause Callback** with the **Date.toString()** Java Library, then show a toast containing the **total.date** in the **onStart Callback**.

Here's the **Final Output** that you will be achieving.



COMMIT to GITHUB at this point. Commit Message: Commit No. 4: Complete Bonus Assignment (Add Update Timestamp)