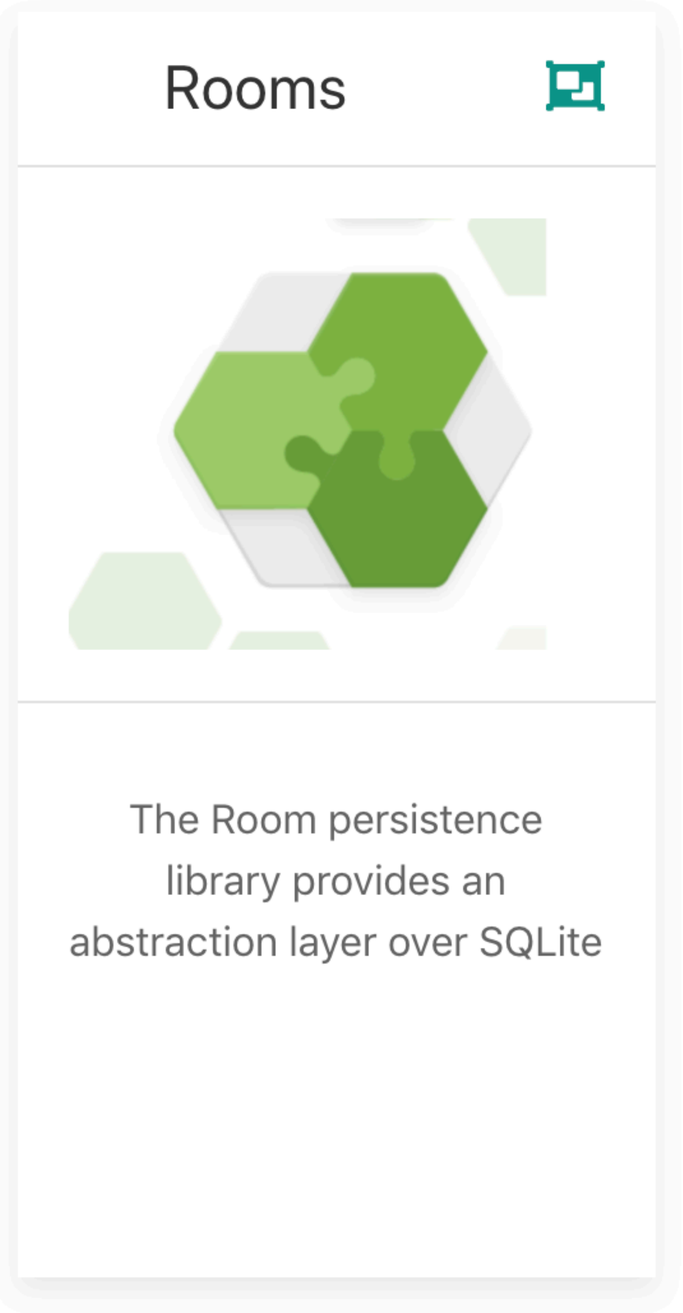
Rooms


Rooms
The Room persistence library provides an abstraction layer over SQLite

Android Architecture Components

A collection of libraries that help you design robust, testable, and maintainable apps. Start with classes for managing your UI component lifecycle and handling data persistence.

Now 1.0 stable

WATCH THE VIDEO GET STARTED

https://developer.android.com/topic/libraries/architecture

2 Major Subsystems:

Room: a SQLite object mapping library

Avoid boilerplate code and easily convert SQLite table data to Java objects using Room. Room provides compile time checks of SQLite statements and can return RxJava, Flowable and LiveData observables.

Manage your app's lifecycle with ease

New lifecycle-aware components help you manage your activity and fragment lifecycles. Survive configuration changes, avoid memory leaks and easily load data into your UI using LiveData, ViewModel, LifecycleObserver and LifecycleOwner.

4 Independent Components

Handling lifecycles

Create a UI that automatically responds to lifecycle events.

LiveData

Build data objects that notify views when the underlying database changes.

ViewModel

Store UI related data that isn't destroyed on app rotations.

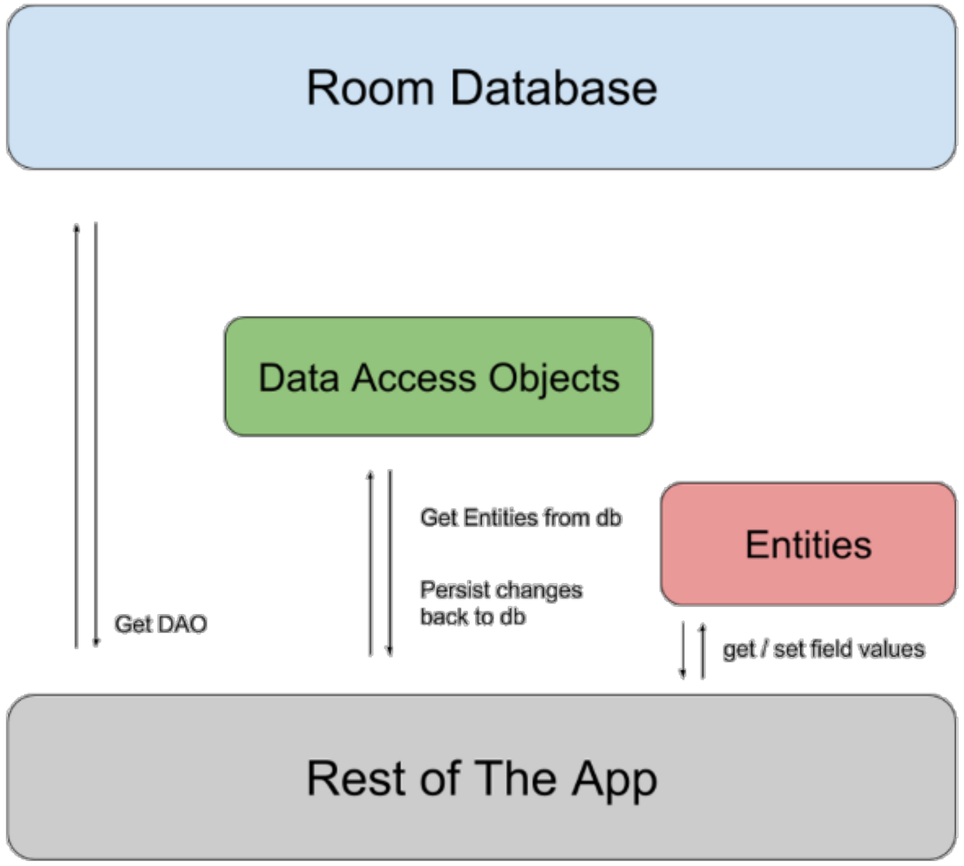
Room

Access your app's SQLite database with in-app objects and compile-time checks.

Room Persistence Library

The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.

The library helps you create a cache of your app's data on a device that's running your app. This cache, which serves as your app's single source of truth, allows users to view a consistent copy of key information within your app, regardless of whether users have an internet connection.

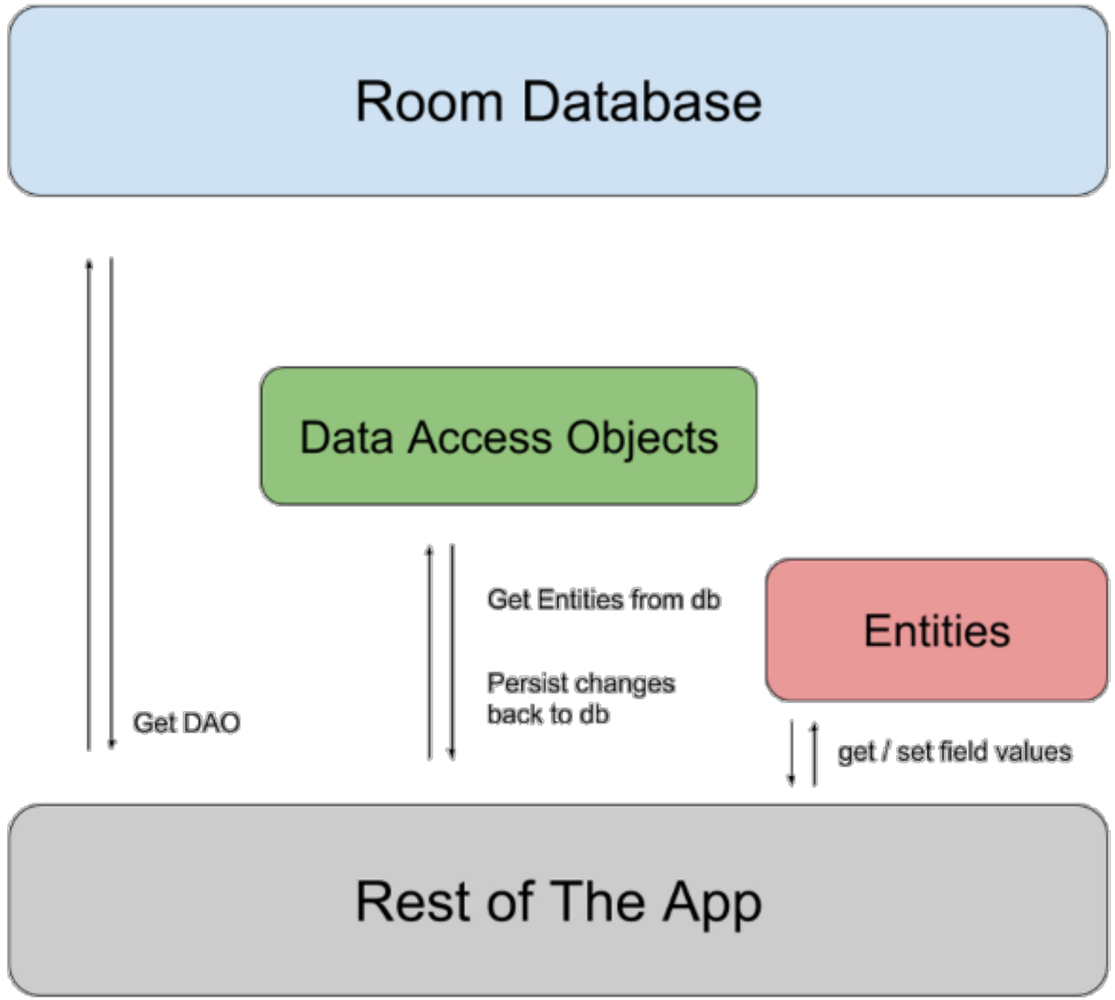


Components of room

@Entity: annotation needed define a table

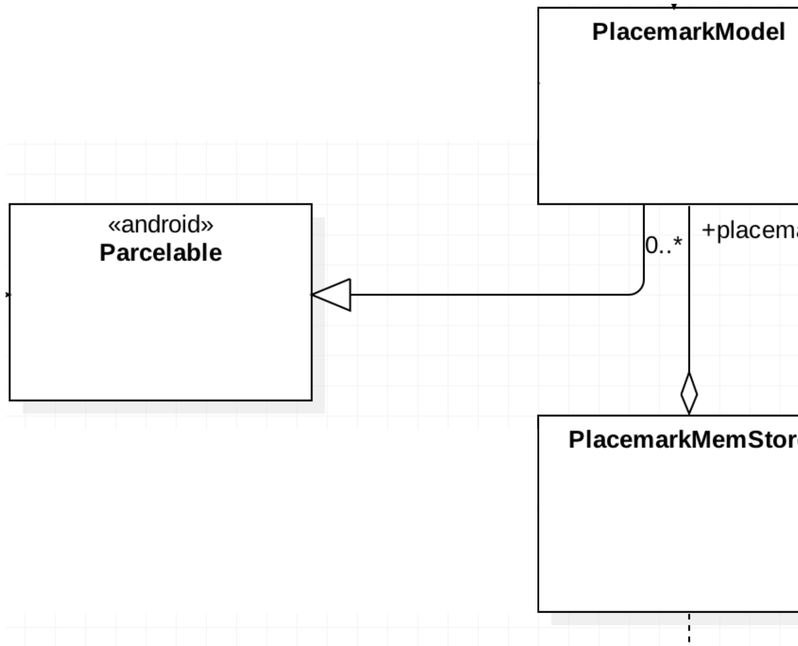
@Dao: annotation needed define a Data Access Object

@Database: annotation to create the database holder



```
@Parcelize
data class PlacemarkModel(var id: Long = 0,
                          var title: String = "",
                          var description: String = "",
                          var image: String = "",
                          var lat : Double = 0.0,
                          var lng: Double = 0.0,
                          var zoom: Float = 0f) : Parcelable
```

This Placemark Model is a Kotlin data class
It is also Parcelable - which means it can be sent between Activities



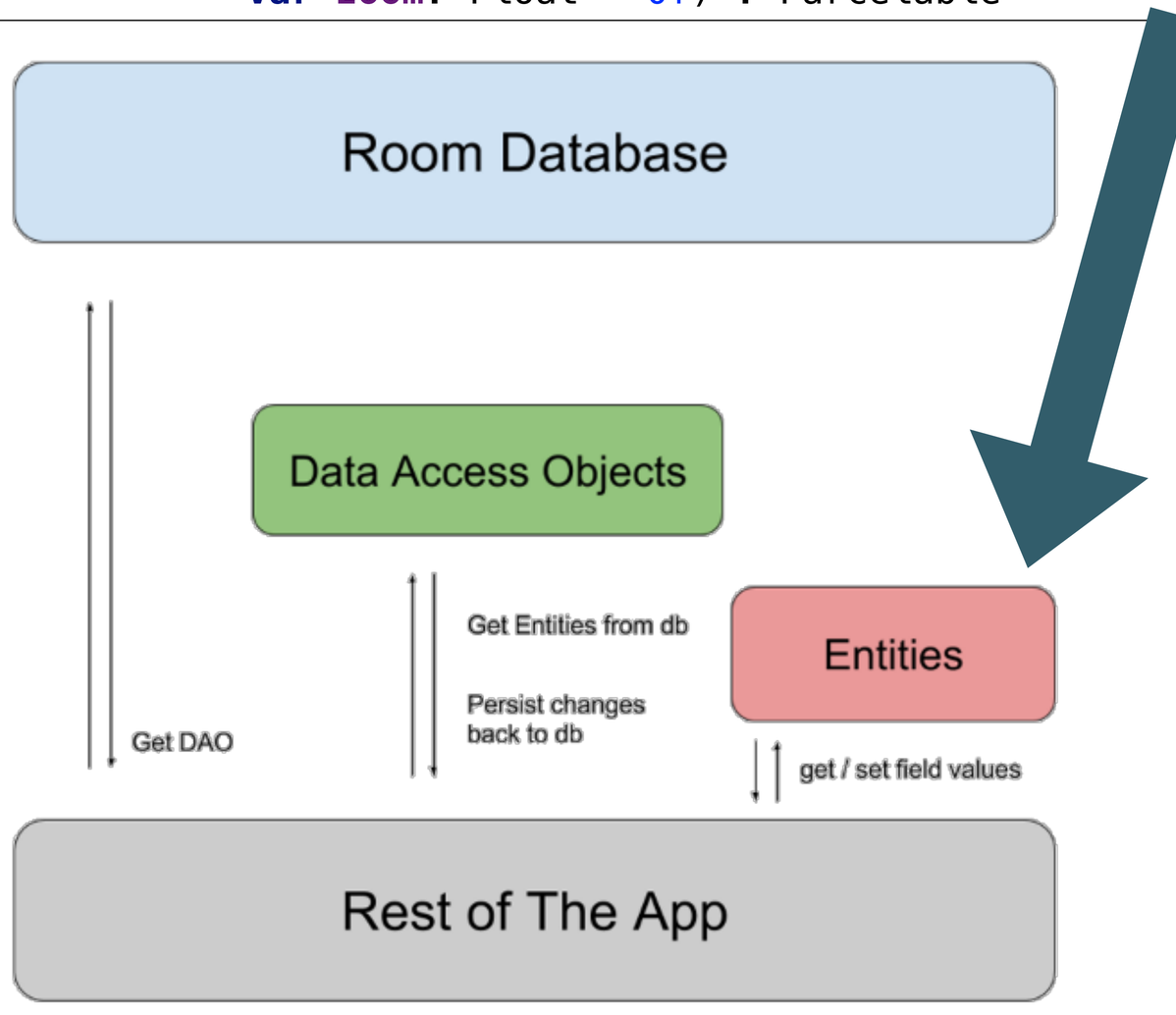
```
@Parcelize
@Entity
data class PlacemarkModel(@PrimaryKey(autoGenerate = true) var id: Long = 0,
                          var title: String = "",
                          var description: String = "",
                          var image: String = "",
                          var lat : Double = 0.0,
                          var lng: Double = 0.0,
                          var zoom: Float = 0f) : Parcelable
```

When using the Room persistence library, you define sets of related fields as entities. For each entity, a table is created within the associated Database object to hold the items.

Annotating the class with **@Entity** enables the class to be also stored in a table in database

Annotating a field with **@PrimaryKey** marks a filed as the key in this table

```
@Parcelize
@Entity
data class PlacemarkModel(@PrimaryKey(autoGenerate = true) var id: Long = 0,
                          var title: String = "",
                          var description: String = "",
                          var image: String = "",
                          var lat : Double = 0.0,
                          var lng: Double = 0.0,
                          var zoom: Float = 0f) : Parcelable
```



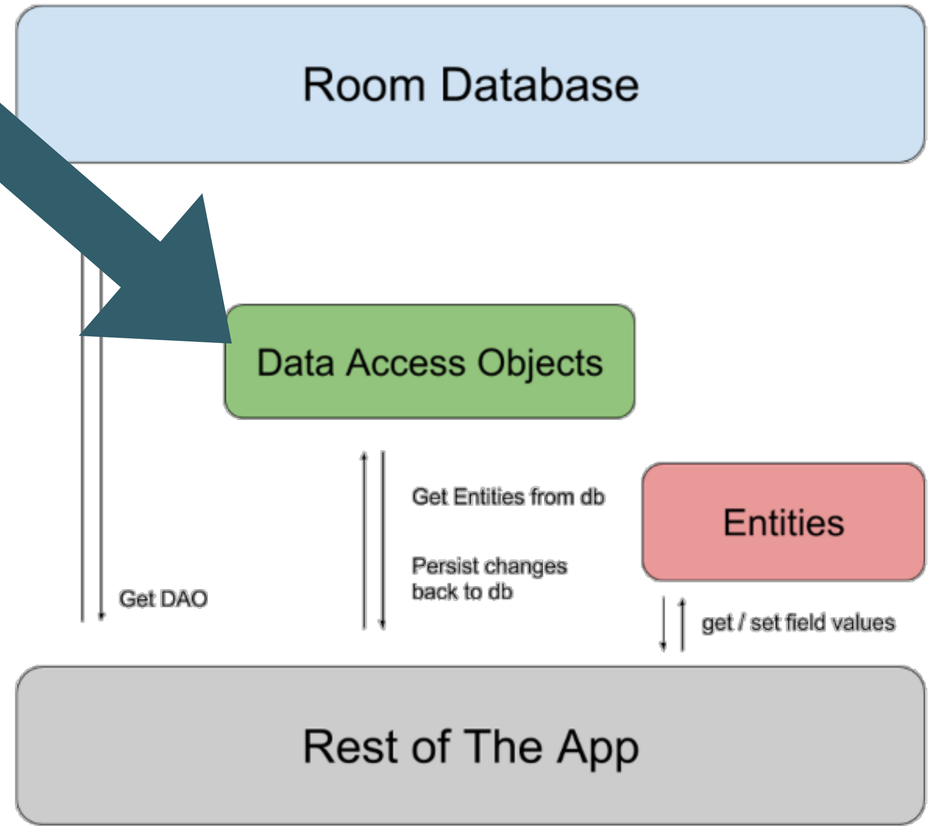
```
@Dao
interface PlacemarkDao {

  @Insert(onConflict = OnConflictStrategy.REPLACE)
  fun create(placemark: PlacemarkModel)

  @Query("SELECT * FROM PlacemarkModel")
  fun findAll(): List<PlacemarkModel>

  @Update
  fun update(placemark: PlacemarkModel)
}
```

To access your app's data using the Room persistence library, you work with data access objects, or DAOs. This set of Dao objects forms the main component of Room, as each DAO includes methods that offer abstract access to your app's database

```
@Dao
interface PlacemarkDao {

  @Insert(onConflict = OnConflictStrategy.REPLACE)
  fun create(placemark: PlacemarkModel)

  @Query("SELECT * FROM PlacemarkModel")
  fun findAll(): List<PlacemarkModel>

  @Update
  fun update(placemark: PlacemarkModel)
}
```

This is an interface to the Placemark Table

We can create, fetch and update placemarks

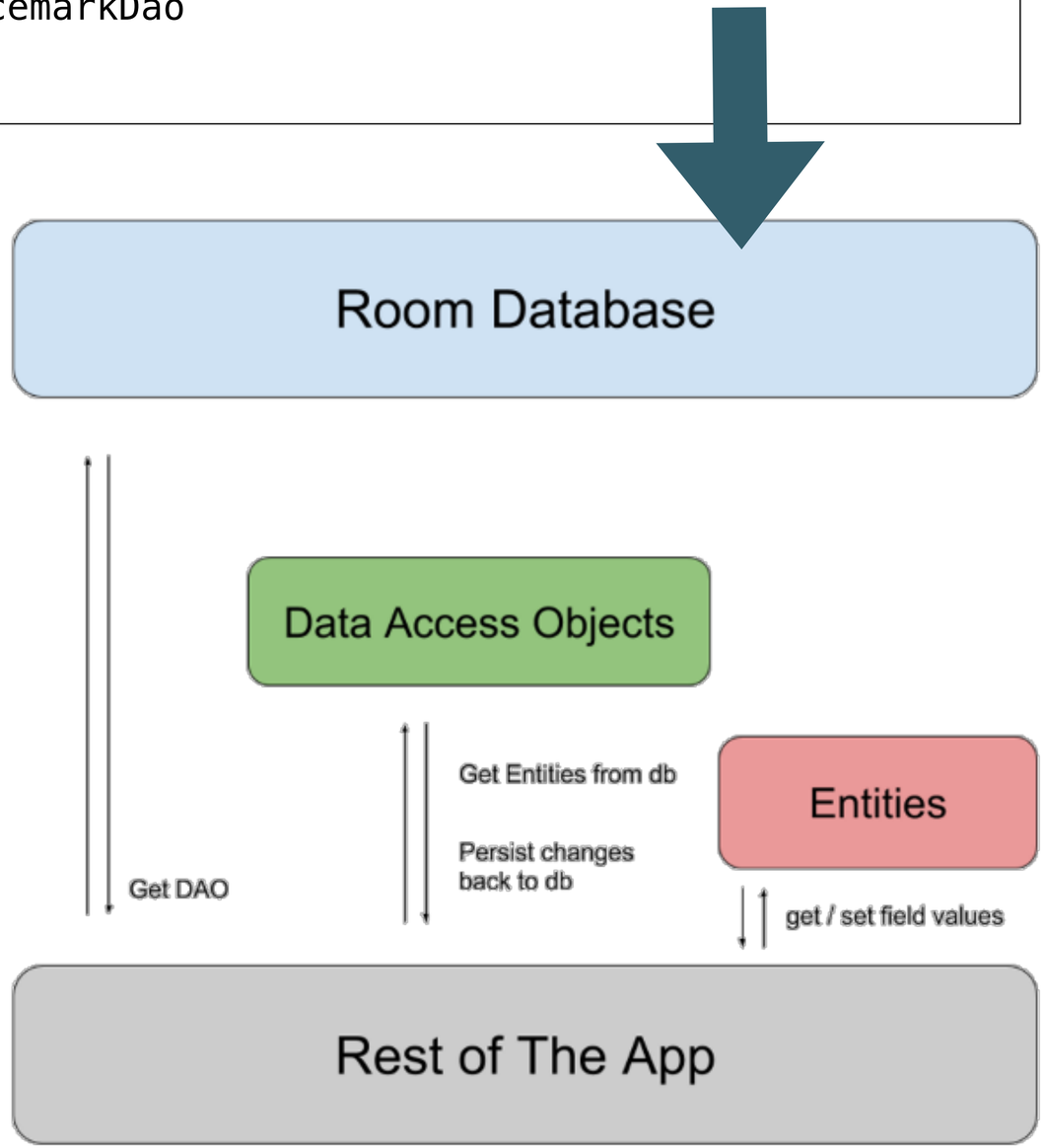


```
@Database(entities = arrayOf(PlacemarkModel::class), version = 1)
abstract class Database : RoomDatabase() {

  abstract fun placemarkDao(): PlacemarkDao
}
```

Represents the Entire Database

Provides a Dao object for each table

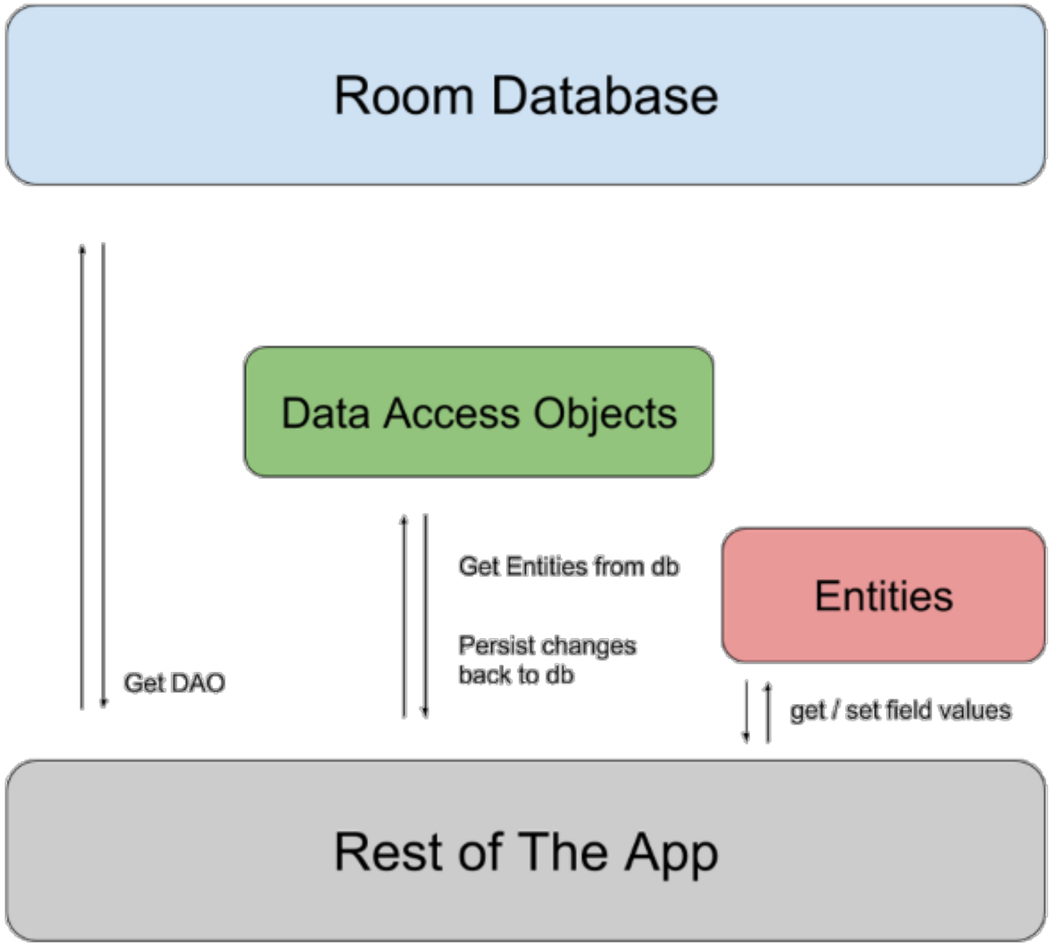
Version number should be changed if structure of database changes

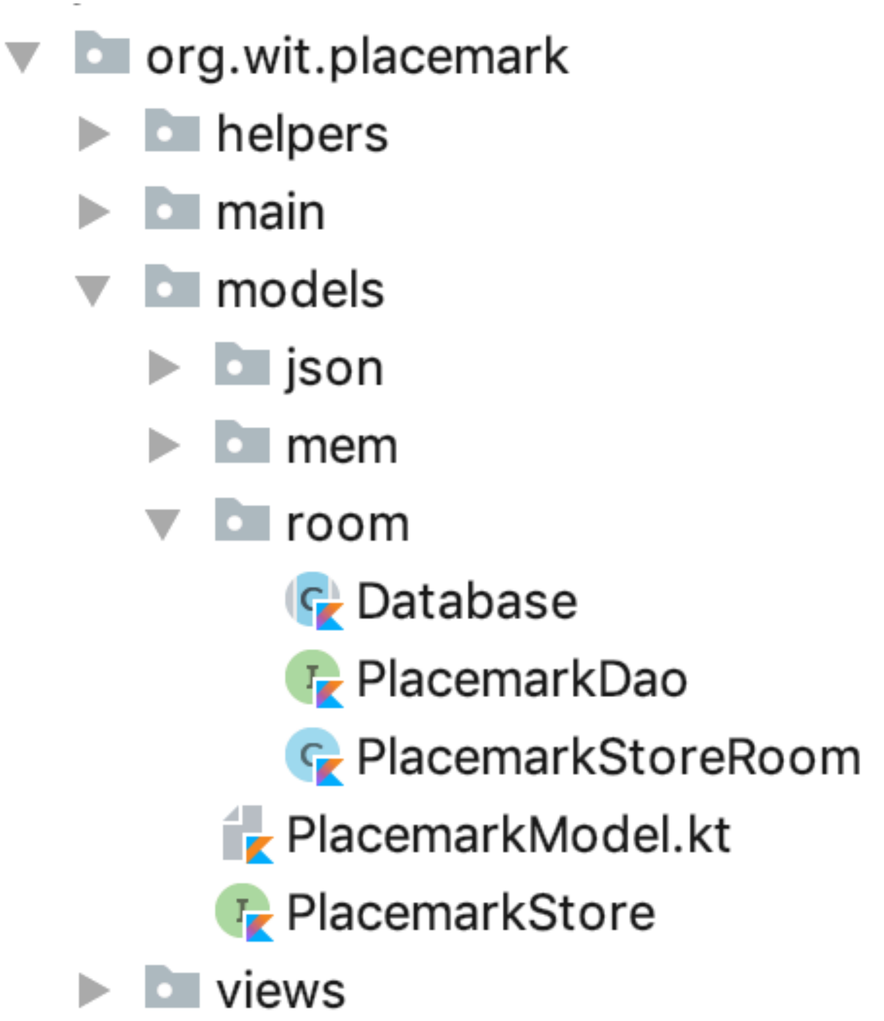


PlacemarkStoreRoom

Implements PlacemarkStore interface
Encapsulates primary database access from the rest of the app

```
class PlacemarkStoreRoom(val context: Context) : PlacemarkStore {

  var dao: PlacemarkDao

  init {
    val database = Room.databaseBuilder(context, Database::class.java, "room_sample.db")
        .fallbackToDestructiveMigration()
        .build()
    dao = database.placemarkDao()
  }

  override fun findAll(): List<PlacemarkModel> {
    return dao.findAll()
  }

  override fun create(placemark: PlacemarkModel) {
    dao.create(placemark)
  }

  override fun update(placemark: PlacemarkModel) {
    dao.update(placemark)
  }
}
```


Room Database
Data Access Objects
Get DAO
Get Entities from db
Persist changes
back to db
Entities
get / set field values
Rest of The App


org.wit.placemark
helpers
main
models
json
mem
room
Database
PlacemarkDao
PlacemarkStoreRoom
PlacemarkModel.kt
PlacemarkStore
views