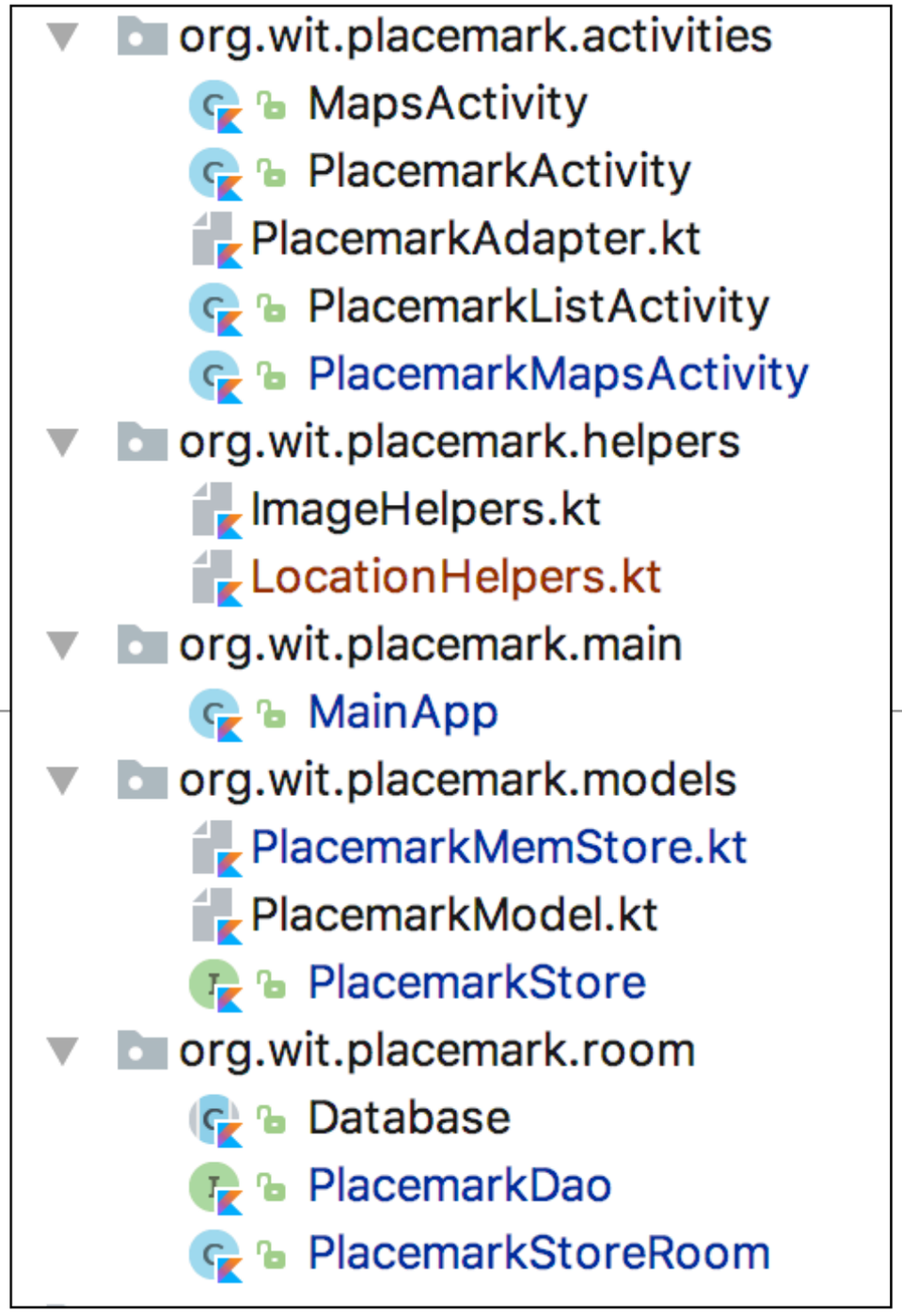
Rooms in Placemark



First Some Housekeeping

```
class MainApp : Application(), AnkoLogger {

  lateinit var placemarks: PlacemarkStore

  override fun onCreate() {
    super.onCreate()
    placemarks = PlacemarkMemStore()
    info("Placemark started")
  }
```

Mark placemarks object as '**lateinit**'

Effectively, we are guaranteeing to initialise it - so no need for null safety checks,

More Housekeeping…

```
private fun loadPlacemarks() {
  showPlacemarks( app.placemarks.findAll())
}

fun showPlacemarks (placemarks: List<PlacemarkModel>) {
  recyclerView.adapter = PlacemarkAdapter(placemarks, this)
  recyclerView.adapter.notifyDataSetChanged()
}
```

In PlacemarkListActivity define methods to manage loading and display of placemarks

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_placemark_list)
  app = application as MainApp

  toolbarMain.title = title
  setSupportActionBar(toolbarMain)

  val layoutManager = LinearLayoutManager(this)
  recyclerView.layoutManager = layoutManager
  loadPlacemarks()
}
```

Call these methods at key points in lifecycle

```kotlin
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
  loadPlacemarks()
  super.onActivityResult(requestCode, resultCode, data)
}
```

```kotlin
class PlacemarkListActivity : AppCompatActivity(), PlacemarkListener {

  lateinit var app: MainApp

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_placemark_list)
    app = application as MainApp

    toolbarMain.title = title
    setSupportActionBar(toolbarMain)

    val layoutManager = LinearLayoutManager(this)
    recyclerView.layoutManager = layoutManager
    loadPlacemarks()
  }

  private fun loadPlacemarks() {
    showPlacemarks( app.placemarks.findAll())
  }

  fun showPlacemarks (placemarks: List<PlacemarkModel>) {
    recyclerView.adapter = PlacemarkAdapter(placemarks, this)
    recyclerView.adapter.notifyDataSetChanged()
  }

  override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu_main, menu)
    return super.onCreateOptionsMenu(menu)
  }

  override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    loadPlacemarks()
    super.onActivityResult(requestCode, resultCode, data)
  }

  override fun onOptionsItemSelected(item: MenuItem?): Boolean {
    when (item?.itemId) {
      R.id.item_add -> startActivityForResult<PlacemarkActivity>(200)
    }
    return super.onOptionsItemSelected(item)
  }

  override fun onPlacemarkClick(placemark: PlacemarkModel) {
    startActivityForResult(intentFor<PlacemarkActivity>().putExtra("placemark_edit", placemark), 201)
  }
}
```

Complete PlacemarkList Activity

Introducing Rooms….

```
compile 'android.arch.persistence.room:runtime:1.0.0'
kapt 'android.arch.persistence.room:compiler:1.0.0'
```

…include in grade

Mark class as an @Entity - it can be stored in a database

```
@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
```

Mark id @PrimaryKey + have it autoGenerated by db

```
@Dao
interface PlacemarkDao {

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

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

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

Define an Interface to the PlacemarkTable
The implementation of this interface is generated by the rooms libraries

```
@Dao
interface PlacemarkDao {

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

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

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

Create a placemark (replace if id already exists)

```
@Dao
interface PlacemarkDao {

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

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

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

Get a List of all Placemarks

```
@Dao
interface PlacemarkDao {

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

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

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

Update an existing Placemark

Interface to Enter Database

Database version number

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

  abstract fun placemarkDao(): PlacemarkDao
}
```

Provide access to all Dao objects (only one so far)

If structure of database changes (new fields etc, this number can be increased

PlacemarkStore - an Interface defining how to access the model

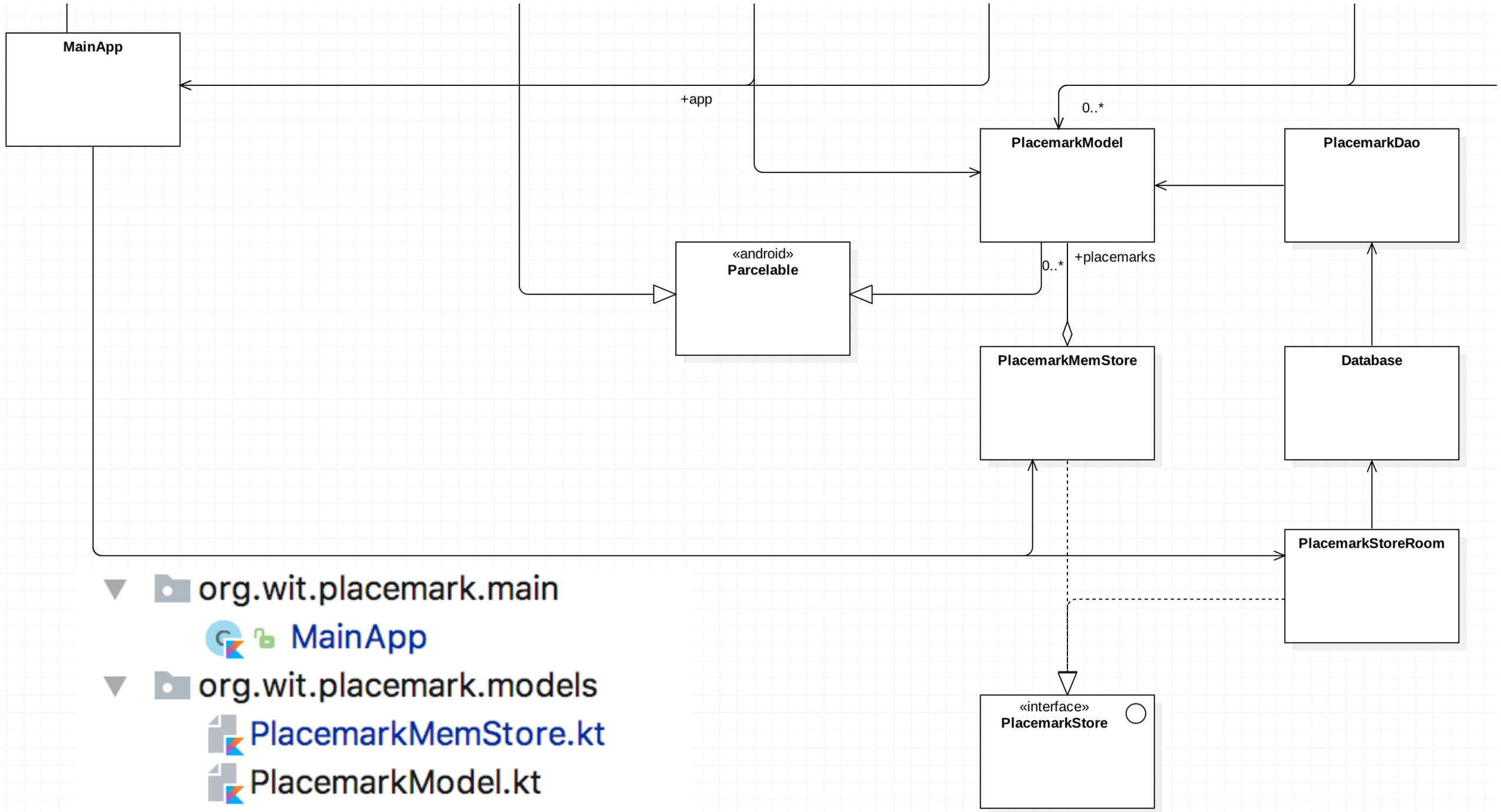
```
interface PlacemarkStore {
  suspend fun findAll(): List<PlacemarkModel>
  suspend fun findById(id:Long) : PlacemarkModel?
  fun create(placemark: PlacemarkModel)
  fun update(placemark: PlacemarkModel)
}
```

2 Implementations:

- PlacemarkMemStore: store placemarks in array
- PlacemarkStoreRoom: store placemarks in database

PlacemarkStoreRoom

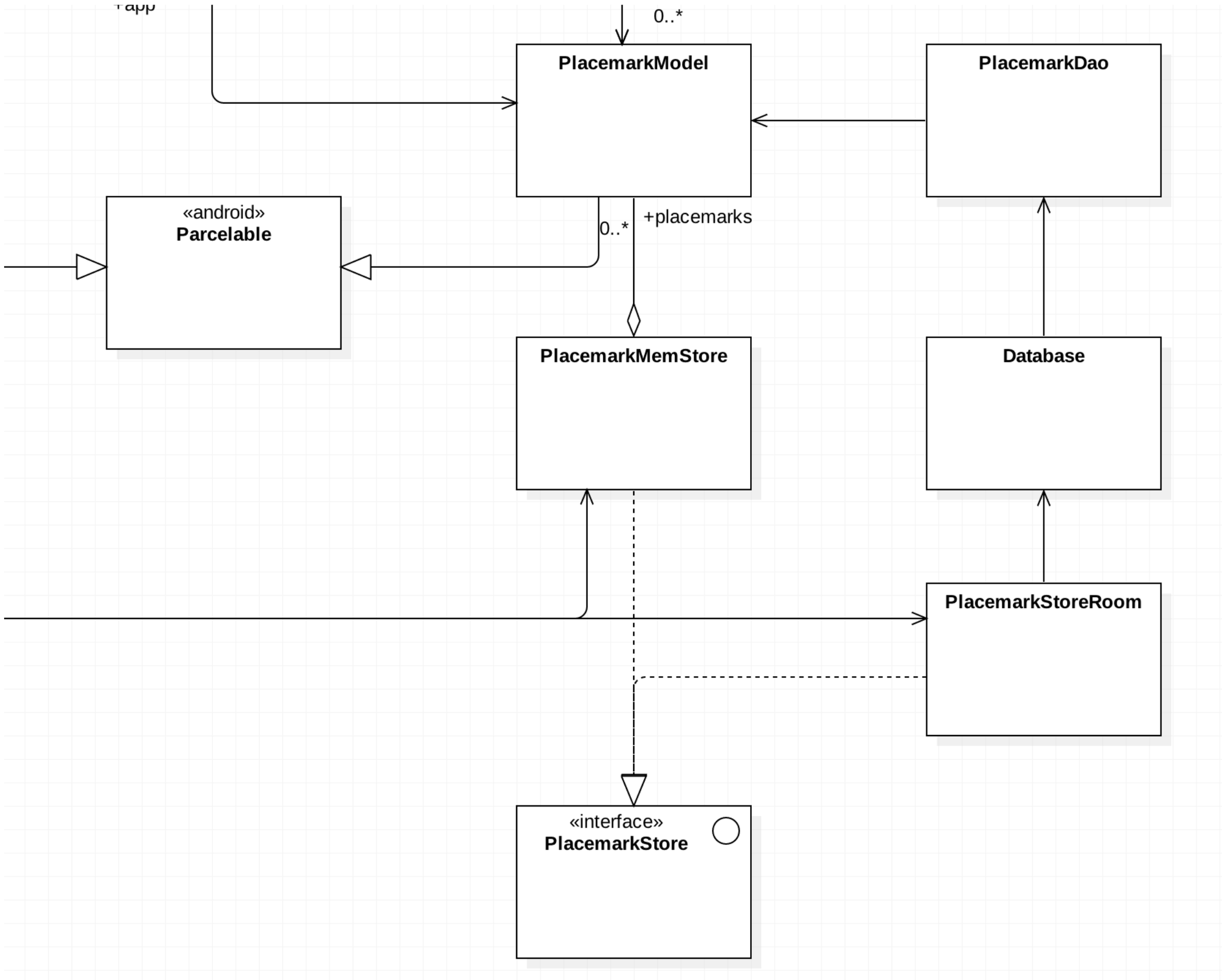
```
class PlacemarkStoreRoom(val context: Context) : PlacemarkStore {

  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)
  }
}
```



- org.wit.placemark.main
  - MainApp
- org.wit.placemark.models
  - PlacemarkMemStore.kt
  - PlacemarkModel.kt
  - PlacemarkStore
- org.wit.placemark.room
  - Database
  - PlacemarkDao
  - PlacemarkStoreRoom

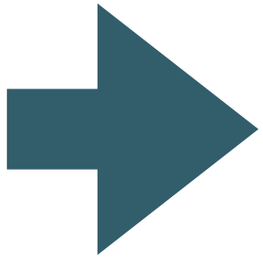
Switch between in-memory and database placemarks

```
class MainApp : Application(), AnkoLogger {

  lateinit var placemarks: PlacemarkStore

  override fun onCreate() {
    super.onCreate()
   // placemarks = PlacemarkMemStore()
    placemarks = PlacemarkStoreRoom (applicationContext)
    info("Placemark started")
  }
}
```


+app
0..*
PlacemarkModel
PlacemarkDao
«android»
Parcelable
0..*
+placemarks
PlacemarkMemStore
Database
PlacemarkStoreRoom
«interface»
PlacemarkStore

Caused by: java.lang.IllegalStateException: Cannot access database on the main thread since it may potentially lock the UI for a long period of time.

This version is terminated by Android



Cannot access database on the main thread

```kotlin
interface PlacemarkStore {
  fun findAll(): List<PlacemarkModel>
  fun create(placemark: PlacemarkModel)
  fun update(placemark: PlacemarkModel)
}
```

```kotlin
class PlacemarkMemStore : PlacemarkStore, AnkoLogger {

  val placemarks = ArrayList<PlacemarkModel>()

  suspend override fun findAll(): List<PlacemarkModel> {
    return placemarks
  }

  override fun create(placemark: PlacemarkModel) {
    placemark.id = getId()
    placemarks.add(placemark)
    logAll()
  }

  override fun update(placemark: PlacemarkModel) {
    var foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.id == placemark.id }
    if (foundPlacemark != null) {
      foundPlacemark.title = placemark.title
      foundPlacemark.description = placemark.description
      foundPlacemark.image = placemark.image
      foundPlacemark.lat = placemark.lat
      foundPlacemark.lng = placemark.lng
      foundPlacemark.zoom = placemark.zoom
    }
  }

  internal fun logAll() {
    placemarks.forEach { info("${it}") }
  }
}
```

Store - In-memory Implementation

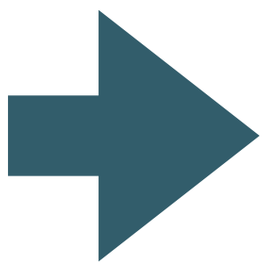
```
interface PlacemarkStore {
  fun findAll(): List<PlacemarkModel>
  fun create(placemark: PlacemarkModel)
  fun update(placemark: PlacemarkModel)
}
```

```
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)
  }
}
```

Store - Database Implementation

```
interface PlacemarkStore {
  fun findAll(): List<PlacemarkModel>
  fun create(placemark: PlacemarkModel)
  fun update(placemark: PlacemarkModel)
}
```

Support background thread invocation vis '**suspend**'



```
interface PlacemarkStore {
  suspend fun findAll(): List<PlacemarkModel>
  fun create(placemark: PlacemarkModel)
  fun update(placemark: PlacemarkModel)
}
```

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

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

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

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

‘**bg**’ ensures all db requests dispatched to background thread

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

  override suspend fun findAll(): List<PlacemarkModel> {
    val deferredPlacemarks = bg {
      dao.findAll()
    }
    val placemarks = deferredPlacemarks.await()
    return placemarks
  }

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

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

```
class PlacemarkListActivity : AppCompatActivity(), PlacemarkListener
{.
  …
   private fun loadPlacemarks() {
     showPlacemarks(app.placemarks.findAll())
  }
}
```

‘**async**’ ensure we wait for background thread to complete before we resume

```
class PlacemarkListActivity : AppCompatActivity(), PlacemarkListener {
  ...
  private fun loadPlacemarks() {
    async(UI) {
      showPlacemarks(app.placemarks.findAll())
    }
  }
}
```