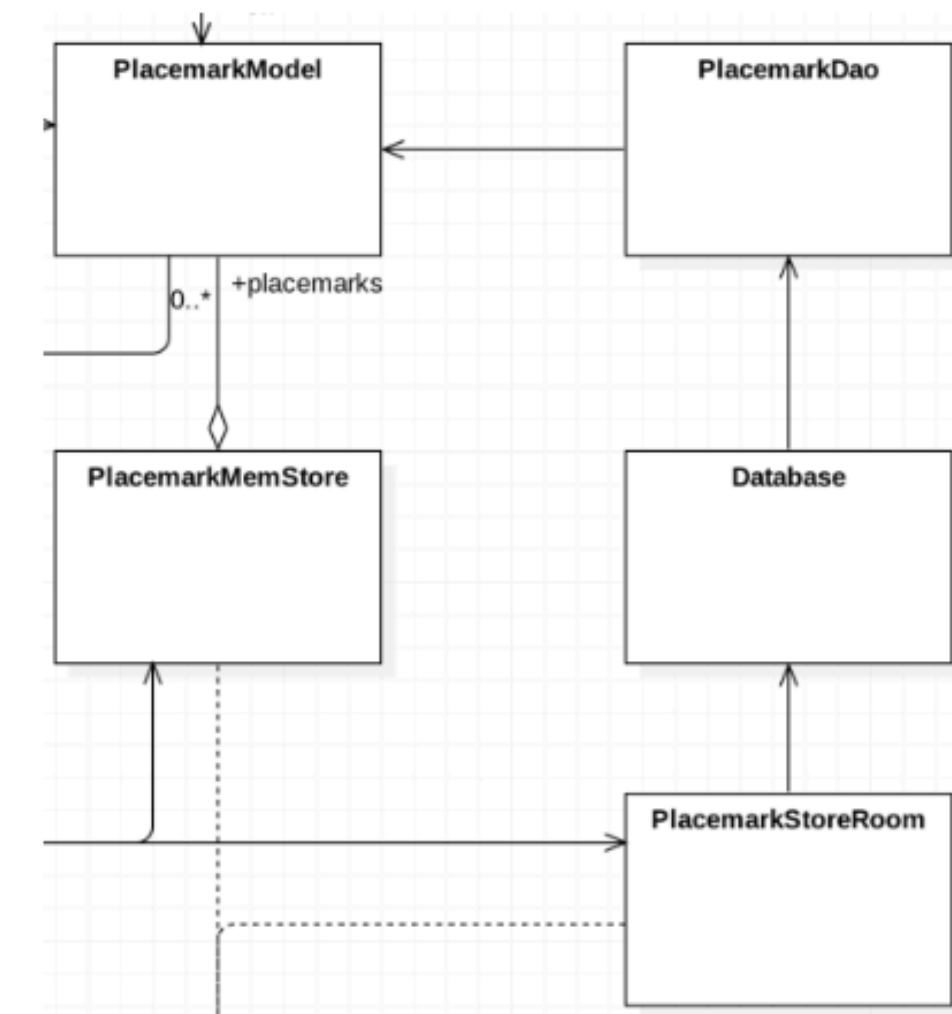


Rooms in Placemark

Rooms in Placemark



Implementation of PlacemarkStoreRoom, which stores placemarks in a SQLite database

build.gradle

A new plugin at the top of the file:

```
apply plugin: "kotlin-kapt"
```

A new version identifier:

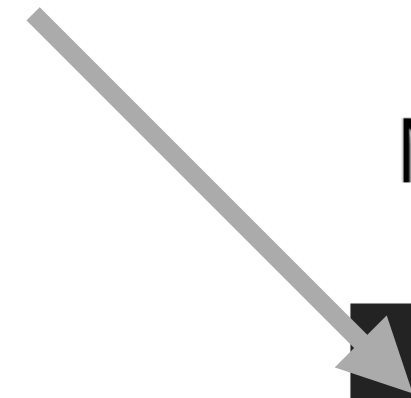
```
room_version = "2.0.0"
```

New libraries:

```
implementation "androidx.room:room-runtime:$room_version"  
kapt "androidx.room:room-compiler:$room_version"
```

Room
Library

Annotation
Processor



Annotation Processing with Kotlin

Annotation processors (see [JSR 269](#)) are supported in Kotlin with the *kapt* compiler plugin.

Being short, you can use libraries such as [Dagger](#) or [Data Binding](#) in your Kotlin projects.

Please read below about how to apply the *kapt* plugin to your Gradle/Maven build.

Using in Gradle

Apply the `kotlin-kapt` Gradle plugin:

```
apply plugin: 'kotlin-kapt'
```

Or you can apply it using the plugins DSL:

```
plugins {  
    id "org.jetbrains.kotlin.kapt" version "1.3.10"  
}
```

Then add the respective dependencies using the `kapt` configuration in your `dependencies` block:

```
dependencies {  
    kapt 'groupId:artifactId:version'  
}
```

PlacemarkModel

```
package org.wit.placemark.models

import android.os.Parcelable
import androidx.room.Entity
import androidx.room.PrimaryKey
import kotlinx.android.parcel.Parcelize

@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
```

We have included 2 additional annotations:

- @Entity
- @PrimaryKey

These annotations will enable PlacemarkModel objects to be stored in a Room database.

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)
}
```

Defines an Interface to
the PlacemarkTable

The implementation of
this interface is
generated by the rooms
libraries

@Dao

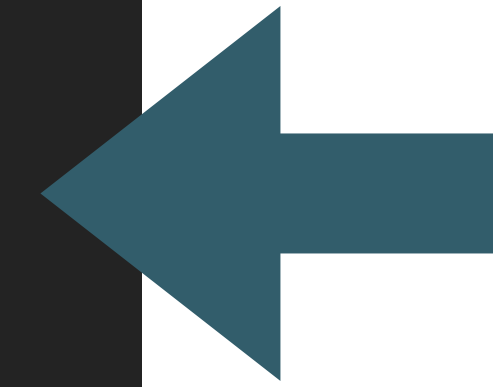
```
interface PlacemarkDao {
```

```
    @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

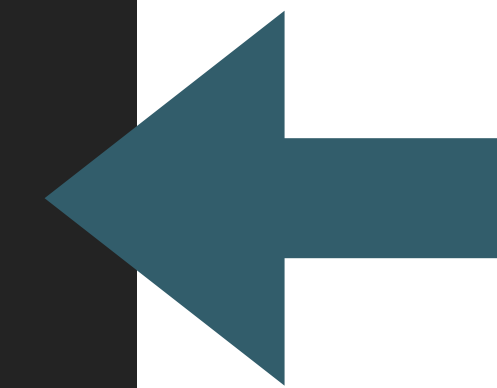
```
interface PlacemarkDao {
```

```
    @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

PlacemarkDao

```
package org.wit.placemark.room

import androidx.room.*
import org.wit.placemark.models.PlacemarkModel

@Dao
interface PlacemarkDao {

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

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

    @Query("select * from PlacemarkModel where id = :id")
    fun findById(id: Long): PlacemarkModel

    @Update
    fun update(placemark: PlacemarkModel)

    @Delete
    fun deletePlacemark(placemark: PlacemarkModel)
}
```

Database

```
package org.wit.placemark.room

import androidx.room.Database
import androidx.room.RoomDatabase
import org.wit.placemark.models.PlacemarkModel

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

    abstract fun placemarkDao(): PlacemarkDao
}
```

Interface to Entier Database

Database

```
package org.wit.placemark.room

import androidx.room.Database
import androidx.room.RoomDatabase
import org.wit.placemark.models.PlacemarkModel

@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)

Database
version number

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

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)  
    }  
}
```


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)  
    }  
}
```

Create a
Database object

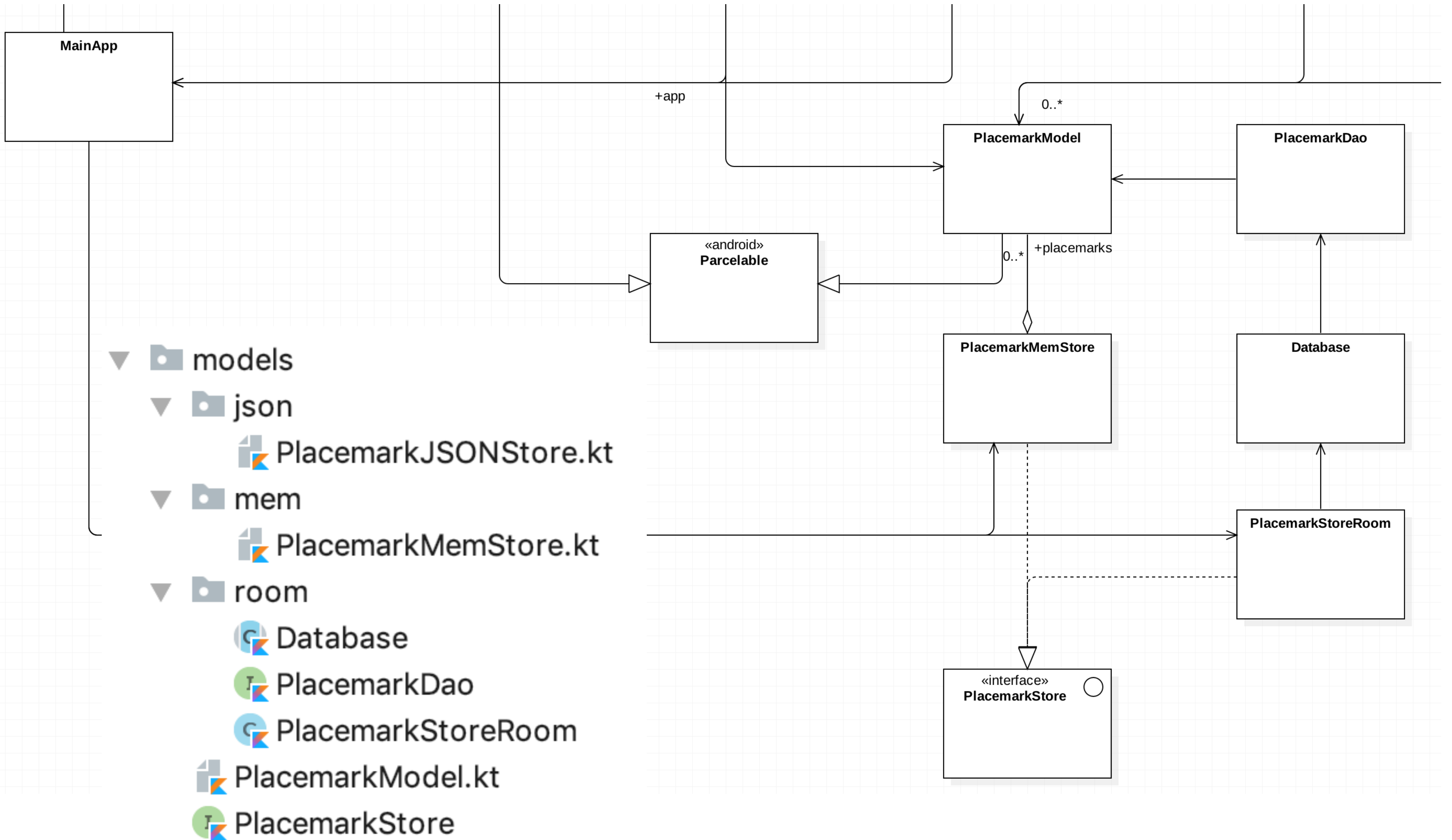


Request a dao
object from the
database



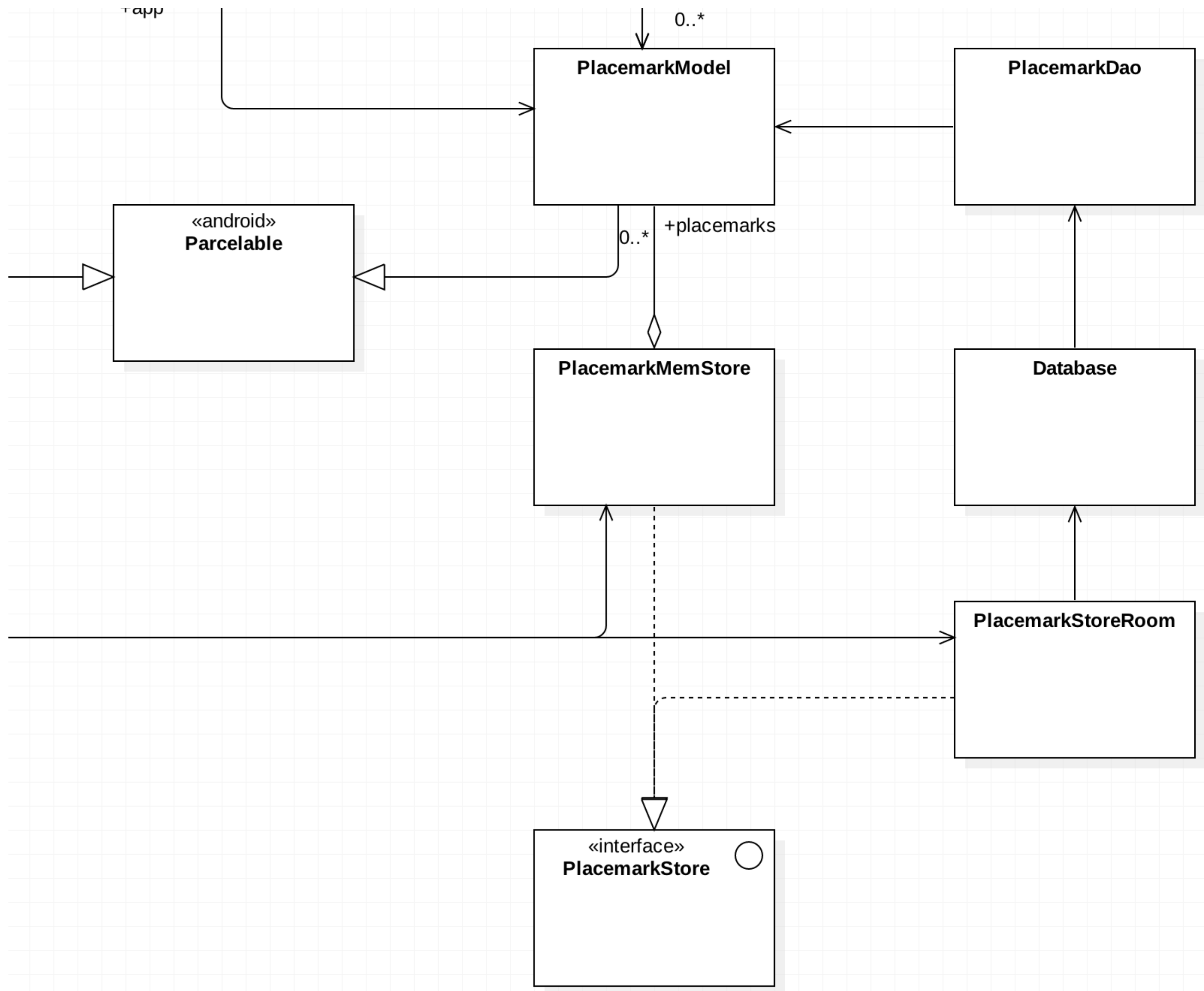
Use the dao to
implement all
PlacemarkStore
features

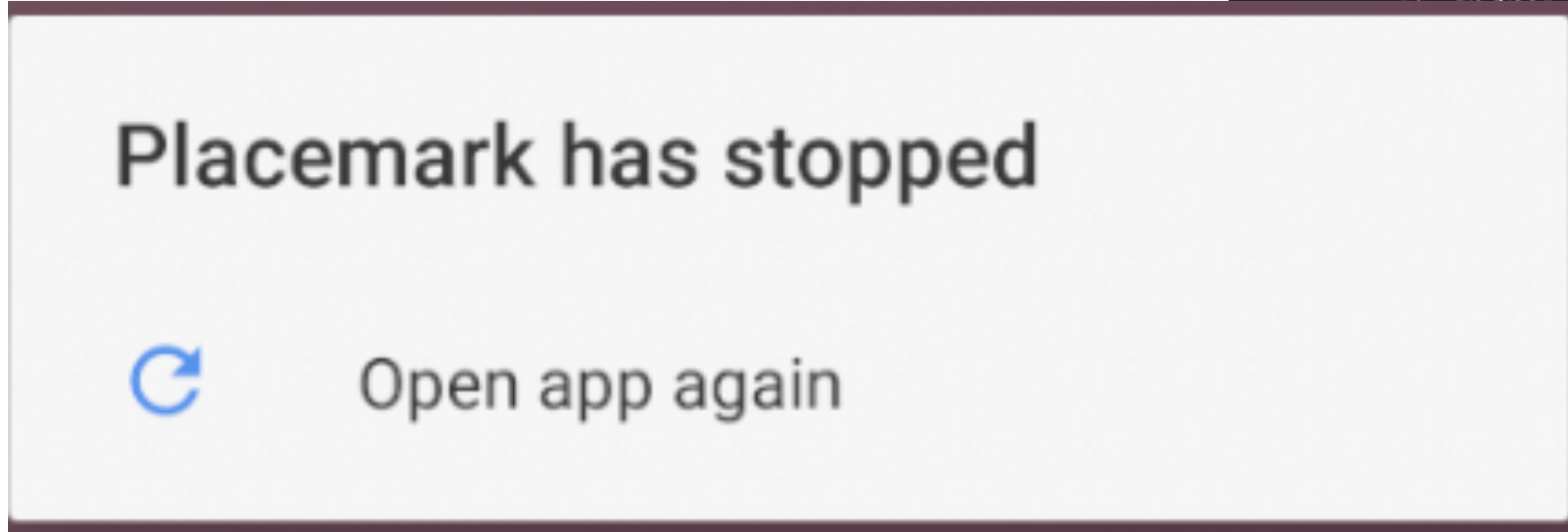




Switch between in-memory and database placemarks

```
class MainApp : Application(), AnkoLogger {  
    lateinit var placemarks: PlacemarkStore  
  
    override fun onCreate() {  
        super.onCreate()  
        // placemarks = PlacemarkMemStore()  
        placemarks = PlacemarkStoreRoom(applicationContext)  
        info("Placemark started")  
    }  
}
```

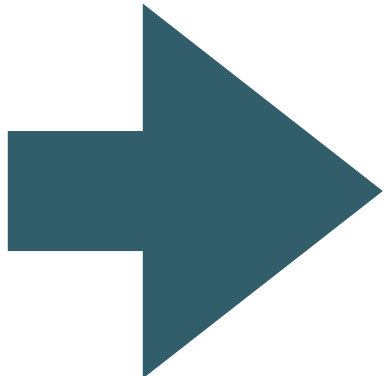





```
2018-11-16 16:50:58.333 12758-12758/org.wit.placemark E/AndroidRuntime: FATAL EXCEPTION: main
Process: org.wit.placemark, PID: 12758
java.lang.RuntimeException: Unable to start activity ComponentInfo{org.wit.placemark/org.wit.placemark.views.placemarklist.PlacemarkListView}
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2665)
at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2726)
at android.app.ActivityThread.-wrap12(ActivityThread.java:)
at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1477)
at android.os.Handler.dispatchMessage(Handler.java:102)
at android.os.Looper.loop(Looper.java:154)
at android.app.ActivityThread.main(ActivityThread.java:6119)
at java.lang.reflect.Method.invoke(Native Method)
at android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:886)
at android.internal.os.ZygoteInit.main(ZygoteInit.java:776)
java.lang.IllegalStateException: Cannot access database on the main thread since it may potentially lock the UI for a long period of time.
at androidx.room.RoomDatabase.assertNotMainThread(RoomDatabase.java:209)
at androidx.room.RoomDatabase.query(RoomDatabase.java:237)
at org.wit.placemark.room.PlacemarkDao_Impl.findAll(PlacemarkDao_Impl.java:137)
at org.wit.placemark.room.PlacemarkStoreRoom.findAll(PlacemarkStoreRoom.kt:21)
at org.wit.placemark.views.placemarklist.PlacemarkListPresenter.loadPlacemarks(PlacemarkListPresenter.kt:23)
at org.wit.placemark.views.placemarklist.PlacemarkListView.onCreate(PlacemarkListView.kt:25)
at android.app.Activity.performCreate(Activity.java:6679)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1118)
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2618)
at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2726)
at android.app.ActivityThread.-wrap12(ActivityThread.java:)
at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1477)
at android.os.Handler.dispatchMessage(Handler.java:102)
at android.os.Looper.loop(Looper.java:154)
at android.app.ActivityThread.main(ActivityThread.java:6119)
at java.lang.reflect.Method.invoke(Native Method)
at android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:886)
at android.internal.os.ZygoteInit.main(ZygoteInit.java:776)
```

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

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

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

Store - In-
memory
Implementation

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

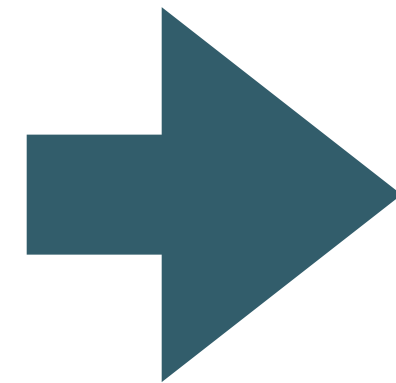
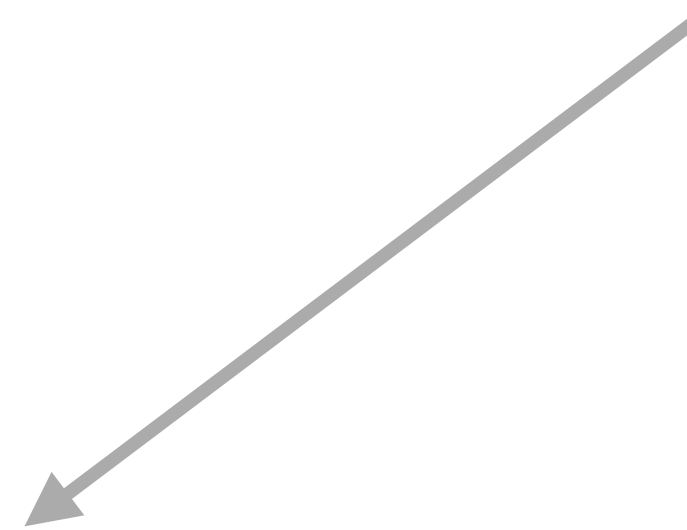
```
class PlacemarkStoreRoom(val context: Context) : PlacemarkStore {  
  
    var dao: PlacemarkDao  
  
    init {  
        val database = Room.databaseBuilder(context, Database::class.java, "room_sample.db")  
            .fallbackToDestructiveMigration()  
            .build()  
        dao = database.placemarkDao()  
    }  
  
    override fun findAll(): List<PlacemarkModel> {  
        return dao.findAll()  
    }  
  
    override fun create(placemark: PlacemarkModel) {  
        dao.create(placemark)  
    }  
  
    override fun update(placemark: PlacemarkModel) {  
        dao.update(placemark)  
    }  
}
```

Any of these
database
calls will
trigger
Termination
of app

Store - Database
Implementation

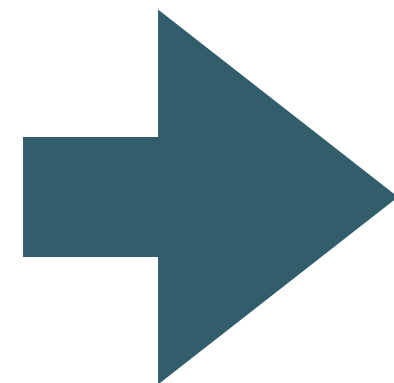
```
interface PlacemarkStore {  
  fun findAll(): List<PlacemarkModel>  
  fun create(placemark: PlacemarkModel)  
  fun update(placemark: PlacemarkModel)  
}
```

Support background
thread invocation via
'suspend'



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

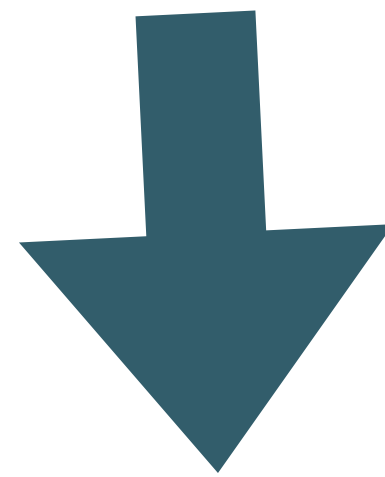
```
class PlacemarkStoreRoom(val context: Context) : PlacemarkStore {  
  
    override fun findAll(): List<PlacemarkModel> {  
        return dao.findAll()  
    }  
  
    override fun create(placemark: PlacemarkModel) {  
        dao.create(placemark)  
    }  
  
    override fun update(placemark: PlacemarkModel) {  
        dao.update(placemark)  
    }  
}
```



'bg' ensures all db
requests
dispatched to
background thread

```
class PlacemarkStoreRoom(val context: Context) : PlacemarkStore {  
    ...  
  
    suspend override fun findAll(): List<PlacemarkModel> {  
        val deferredPlacemarks = bg {  
            dao.findAll()  
        }  
        val placemarks = deferredPlacemarks.await()  
        return placemarks  
    }  
  
    suspend override fun create(placemark: PlacemarkModel) {  
        bg {  
            dao.create(placemark)  
        }  
    }  
  
    suspend 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 ‘suspend’ denoted
calls completes before
we resume

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