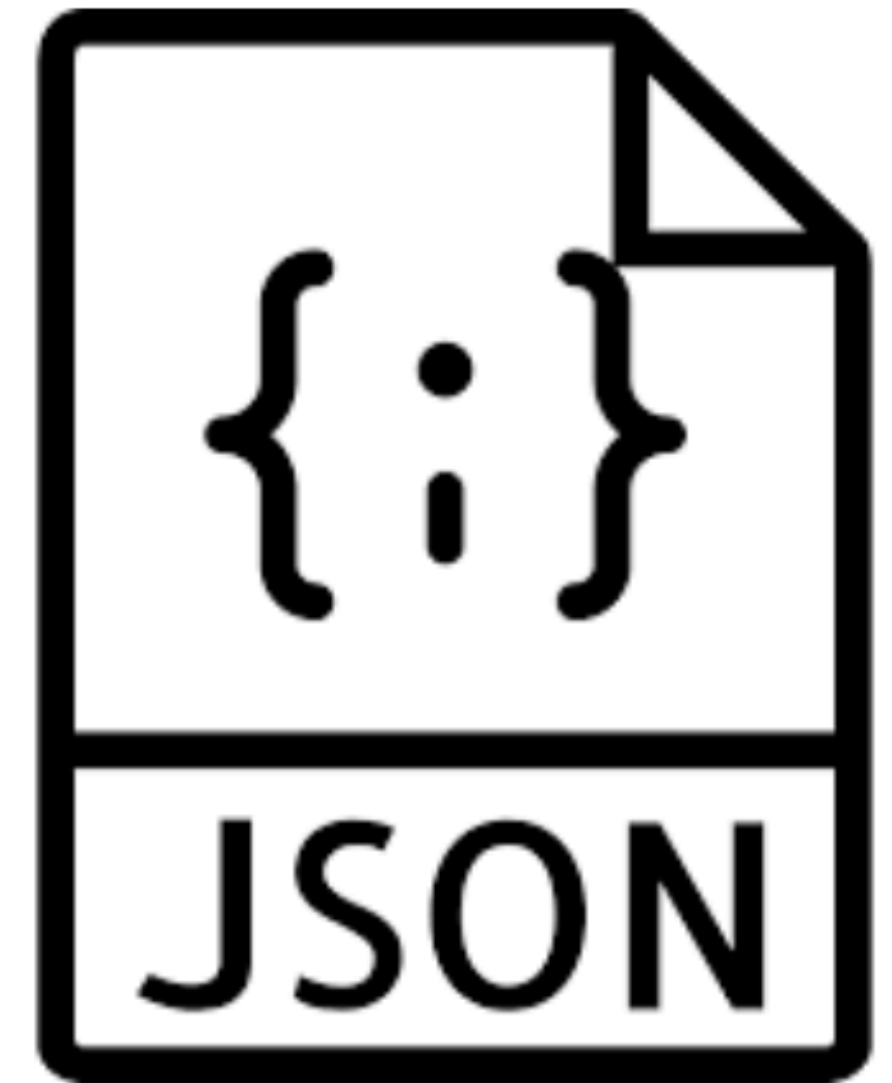


PlacemarkJSONStore

JSON Store



A new PlacemarkStore implementation - PlacemarkJSONStore - to persist placemarks to a JSON file.

PlacemarkStore Initialisation

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

Declare placemarks as
“PlacemarkStore” type

Then create a
PlaceMemStore on
initialisation

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

PlacemarkStore

```
class PlacemarkMemStore : PlacemarkStore, AnkoLogger {

    val placemarks = ArrayList<PlacemarkModel>()

    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
            logAll();
        }
    }

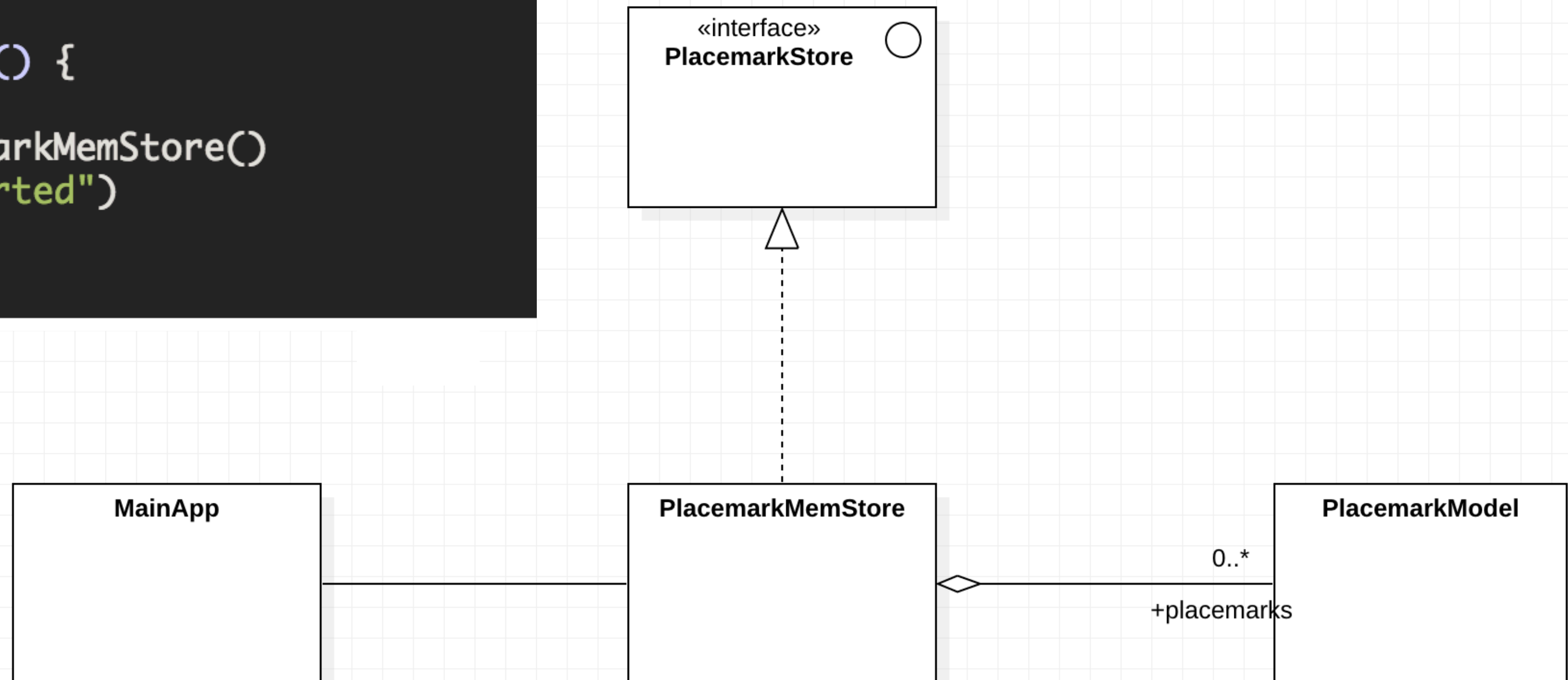
    override fun delete(placemark: PlacemarkModel) {
        placemarks.remove(placemark)
    }

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

PlacemarkMemStore

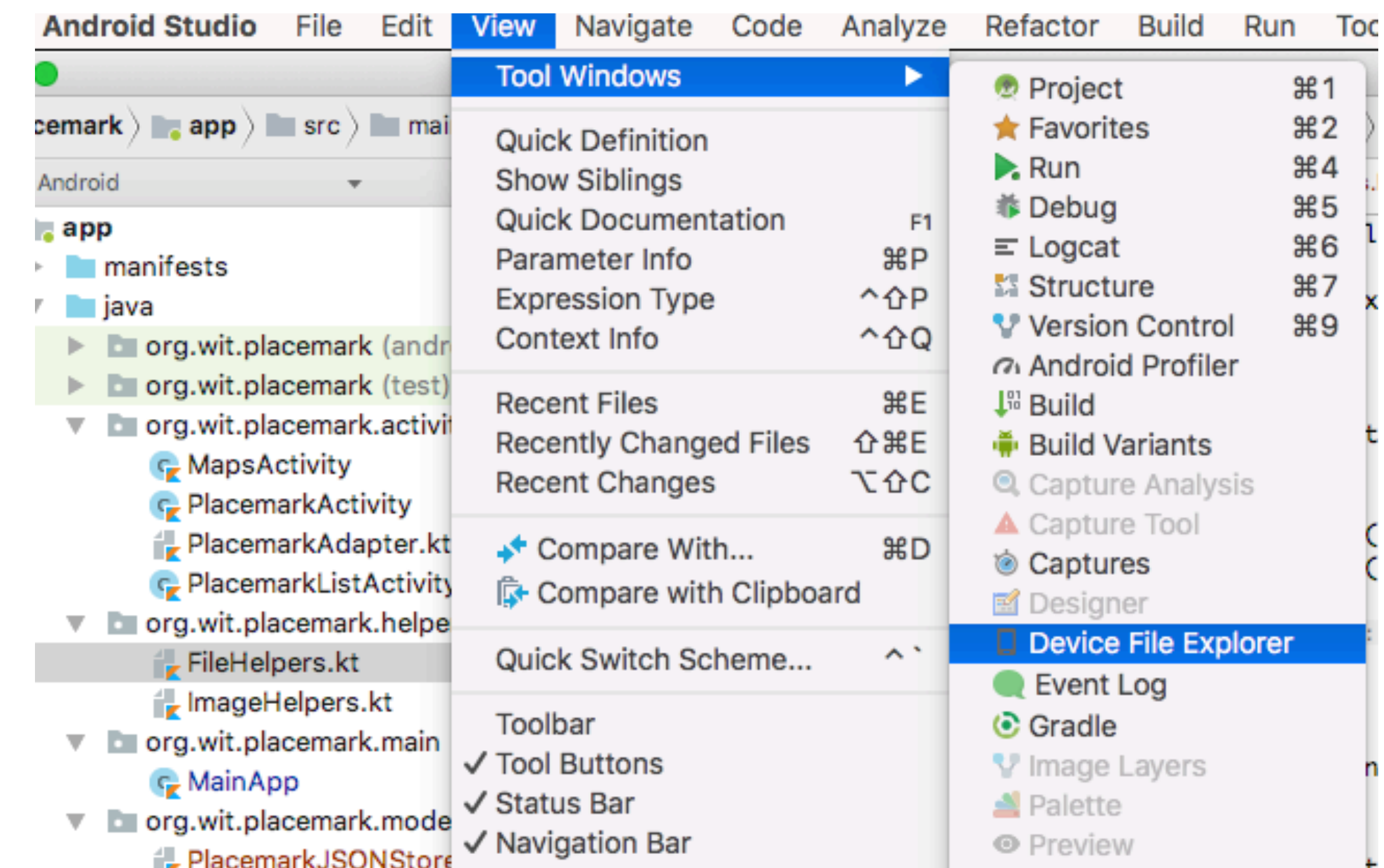
PlacemarkStore & PlacemarkMemStore

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

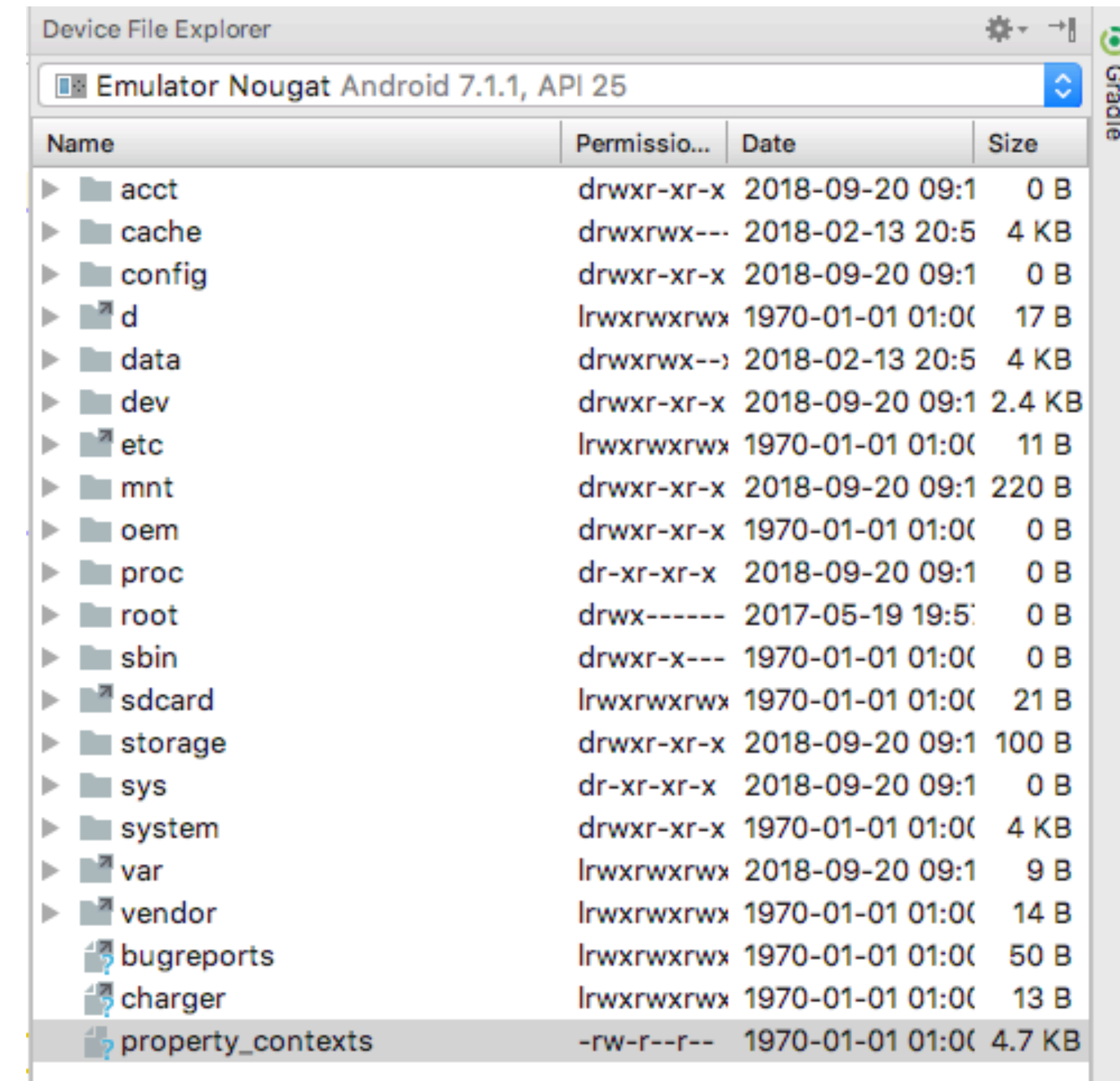


Android File System

Android Devices support a full filesystem

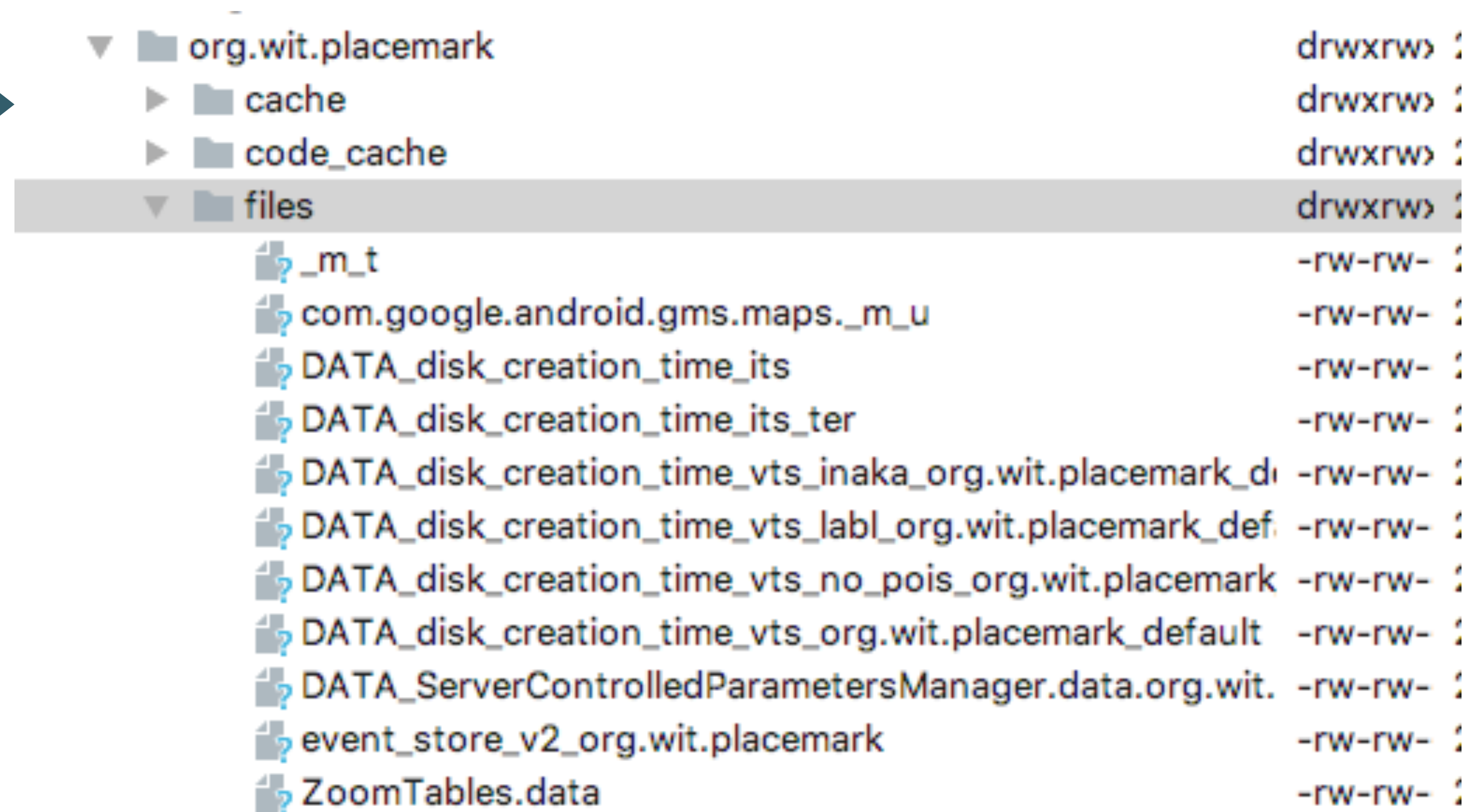
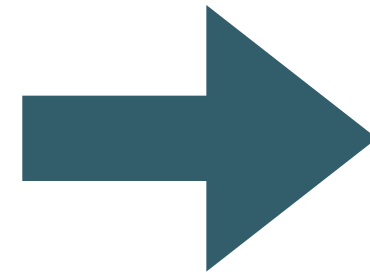


This file system
can be browsed
from Studio
(emulator or
actual device)



Application Folder

Applications will be largely confined to a specific folder created when the app is installed




We can read/write files too this folder - and also browse their contents in Studio

`/data/data/org.wit.placemark/files`

File Helper functions: write()

Simple function to write a string to a file

```
fun write(context: Context, fileName: String, data: String) {  
    try {  
        val outputStreamWriter = OutputStreamWriter(context.openFileOutput(fileName, Context.MODE_PRIVATE))  
        outputStreamWriter.write(data)  
        outputStreamWriter.close()  
    } catch (e: Exception) {  
        Log.e("Error: ", "Cannot read file: " + e.toString());  
    }  
}
```



Android 'context' object
required to locate and open
file in Application data folder

Uses standard
Java Streams
methods

File Helper functions: read()

```
fun read(context: Context, fileName: String): String {
    var str = ""
    try {
        val inputStream = context.openFileInput(fileName)
        if (inputStream != null) {
            val inputStreamReader = InputStreamReader(inputStream)
            val bufferedReader = BufferedReader(inputStreamReader)
            val partialStr = StringBuilder()
            var done = false
            while (!done) {
                var line = bufferedReader.readLine()
                done = (line == null);
                if (line != null) partialStr.append(line);
            }
            inputStream.close()
            str = partialStr.toString()
        }
    } catch (e: FileNotFoundException) {
        Log.e("Error: ", "file not found: " + e.toString());
    } catch (e: IOException) {
        Log.e("Error: ", "cannot read file: " + e.toString());
    }
    return str
}
```

Also uses standard Java Streams methods + context to open file

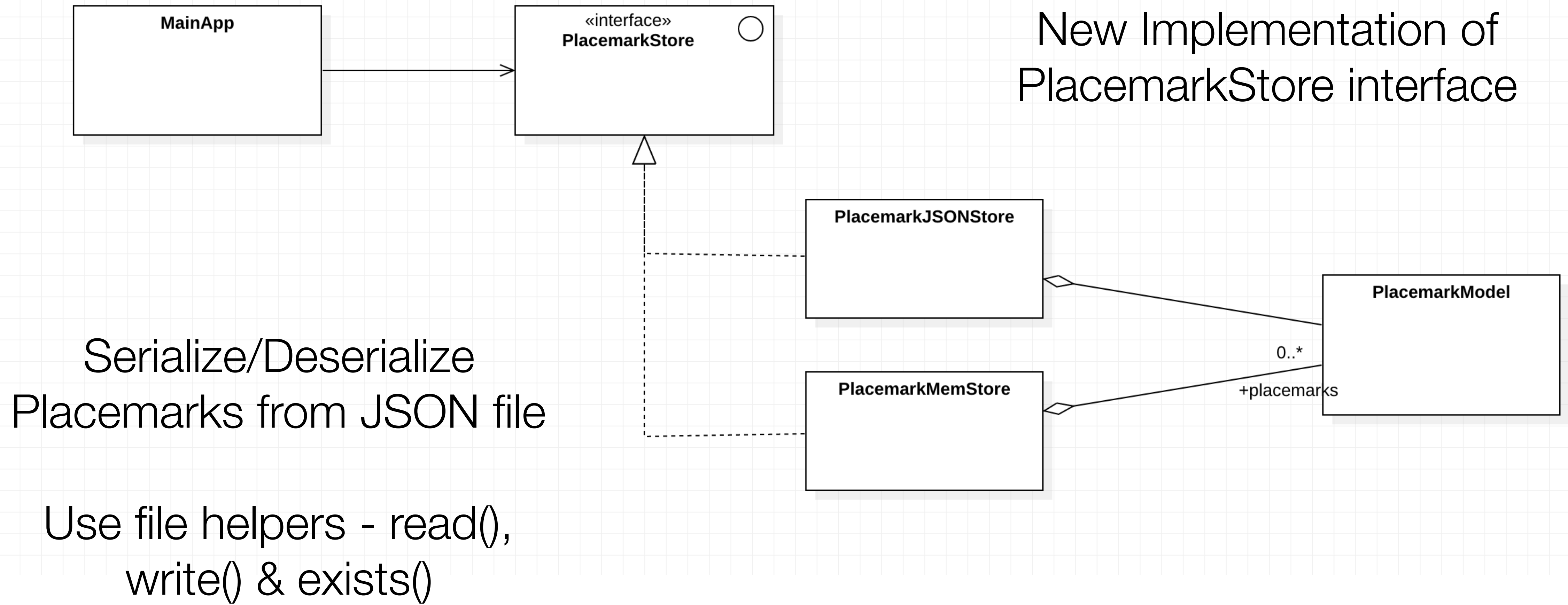
Read string line by line and return complete file contents.

Could be optimised - but keep simple for debug purposes (for the moment)

File Helper functions: exists()

```
fun exists(context: Context, filename: String): Boolean {  
    val file = context.getFileStreamPath(filename)  
    return file.exists()  
}
```

PlacemarkJSONStore Specification



Google Library to support JSON Encoding/ Decoding in Java

📖 README.md

Gson

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

There are a few open-source projects that can convert Java objects to JSON. However, most of them require that you place Java annotations in your classes; something that you can not do if you do not have access to the source-code. Most also do not fully support the use of Java Generics. Gson considers both of these as very important design goals.

Goals

- Provide simple `toJson()` and `fromJson()` methods to convert Java objects to JSON and vice-versa
- Allow pre-existing unmodifiable objects to be converted to and from JSON
- Extensive support of Java Generics
- Allow custom representations for objects
- Support arbitrarily complex objects (with deep inheritance hierarchies and extensive use of generic types)

build.gradle

```
implementation "com.google.code.gson:gson:2.8.5"
```

PlacemarkJSONStore

```
package org.wit.placemark.models

import android.content.Context
import com.google.gson.Gson
import com.google.gson.GsonBuilder
import com.google.gson.reflect.TypeToken
import org.jetbrains.anko.AnkoLogger
import org.wit.placemark.helpers.*
import java.util.*

val JSON_FILE = "placemarks.json"
val gsonBuilder = GsonBuilder().setPrettyPrinting().create()
val listType = object : TypeToken<java.util.ArrayList<PlacemarkModel>>() {}

fun generateRandomId(): Long {
    return Random().nextLong()
}

class PlacemarkJSONStore : PlacemarkStore, AnkoLogger {

    val context: Context
    var placemarks = mutableListOf<PlacemarkModel>()

    constructor (context: Context) {
        this.context = context
        if (exists(context, JSON_FILE)) {
            deserialize()
        }
    }

    override fun findAll(): MutableList<PlacemarkModel> {
        return placemarks
    }
}
```

```
override fun create(placemark: PlacemarkModel) {
    placemark.id = generateRandomId()
    placemarks.add(placemark)
    serialize()
}

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

override fun delete(placemark: PlacemarkModel) {
    ...
}

private fun serialize() {
    val jsonString = gsonBuilder.toJson(placemarks, listType)
    write(context, JSON_FILE, jsonString)
}

private fun deserialize() {
    val jsonString = read(context, JSON_FILE)
    placemarks = Gson().fromJson(jsonString, listType)
}
}
```

imports

```
import android.content.Context
import com.google.gson.Gson
import com.google.gson.GsonBuilder
import com.google.gson.reflect.TypeToken
import org.jetbrains.anko.AnkoLogger
import org.wit.placemark.helpers.*
import java.util.*
```

PlacemarkJSONStore -
preamble

```
val JSON_FILE = "placemarks.json"
```

Filename for
placemarks store

Helper
variables for
use with
GSON parser

```
val gsonBuilder = GsonBuilder().setPrettyPrinting().create()
val listType = object : TypeToken<java.util.ArrayList<PlacemarkModel>>() {}.type
```

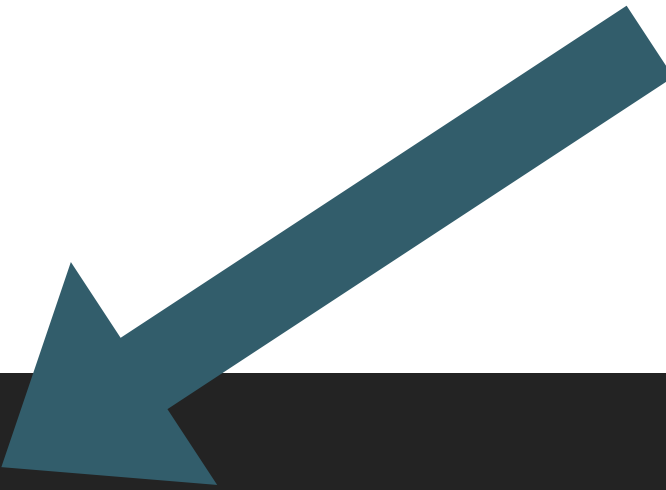
```
fun generateRandomId(): Long {
    return Random().nextLong()
}
```

Unique ID
generator


Methods to
Read & Write
Placemarks
Array to/from
Json file

```
private fun serialize() {  
    val jsonString = gsonBuilder.toJson(placemarks, listType)  
    write(context, JSON_FILE, jsonString)  
}  
  
private fun deserialize() {  
    val jsonString = read(context, JSON_FILE)  
    placemarks = Gson().fromJson(jsonString, listType)  
}
```

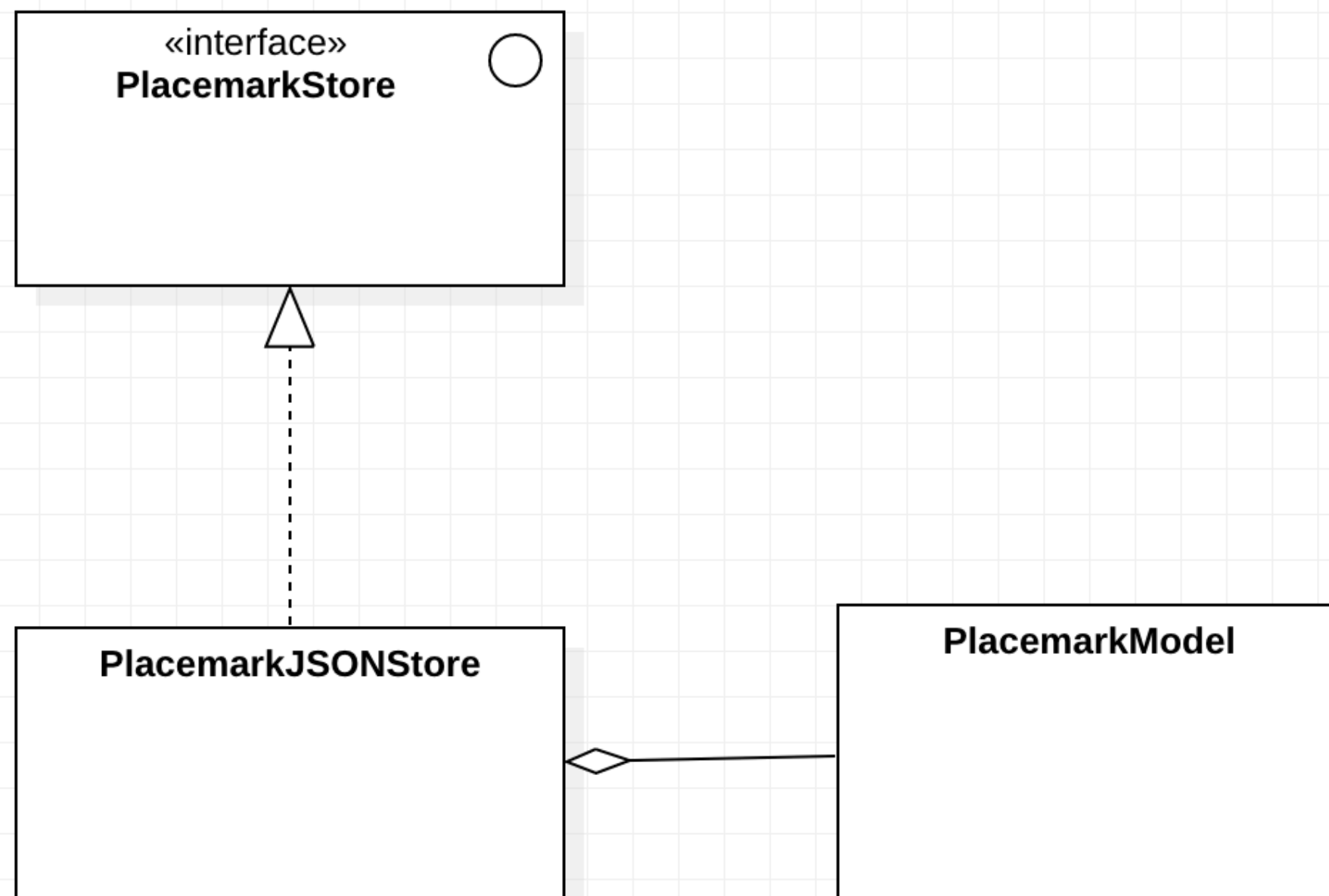
Gson parser converts
Placemarks to JSON
string



Gson parser converts a
JSON string to
Placemarks list



Load placemarks when Store created



```
class PlacemarkJSONStore : PlacemarkStore, AnkoLogger {

    val context: Context
    var placemarks = mutableListOf<PlacemarkModel>()

    constructor (context: Context) {
        this.context = context
        if (exists(context, JSON_FILE)) {
            deserialize()
        }
    }

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

    override fun create(placemark: PlacemarkModel) {
        placemark.id = generateRandomId()
        placemarks.add(placemark)
        serialize()
    }

}
```

Save place marks whenever each Placemark created

```

package org.wit.placemark.models

import android.content.Context
import com.google.gson.Gson
import com.google.gson.GsonBuilder
import com.google.gson.reflect.TypeToken
import org.jetbrains.anko.AnkoLogger
import org.wit.placemark.helpers.*
import java.util.*

val JSON_FILE = "placemarks.json"
val gsonBuilder = GsonBuilder().setPrettyPrinting().create()
val listType = object : TypeToken<java.util.ArrayList<PlacemarkModel>>() {}.type

fun generateRandomId(): Long {
    return Random().nextLong()
}

class PlacemarkJSONStore : PlacemarkStore, AnkoLogger {

    val context: Context
    var placemarks = mutableListOf<PlacemarkModel>()

    constructor (context: Context) {
        this.context = context
        if (exists(context, JSON_FILE)) {
            deserialize()
        }
    }

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

```



```

override fun create(placemark: PlacemarkModel) {
    placemark.id = generateRandomId()
    placemarks.add(placemark)
    serialize()
}

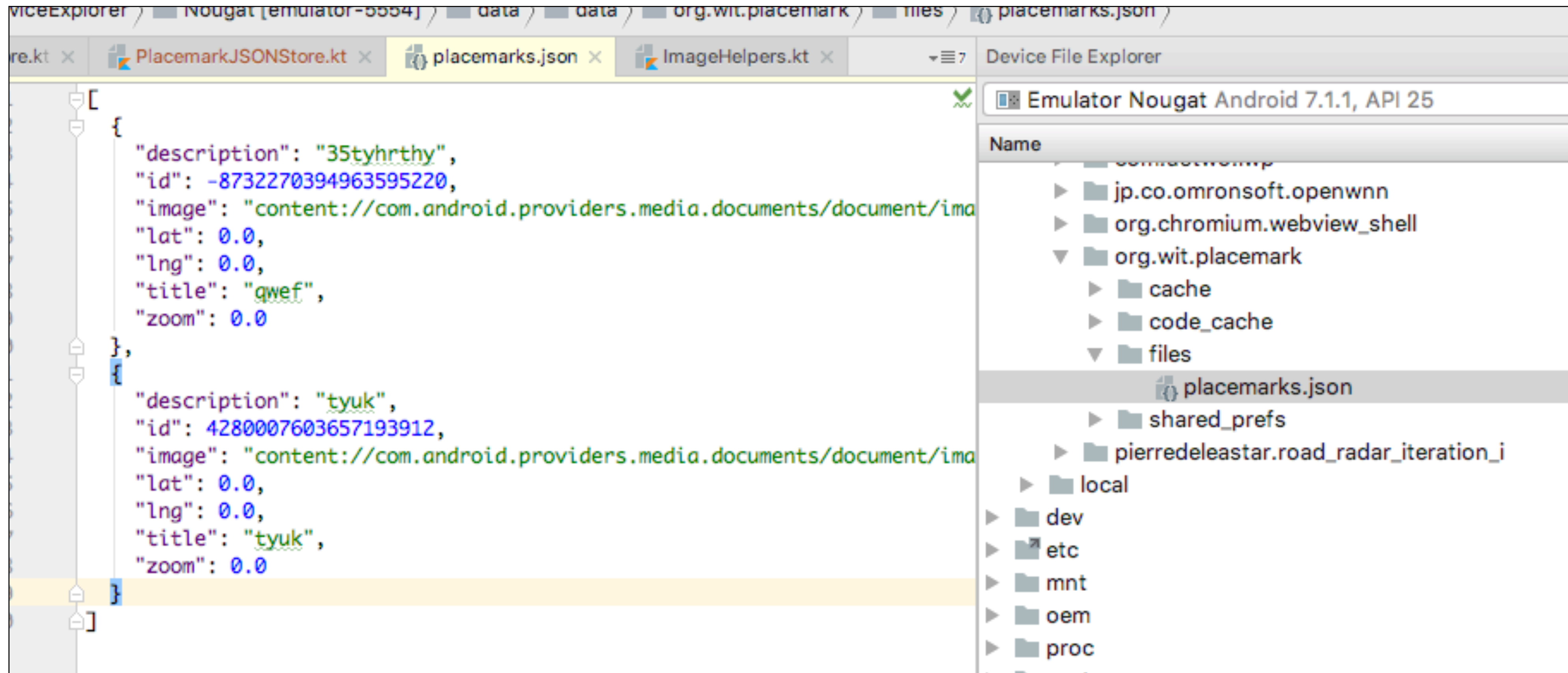
override fun update(placemark: PlacemarkModel) {
    ...
}

override fun delete(placemark: PlacemarkModel) {
    ...
}

private fun serialize() {
    val jsonString = gsonBuilder.toJson(placemarks, listType)
    write(context, JSON_FILE, jsonString)
}

private fun deserialize() {
    val jsonString = read(context, JSON_FILE)
    placemarks = Gson().fromJson(jsonString, listType)
}
}

```

Browse File in Studio

