Retrofit and Room

Android 102:
Networking and Persistence

Kshitij Chauhan

@haroldadmin @haroldadmin kshitijchauhan.me

Why?

- Almost all Android apps use Networking and Databases.
- Networking lets your app connect to the outside world.
- Databases are the best way to persist any non-trivial amount of data in your app.
- They can work together to create great user experiences.

Networking

Working with APIs

Again, why?

- Phones can not serve as centralized data stores, so we need servers.
- Even when we can do the task on-device, we should not.
- Servers are powerful, phones are not.
- Processing a lot of data on a phone is a drain on its resources: Battery, CPU, Memory.
- As good citizens on an Android phone, our apps should consume as little resources as possible.

How?

- We communicate with servers using ***APIs***.
- An API is a way of exchanging data between different computers.
- A ***client*** makes a ***request*** to a ***server*** for some ***data***, and the server sends a ***response*** with the requested data.
- The most popular method of requesting data from servers is a ***REST API***.

REST APIs

Representational State Transfer

What is a REST API?

- The REST protocol is modeled using HTTP requests.
- Four common operations: Create, Read, Update, Delete (CRUD).
- Existing request types in the HTTP protocol serve these use cases quite well.
- HTTP responses are accompanied by a response code in the 2xx-5xx range. Servers use standard response codes to communicate more information.

Example

- To request a server to send you list of articles in the news feed, we make a GET request for all articles, and the server responds with a list of articles in a format which we can understand.
- To create a new article on the server, we will send it a POST request with the data of the article in a format which the server can understand.
- ...and so on.

In all requests and responses, it is important to share data in a **format which both the client and server can understand.**

JSON

Javascript Object Notation

JSON

- The data between a client and the server needs to be transferred in a language-independent format, so that it can be parsed by both regardless of their programming language or operating system.
- The most popular format today for this is JSON.
- JSON is not tied to Javascript in any way.

Parsing JSON

- JSON is just text structured in a special way.
- In JSON, data is represented using two primitives: Objects and Arrays.
- Objects are key-value pairs, and Arrays are collections of objects

JSON Objects

Key-Value pairs

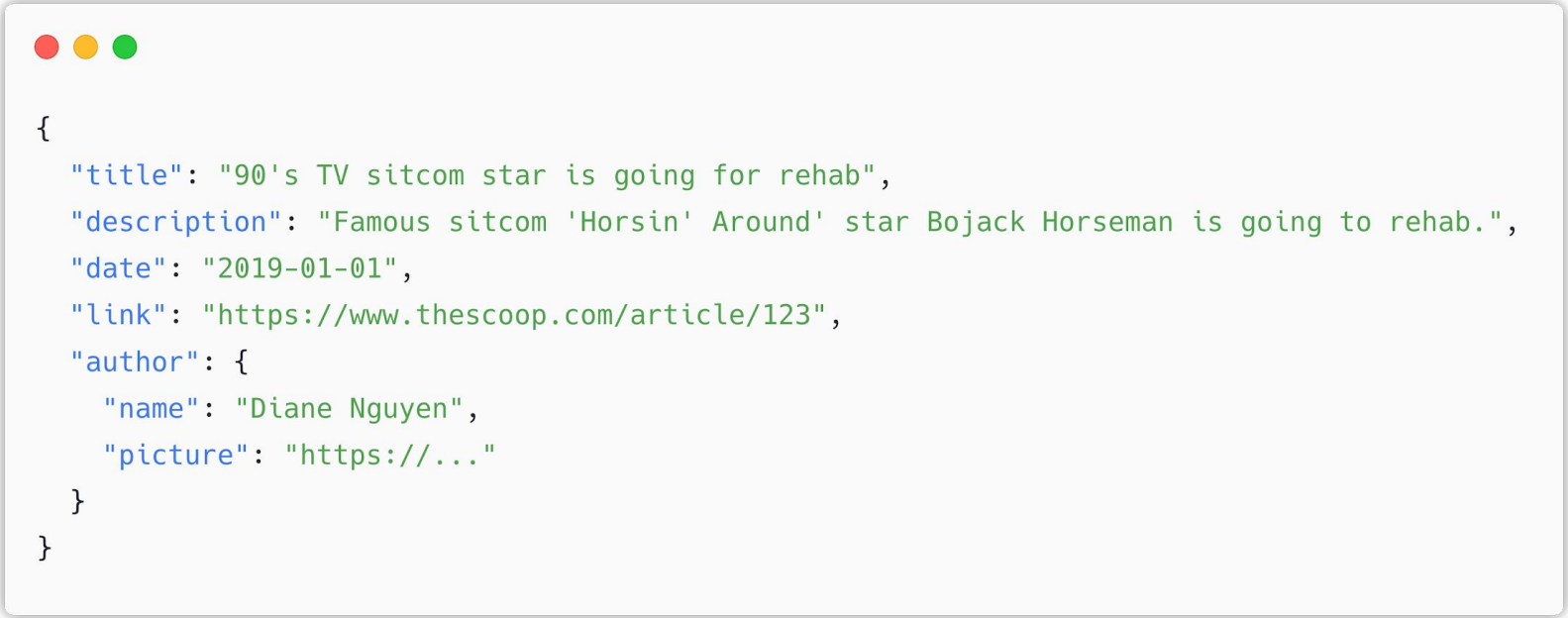
JSON Object

```
{
  "title": "90's TV sitcom star is going for rehab",
  "description": "Famous sitcom 'Horsin' Around' star Bojack Horseman is going to rehab.",
  "date": "2019-01-01",
  "link": "https://www.thescoop.com/article/123"
}
```

JSON Objects

- An object starts with a *{* and ends with a *}*.
- Between the braces is a collection of key-value pairs.
- Everything is represented as a string.
- Objects can have nested objects in them.

Nested Objects



```
{
  "title": "90's TV sitcom star is going for rehab",
  "description": "Famous sitcom 'Horsin' Around' star Bojack Horseman is going to rehab.",
  "date": "2019-01-01",
  "link": "https://www.thescoop.com/article/123",
  "author": {
    "name": "Diane Nguyen",
    "picture": "https://..."
  }
}
```

JSON Arrays

Collections of objects

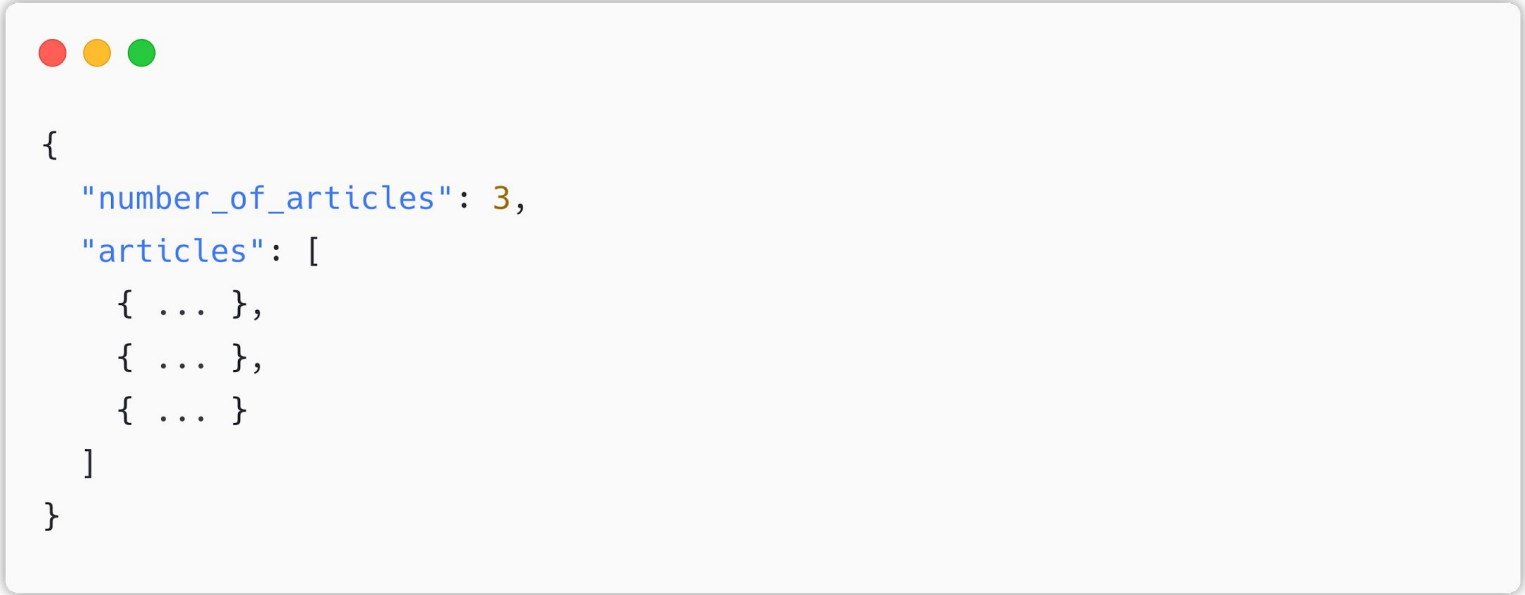
JSON Arrays

```
[
  {
    "title": "90's TV sitcom star is going for rehab",
    "description": "Famous sitcom 'Horsin' Around' star Bojack Horseman is going to rehab.",
    "date": "2019-01-01",
    "link": "https://www.thescoop.com/article/123",
   },
  {
    "title": "Bojack Horseman helps woman in rehab escape",
    "description": "'Horsin' Around' star Bojack Horseman helps a woman escape from his...",
    "date": "2019-01-02",
    "link": "https://www.thescoop.com/article/124",
  }
]
```

JSON Arrays

- JSON Arrays are used for representing a series of objects.
- Unlike objects which use curly braces, arrays use *[* and *]*
- Objects and Arrays can be nested in each other.

Nesting Objects and Arrays



```
{
  "number_of_articles": 3,
  "articles": [
    { ... },
    { ... },
    { ... }
  ]
}
```

With the basics out of the way, let's move on to how do we do these things on Android.

While JSON and REST APIs are very flexible, it is important to understand that there is a very well established contract by the API server about the structure of data it is going to send.

Every API has a different format. We need to consult their documentation to understand how to communicate with them.

Retrofit

*RESTin' on Android made easy**

* Well, not *that* easy

Retrofit

- Retrofit makes networking on Android easy.
- It is an HTTP client for Android, written by Square.
- Pretty much the standard networking library used by every Android app.

https://square.github.io/retrofit/

Live coding demo

We will work with the awesome SpaceX API

`https://github.com/r-spacex/SpaceX-API`

Sooo why did we crash? 🤔

Because Retrofit does not know how to parse JSON to create Java/Kotlin objects.

Let's fix that next.

Moshi

Parsing JSON on Android

Moshi

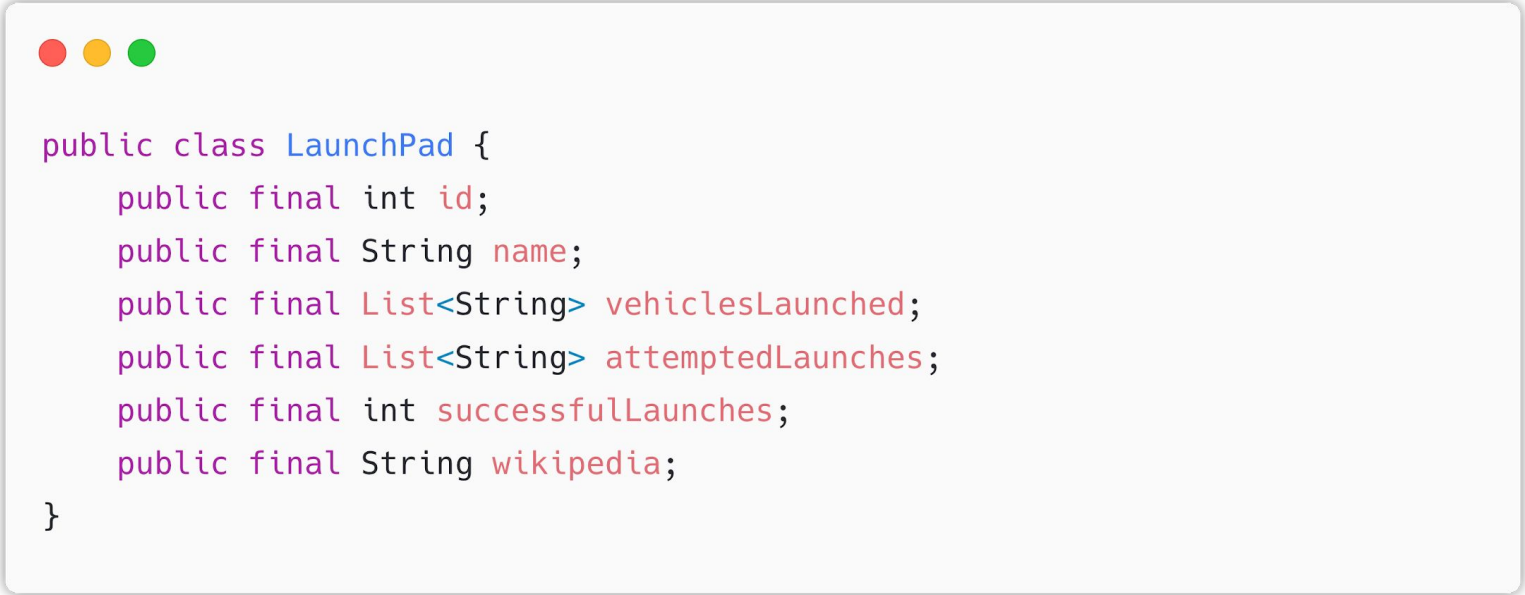
- Retrofit needs something to help it parse responses in JSON to Java/Kotlin objects.
- The task of converting JSON to POJOs is called **“*Deserialization*”**.
- The reverse process is called ***“Serialization”.***
- Deserializing JSON responses is where Moshi comes.

https://github.com/square/moshi/

How to use Moshi

- Annotate the fields in Model classes with their equivalent JSON names.
- Add the Moshi-Converter to Retrofit
- And that’s it.

Turn this...



```
public class LaunchPad {
    public final int id;
    public final String name;
    public final List<String> vehiclesLaunched;
    public final List<String> attemptedLaunches;
    public final int successfulLaunches;
    public final String wikipedia;
}
```

...Into this

```
public class LaunchPad {
    @Json(name = "id")
    public final int id;

    @Json(name = "name")
    public final String name;

    @Json(name = "vehicles_launches")
    public final List<String> vehiclesLaunched;

    @Json(name = "attempted_launches")
    public final int attemptedLaunches;

    @Json(name = "successful_launches")
    public final int successfulLaunches;

    @Json(name = "wikipedia")
    public final String wikipedia;
}
```

Live coding demo

Setting up Moshi

So why did that crash again? 😓

To understand that, we need to learn about Threads.

Threads and Asynchrony

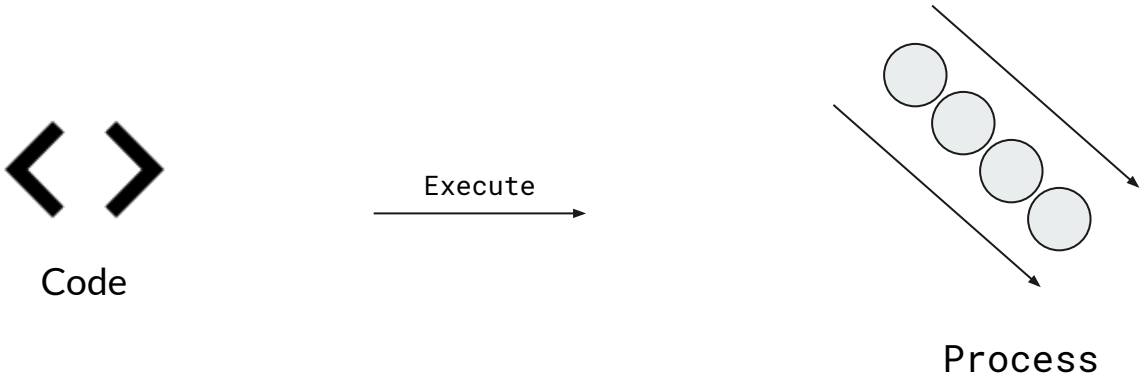
Doing multiple things at once

Processes

- A program is a passive entity. It does not do anything unless it is executed.
- Upon execution, the operating system starts a “Process” which executes our program.
- Each process has its own memory space.
- The process reads instructions from our program and executes them.
- Modern operating systems can run multiple processes simultaneously.

Code is executed in a process

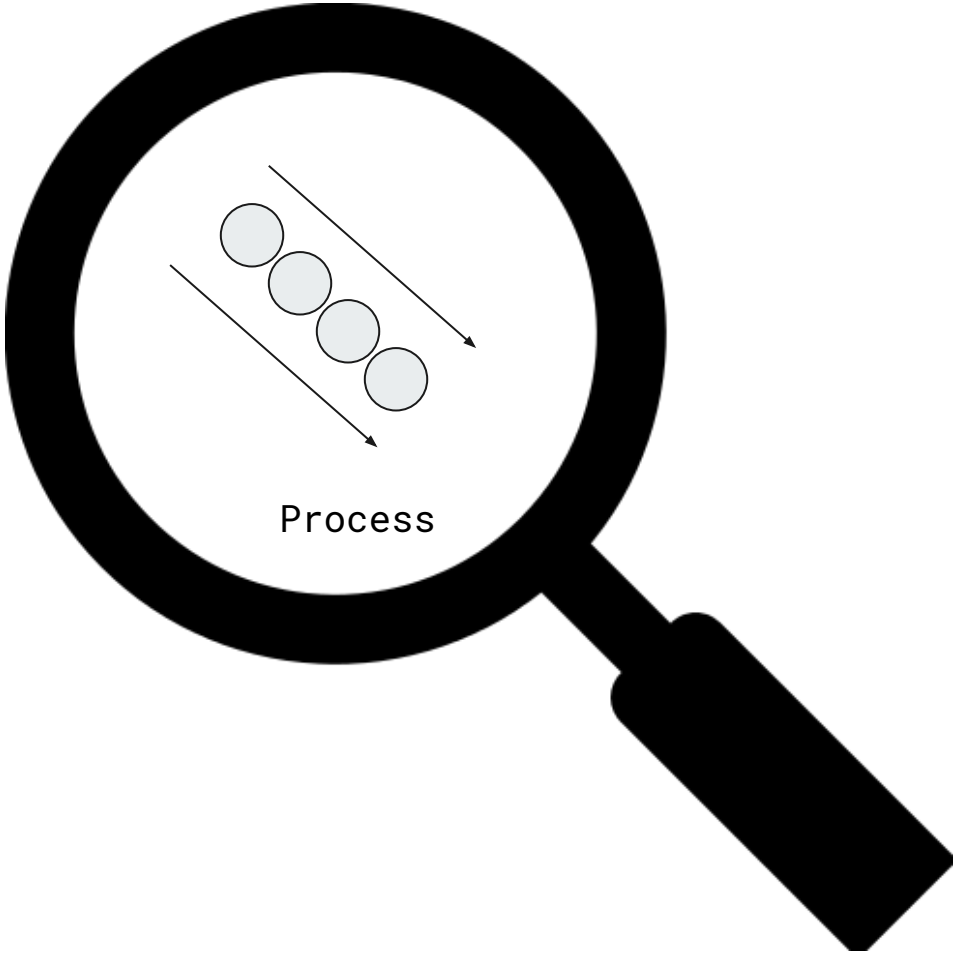
Each Process contains its own memory space and series of instructions

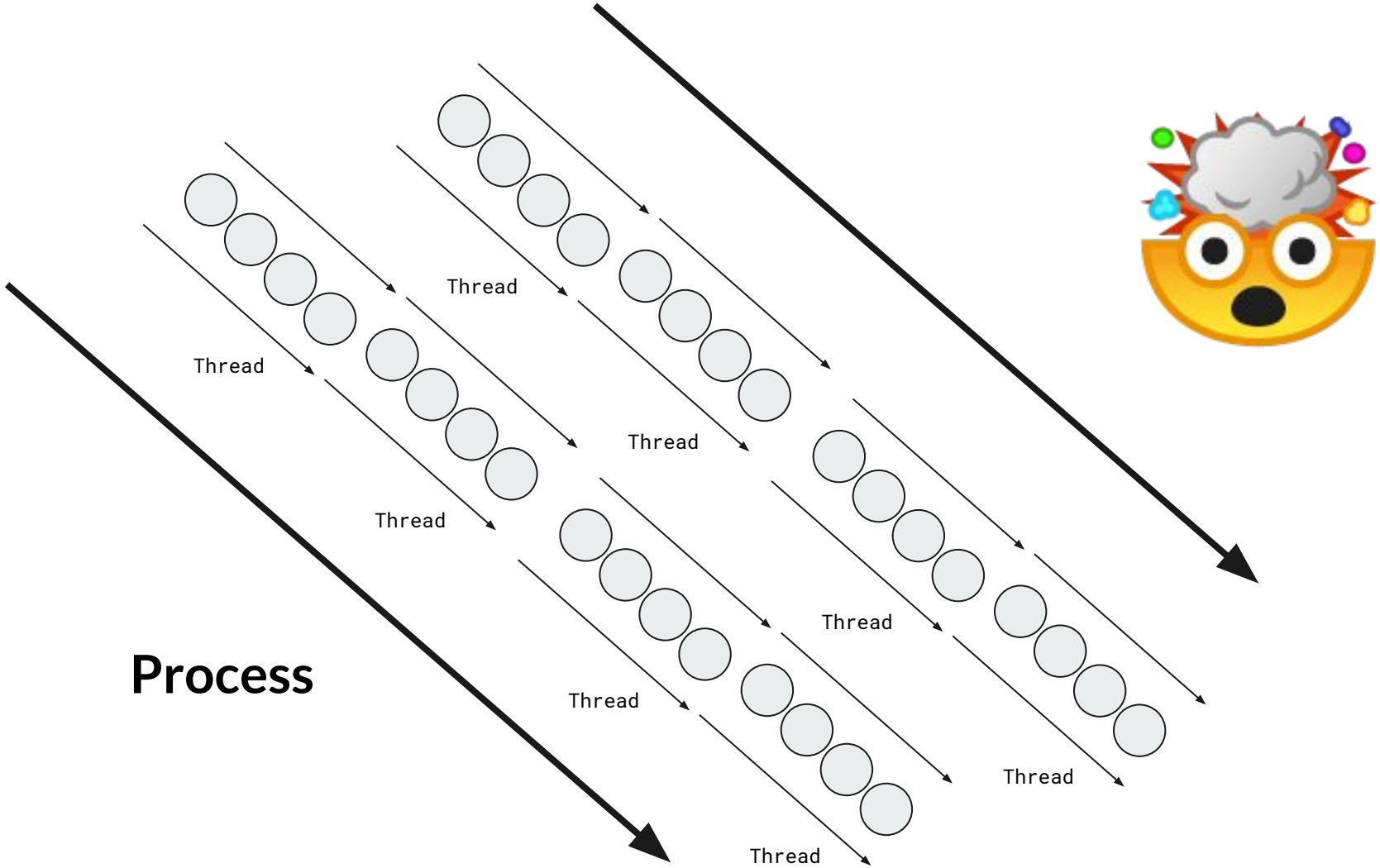


Apps on Android also execute in a process.

Processes are made up of smaller elements called “threads”

- Threads are like smaller, lighter processes.
- A process can have as many threads as it wants, until it runs out of memory.
- Each thread in a process has its own memory space inside the process’s memory space.
- Each thread can access its own memory space, as well as the process’s memory space.
- Threads can communicate with each other.


Process


Thread
Thread
Thread
Thread
Thread
Thread
Thread
Thread
Process

Threads

- When the OS starts executing a program, it creates one process, with one thread.
- This thread is called as the main thread, and it executes the program.
- The main thread can create new threads when needed, and they run in parallel.
- This is known as multi-threading.
- When used effectively, it can greatly speed up a program on multi-core processors.

Theads

- A program executes sequentially
- A thread can not move on to execute another task before it can complete the current one.
- Therefore, while a thread is processing one task it is busy, and can not handle any other tasks.

Threads on Android

- On Android, the main thread is responsible for drawing the UI and responding to user inputs such as touches and button presses.
- When we execute a network request in our app, the main thread gets blocked because it is waiting for the a from the server.
- It can not perform UI updates or respond to user inputs during this time.
- This means our app freezes, or “hangs” from the perspective of the user.

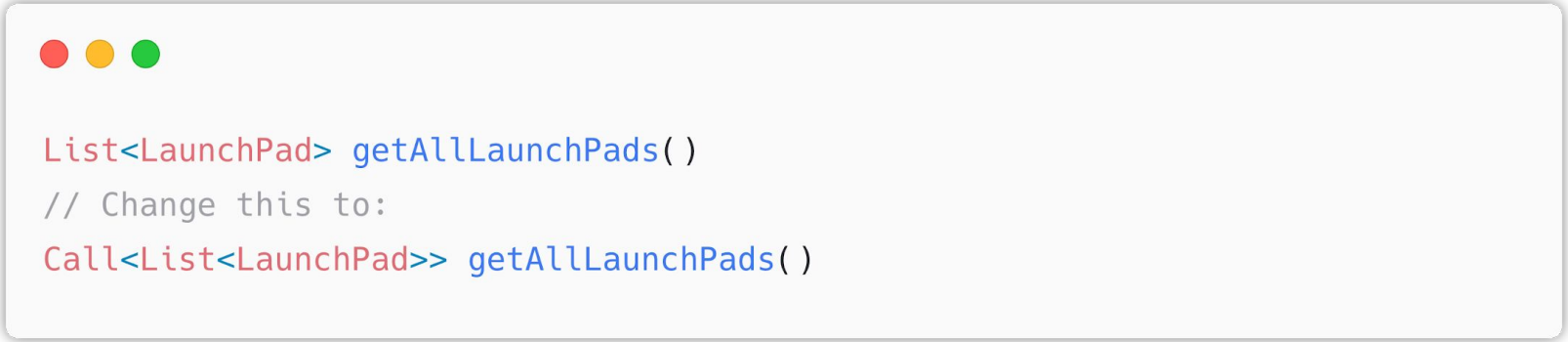
Android forbids this, and crashes our app as soon as we execute a network request on the main thread.

The solution is to execute the network request on a background thread.

Luckily, Retrofit makes this easy.

Asynchronous Requests

First, change the network request definition



```
List<LaunchPad> getAllLaunchPads()
// Change this to:
Call<List<LaunchPad>> getAllLaunchPads()
```

Asynchronous Requests

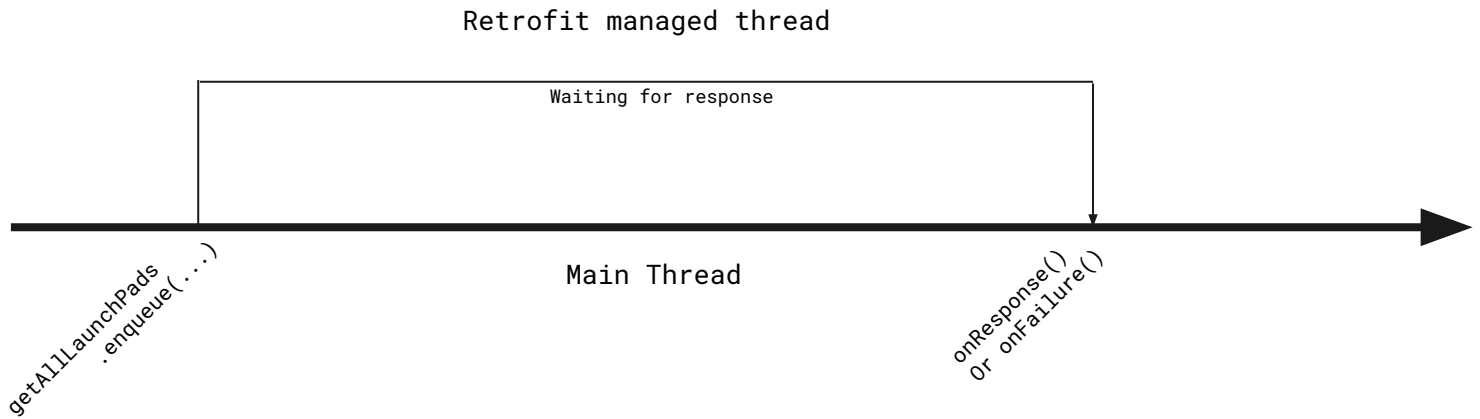
Second, change how the request is executed

```
List<LaunchPad> launchPads = launchPadService.getAllLaunchPads();

// Change this to:

launchPadService.getAllLaunchPads().enqueue(new Callback<List<LaunchPad>>() {
  @Override
  public void onResponse(Call<List<LaunchPad>> call, Response<List<LaunchPad>> response) {
    // Handle response
  }
  @Override
  public void onFailure(Call<List<LaunchPad>> call, Throwable t) {
    // Handle Error
  }
});
```

Order of events



Live coding demo

Asynchronous Retrofit Requests

So that's all about networking.

Moving on to Databases.

Databases

Persisting structured data

Databases

- Databases are used in almost every Android app.
- They provide us with a way to save data to local device storage.
- They are great for caching data, so that our app can work even when the user is offline.
- There is a wide variety of databases out there: MySQL, MongoDB, Firebase Firestore, etc.
- Android ships with the SQLite database out of the box.

SQLite

- SQLite is a tiny, fast, embedded database.
- We interact with it using SQL.
- SQL is a language for interacting with a particular class of databases.
- SQL makes it easy to perform CRUD operations on databases

Structure of an SQLite database

- Data is stored in SQLite inside tables.
- A table is used to model real world entity.
- A table has a schema, and a number of rows containing data conforming to that schema
- Each row represents an entity of the type of the object which the table is being used to model.

Tables

Table: Launchpad					
ID	Name	Status	Attempted Launches	Successful Launches	...
1	Kwajalein Atoll	Retired	5	2	...
2	CCAFS SLC 40	Active	45	43	...
...	...	...	...	...	...

How do we use SQL to query the database?

READ

```
SELECT (<column1>, <column2>, <column3>, ...)
FROM <table-name>
WHERE <some-condition>;
```

Reading from the database

- We can pass a wildcard character (*) in place of the column names to select all columns.
- The query returns to us all the rows which match the condition given in the ‘WHERE’ clause.
- If no ‘WHERE’ clause is specified, then the query returns all the rows in the table.

Fetching all launchpads

```
SELECT * FROM launchpad;
```

INSERT

```
INSERT INTO <table_name> (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

UPDATE

```sql
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

DELETE

```
DELETE FROM table_name WHERE condition;
```

All of this gets complicated pretty quickly.

Luckily, Room is here to help us.

Room

Persisting structured data

Working with Room

- Tables in the database are modeled as regular classes.
- Queries for Insertion, deletion, and updating are handled automatically.
- Database interactions are defined in DAO interfaces.
- Reading from the database is flexible, and powerful.

Working with Room

1. Define model classes
2. Define DAOs
3. Interact with database using DAOs

Defining Entities

```
@Entity(tableName = "launchpad")
public class LaunchPadDb {

    @ColumnInfo(name = "id")
    @PrimaryKey
    public final int id;

    @ColumnInfo(name = "name")
    public final String name;

    ...
}
```

Defining DAOs

```java
@Dao
public interface LaunchPadDao {

    @Query("SELECT * FROM launchpad")
    List<LaunchPadDb> getAllLaunchPads();

    @Insert
    void saveAllLaunchPads(List<LaunchPadDb> launchPads);

}
```

Defining Database

```
@Database(entities = {LaunchPadDb.class}, version = 1, exportSchema = false)
@TypeConverters(value = {Converters.class})
public abstract class LaunchPadDatabase extends RoomDatabase {

    public abstract LaunchPadDao getLaunchPadDao();

}
```

Live coding demo

Working with Room

How can we do better?

Observable Queries with Room

Automatic notifications when data changes

Observable Queries

- Observable Queries in Room notify us about changes in the data we are observing automatically.
- We can accomplish this using LiveData, a lifecycle-aware observable value holder.
- Room supports other observable streams too, including RxJava's Flowable/Observable, as well as Kotlin Coroutine Flow.
- We are going to use LiveData in this example.

LiveData

Lifecycle-aware observable data holder

LiveData

- LiveData allows us to observe changes in the value of something, and execute some code whenever this value changes.
- To use it with Room, we wrap the return type of our DAO methods with LiveData.

LiveData in DAO

```
@Query("SELECT * FROM launchpad")
List<LaunchPadDb> getAllLaunchPads();
// Change this to:
@Query("SELECT * FROM launchpad")
LiveData<List<LaunchPadDb>> getAllLaunchPads();
```

Observing LiveData

```
launchPadDao.getAllLaunchPads().observe(this, new Observer<List<LaunchPadDb>>() {
  @Override
  public void onChanged(List<LaunchPadDb> launchPadDbs) {
    // Handle new value
  }
});
```

Live coding demo

Observable Queries

Now that we are observing the database for all launchpads, whenever Room detects that we save data to this table our LiveData observer will be called with the new list of launchpads.

And that brings us to the end of this presentation.

Recycler Views Fragments
RxJava Kotlin Coroutines MVVM
MVI Dependency Injection
Multi-module apps Navigation
Room Persistence Firebase
Play Services Espresso JUnit

There's a LOT more stuff out there

Gradle builds take too long!

Newer Java features aren't available to me

It's overwhelming

Testing is difficult!

What is the difference between a DI library and service locator?

Heterogenous Recycler Views?

Jetpack Compose? Redux? State?

Why does Google hate me?

Just keep practicing. All it takes is time.

Retrofit and Room

https://github.com/dsc-dtu/Android-102-Room-Retrofit

Kshitij Chauhan

 @haroldadmin  @haroldadmin  kshitijchauhan.me