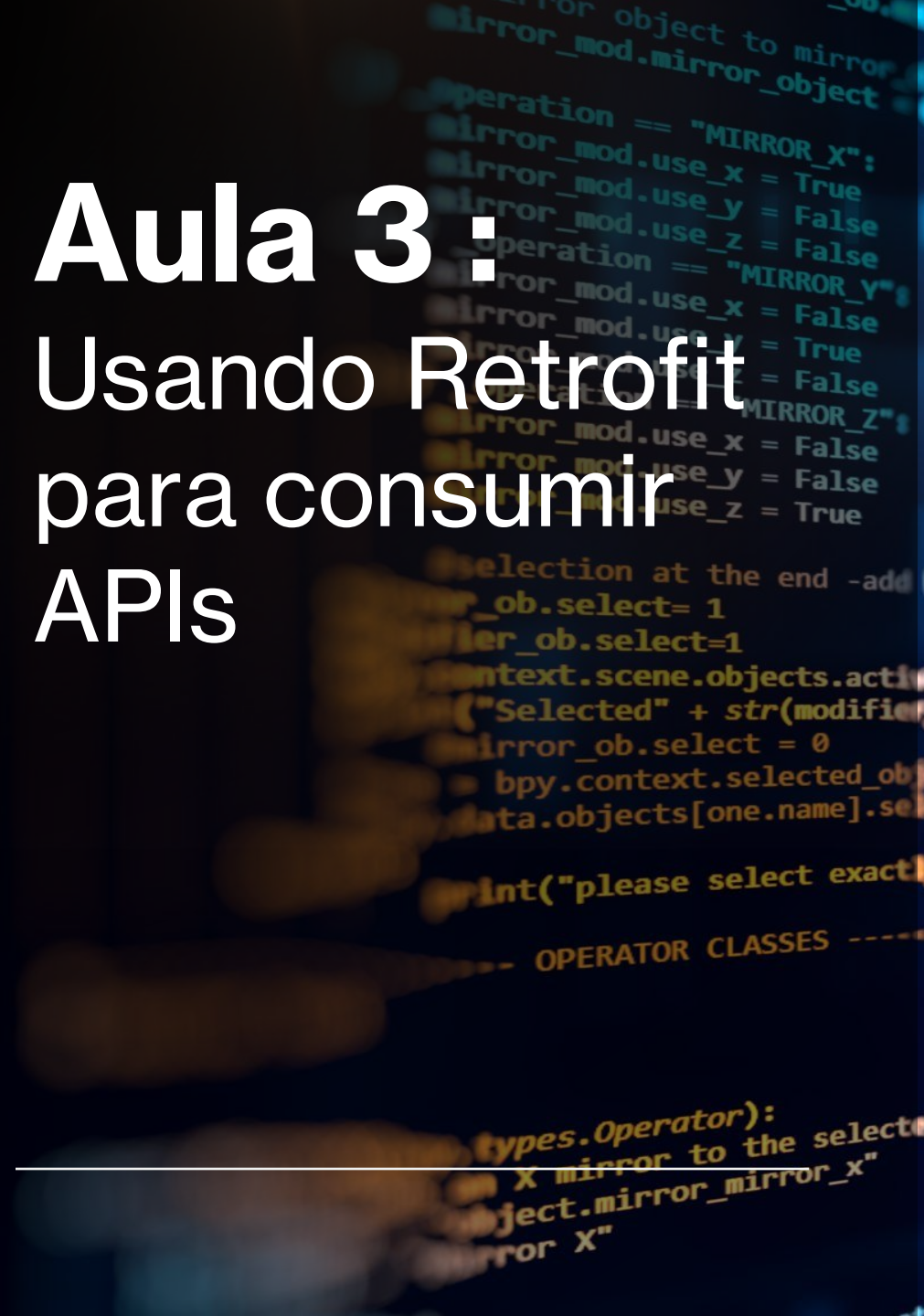
Aula 3 :
Usando Retrofit para consumir APIs



Introdução

- Nas aulas anteriores vimos como funcionam as API's Rest, nesta aula vamos começar a parte mais divertida, como conseguimos utilizar o Retrofit para fazer nosso aplicativo obter informações de um API

Porque escolhemos o Retrofit?

- Existem diversas forma diferentes de consumir API no Android, mas escolhemos o Retrofit pois ela é simples de utilizar e possui vários recursos úteis, além disso ela é sem dúvida a biblioteca mais utilizada no Android para a comunicação com API’s

O que é o Retrofit

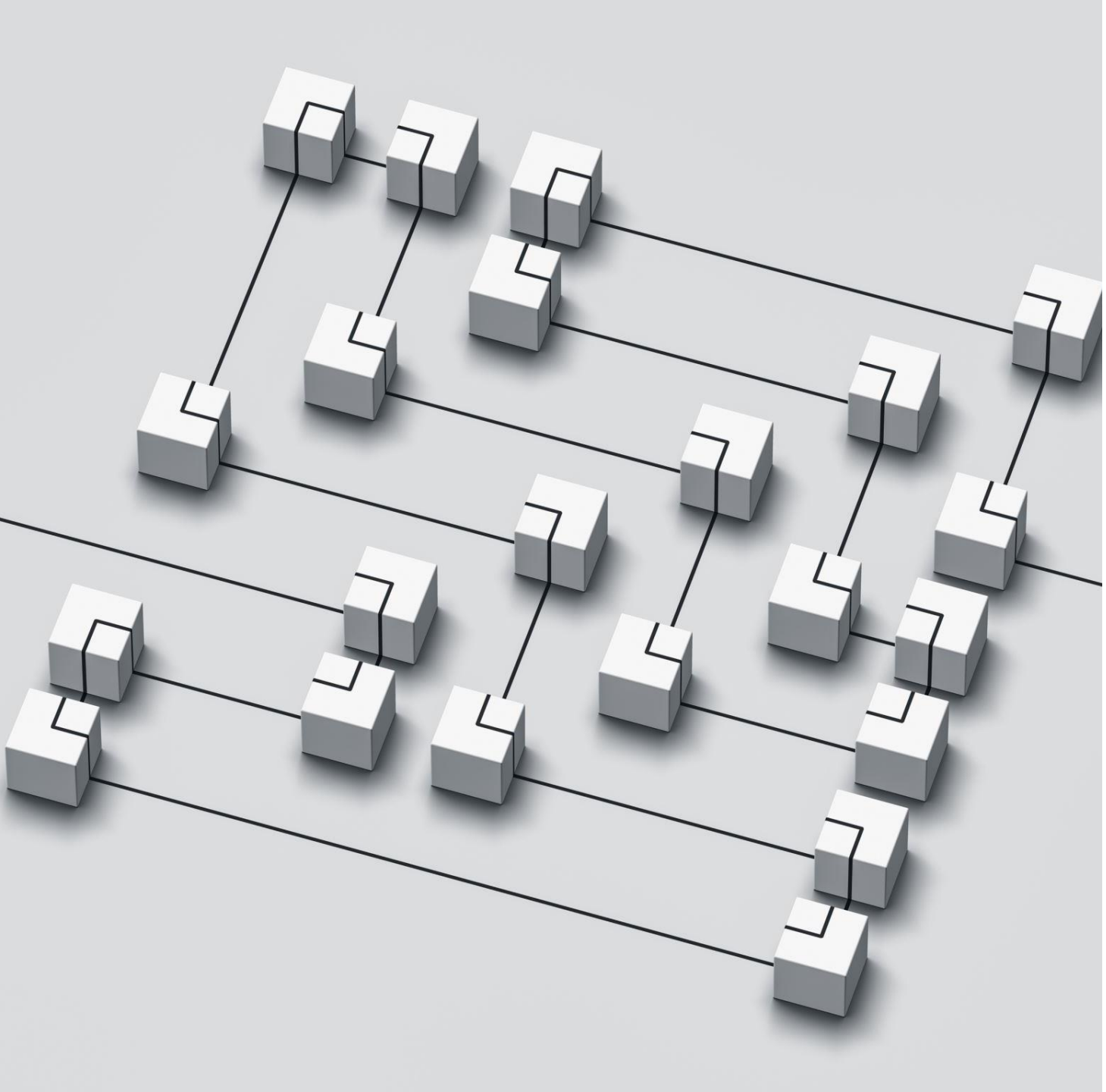
- É uma biblioteca que permite de forma simples, utilizando interfaces, criar requisições para uma API
- Usando o retrofit em conjunto com uma biblioteca de serialização converte automaticamente o JSON recebido em um objeto, facilitando seu uso

Como criar uma requisição no retrofit

- Cada requisição vai ser um método definido em uma interface
- Vamos utilizar uma anotação com o verbo e o endereço do recurso para definir qual o endpoint que vai ser consumido
- O retorno do nosso método vai ser do tipo **Call**, um objeto definido no retrofit que vai permitir depois executarmos a chamada

```
interface ComicsService {

    @GET( value: "v1/public/comics")
    fun getComics() : Call<Comic>

    @DELETE( value: "v1/public/comics")
    fun deleteComic() : Call<Data>

    @POST( value: "v1/public/comics")
    fun createComic() : Call<Data>

    @PUT( value: "v1/public/comics")
    fun updateComic() : Call<Data>
}
```

Requisição exemplo



Passando informações utilizando parâmetros na query

- Basta adicionar a anotação **@QUERY** passando entre parênteses o nome do parâmetro esperado pelo backend
- Logo depois da anotação teremos um parâmetro da função que representará a informação

```
@GET( value: "v1/public/comics")
fun getComics(@Query( value: "id") id: String, @Query( value: "available") isAvailable: Boolean): Call<Comic>

@POST( value: "v1/public/store")
fun createStore(@Query( value: "strange_parameter_v2") data: String)
```

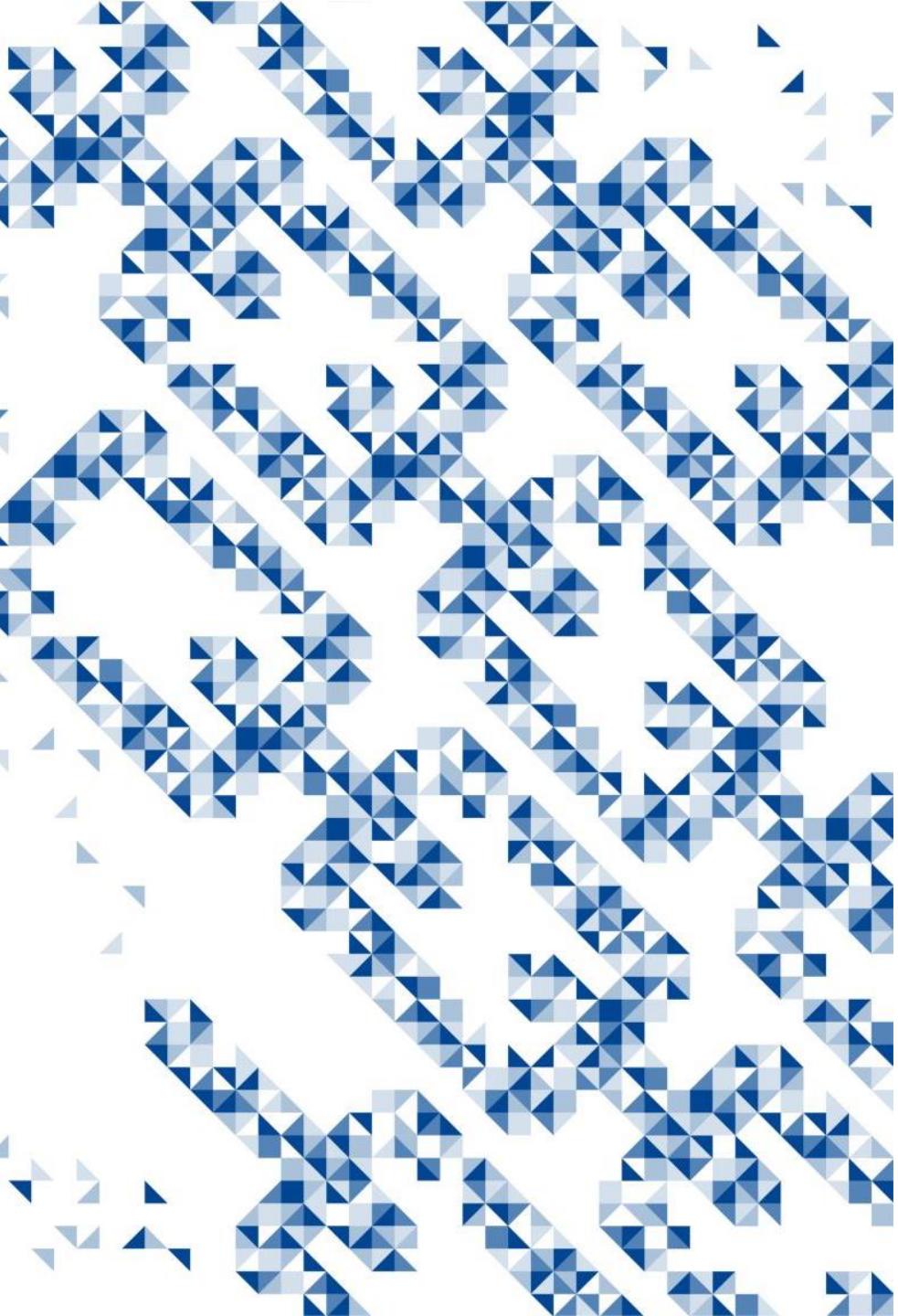
Requisição com parâmetro na query

Passando informações utilizando parâmetros no path

- Primeiro precisamos adicionar no path dentro da anotação do verbo http o nome do parâmetro entre **{}**
- Depois de forma similar a query, adicionamos um parâmetro na nossa função começando com um **@Path** e dentro do parênteses **o mesmo nome do parâmetro usado na anotação anterior**

Requisição com parâmetro no Path

```
@GET( value: "v1/public/comics/{comicsId}")
fun getComics(@Path( value: "comicsId") id: String): Call<Comic>

@POST( value: "v1/public/{comicsId}/images/{image_name}")
fun createStore(@Path( value: "comicsId") id: String, @Path( value: "image_name") image: String): Call<Data>
```



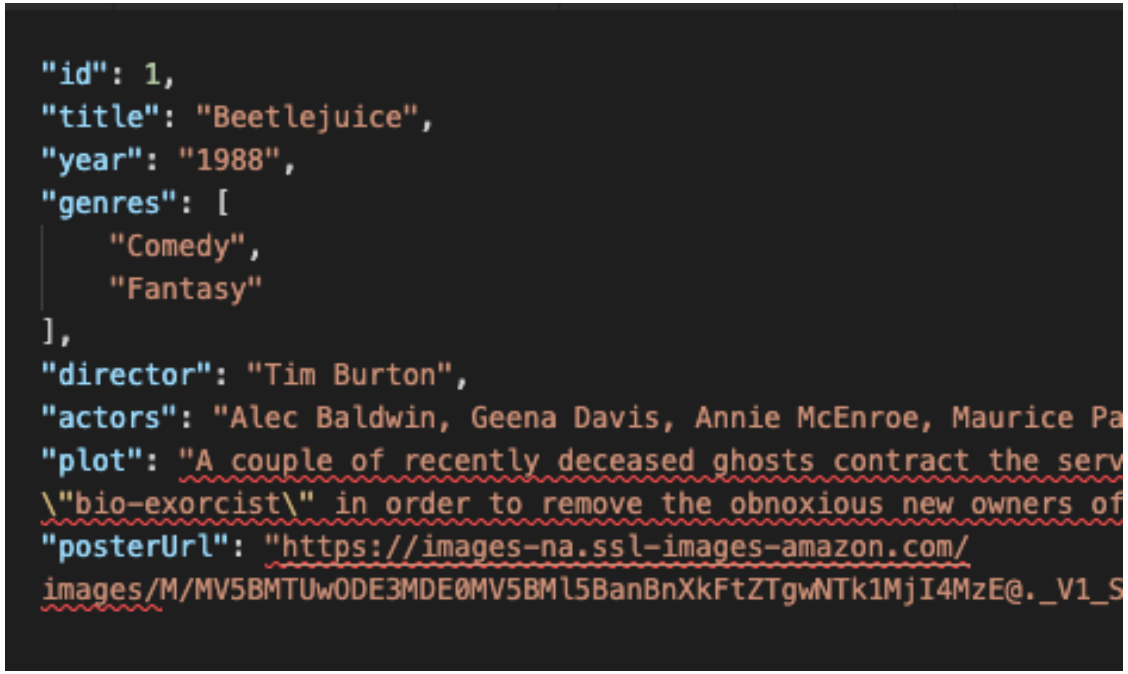
Convertendo a resposta em um objeto

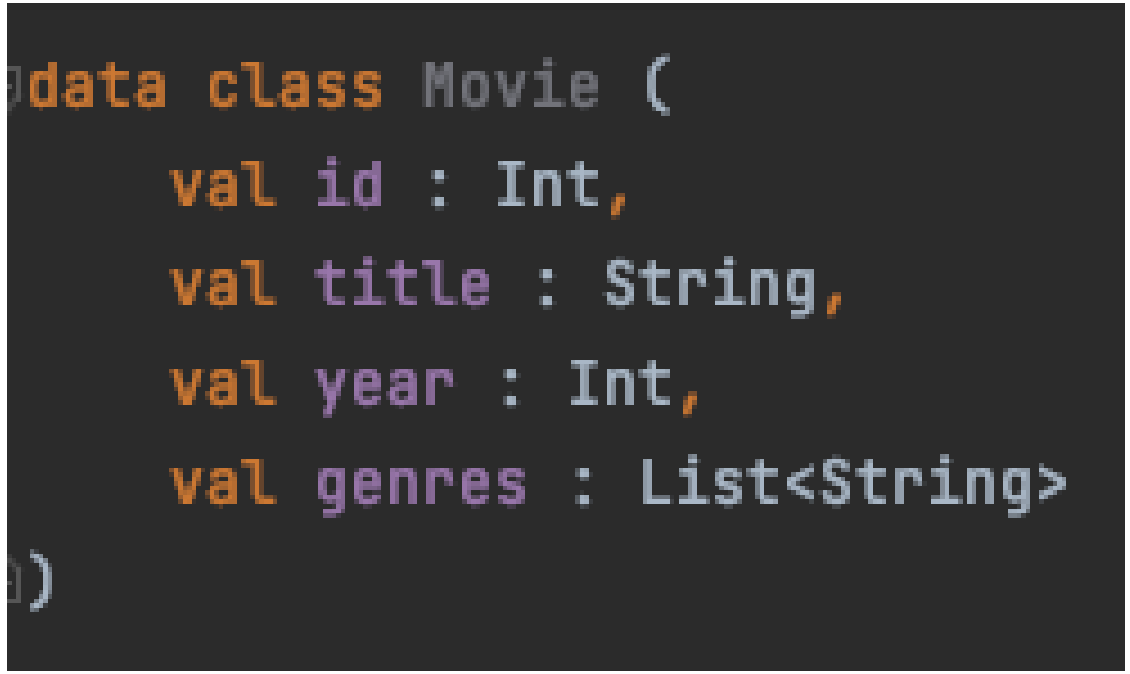
- O Retrofit permite utilizar uma série de adpater que vão converter automaticamente o JSON recebido nas respostas em Objetos
- Nos exemplos anterior, a resposta dos método era sempre **Call<T>** aonde **T** é objeto que vai conter a resposta convertida
- No nosso projeto vamos utilizar a biblioteca **Moshi** para fazer a conversão

Mapeamento JSON para Objeto

- Para conseguir mapear corretamente um JSON para um objeto, precisamos olhar para o JSON e entender quais são as suas propriedades e os tipos delas, depois criamos uma classe que representa aquele JSON
- Importante: **Não é necessário mapear todas as propriedades**, podemos ignorar as informações que não são necessárias e só mapear as que temos interesse

Exemplo de mapeamento de JSON para objeto





Melhorando a performance do Moshi

- Somente criando uma classe que representa o JSON já suficiente para o moshi conseguir fazer o mapeamente
- Mas o moshi funciona de forma mais otimizada se incluirmos a seguinte anotação nas classes que representam o JSON

```
@JsonClass(generateAdapter = true)
```

Criando uma instância do Retrofit

- **baseUrl** é a Url aonde se encontra o servidor que vamos utilizar, incluir aqui somente a informação comum a todos os serviços
- **Converter**: biblioteca que será utilizada para converter o JSON em objeto

```
val retrofit = Retrofit.Builder()
    .baseUrl( baseUrl: "https://api.example.com")
    .addConverterFactory(MoshiConverterFactory.create())
    .build()
```

Executando uma requisição de forma assíncrona

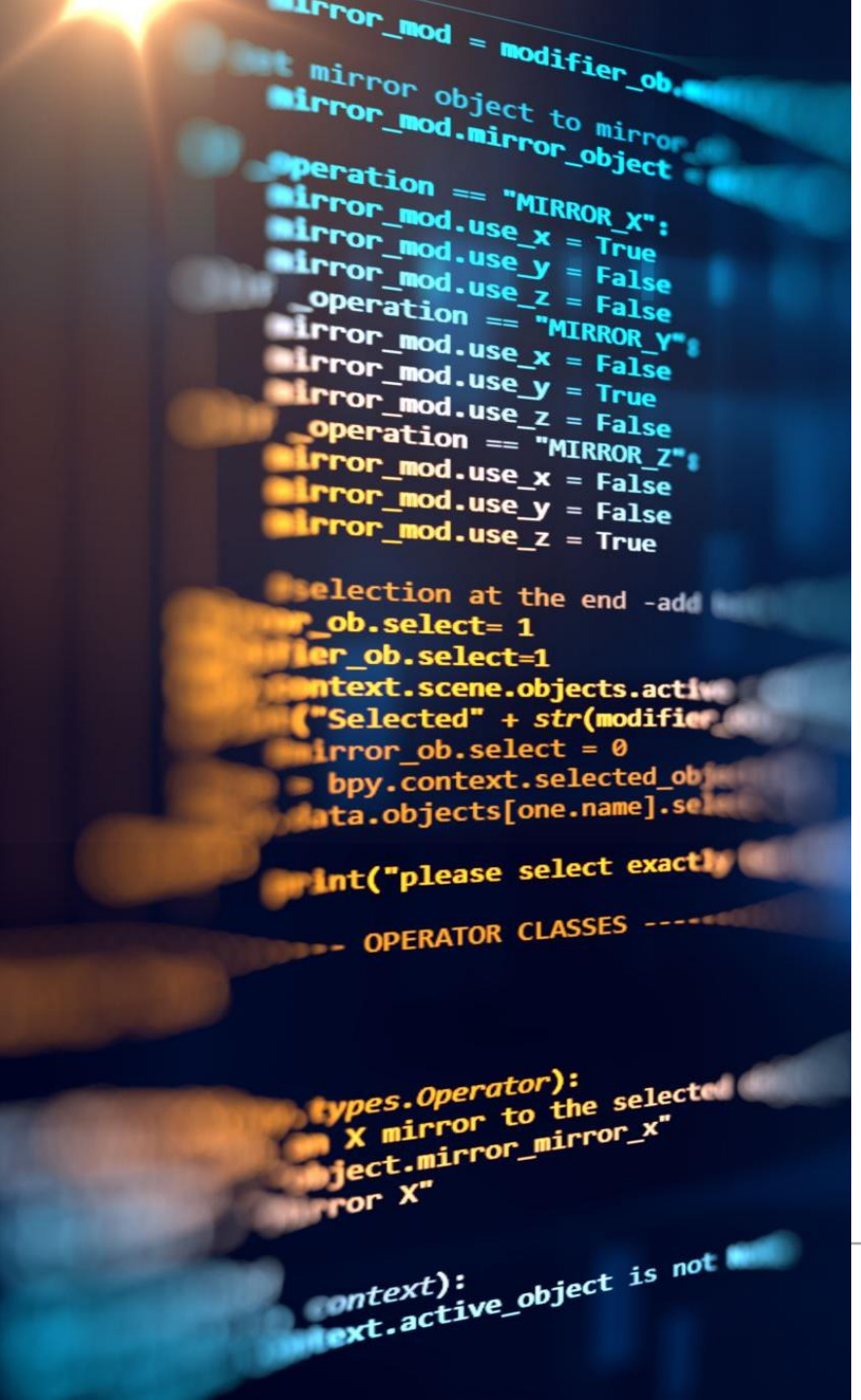
- Primeiro precisamos gerar um serviço baseado na nossa interface com as requisições
- Utilizando o serviço usamos o método **enqueue** para executar a chamada

```
val comicService = retrofit.create(ComicsService::class.java)

comicService.getComics( id: "test").enqueue(object : Callback<Comic>{
    override fun onResponse(call: Call<Comic>, response: Response<Comic>) {
        if(response.isSuccessful){
            //sucesso!!!
        }else{
            //algum erro ocorreu
        }
    }

    override fun onFailure(call: Call<Comic>, t: Throwable) {
        //erro
    }
})
}
```

Adicionando o Retrofit e o Moshi

- Incluir a seguinte dependência no arquivo build.gradle do módulo

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation("com.squareup.moshi:moshi-kotlin:1.13.0")
implementation "com.squareup.retrofit2:converter-moshi:2.9.0"
kapt("com.squareup.moshi:moshi-kotlin-codegen:1.13.0")
```



Bora programar?!

Quer saber mais a respeito?

- Retrofit – Documentação
- Retrofit - Github
- ***Moshi – Github***
- ***Documentação da API da marvel***