Method 1:

In the above code, we create an **observables** list by mapping each ID to an **Observable** using the Retrofit API service. We apply the **subscribeOn(Schedulers.io())** operator to perform the API requests on a background thread.

We use **Observable.merge(observables)** to combine all the observables into a single observable that emits each response one by one. We apply the **subscribeOn(Schedulers.io())** operator to continue the API requests on a background thread and **observeOn(AndroidSchedulers.mainThread())** to receive the responses on the main thread.

We subscribe to the combined observable using an **Observer** and implement its methods (**onSubscribe**, **onNext**, **onError**, and **onComplete**) to handle the disposable, response, error, and completion respectively

Steps

1. **Add the necessary dependencies to your project's build.gradle file:**

implementation 'io.reactivex.rxjava3:rxjava:3.1.1'  
implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'  
  
implementation 'com.squareup.retrofit2:retrofit:2.9.0'  
  
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'  
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'  
  
implementation 'com.squareup.retrofit2:adapter-rxjava3:2.9.0'

1. **Create an API interface Service class that defines your API endpoints using Retrofit annotations. For example:**

import com.example.kotlinrndproject.data.ApiResponse\_UserDetails  
import io.reactivex.rxjava3.core.Observable  
import okhttp3.ResponseBody  
import retrofit2.http.GET  
import retrofit2.http.Path  
  
  
interface ApiService {  
  
 @GET("todos")  
 fun getTodosList(): Observable<ResponseBody> //Observable<ApiResponse\_UserDetails>  
  
 @GET("todos/{id}")  
 fun getUserDetails(@Path("id") id: Int): Observable<ResponseBody> //Observable<ApiResponse\_UserDetails>  
  
 @GET("todos/{id}")  
 fun getUserDetails2(@Path("id") id: Int): Observable<ApiResponse\_UserDetails> //Observable<ApiResponse\_UserDetails>  
  
}

1. **Create a data class Api Response data class to represent the response from the API:**

class ApiResponse\_UserList : ArrayList<ApiResponse\_UserDetails>()

data class ApiResponse\_UserDetails(  
 val completed: Boolean,  
 val id: Int,  
 val title: String,  
 val userId: Int  
)

1. Create a Retrofit instance and an implementation of the APIRequest interface:

import retrofit2.Retrofit  
import retrofit2.adapter.rxjava3.RxJava3CallAdapterFactory  
import retrofit2.converter.gson.GsonConverterFactory  
  
  
object APIRequest {  
  
 private val retrofit: Retrofit = Retrofit.Builder()  
 .baseUrl("https://jsonplaceholder.typicode.com/")  
 .addConverterFactory(GsonConverterFactory.create())  
 .addCallAdapterFactory(RxJava3CallAdapterFactory.create())  
 .build()  
  
 val apiService: ApiService = retrofit.create(ApiService::class.*java*)  
}

1. **In your activity or fragment, use RxJava to make multiple API requests concurrently:**
   1. **Using RxJava Observable.merge method**

private fun CallMultipleAPI\_RxJaavOservable\_Marge(){  
 //Create list of Observable api request  
 val listOfAPiName = *listOf*(1, 2, 3, 4, 5)  
 val observables: List<Observable<ResponseBody>> = listOfAPiName.*map* **{** id **->** if(id == 1){  
 *apiService*.getTodosList().subscribeOn(Schedulers.io())  
 }else{  
 *apiService*.getUserDetails(id).subscribeOn(Schedulers.io())  
 }  
 **}** //Marge all request with Observable.merge function  
 Observable.merge(observables)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe(object : Observer<ResponseBody> {  
 override fun onSubscribe(d: Disposable) {  
 *println*("CallMultipleAPI\_RxJaavOservable\_Marge onSubscribe method call: ${d}")  
 // Handle disposable  
 }  
  
 override fun onNext(response: ResponseBody) {  
 // Handle response  
 *println*("CallMultipleAPI\_RxJaavOservable\_Marge onNext method call: $response")  
 var json = JSONTokener(response.string()).nextValue()  
 when (json) {  
 is JSONObject -> { //it is a JsonObject  
 *println*("CallMultipleAPI\_RxJaavOservable\_Marge JSONObject: $json")  
 }  
 is JSONArray -> { //it is a JsonArray  
 *println*("CallMultipleAPI\_RxJaavOservable\_Marge JSONArray: $json")  
 }  
 else -> { //handle the odd scenario  
 *println*("CallMultipleAPI\_RxJaavOservable\_Marge else block : $json")  
 }  
 }  
 }  
  
 override fun onError(e: Throwable) {  
 // Handle error  
 *println*("CallMultipleAPI\_RxJaavOservable\_Marge onError method call: ${e}")  
 }  
  
 override fun onComplete() {  
 // Handle completion  
 *println*("CallMultipleAPI\_RxJaavOservable\_Marge onComplete method call")  
 }  
 });  
}

* 1. Using RxJava **Observable.zip** method

@SuppressLint("CheckResult")  
fun CallMultipleAPI\_RXJava\_OservableZip() {  
  
  
 val ids = *listOf*(1, 2, 3, 4, 5, 6, 7, 8, 9)  
 val observables: List<Observable<ApiResponse\_UserDetails>> = ids.*map* **{** url **->** *apiService*.getUserDetails2(url)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 **}** Observable.zip(observables) **{** responses **->** // Handle the responses  
 val responseList = responses.*map* **{ it** as ApiResponse\_UserDetails **}** try {  
 val jsonObject = responseList.get(0).title;  
 *println*("getCallAPI jsonObject======= $jsonObject");  
 } catch (e: JSONException) {  
 e.printStackTrace()  
 *println*("getCallAPI Exception============ ${e.printStackTrace()}");  
 }  
 **}** .subscribe **{** responseList **->** // Handle the combined response list  
 // ...  
 *println*("getCallAPI responseList method call: $responseList")  
 **}**}

* 1. **Using Corouting with async and wait**

fun CallMultipleAPI\_Corouting\_RxJavaObserbable(){  
 val urls = *listOf*(1, 2, 3, 4, 5, 6, 7, 8, 9)  
 val coroutineScope = *CoroutineScope*(Dispatchers.Main)  
  
 coroutineScope.*launch* **{** val responses = *mutableListOf*<String>()  
 // Make API requests concurrently  
 val jobs = urls.*map* **{** url **->** delay(100)  
 *async* **{** delay(100)  
 *apiService*.getUserDetails2(url).subscribe(*Consumer* **{**it->  
 *println*("coroutineScope subscribe response Data: ${it} =========== ${it.title}")  
 **}**)  
 **}  
 }** *println*("coroutineScope subscribe !!!!!!! Before wait")  
 jobs.awaitAll()  
 *println*("coroutineScope subscribe !!!!!!! After wait")  
 // Handle the responses  
 // ...  
 **}**}