Practical class # 15 – Get data from the Internet

1) Download the starting code from the course website

android-basics-kotlin-mars-photos-app-starter.zip

OverviewFragment:

- This is the fragment displayed within the MainActivity. The placeholder text you saw in the previous step is displayed in this fragment.
- This class holds a reference to the OverviewViewModel object.
- The OverviewFragment has an onCreateView() function that inflates the fragment_overview layout using Data Binding, sets the binding lifecycle owner to itself, and sets the viewModel variable in the binding object to it.
- Because the lifecycle owner is assigned, any LiveData used in Data Binding will automatically be observed for any changes, and the UI will be updated accordingly.

OverviewViewModel:

- This is the corresponding view model for the OverviewFragment.
- This class contains a MutableLiveData property named _status along with its backing property. Updating the value of this property, updates the placeholder text displayed on the screen.
- The getMarsPhotos() method updates the placeholder response.

res/layout/fragment_overview.xml:

- This layout is set up to use data binding and consists of a single TextView.
- It declares an OverviewViewModel variable and then binds the status from the ViewModel to the TextView.

MainActivity.kt: the only task for this activity is to load the activity's layout, activity_main.

layout/activity_main.xml: this is the main activity layout with a single FragmentContainerView pointing to the fragment_overview, the overview fragment will be instantiated when the app is launched.

2) Add Retrofit dependencies

1. Open the project's top-level level build.gradle(Project: MarsPhotos) file. Notice the repositories listed under the repositories block. You should see two repositories, google(), mavenCentral().

```
repositories {
        google()
        mavenCentral()
}
```

2. Open module level gradle file, build.gradle (Module: MarsPhots.app).
3. In the dependencies section, add these lines for the Retrofit libraries:

```
// Retrofit
implementation "com.squareup.retrofit2:retrofit:2.9.0"
// Retrofit with Scalar Converter
implementation "com.squareup.retrofit2:converter-scalars:2.9.0"
```

4. Click sync now

3) Create a network layer

1. Create a new package called network. In your Android project pane, right-click on the package, com.example.android.marsphotos. Select **New > Package**. In the popup, append **network** to the end of the suggested package name.
2. Create a new Kotlin file under the new package **network**. Name it MarsApiService.
3. Open network/MarsApiService.kt. Add the following constant for the base URL for the web service.

```
private const val BASE_URL =
  "https://android-kotlin-fun-mars-server.appspot.com"
```

4. Just below that constant, add a Retrofit builder to build and create a Retrofit object.

```
private val retrofit = Retrofit.Builder()
```

5. Import retrofit2.Retrofit, when prompted.
6. Retrofit needs the base URI for the web service, and a converter factory to build a web services API. The converter tells Retrofit what to do with the data it gets back from the web service. In this case, you want Retrofit to fetch a JSON response from the web service, and return it as a String. Retrofit has a ScalarsConverter that supports strings and other primitive types, so you call addConverterFactory() on the builder with an instance of ScalarsConverterFactory.

```
private val retrofit = Retrofit.Builder()
  .addConverterFactory(ScalarsConverterFactory.create())
```

   Import retrofit2.converter.scalars.ScalarsConverterFactory when prompted.

7. Add the base URI for the web service using baseUrl() method. Finally, call build() to create the Retrofit object.

```
private val retrofit = Retrofit.Builder()
  .addConverterFactory(ScalarsConverterFactory.create())
  .baseUrl(BASE_URL)
  .build()
```

8. Below the call to the Retrofit builder, define an interface called MarsApiService, that defines how Retrofit talks to the web server using HTTP requests.

```
interface MarsApiService {
}
```

9. Inside the MarsApiService interface, add a function called getPhotos() to get the response string from the web service.

```
interface MarsApiService {
   fun getPhotos()
}
```

10. Use the @GET annotation to tell Retrofit that this is GET request, and specify an endpoint, for that web service method. In this case the endpoint is called photos.

```
interface MarsApiService {
   @GET("photos")
   fun getPhotos()
}
```

Import retrofit2.http.GET when requested.

When the getPhotos() method is invoked, Retrofit appends the endpoint photos to the base URL (which you defined in the Retrofit builder) used to start the request. Add a return type of the function to String.

```
interface MarsApiService {
   @GET("photos")
   fun getPhotos(): String
}
```

4) Declare an object to avoid creating different Retrofits

1. Outside the MarsApiService interface declaration, define a public object called MarsApi to initialize the Retrofit service. This is the public singleton object that can be accessed from the rest of the app.

```
object MarsApi {


}
```

2. Inside the MarsApi object declaration, add a lazily initialized retrofit object property named retrofitService of the type MarsApiService. You make this lazy initialization, to make sure it is initialized at its first usage. You will fix the error in the next steps.

```
object MarsApi {
  val retrofitService : MarsApiService by lazy {
    }
}
```

3. Initialize the retrofitService variable using the retrofit.create() method with the MarsApiService interface.

```
object MarsApi {
  val retrofitService : MarsApiService by lazy {
    retrofit.create(MarsApiService::class.java) }
}
```

The Retrofit setup is done! Each time your app calls MarsApi.retrofitService, the caller will access the same singleton Retrofit object that implements MarsApiService which is created on the first access. In the next task, you will use the Retrofit object you have implemented.

4. In MarsApiService, make getPhotos() a suspend function. So that you can call this method from within a coroutine.

```
@GET("photos")
suspend fun getPhotos(): String
```

5. Open overview/OverviewViewModel. Scroll down to the getMarsPhotos() method. Delete the line that sets the status response to "Set the Mars API Response here!". The method getMarsPhotos() should be empty now.

```
private fun getMarsPhotos() {


}
```

6. Inside getMarsPhotos(), launch the coroutine using viewModelScope.launch.

```
private fun getMarsPhotos() {
  viewModelScope.launch {
  }
}
```

Import androidx.lifecycle.viewModelScope and kotlinx.coroutines.launch when prompted.

7. Inside viewModelScope, use the singleton object MarsApi, to call the getPhotos() method from the retrofitService interface. Save the returned response in a val called listResult.

```
viewModelScope.launch {
   val listResult = MarsApi.retrofitService.getPhotos()
}
```

Import com.example.android.marsphotos.network.MarsApi when prompted.

8. Assign the result we just received from the backend server to the _status.value.

```
val listResult = MarsApi.retrofitService.getPhotos()
_status.value = listResult
```

9. Run the app, notice that the app closes immediately, it may or may not display an error popup.
10. Click the Logcat tab in Android Studio and note the error in the log, which starts with a line like this, "------- beginning of crash"

```
   --------- beginning of crash
22803-22865/com.example.android.marsphotos E/AndroidRuntime: FATAL EXCEPTION: OkHttp
Dispatcher
   Process: com.example.android.marsphotos, PID: 22803
   java.lang.SecurityException: Permission denied (missing INTERNET permission?)
...
```

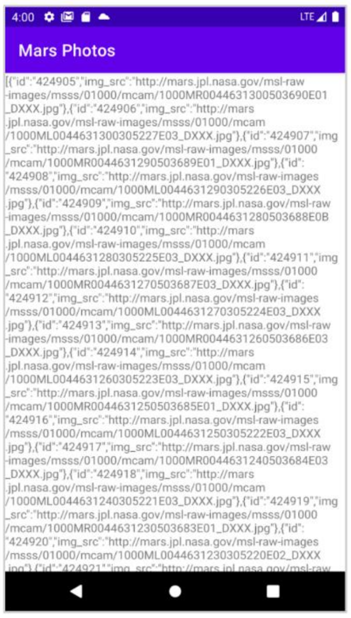
This error message indicates the app might be missing the INTERNET permissions. You will resolve this by adding internet permissions to the app in the next task.

5) Add Internet permission

1. Open manifests/AndroidManifest.xml. Add this line just before the <application> tag:

```
<uses-permission android:name="android.permission.INTERNET" />
```

2. Compile and run the app again. If you have a working internet connection, you should see the JSON text containing data related to Mars photos.



[{"id":"424905","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631300503690E01_DXXX.jpg"},{"id":"424906","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631300305227E03_DXXX.jpg"},{"id":"424907","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631290503689E01_DXXX.jpg"},{"id":"424908","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631290305226E03_DXXX.jpg"},{"id":"424909","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631280503688E0B_DXXX.jpg"},{"id":"424910","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631280305225E03_DXXX.jpg"},{"id":"424911","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631270503687E03_DXXX.jpg"},{"id":"424912","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631270305224E03_DXXX.jpg"},{"id":"424913","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631260503686E03_DXXX.jpg"},{"id":"424914","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631260305223E03_DXXX.jpg"},{"id":"424915","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631250503685E01_DXXX.jpg"},{"id":"424916","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631250305222E03_DXXX.jpg"},{"id":"424917","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631240503684E03_DXXX.jpg"},{"id":"424918","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631240305221E03_DXXX.jpg"},{"id":"424919","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631230503683E01_DXXX.jpg"},{"id":"424920","img_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631230305220E02_DXXX.jpg"},{"id":"424921","img_src":"http://mars.jpl.nasa.gov/msl-raw

3. Tap the Back button in your device or emulator to close the app.
4. Put your device or emulator into airplane mode, to simulate a network connection error. Reopen the app from the recents menu, or restart the app from Android Studio.
5. Click the Logcat tab in Android Studio and note the fatal exception in the log, which looks like this:

```
3302-3302/com.example.android.marsphotos E/AndroidRuntime: FATAL EXCEPTION: main
   Process: com.example.android.marsphotos, PID: 3302
   java.net.SocketTimeoutException: timeout
...
```

This error message indicates the application tried to connect and timed out. Exceptions like this are very common.

6) Handle the error

1. Open overview/OverviewViewModel.kt. Scroll down to the getMarsPhotos() method. Inside the launch block, add a try block around MarsApi call to handle exceptions. Add catch block after the try block:

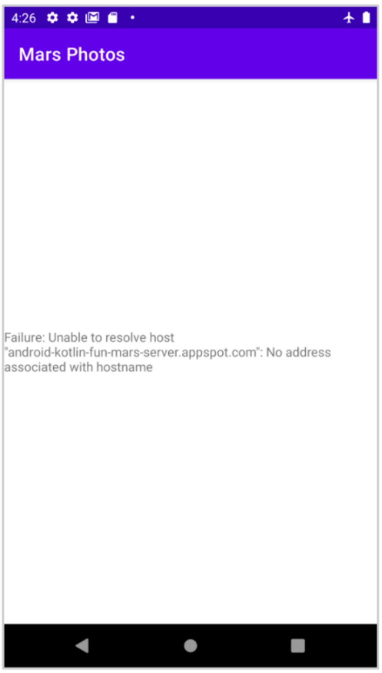
```
viewModelScope.launch {
  try {
     val listResult = MarsApi.retrofitService.getPhotos()
     _status.value = listResult
  } catch (e: Exception) {

  }
}
```

2. Inside the catch {} block, handle the failure response. Display the error message to the user by setting the e.message to the _status.value.

```
catch (e: Exception) {
  _status.value = "Failure: ${e.message}"
}
```

3. Run the app again, with the airplane mode turned on. The app does not close abruptly this time, but displays an error message instead.



```
Failure: Unable to resolve host
"android-kotlin-fun-mars-server.appspot.com": No address
associated with hostname
```

4. Turn off airplane mode on your phone or emulator. Run and test your app, make sure everything is working fine and you are able to see the JSON string.

7) Add Moshi library dependencies

1. Open build.gradle (Module: app).
2. In the dependencies section, add the code shown below to include the Moshi dependency. This dependency adds support for the Moshi JSON library with Kotlin support.

```
// Moshi
implementation 'com.squareup.moshi:moshi-kotlin:1.13.0'
```

3. Locate the lines for the Retrofit scalar converter in the dependencies block and change these dependencies to use converter-moshi:

   Replace this

```
// Retrofit
implementation "com.squareup.retrofit2:retrofit:2.9.0"
```

```
// Retrofit with scalar Converter
implementation "com.squareup.retrofit2:converter-scalars:2.9.0"
```

with this

```
// Retrofit with Moshi Converter
implementation 'com.squareup.retrofit2:converter-moshi:2.9.0'
```

4. Click Sync Now to rebuild the project with the new dependencies.

8) Implement the Mars Photo dataclass

1. Right-click on the network package and select New > Kotlin File/Class.
2. In the popup, select Class and enter MarsPhoto as the name of the class. This creates a new file called MarsPhoto.kt in the network package.
3. Make MarsPhoto a data class by adding the data keyword before the class definition. Change the {} braces to () parentheses. This leaves you with an error, because data classes must have at least one property defined.

```
data class MarsPhoto(
)
```

4. Add the following properties to the MarsPhoto class definition.

```
data class MarsPhoto(
  val id: String, val img_src: String
)
```

Replace the line for the img_src key with the line shown below. Import com.squareup.moshi.Json when requested.

```
@Json(name = "img_src") val imgSrcUrl: String
```

9) Update MarsApiService

1. Open network/MarsApiService.kt. Notice the unresolved reference errors for ScalarsConverterFactory. This is because of the Retrofit dependency change you made in a previous step. Delete the import for ScalarConverterFactory. You will fix the other error soon.

Remove:

```
import retrofit2.converter.scalars.ScalarsConverterFactory
```

2. At the top of the file, just before the Retrofit builder, add the following code to create the Moshi object, similar to the Retrofit object.

```
private val moshi = Moshi.Builder()
```

3. Import com.squareup.moshi.Moshi and com.squareup.moshi.kotlin.reflect.KotlinJsonAdapterFactory when requested.
4. For Moshi's annotations to work properly with Kotlin, in the Moshi builder, add the KotlinJsonAdapterFactory, and then call build().

```
private val moshi = Moshi.Builder()
  .add(KotlinJsonAdapterFactory())
  .build()
```

5. In the retrofit object declaration change the Retrofit builder to use the MoshiConverterFactory instead of the ScalarConverterFactory, and pass in the moshi instance you just created.

```
private val retrofit = Retrofit.Builder()
  .addConverterFactory(MoshiConverterFactory.create(moshi))
  .baseUrl(BASE_URL)
  .build()
```

6. Import retrofit2.converter.moshi.MoshiConverterFactory when requested.
7. Now that you have the MoshiConverterFactory in place, you can ask Retrofit to return a list of MarsPhoto objects from the JSON array instead of returning a JSON string. Update the MarsApiService interface to have Retrofit return a list of MarsPhoto objects, instead of returning String.

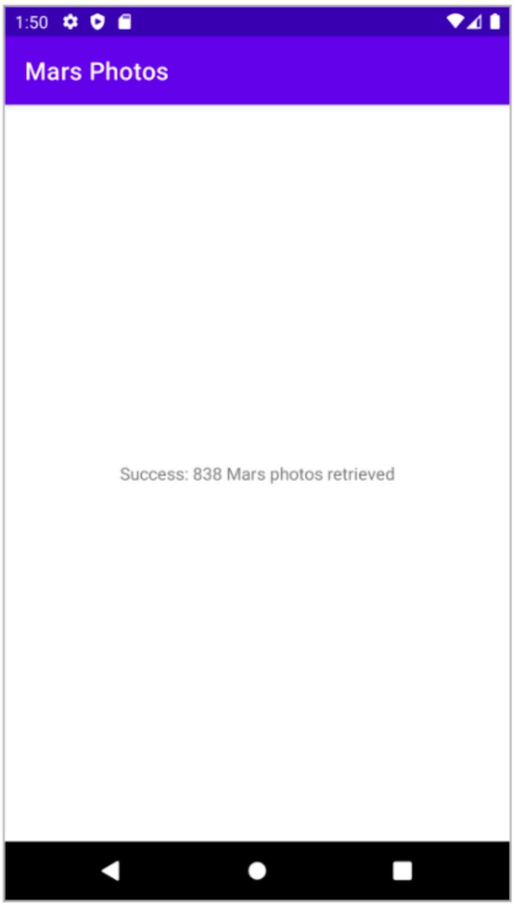
```
interface MarsApiService {
  @GET("photos")
  suspend fun getPhotos(): List<MarsPhoto>
}
```

8. Do similar changes to the viewModel, open OverviewViewModel.kt. Scroll down to getMarsPhotos() method.
9. In the method getMarsPhotos(), listResult is a List<MarsPhoto> not a String anymore. The size of that list is the number of photos that were received and parsed. To print the number of photos retrieved update _status.value as follows.

```
_status.value = "Success: ${listResult.size} Mars photos retrieved"
```

10. Import com.example.android.marsphotos.network.MarsPhoto when prompted.

11. Make sure airplane mode is turned off in your device or emulator. Compile and run the app. This time the message should show the number of properties returned from the web service not a big JSON string:



Success: 838 Mars photos retrieved