6. Writing Robust Apps

“Without Requirements or Design, Programming is the art of adding Bugs to an Empty Text File.”

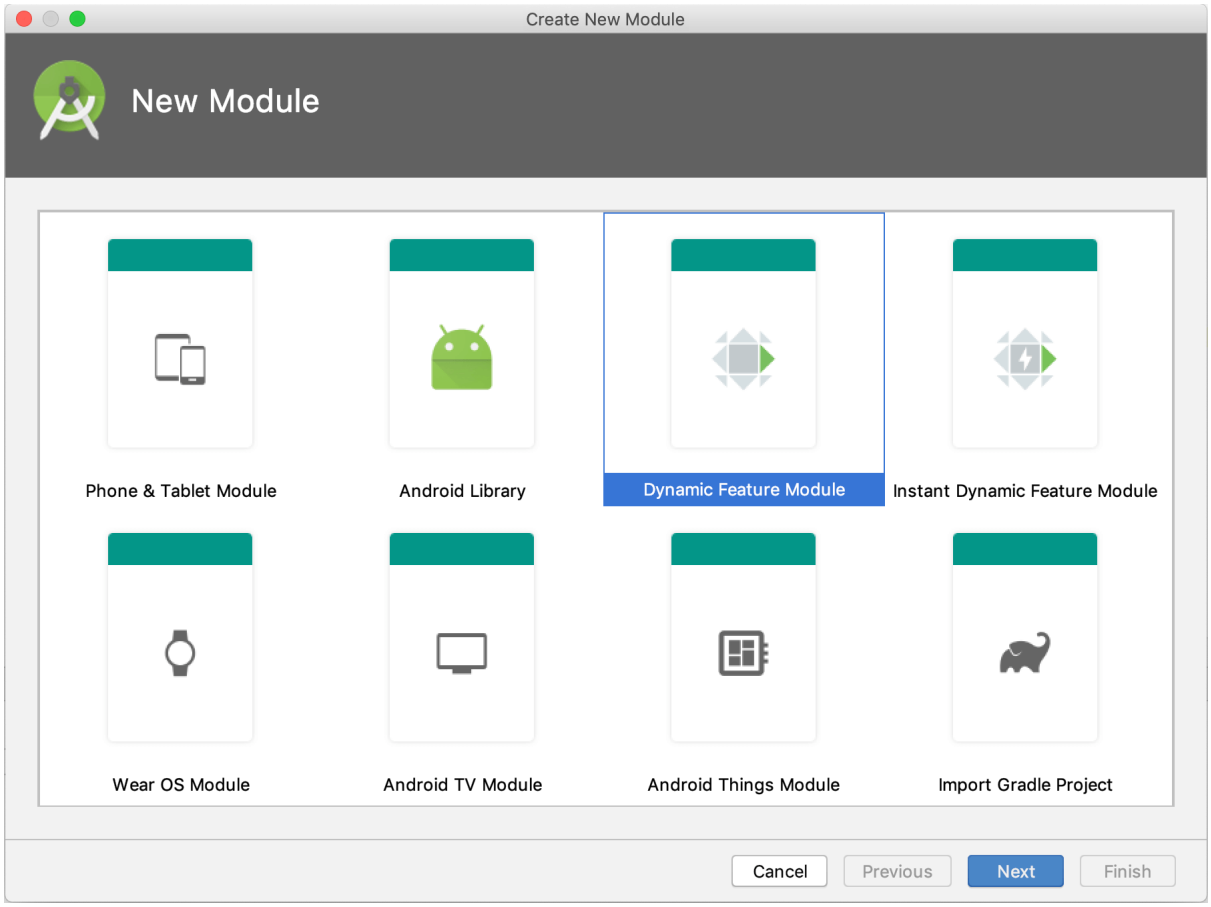
— Louis Srygley

Nowadays, it is going to be hard to build a robust android application, an application that fits the user experience, offline first, not buggy, scalable, maintainable and testable application. So, what we are going to do in this section is to offer you some Tips and Tricks — according to our experience — to how to build a robust android application from scratch.

Now, Let's get started with our project

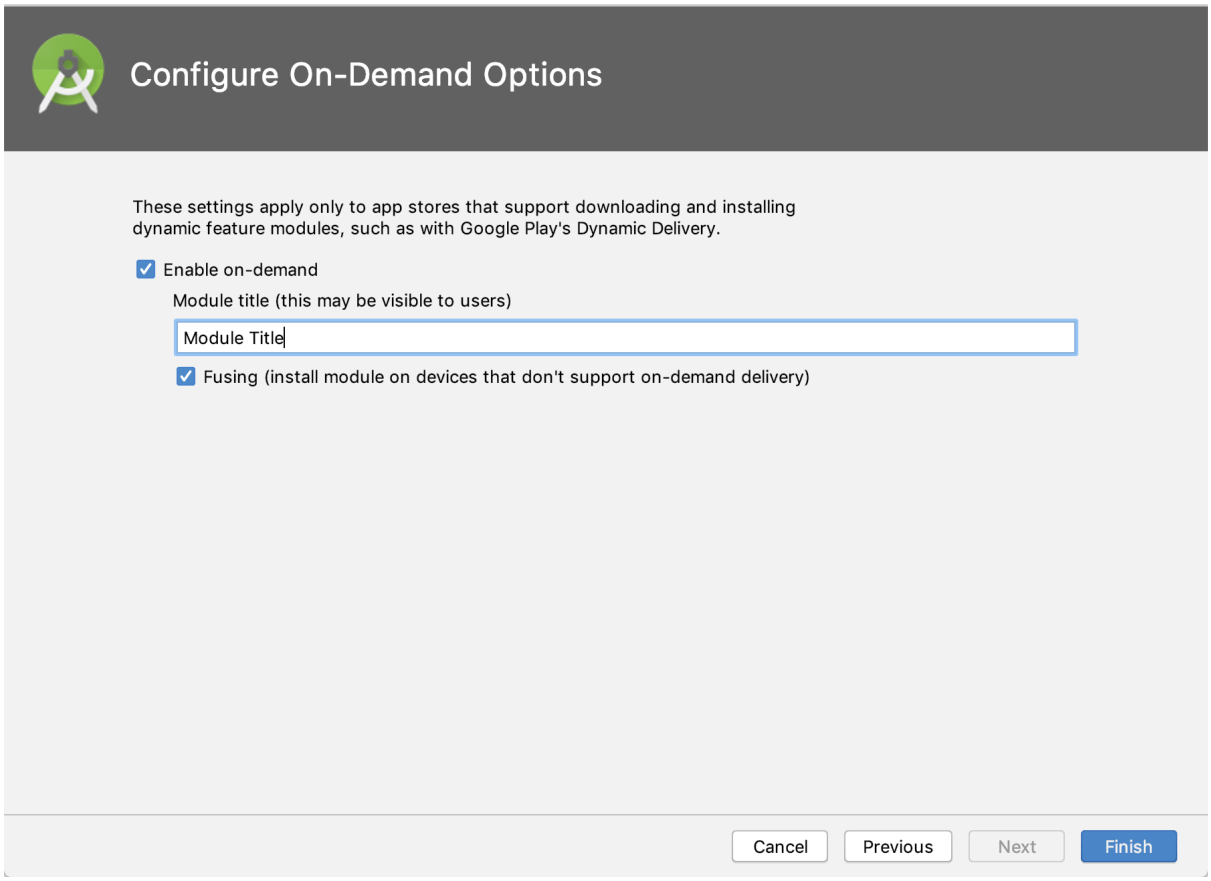
Create a project in Android Studio and we will get an **app** module on creation.

Now, as a next step, we will create another module (Dynamic Feature Module). To do it, Goto **File-> New -> New Module and select Dynamic Feature Module**



and press next. In the next screen, add the required info like **module name** and **package name** and press next.

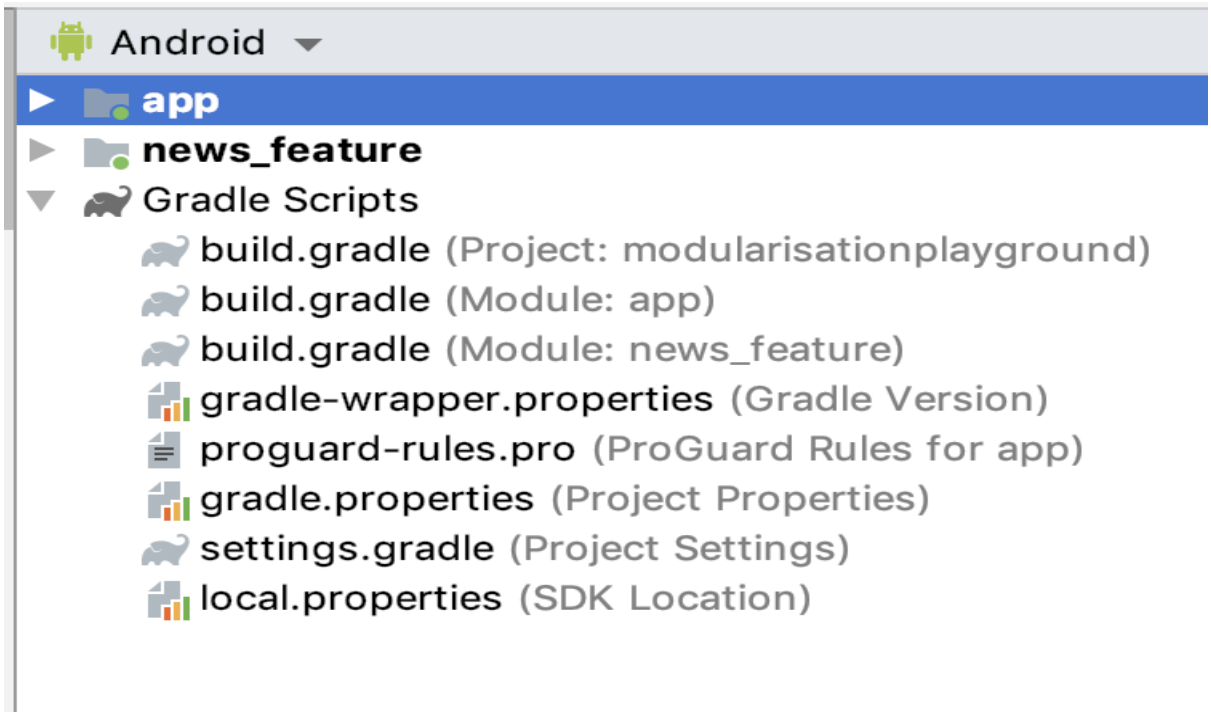
On the next screen, you will get a few options to configure your module.



Press finish and your module will be created (I name my module as news_feature).

Here, enable on-demand checked means that it can be downloaded on demand and Fusing means it can be available for the device running kitkat.

Finally, your project will look like,



Now, both **app** and **news_feature** are created and will have their own gradle files.

What do we want to achieve in this project?

In this project, we will have a launching activity in **app-module** which would have one button and on click of it we will download the **news-feature module** and in the news-feature module, we will have a RecyclerView displaying the list of dummy news.

Let's start building the project, In the app-module's MainActivity, let me design activity_main.xml,

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">
    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:id="@+id/buttonClick"
            android:text="Click to Download News Module"
            android:layout_centerInParent="true"/>
    <Button android:layout_width="wrap_content"
            android:text="Open News Module"
            android:id="@+id/buttonOpenNewsModule"
            android:layout_height="wrap_content"
            android:visibility="gone"
            android:layout_below="@+id/buttonClick"
            android:layout_centerInParent="true"/>
    <Button android:layout_width="wrap_content"
            android:text="Delete News Module"
            android:id="@+id/buttonDeleteNewsModule"
            android:layout_height="wrap_content"
            android:visibility="gone"
            android:layout_below="@+id/buttonOpenNewsModule"
            android:layout_centerInParent="true"/>
</RelativeLayout>
```

in this we will have 3 buttons in which 2 buttons are hidden and it will be visible only when the dynamic module is downloaded properly. And it will be used to open the activity with the recyclerView in **news-feature** module and other is to delete the dynamic module.

When we created the **news-feature** module, in app's **build.gradle** the following is added,

```
android {
    .......
    dynamicFeatures = [":news_feature"]
}
```

It means that we have a dynamic module in the project and in the news-feature's build.gradle,

```
dependencies {
    .....
    implementation project(':app')
}
```

The dynamic feature module implements the app module and uses it as a dependency. Also, in dynamic feature

Gradle plugin to the module's `build.gradle` file:

```
apply plugin: 'com.android.dynamic-feature'
```

It specifies that the module is a dynamic-feature module. In the manifest of the dynamic Module,

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:dist="http://schemas.android.com/apk/distribution"
          package="com.mindorks.news_feature">

   <dist:module
            dist:instant="false"
            dist:onDemand="true"
            dist:title="@string/title_news_feature">
        <dist:fusing dist:include="true"/>
    </dist:module>
</manifest>
```

- <dist> tag helps the project to know how the module is being packed
- `dist:onDemand="true"` : It specifies if the module is available as an on demand download.

and to make your module downloadable on demand add the following in the app-module's build.gradle,

```
implementation 'com.google.android.play:core:1.6.1'
```

The play core library is responsible for the dyamic modules download on demand.

Now, let's start to design the view to display the list of dummy news in news-feature dynamic module. In the src package, we create a package **newsloader** and in that, we have sub-packages

of **adapter**, **data source**, **model** and **NewsLoaderActivity**. In this,

- **adapter** package is responsible for providing the binding between the recyclerView and the data
- **data source** provides the data for the recyclerView
- **model** package is responsible for providing the structure of the data which has to be added in the recyclerView
- and **NewsLoaderActivity** file is the Activity file which we will open after downloading the dynamic feature from app-module.

We can treat the dynamic-module as the individual app itself. So, if we use any 3rd party library in the dynamic module it will only be used when the dynamic module is downloaded on demand and not when the app is being uploaded in the play store.

If we want to share the 3rd party library from the main app's module to the dynamic module, we add the libraries in the app's **build.gradle** using,

```
api 'androidx.constraintlayout:constraintlayout:1.1.3'
```

*When we use **api** the libraries can be used by the other modules as well as the dynamic modules **implements** the app module. If we use **implementation** then the library is only used by module it is implemented in.*

Now, Let's learn how to make the dynamic-feature downloadable from the app's module.

We have already created the layout for MainActivity.kt file in app's module. Now, let us write the logic for downloading the modules.

Firstly, we will create 3 variables,

```
lateinit var splitInstallManager: SplitInstallManager
lateinit var request: SplitInstallRequest
val DYNAMIC_FEATURE = "news_feature"
```

- **SplitInstallManager** is responsible for downloading the module. The app has to be in Foreground to download the dynamic module.
- **SplitInstallRequest** will contain the request information that will be used to request our dynamic feature module from Google Play.

Now, we have to initialize the lateinit variables in onCreate of the MainActivity,

```
override fun onCreate(savedInstanceState: Bundle) {
    ....
    initDynamicModules()
}
```

and initDynamicModules will look like,

```
private fun initDynamicModules() {
splitInstallManager = SplitInstallManagerFactory.create(this)
request = SplitInstallRequest
        .newBuilder()
        .addModule(DYNAMIC_FEATURE)
        .build();
}
```

Here,

- First we create the factory for splitInstallManager and then we create the SplitInstallRequest instance.
- We add the module name, in **addModule(/** Module Name **/)** and we can also add multiple multiple modules by add multiple addModule.

Now, we are ready to setup the download feature of the dynamic-module. We will setup setOnClickListeners for the button,

```
private fun setClickListeners() {
    buttonClick.setOnClickListener {
        if (!isDynamicFeatureDownloaded(DYNAMIC_FEATURE)) {
            downloadFeature()
        } else {
```

```
            buttonDeleteNewsModule.visibility = View.VISIBLE
            buttonOpenNewsModule.visibility = View.VISIBLE
        }
    }
}
```

Here , first we check if the module is already downloaded or not and then we show the rest two buttons to delete or open modules.

isDynamicFeatureDownladed looks like,

```
private fun isDynamicFeatureDownloaded(feature: String):
Boolean =
    splitInstallManager.installedModules.contains(feature)
```

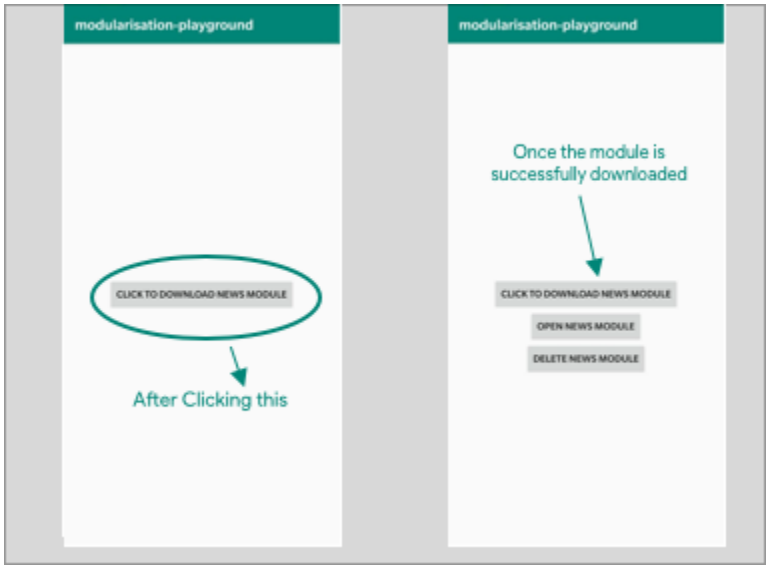
It will return if the installed module contains the dynamic-module or not.

If the dynamic-module was not downloaded initially, it will call ***downloadFeature(),***

```
private fun downloadFeature() {
    splitInstallManager.startInstall(request)
        .addOnFailureListener {
        }
        .addOnSuccessListener {
            buttonOpenNewsModule.visibility = View.VISIBLE
            buttonDeleteNewsModule.visibility = View.VISIBLE
        }
        .addOnCompleteListener {
        }
}
```

Here, the SplitInstallManager will start installing the module and when the the download success is successful , it will show the open and delete button.

The before and after of downloading the dynamic-module,



The dynamic-feature-module is downloaded successful and it can be verified because the rest two buttons are visible.

Now, to open the Activity of dynamic-feature-module i.e NewsLoaderActivity we can't use normal intent as we open in same modules using Intent. To open the activity of different module on click of **Open News Module** button we use,

```
buttonOpenNewsModule.setOnClickListener {
    val intent = Intent().setClassName(this,
"com.mindorks.news_feature.newsloader.NewsLoaderActivity")
    startActivity(intent)
}
```

Here, we use **setClassName** we have to pass context and the path of the Activity in string format.

Note : If we refactor the code of the module we have to manually change the path in the intent to open the activity.

And now, to delete the downloaded dynamic-feature-module in Android on the click of **Delete News Module** we call **uninstallDynamicFeature** function**,**

```
buttonDeleteNewsModule.setOnClickListener {
    val list = ArrayList<String>()
    list.add(DYNAMIC_FEATURE)
    uninstallDynamicFeature(list)
```

```
}
```

It takes list of strings which contains the name of dynamic module as parameter and **uninstallDynamicFeature** looks like,

```
private fun uninstallDynamicFeature(list: List<String>) {
    splitInstallManager.deferredUninstall(list)
        .addOnSuccessListener {
            buttonDeleteNewsModule.visibility = View.GONE
            buttonOpenNewsModule.visibility = View.GONE
        }
}
```

Here, in **deferredUninstall** we take list as parameter and not string. And when the uninstall is successful both the open and delete buttons are hidden.

- If we have multiple modules in the app, we can get the list of all the installed modules using**,**

```
SplitInstallManager.getInstalledModules()
```

Finally,

We can also monitor the state of the request in the process when we request any dynamic-module,

```
var mySessionId = 0
val listener = SplitInstallStateUpdatedListener {
    mySessionId = it.sessionId()
    when (it.status()) {
        SplitInstallSessionStatus.DOWNLOADING -> {
            val totalBytes = it.totalBytesToDownload()
            val progress = it.bytesDownloaded()
            // Update progress bar.
        }
        SplitInstallSessionStatus.INSTALLING ->
Log.d("Status", "INSTALLING")
        SplitInstallSessionStatus.INSTALLED ->
Log.d("Status", "INSTALLED")
```

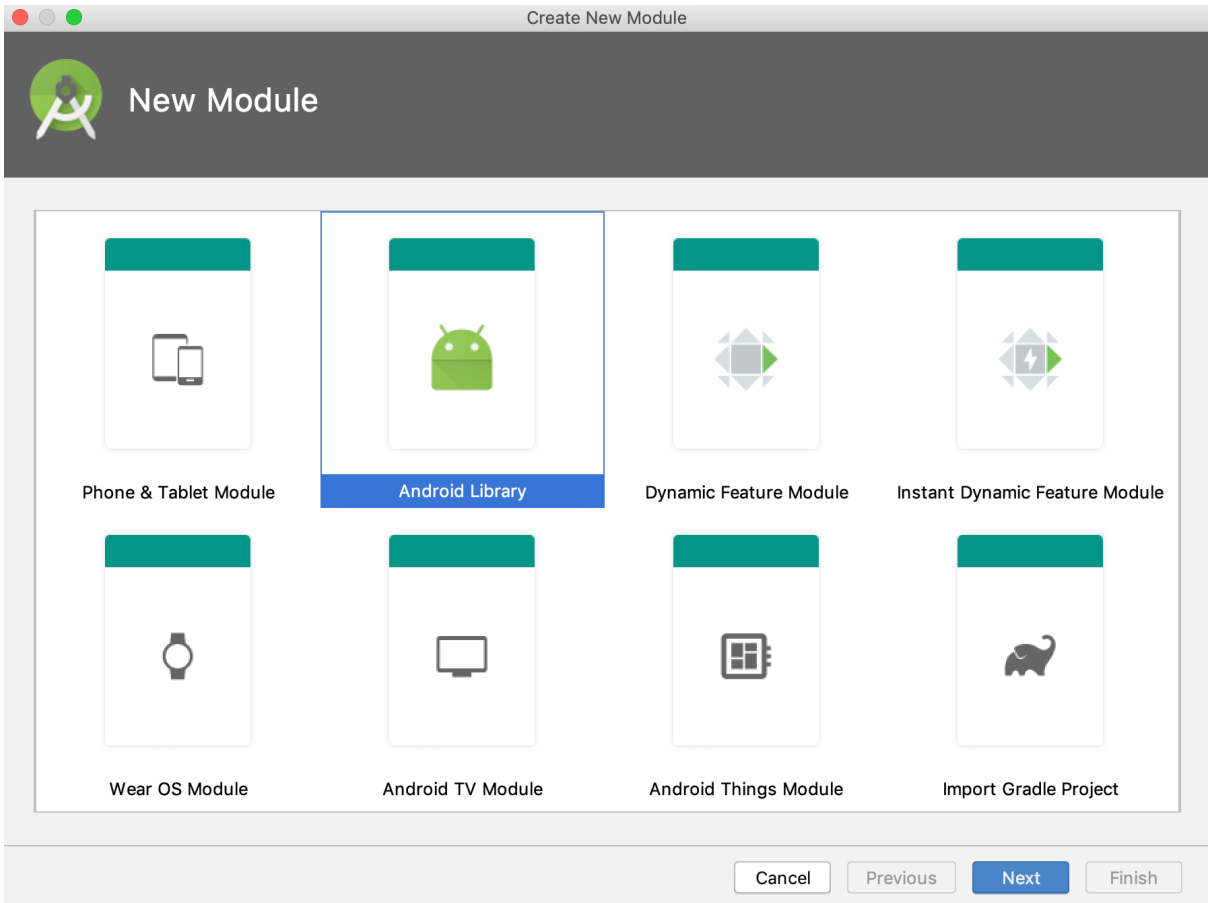
```
        SplitInstallSessionStatus.FAILED -> Log.d("Status",
"FAILED")
        SplitInstallSessionStatus.REQUIRES_USER_CONFIRMATION
-> Log.d("Status", "REQUIRES_USER_CONFIRMATION")
    }
}
```

SplitInstallSessionStatus.REQUIRES_USER_CONFIRMATION *only occur when attempting to download a sufficiently large module and to test it we have to upload it on playstore. To get the user confirmation we use,*

```
SplitInstallSessionStatus.REQUIRES_USER_CONFIRMATION ->
startIntentSender(it.resolutionIntent().intentSender, null,0,
0, 0)
```

If we want to use small features and which doesn't require much size when compiled in an APK, we can use **libray-module** and it will work similarly as third party libraries. To create a **library-module**, we goto,

File -> New -> New Module, and we see



And here it gets the module created as same as the **news-feature.**

- In this, the code of the library module is directly accessible by the app module

and in the **build.gradle** of the library, we add

```
apply plugin: 'com.android.library'
```

which is different from what we added for **dynamic-modules.**

But, the bigger question is how we decide which module to keep in dynamic-module and which module in library.

- If the module is very less used feature in the app but is an important flow of the app we keep that in dynamic-module. For example, Paid Features, OnBoarding Flow etc.

If the dynamic module is initially integrated in the first build of APK can also be deleted later after the use. Like after the OnBoarding of the user, we can delete the module to save some storage.

- If the module is very light in size we can keep as a library module.

Benefits of Modular Android App Architecture :

- **Easily Managable By Multiple Devs** : The app can be handled properly by multiple devs in a team if they work on different modules at a time. It will not hamper anyone's code in the process
- **Maintainability**: Consider having multiple XML files in layout folder if we are using single module architecture. Dividing it in multiple different modules helps to maintain it properly. It will also help in the fast compilation as the gradle will be responsible for building individual modules rather than one module.

- **Reduced APK Size:** If we use Android App bundle with dynamic delivery it will reduce the app size by a lot and it will helps the user to quickly download the app and hence the app will have more signups
- **Testing New Features :** If you want to test some new features, you can build it in separate modules and keep for users to test it. In this way, it will not interfere in the core flow of the app and we can also test the feature simultaneously.