Android, Java, Gradle - Custom Library Setup

By John Bentley

The Goal

The goal is to detail the procedures for setting up custom libraries with the following features:

- That are a custom Android Library (i.e. that provides Android resources, with some minor Java code) and a custom Pure Java Library.
- To work whether consumed by an Android App (consuming an custom Android Library and a custom Java Library) or a Pure Java App (consuming a custom Java Library).
- To allow the custom libraries to live outside the directory hierarchy of the consuming app.
- To work whether working in Android Studio (on a consuming Android App) or IntelliJ IDEA (on a consuming Pure Java App).
- Uses Gradle.

The conventional setup for referencing libraries in Android Studio has Android *import* the libraries into the root directory of our main project. That's not really what we want as the essential benefit of a library, that it is shared between many projects, would be lost. By allowing our custom libraries to live outside the directory hierarchy of the consuming app, we retain the benefit of making a library reusable by many projects/apps.

We'll walkthrough the setup of the following demonstration projects:

- MyJavaLibrary (a Pure Java Library).
- MyAndroidLibrary (that provides Android resources, with some minor Java code).
- MyAndroidApp. This will consume MyAndroidLibrary and MyJavaLibrary.
- MyJavaApp. This will consume MyJavaLibrary.

Gradle and Android Studio Setup

Setting versions of Gradle and the Android Plugin For Gradle

What?	Url	Path

Android Plugin Version	See the latest heading under https://developer.android.com/studio/releases/gradle-plugin#updating-gradle e.g. "3.3.0 (January 2019)	Android Studio > File > Project Structure > Project > "Android Plugin Version
Gradle Version	https://gradle.org/downloads/	Android Studio > File > Project Structure > Project > "Gradle Version").
Gradle and Android Plugin Version compatibility	https://developer.android.com/studio/releases/gradle-plugin#updating-gradle	

Gradle build process

The build process results in an Android Application Package (APK). This will either be a "debug APK" or a "release APK:.

Invalid source specified., *Configure your build, https://developer.android.com/studio/build/index.html*

A build configuration may stipulate all of the following aspects:

Aspect	**Descrption**	**Example**
Build type	Defines different properties related to buid, typically to do with the development lifecycle. You must have at least one build type defined.	debug, release (created by default).
Product flavour	Different versions of your app released to users. Optional	free, paid
Build variant	A "cross product" of build type and product flavour. "Although you do not configure build variants directly, you do configure the build types and product flavours that form them">	freeDebug, paidRelease
Manifest entry override	For each build variant you can specify values that override values in the manifest file.	
Dependency	Project dependencies can come from the local filesystem or remote repositories.	
Signing	Debug versions are automatically signed with a default key. Release versions will have to be explicitly signed by you.	
ProGuard	ProGuard shrinks and obfuscates classes. You can define rules for ProGuard.	

Multiple APK support	You can build multilple APKs for different purposes.	

Invalid source specified., "*Configure Your Build*", "*Custom Build Configurations*", https://developer.android.com/studio/build/index.html#build-config
Invalid source specified., "*Configure Build Variants*", "*Configure Product Flavours*", https://developer.android.com/studio/build/build-variants.html#product-flavors

Build configurartion files, aka Gradle files, are: plaintext files; use "Domain Specific Language (DSL)"; and build logic with "Groovy".

Build configurartion files, aka Gradle files, types are:

Level	**File name**	**Example contents**
Project	gradle.properties	org.gradle.jvmargs=-Xmx1536m
Project	gradle-wrapper.properties	distributionUrl=https\://services.gradle.org/distributions/gradle-4.1-all.zip
Project	settings.graddle	include ':app' include ':standard-java-library' project(':standard-java-library').projectDir = new File('../../../Libraries/standard-java-library')

Project	build.gradle (e.g. MyThirdApp). "Top-Level Build File", located in theh project root directory.	buildscript {
		  repositories {
		  google()
		  jcenter()
		  }
		  dependencies {
		  classpath 'com.android.tools.build:gradle:3.0.1'
		  // NOTE: Do not place your application dependencies here; they belong
		  // in the individual module build.gradle files
		  }
		}

Module	build.gradle E.g. "(Module: app)", the root module of the project.	apply plugin: 'com.android.application'
		
		android {
		  compileSdkVersion 26
		  defaultConfig {
		  applicationId "au.com.softmake.mythirdapp"
		  minSdkVersion 15
		  targetSdkVersion 26
		  versionCode 1
		  versionName "1.0"
		  }
		  buildTypes {
		  release {
		  minifyEnabled false
		  proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
		  }
		  }
		}
		
		dependencies {
		  implementation project(':standard-java-library')

		`implementation project(':standard-app-library')`
		…
		`}`

Module	`build.gradle e.g. "(Module: standard-java-library)"`	`apply plugin: 'java-library'`
		
		`dependencies {`
		`//implementation fileTree(dir: 'libs', include: ['*.jar'])`
		`}`
		
		`sourceCompatibility = "1.7"`
		`targetCompatibility = "1.7"`

Invalid source specified., *"Configure Your Build" > "Build Configuration File", https://developer.android.com/studio/build/index.html#build-files*

Experimental findings on MyJavaLibrary

So far following experiments, using my techniques (others might find a way around any limits), have shown that MyJavaLibrary:

- Can be created in Intellij IDEA and:
  - Can't be opened directly in Android Studio (the directory/module views are fucked up in the UI); but
  - Can be referenced, using gradle, from MyAndroidApp in Android Studio; and
  - Can be independently run, using gradle, when referenced from MyAndroidpp in Android Studio.
- Can be created in Android Studio and:
  - Can be opened directly in IntelliJ IDEA. Gradle can be used to run it.

Therefore, because MyJavaLibrary is a pure java library and probably needs to be shared between being consumed by Android Apps and (pure) Java App, we are enabled to create it in Intellij IDEA for relevant use in Android Studio (with the caveat we can't open it directly in Android Studio).

Conclusion:

- Create MyJavaLibrary in IntelliJ IDEA;
- Reference it from MyAndroidApp in Android Studio (and from MyJavaApp in Intellij IDEA);
- If you want to run MyJavaLibrary independently in Android Studio, do so indirectly by opening MyAndroidApp, which references MyJavaLibrary, and use a custom gradle run configuration (see Verify MyJavaLibrary (mjl) module can run independently ... below).

Also:

- In Intellij a gradle configured Project can reference a Native Module, see ToturialAtOracle02 (the project configure under gradle) referencing JDBCTutorial (a

native/ant module). That is, so long as the reference module lives outside the directory of the project.

Naming Conventions and Config Summary

Firstly we establish our naming and directory conventions (if you don't like the following, establish your own convention).

Overview

Project name	Package Name	Module name	Project name in file system
MyAndroidApp	au.com.example.myandroidapp	app	..\MyAndroidApp
MyJavaApp	au.com.example.myjavaapp	MyJavaApp	..\MyJavaApp
MyAndroidLibrary	au.com.example.mal	mal	..\Libraries\MyAndroidLibrary
MyJavaLibrary	au.com.example.mjl	mjl	..\Libraries\MyJavaLibrary

Main (entry) class name:

```
// In android
public class MainActivity extends AppCompatActivity {
    ...

// In (pure) Java
public class Main {
    // The main method within the Main class ...
    public static void main(String[] args) {
    ...
```

Project name

Project name in the file system: PascalCase.

```
..\LibraryRefDemo\MyAndroidApp
..\LibraryRefDemo\MyJavaApp
..\LibraryRefDemo\Libraries\MyAndroidLibrary
..\LibraryRefDemo\Libraries\MyJavaLibrary
```

```
Android Studio "Create New Project Wizard" Naming Tip:

Create New Project Wizard > Create Android Project:

    Application Name: MyAndroidApp
    Project Location: ..\MyAndroidApp
```

Package name

Package name: lowercase.

- For the application: full project name.

```
// For MyAndroidApp's root module: "app" we have the following package name
au.com.example.myandroidapp

// For MyJavaApp's module:
au.com.example.myjavaapp
```

- For the library modules: acronym of library project name.

```
// (For MyAndroidLibrary's root module: "mal")
au.com.example.mal

// (For MyJavaLibrary' module where we placed our java code to be shared: "mjl")
au.com.example.mjl
```

Module name

IDE	Open Project	Module name in UI	Module in file system
Android Studio	MyAndroidApp	app mal mjl	LibraryRefDemo**MyAndroidApp\app**\src LibraryRefDemo\Libraries**MyAndroidLibrary\mal**\src LibraryRefDemo\Libraries**MyJavaLibrary**\src
Android Studio	MyAndroidLibrary	mal	LibraryRefDemo\Libraries**MyAndroidLibrary\mal**\src
Android Studio	MyJavaLibrary	N/A	{Can't be opened/don't open directly in Android Studio, when created in IntelliJ}
Intellij IDEA	MyJavaApp	MyJavaApp mjl\MyJavaLibary [mjl]	LibraryRefDemo**MyJavaApp**\src LibraryRefDemo\Libraries**MyJavaLibrary**\src
Intellij IDEA	MyJavaLibrary	MyJavaLibary	LibraryRefDemo\Libraries**MyJavaLibrary**\src

Gradle `settings.gradle (Project settings)`:

- Open in Android Studio:
  - Open MyAndroidApp.
    - app module: lowercase.

```
include ':app' // Set automatically by Android Studio
```

        - mal module: lowercase

```
include ':mal'
// Despite the name "project" we actually need to reference the module
directory
// of a project. It doesn't matter whether there is a trailing slash
'/' or not.
project(':mal').projectDir =
        new File(settingsDir, '../Libraries/MyAndroidLibrary/mal/')
```

        - mjl module: lowercase

```
// include line must precede projectDir setting.
// Ensure colon prefixes exist.
include ':mjl`
// It doesn't matter whether there is a trailing slash '/' or not.
project(':mjl').projectDir = new File(settingsDir,
'../Libraries/MyJavaLibrary/')
```

    - Open MyAndroidLibrary. mal module: lowercase abbreviation.

```
include ':mal'
```

    - Open MyJavaLibrary. Don't.

- Open in IntelliJ IDEA:
    - Open MyJavaApp. 'MyJavaApp' module: PascalCase.

```
// Generally no need for setting rootProject.name in a settings.gradle as
// case will inherit from file system basename.
// However, if necessary ..\LibraryRefDemo\MyJavaApp\settings.gradle
rootProject.name = 'MyJavaApp'

// include line must precede projectDir setting.
// Ensure colon prefixes exist.
include ':mjl`
// It doesn't matter whether there is a trailing slash '/' or not.
project(':mjl').projectDir = new File(settingsDir,
'../Libraries/MyJavaLibrary/')
// Not '../Libraries/MyJavaLibrary/mjl/'. We need the parent directory of
'src'.
```

    - Open MyJavaLibrary. 'MyJavaLibrary' module: PascalCase.

```
// ..\LibraryRefDemo\Libraries\MyJavaLibrary
// Generally no need for setting rootProject.name in a settings.gradle as
// case will inherit from file system basename.

// However, if we wanted to have a name different from the file system
// folder name we'd do something like ...
// ..\LibraryRefDemo\Libraries\MyJavaLibrary\settings.gradle
rootProject.name = 'mjl'
```

Pure Java Libraries may be used in an Android context. In an Android context modules are lowercase.

Gradle `build.gradle` (of MyAndroidApp's app module; or MyJavaApp's module). For referencing a library from an application:

```
// E.g. ..\LibraryRefDemo\MyJavaApp\build.gradle
dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    implementation  project(':mjl')
```

```
    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}
```

Main (entry) class name

Android main (entry) class (Activity): "MainActivity" as PascalCase.

```
// Directory Structure
// \LibraryRefDemo\MyAndroidApp\app

// The Main Entry Class base name "MainActivity"
// \LibraryRefDemo\MyAndroidApp\app\src\main\java
   \au\com\example\myandroidapp\MainActivity.java

package au.com.example.myandroidapp;

...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
...

    }
}
```

Java main (entry) class: "Main" as PascalCase, rather than taking on the name of the application or library.

```
// Directory Stucture
..\LibraryRefDemo\MyJavaApp
..\LibraryRefDemo\Libraries\MyJavaLibrary


// The Main Entry Class base name "Main" is not the same as "MyJavaLibrary"
...LibraryRefDemo\Libraries\MyJavaLibrary\src\au\com\example\myjavalibrary\Main.java

package au.com.example.mjl;

public class Main {
    // The main method within the Main class ...
    public static void main(String[] args) {
        System.out.println("MyJavaLibrary! " +
au.com.example.myjavalibrary.MyStrings.GetStringFromLibrary());
    }
}
```

Using IntelliJ IDEA (with custom libraries)

Overview

Using Intillij IDEA our goal is to create a custom java library that we can reference from multiple java projects, while being able to edit the library code when working on a consuming Java, or Android app (project).

The key to native (as opposed to gradle) referencing is to add to your project the Library:

- As a module from existing sources (targeting an existing .iml file); and
- As a module dependency.

Later on we will do a gradle conversion, if you prefer to work with gradle and/or you want to reference MyJavaLibrary from an MyAndroidApp.

Steps (for a module dependency)

Create Custom Java Library

Create MyJavaLibary as an independent project and run:

1. IntelliJ IDEA > 'Welcome to Intellij IDEA' > Create New Project ...
    a. Java. [Next]. [Next]
    b. Settings ...
        i. Project Name: **MyJavaLibrary**
        ii. Project Location: C:\Users\John\Documents\Sda\Code\Android\Examples\LibraryRefDemo**Libraries****MyJavaLibrary**
    c. [Finish].
    d. "The project file ... does not exist. ... It will be created ...?" [OK].
2. In the Project pane verify "Src" folder is blue, designating a "source" folder. If grey:
    a. File > Project Structure > Project Settings > Modules.
    b. Click on "MyJavaLibrary" module > |Sources| > Right click on "src" folder. Choose [Sources]. Src folder should now be blue. [OK].
3. In the Project pane add a package "au.com.example.mjl" underneath src.
4. Under "src/au.com.example.mjl" add a test class of MyJavaLibrary, that does work (return a string).

```
package au.com.example.mjl;

public class MyStrings {
    public static String GetStringFromLibrary() {
        return "From MyJavaLibrary.";
    }
}
```

5. Under "src/au.com.example.mjl" add a Main class. Add code that calls the test class (MyStrings).

```
package au.com.example.mjl;

public class Main {
    public static void main(String[] args) {
        System.out.println("MyJavaLibrary Main Class: " +
au.com.example.mjl.MyStrings.GetStringFromLibrary());
    }
}
```

6. Create a run configuration. Add Cofiguration > [+] > Application:
    a. Name: **MyJavaLibraryRun**
    b. Main Class: **au.com.example.mjl.Main**
    c. Working Directory: C:\Users\John\Documents\Sda\Code\Android\Examples\LibraryRefDemo\Libraries\MyJavaLibrary
    d. Use classpath of module: MyJavaLibrary.
    e. [OK]
7. Run MyJavaLibraryRun (e.g. Run > Run 'MyJavaLibraryRun') . Verify output in the Run Pane:

```
MyJavaLibrary Main Class: From MyJavaLibrary.
```

8. Copy any Java code modules, into \Libraries\SoftmakeJavaLibrary\src\au\com\softmake\sjl from elsewhere.
9. Run again to verify everything still works.

Create Java App

Create MyJavaApp as an independent project and run:

1. Close the MyJavaLibrary Project.
2. 'Welcome to Intellij IDEA' > Create New Project ...
    a. Java > [Next] > [Next]
    b. Project Name: **MyJavaApp**
    c. Project Location: C:\Users\John\Documents\Sda\Code\Java**\Examples\LibraryRefDemo\MyJavaApp** (Not …\LibraryRefDemo**Libraries**\MyJavaApp).
    d. [Finish].
    e. "The project file ... does not exist. ... It will be created ...?" [OK].
3. In the Project pane verify "src" folder is blue, designating a "source" folder. If grey:
    a. File > Project Structure > Project Settings > Modules.
    b. Click on "MyJavaApp" module > |Sources| > Right click on "src" folder. Choose [Sources]. Src folder should now be blue.
4. Create Package underneath "src" folder: **au.com.example.myjavaapp**
5. Create Main class under src/au.com.example.mymainapp and add some dummy code:

```
package au.com.example.myjavaapp;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

10. Create a run configuration. [+] > Application >
    a. Name: **MyJavaAppRun**
    b. Main Class: **au.com.example.myjavaapp.Main**
    c. Working Directory: C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\MyJavaApp
    d. Use classpath of module: MyJavaApp.
    e. [OK]
11. Run MyJavaAppRun. Verify output:

```
Hello World!
```

Reference Java Library From Java App

To the MyJavaApp Project add the Library, MyJavaLibary, as a module and a module dependency:

1. Open MyJavaApp as a project (which will be already open if you are continuing from above).
2. File > New ... > **Module from existing sources** {Not "Project from existing sources}...
    a. Select ..\LibraryRefDemo\Libraries\MyJavaLibrary**MyJavaLibrary.iml**. [OK]
    b. Observe in the Project Pane MyJavaLibrary is added as module (and you can edit its source code).
3. (Continuing with MyJavaApp open as a project) File > Project Structure ... > Project Settings > Modules > MyJavaApp > Dependencies > [+] **> Module Dependency** > Select "MyJavaLibrary". [OK]. [OK].

Verify you can reference MyJavaLibrary code from MyJavaApp:

1. Change MyJavaApp's Main class as follows

```
package au.com.example.myjavaapp;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
        System.out.println(au.com.example.mjl.MyStrings.GetStringFromLibrary());
    }
}
```

2. Run MyJavaAppRun. Observe:

```
Hello World!
From MyJavaLibrary.
```

Verify your edits to MyJavaLibrary code will be picked up by MyJavaApp.

1. Edit MyJavaLibrary code. Change the test string.

```
public class MyStrings {
    public static String GetStringFromLibrary() {
        return "From MyJavaLibrary XXX.";
    }
}
```

2. Run MyJavaAppRun. Observe:

```
Hello World!
From MyJavaLibrary xxx.
```

Gradle Conversion

Delete native library referencing

Assuming "Steps (for a module dependency)", above, has been followed ...

- From MyJavaApp delete the native reference to MyJavaLibrary. With MyJavaApp open... File > Project Structure ... > Project Settings > Modules > MyJavaLibrary [-]. [Yes]. [OK].
- MyJavaApp
  - ... File > Project Structure ... > Project Settings > Modules > Select MyJavaLibrary > [-].
  - "Remove module 'MyJavaLibrary' from the Project? No files will be deleted on disk.". [Yes]. [Ok]
- Observe MyJavaLibrary is no longer available in the Project Pane.

Convert Custom Java Library

Change a native IntelliJ MyJavaLibrary to use Gradle.

1. Close MyJavaApp. Open MyJavaLibrary.
2. Create an empty `build.gradle` file in the root of MyJavaLibrary. `..\LibraryRefDemo\Libraries\MyJavaLibrary\build.gradle`
3. Add the following. We use `sourceSets.main.java.srcDirs = ['src']` and `id 'java-library'` ...

```
plugins {
//    // Apply the java plugin to add support for Java
//    id 'java'
//
//    // Apply the application plugin to add support for building an application
//    id 'application'

    // Apply the java-library plugin to add support for Java Library
    id 'java-library'

}

sourceSets.main.java.srcDirs = ['src']

// Define the main class for the application
//mainClassName = 'au.com.example.myjavalibrary.Main'

dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}

// In this section you declare where to find the dependencies of your project
repositories {
```

```
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

...\LibraryRefDemo\Libraries\MyJavaLibrary\build.gradle

4. To build.gradle, add a runLib task to allow gradle run the library for testing purposes.

```
plugins {
    // Apply the java-library plugin to add support for Java Library
    id 'java-library'
}

sourceSets.main.java.srcDirs = ['src']

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'
    implementation 'junit:junit:4.12'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}

// https://stackoverflow.com/a/42259553/872154
task runLib (type:JavaExec) {
    classpath = sourceSets.main.runtimeClasspath
    main = project.hasProperty("mainClass") ? project.getProperty("mainClass") :
"au.com.example.mjl.Main"
}
```

In MyMainApp we'll have a "runApp" config. These gradle task names need to be different otherwise they will run at once (although in some circumstances you might want to run them together).

..\LibraryRefDemo\Libraries\MyJavaLibrary\mjl\build.gradle

5. Close and reopen MyJavaLibrary. This will prompt:

```
"Unlinked Gradle project?
 Import Gradle project, this will also enable Gradle Tool Window ...
```

6. Click the "import Gradle project" link. Accept defaults. [OK].

7. Allow gradle to build. Then:
    - Open the Gradle Tool Window.
    - MyJavaLibrary > Tasks > other > runApp. Right Click: Create 'MyJavaLibrary [runApp] ...'.
    - Accept defaults. [OK].

8. Click on the toolbar green arrow to execute 'MyJavaLibrary [runApp]'.
9. Observe expected output in the Run Pane.

```
12:59:17: Executing task 'runApp'...

> Task :compileJava
> Task :processResources NO-SOURCE
```

```
> Task :classes

> Task :runApp
MyJavaLibrary Main Class: From MyJavaLibrary XXX.

BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed
12:59:18: Task execution finished 'runApp'.
```

10. Delete Intellij IDEA native referencing fossils (since we are now using gradle):
    - Delete \LibraryRefDemo\Libraries\MyJavaLibrary\mjl.iml & MyJavaLibrary.iml.
    - Delete "MyJavaAppRun" (native run configuration). Leave "MyJavaApp [run]" (the gradle run configuration).

Convert Java app

Change a native IntelliJ MyJavaApp to use Gradle:

1. Close MyJavaLibrary. Open MyJavaApp.
2. Create an empty build.gradle file in the root of MyJavaApp. ..\LibraryRefDemo\MyJavaApp\build.gradle.
3. Add the following. We use sourceSets.main.java.srcDirs = ['src'] and mainClassName = ''au.com.example.myjavaapp.Main".

```
plugins {
    // Apply the java plugin to add support for Java
    id 'java'

    // Apply the application plugin to add support for building an application
    id 'application'
}

sourceSets.main.java.srcDirs = ['src']

// Define the main class for the application
mainClassName = 'au.com.example.myjavaapp.Main'

dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

C:\Users\John\Documents\Sda\Code\Android\Examples\LibraryRefDemo\MyJavaApp\build.gradle

4. Close and reopen MyJavaApp. This will prompt:
   a. "Unlinked Gradle project?
      Import Gradle project, this will also enable Gradle Tool Window ...
   b. Click the "import Gradle project" link. Gradle JVM: Use JAVA_HOME. [OK].
5. Allow gradle to build the project.
6. Temporarily comment out code reference to MyJavaLibrary from MyJavaApp

```
package au.com.example.mymainapp;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
//        System.out.println(au.com.example.mjl.MyStrings.GetStringFromLibrary());
    }
}
```

9. Create a Gradle based run configuration.
    a. Gradle Pane > MyJavaApp > Tasks > application > run
    b. Right click > Create 'MyJavaApp [run]'. Accept Defaults. [OK]
    c. Observe in Intellij's toolbar the run configuration 'MyJavaApp [run]' now exists.
10. Click on the toolbar green arrow to execute 'MyJavaApp [runApp]'.
11. Observe expected output in the Run Pane.

```
12:56:10: Executing task 'run'...

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :run
Hello World!

BUILD SUCCESSFUL in 0s
2 actionable tasks: 2 executed
12:56:10: Task execution finished 'run'.
```

Reference Custom Java Library from Java App

To reference MyJavaLibrary (already converted to use gradle) from MyJavaApp (already converted to use gradle):

1. Within MyJavaApp's root directory create a `settings.gradle` file, e.g. ...\Examples\LibraryRefDemo\MyJavaApp\settings.gradle, as follows:

```
// Generally no need for setting rootProject.name in a settings.gradle as
// case will inherit from file system basename.
// However, if necessary ..\LibraryRefDemo\MyJavaApp\settings.gradle ...
// rootProject.name = 'MyJavaApp'

// include line must precede projectDir setting.
// Ensure colon prefixes exist.
include ':mjl'
// It doesn't matter whether there is a trailing slash '/' or not.
project(':mjl').projectDir = new File(settingsDir, '../Libraries/MyJavaLibrary/')
```

2. Within MyJavaApp's `build.gradle` file reference MyJavaLibrary as a dependency.

```
dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    implementation  project(':mjl')

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}
```

C:\Users\John\Documents\Sda\Code\Java\Examples\LibraryRefDemo\MyJavaApp\build.gradle

3. Close MyJavaApp and reopen. When prompted import gradle changes.

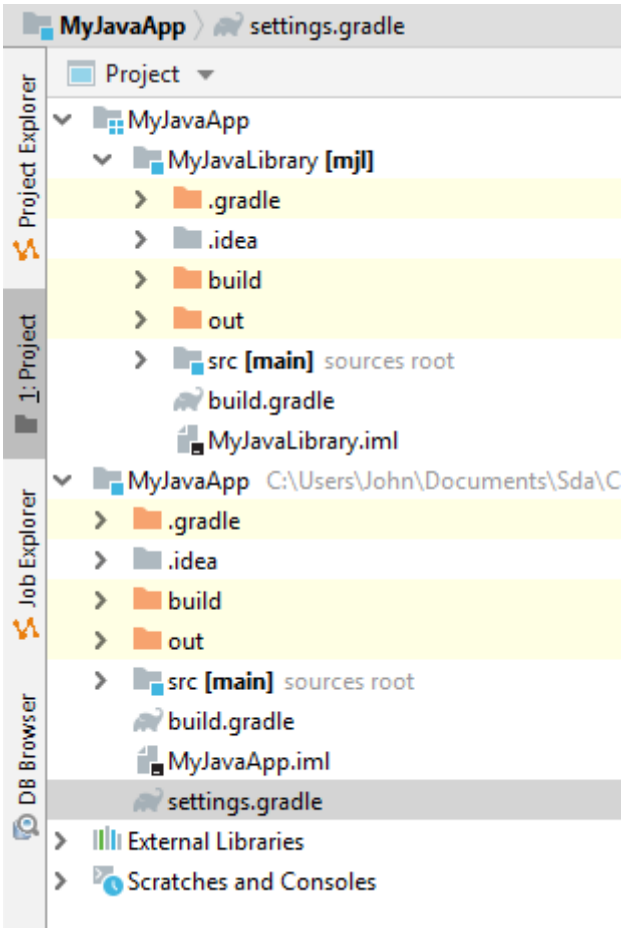
4. Adjust your MyJavaApp code to reference MyJavaLibrary code (see above).

```
package au.com.example.myjavaapp;

public class MyJavaApp {
    public static void main(String[] args) {
       System.out.println("Hello World!");
       System.out.println(au.com.example.mjl.MyStrings.GetStringFromLibrary());
    }
}
```

5. (If not already done so) Create a Gradle run configuration (see above).
6. Run the Gradle Configuration "MyJavaApp [run]" and verify it picks up MyJavaLibrary Code.

```
13:05:32: Executing task 'run'...

> Task :mjl:compileJava
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :mjl:processResources NO-SOURCE
> Task :mjl:classes
> Task :mjl:jar

> Task :run
Hello World!
From MyJavaLibrary XXX.

BUILD SUCCESSFUL in 0s
4 actionable tasks: 4 executed
13:05:33: Task execution finished 'run'.
```

7. Note the UI (as at 2019-01-20; V 2018.3.3) project hierarchy naming is bit clunky. Don't worry about it ....



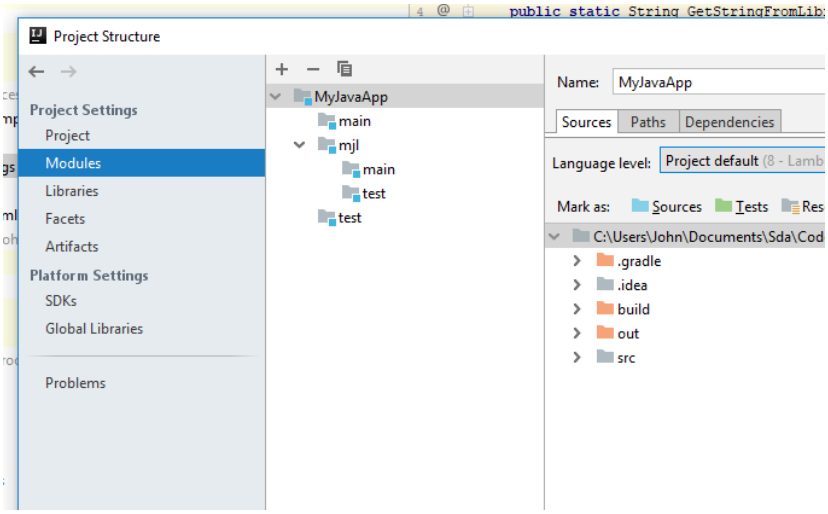
8. Change the MyJavaLibrary Code: alter "MyJavaLibrary [mjl]/src/au.com.example.mjl/MyStrings"

```
package au.com.example.mjl;

public class MyStrings {
    public static String GetStringFromLibrary() {
        return "From MyJavaLibrary ZZZ.";
    }
}
```

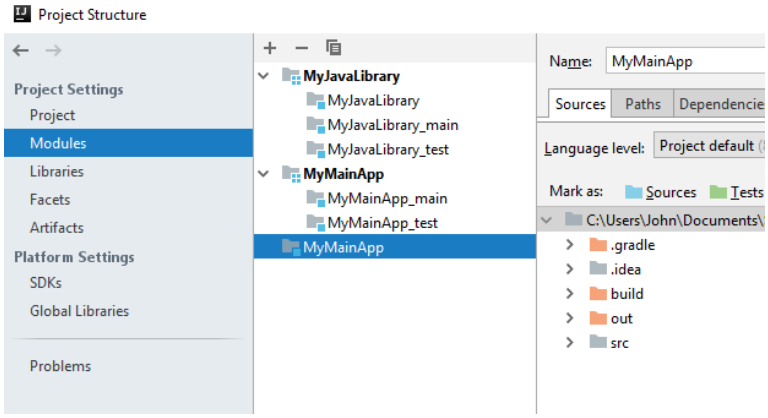
9. Rerun 'MyJavaApp [run]', and verify changed MyJavaLibrary code flows through to MyJavaApp.

```
...
Hello World!
From MyJavaLibrary ZZZ.
...
```

10. Note convoluted Intellij Native structure at File > Project Structure ... > Project Settings > Modules. Don't fuck with it.



... formerly ...



Using Android Studio (with custom libraries)

Overview

Stackoverflow version

An earlier version of "Custom Libraries" section online at Stackoverflow, *How do we reference custom Android and Java Libraries living outside the directory of our Main Android Project?*, . This version still has public utility but my private thoughts here in AndroidJavaGradle-CustomLibrarySetup.docx are the master.

Basic tricks

The following basic tricks, and the subsequent step-by-step procedures, are current as of 2018-03-10 for Android Studio 3.0.1.

Before going through the step-by-step procedures, the basic tricks to make this work are:

- Use Android Studio, rather than another IDE (like IDEA IntelliJ), for creating MyAndroidApp. Code and resources can be copied (e.g. using Windows Explorer) from any existing projects once the basic framework has been setup.
- In MyAndroidApp `settings.gradle (Project Settings)` we reference Library modules (rather than Library Projects as such) using something like the following:

```
include ':app'

include ':mal'
// Despite the name "project" we actually need to reference the module directory
// of a project. It doesn't matter whether there is a trailing slash '/' or not.
project(':mal').projectDir =
        new File(settingsDir, '../Libraries/MyAndroidLibrary/mal/')

include ':mjl'
// Despite the name "project" we actually need to reference the module directory
// of a project. It doesn't matter whether there is a trailing slash '/' or not.
project(':mjl').projectDir =
        new File(settingsDir, '../Libraries/MyJavaLibrary/')
// Not '../Libraries/MyJavaLibrary/mjl/'. We need the parent directory of 'src'.

// or
// project(':mjl').projectDir = new File(settingsDir,
'../../../../Java/Examples/LibraryRefDemo/Libraries/MyJavaLibrary/')
```

- In MyAndroidApp root module's `build.gradle (Module: app)` file, add:

```
dependencies {
    ...
    // Reference module rather than project.
    implementation project(':mal')
    implementation project(':mjl')
    ...
}
```

- When creating MyJavaLibrary use an independently installed JDK, not Android's Embedded JDK:
    - File > Project Structure > |SDK Location| > JDK location:
    - "Use embedded JDK (recommended)": unticked.
    - C:\Program Files\Java\jdk1.8.0_161

The step-by-step procedure is as follows. A forward slash "/" in a string will represent a path in the Project Pane, Android View (chose from the combo box). A back slash "\" in a string will represent a path in the file system …

Create Android App

Create an Android Project ("MyAndroidApp"):

- Open {Android Studio} to the 'Welcome to Android Studio' screen > [Start a new Android Studio project] …
- Wizard 'Create New Project':
  - |Choose your project|: **Empty Activity**. [Next]
  - |Configure your project|:
    - Name: MyAndroidApp
    - Package name: au.com.example.myandroidapp
    - Save Location: C:\Users\John\Documents\Sda\Code\Android\Examples\LibraryRefDemo\MyAndroidApp
    - (Remaining settings: as desired).
    - [Finish]
- Add some "hello world" resources and code: wait for the gradle build to finish; then in the Project Pane switch to Android View (from the combo box):
  - res/layout/activity_main.xml. Add id.

```
<TextView
    android:id="@+id/myandroidapp_output"
    android:layout_width="wrap_content"
```

  - res/values/strings.xml. Add string resource.

```
<resources>
    <string name="app_name">MyAndroidApp</string>
    <string name="myandroidapp_cool_string">From MyAndroidApp.</string>
</resources>
```

  - au.com.example.myandroidapp.MainActivity. Reference string rescource from java code.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = (TextView) findViewById(R.id.myandroidapp_output);
        String someString =
getResources().getString(R.string.myandroidapp_cool_string) + "\r\n";
        textView.setText(someString);
    }
}
```

    To properly import the referenced classes you may have to: insert your cursor over a TextView; alt+enter;

- Run against the emulator to verify OK:
  - Run > Run app (Shift+F10)... "Select Deployment Target" > (Click on your desired Available Virtual Device; Or create one if none exist) > [OK].
- Wait for gradle to finish building and the emulator to load the app. Observe the string in your application:

```
From MyAndroidApp.
```

Create Custom Android Library

Create the library, firstly, as another ordinary Android Project:

- In Android Studio close the open project, if open.
- Open {Android Studio} to the 'Welcome to Android Studio' screen > [Start a new Android Studio project] …
- Wizard 'Create New Project':
  - |Choose your project|: **Empty Activity**. [Next]
  - |Configure your project|:
    - Name: MyAndroidLibrary
    - Package name: au.com.example.**mal**
    - Save Location: C:\Users\John\Documents\Sda\Code\Android\Examples\LibraryRefDemo**Libraries**\MyAndroidLibrary
    - (Remaining settings: as desired).
    - [Finish]
- Wait for gradle to finish building.
- {Android Studio} > Project Pane > Android View (from combo box) > "app" > Right Click > Refactor ... > Rename: mal
  - What would you like to do?: Rename Directory. [OK]. "mal". [OK]
  - (Repeat above) What would you like to do?: Rename Module. [OK]. "mal". [OK].
- Add some hello-world type code:
  - In mal/res/values/strings.xml, add:

```
<resources>
    <string name="app_name">MyAndroidLibrary</string>
    <string name="myandroidlibrary_cool_string">From MyAndroidLibrary.</string>
</resources>
```

  - ../res/layout/activity_main.xml (Text View). Add id.

```
<TextView
    android:id="@+id/myandroidlibrary_output"
    android:layout_width="wrap_content"
```

  - In ..\LibraryRefDemo\Libraries\MyAndroidLibrary\mal\src\main\java\au\com\example\mal\MainActivity.java, add to onCreate method (and import relevant classes, alt+enter with cursor over TextView):

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView myText = (TextView) findViewById(R.id.myandroidlibrary_output);
        String someString = getResources().getString(R.string.
myandroidlibrary_cool_string) + "\r\n";
        myText.setText(someString);
    }
}
```

To properly import the referenced classes you may have to: insert your cursor over a TextView; alt+ enter;

- Copy any existing code (and resources) from elsewhere. Copy android code into: ..\LibraryRefDemo\Libraries\MyAndroidLibrary\mal\src\main\java\au\com\example\mal
- Change the run configuration name from "app" to "mal".
- Verify it can run as a project: Run > Run "mal". Wait for gradle to finish building. Observe string in the emulator:

```
From MyAndroidLibrary.
```

(Following on from above) convert a project's module to an Android library:

- {Android Studio} > Project Panel > Android View > Gradle Scripts > build.gradle (Module: mal) > Double Click (to open). Then ...

```
// Comment out ...
apply plugin: 'com.android.application'

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

(Goolge Android Tools, 2014. Develop > Tools) http://developer.android.com/sdk/installing/create-project.html#SettingUpLibraryModule
(Google, n.d. Gradle Plugin User Guide) http://tools.android.com/tech-docs/new-build-system/user-guide#TOC-Library-projects

- Also in build.gradle (Module: mal) comment out applicationID.

```
...
android {
    compileSdkVersion 26
    defaultConfig {
//        applicationId "au.com.example.mal"
        ...
```

Comment out rather than delete, in case we want to re-run mal as an ordinary Android Project for testing purposes.

https://stackoverflow.com/questions/27374933/android-studio-1-0-and-error-library-projects-cannot-set-applicationid

- Tools > Android > Sync Project with Gradle files (or click "Sync Now" link in yellow tip bar). Wait for gradle build to finish.

Reference Custom Android Library from Android Project

To reference a custom android library from your android app:

Reference, rather than import, a custom Android Library Project (Unorthodox and recommended method)

From MyAndroidApp reference MyAndroidLibrary, a Custom Android Library Project living outside the Android App Project's directory. …

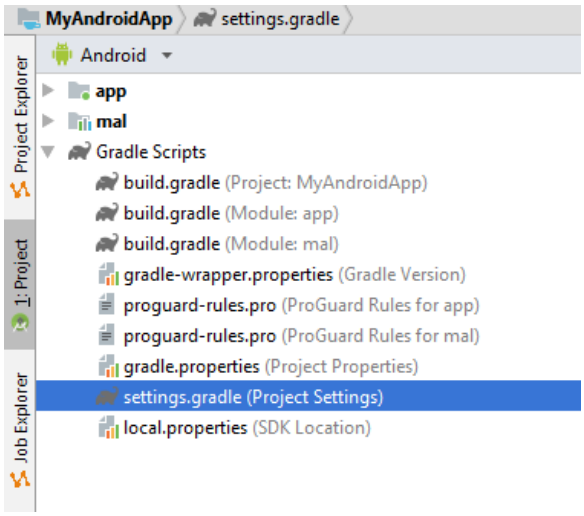
- {Android Studio} > File > Open Recent > MyAndroidApp > 'Open Project' > [This Window].
- Project Pane > Android View (from drop down).
- Open MyAndroidApp's `settings.gradle (Project Settings)` file. Add `:mal` references as follows ..

```
include ':app'

include ':mal'
// Despite the name "project" we actually need to reference the module directory
// of a project. It doesn't matter whether there is a trailing slash '/' or not.
project(':mal').projectDir =
        new File(settingsDir, '../Libraries/MyAndroidLibrary/mal/')
```

- Open MyAndroidApp's root module's build.gradle file (i.e. `build.gradle (Module: app)`).Then add ...

```
dependencies {
    ...
    // Reference module rather than project.
    implementation project(':mal')
    ...
}
```

- File > Sync Project with Gradle files (or click "Sync Now" link in yellow tip bar).
- Observe that mal's source files are now accessible via the project pane (e.g. in Android View).



- Verify you can reference a string resource from the Library:
  - Check ..**mal**/res/values/strings.xml for the following string ...

```
<string name="myandroidlibrary_cool_string">From MyAndroidLibrary.</string>
```

  - Check MyAndroidApp **app**/res/layout/activity_main.xml ...

```
<TextView
    android:id="@+id/myandroidapp_output"
    android:layout_width="wrap_content"
...
```

- In MyAndroidApp app/java/au.com.example.myandroidapp/MainActivity add a reference to `myandroidlibrary_cool_string`.

```
public class MainActivity extends Actvity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView myText = (TextView) findViewById(R.id.myandroidapp_output);
        String someString =
getResources().getString(R.string.myandroidapp_cool_string) + "\r\n";
        someString +=
getResources().getString(au.com.example.mal.R.string.myandroidlibrary_cool_string)
+ "\r\n";
        myText.setText(someString);

    }
}
```

- In MyAndroidLibrary/manifests/AndroidManifest.xml comment out the MAIN/LAUNCHER intent. Otherwise two icons for your app will appear in the App drawer (keywords: prevent icons from appearing twice).

```
<activity android:name=".MainActivity">
    <!-- Don't make a main/launcher activity when being called by main app.
         Otherwise two icons for your app will appear in the App drawer.  -->
    <!--<intent-filter>-->
        <!--<action android:name="android.intent.action.MAIN" />-->
        <!--<category android:name="android.intent.category.LAUNCHER" />-->
    <!--</intent-filter>-->
</activity>
```

- Run MyAndroidApp Project. Run > Run app (Shift+F10). Wait for gradle to build and for the emulator to load your app. Observe the string coming from your library is reflected in your emulator:

```
From MyAndroidApp.
From MyAnroidLibrary.
```

- Alter a string from ..**mal**/res/values/strings.xml

```
<string name="myandroidlibrary_cool_string">>From MyAndroidLibrary XXX.</string>
```

- Run > Apply changes (Ctrl+F10). Wait for gradle to build and for the emulator to reload your app. Observe that the changed string is reflected in your emulator.

```
From MyAndroidApp.
From MyAnroidLibrary XXX.
```

(Bentley, n.d.) 2018-02-19 C:\Users\John\Documents\Sda\Code\Android\Examples\LibraryRefDemo
(Google, n.d. Gradle Plugin User Guide) http://tools.android.com/tech-docs/new-build-system/user-guide#TOC-Library-projects
(Goolge Android Tools, 2014. Develop > Tools) https://developer.android.com/studio/projects/android-library.html

Import a custom Android Library Project *.aar ([Unverified] Orthodox method. Not recommended)

If you follow https://developer.android.com/studio/projects/android-library.html#AddDependency and get an error like "you can't import this module as another with the same name exists" then delete the "same name" subfolder (e.g. app-debug") from the project into which you are importing.

https://stackoverflow.com/questions/33172065/project-already-contains-module-with-this-name-android-studio

(Deprecated) Create Custom Java Library from Android Studio

{We no longer create a custom (Pure) Java Library from within Android Studio. Instead we create one from within IntelliJ, in order to ensure this sort of library can be called from both MyAndroidApp (run from Android Studio) and MyJavaApp (run from Intellij IDEA).

Instead see above: "Using Intellij IDEA (with custom Libraries) > Create Custom Java Library and Convert Custom Java Library.}

Create a Project with only a Java-Library module...

First create an ordinary Android Project with an Android Module:

- Close current project if open.
- Open {Android Studio} to the 'Welcome to Android Studio' screen > [Start a new Android Studio project]:
- Wizard "Create New Project":
  - |Create Android Project|
  - Application name: `MyJavaLibrary`
  - Company domain: `example.com.au`
  - Project location: `..\LibraryRefDemo\Libraries\MyJavaLibrary`
  - Package name: `au.com.example.mjl` (this will be the namespace for the root module, our java code will live in another module and namespace that we'll subsequently create). [Next]
  - |Target Android Devices|. Accept defaults (or as desired). [Next]
  - |Add an Activity to Mobile| > "Add No Activity". [Finish]
- Wait for gradle to finish building.
- Open Project Pane. Select "Android" View from combobox.
- Observe there is an "app" module. We are going to create an additional module ...

Create a Java Library Module:

- File > New ... > New Module ...
- Wizard "Create New Module".
- |New Module| > Java Library. [Next]
- |Java Library| "Configure your new module".
- Library name: `mjl`
- Java package name: `au.com.example.mjl`
- Java class name: `Start`
- [Finish]

Do not delete the MyJavaLibrary Project's root module, app. If you do then, by default, your mjl will run once successfully but subsequent updates to code won't be incorporated into the second and subsequent runs.

There may be a way to properly sort out the gradle files but my gradle knowledge is limited.

(Bentley, n.d.) 2018-03-03

In your Java Library Module shove some code in:

- In your newly created Java Class, Start, add a main method. Ensure your main method has the correct signature (specifically include "String[] args"). For example use the following class for hello world purposes. mjl/java/au.com.example.mjl/Start ...

```
package au.com.example.mjl;

public class Start {
    public static void main(String[] args) {
        System.out.println(getCoolString());
    }

    // Must be public because we want to reference the method directly later
    public static String getCoolString() {
        return "From My Java Library";
    }
}
```

- Project Pane > Android View (from combobox) > Gradle Scripts > `build.gradle (Module: mjl)`:
- Verify apply plugin set propery:

```
apply plugin: 'java-library'
```

- Verify dependency as follows:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
}

// You can delete or comment out ...
//sourceCompatibility = "1.7"
//targetCompatibility = "1.7"
```

- Use an independently installed JDK, not Android's Embedded JDK.
- File > Project Structure ... > |SDK Location| > JDK location:
  - "Use embedded JDK (recommended)": unticked.
  - C:\Program Files\Java\jdk1.8.0_161
- Tools > Android > Sync Project with Gradle files (or click "Sync Now" link in yellow tip bar).
- Run > Edit Configurations …
  - Click on plus + symbol > Application
  - Name: MyJavaRun
  - Main class: au.com.example.mjl.Start
  - Working Directory: ... \LibraryRefDemo\Libraries\MyJavaLibrary
  - Use classpath of module: mjl
  - JRE: Default
  - [OK]
- Verify module can run...
- On the toolbar near the run triangle ensure the " MyJavaRun" configuration is selected. Click on the run green triangle.
- (After gradle build) observe in the "Run" pane the string output:

```
From My Java Library
```

Copy java code files from elsewhere (if you have any):

- Copy java code into: ..\LibraryRefDemo\Libraries\MyJavaLibrary\mjl\src\main\java\au\com\example\mjl.

You might have to change package names in the copied code.

- In Android Studio, Project Pane, observe these files show up (you might have to wait for a refresh event. Restart {Android Studio} if necessary to force a refresh).

Test run configuration OK again:

- Make some changes to your code, e.g.:

```
public class Start {
    public static void main(String[] args) {
        System.out.println(getCoolString());
    }

    // Must be public because we want to reference the method directly later
    public static String getCoolString() {
        return "From My Java Library YYY";
    }
}
```

- On the toolbar near the run triangle ensure the " MyJavaRun" configuration is select. Click on the run green triangle.
- Observe the updated string comes through in the "Run" pane.

```
From My Java Library YYY
```

Note as of 2018-03-03 Intellij IDEA 2017.3 can't, now, run MyJavaLibrary due to not supporting the latest gradle plug-in. I'm expecting this will change from IDEA 2018.1.

> It is going to stop working sometimes between 3.0-alpha2 and the final 3.0 version of the plugin.
>
> The model that the gradle plugin export to Studio/IJ to setup the projects is changing in a breaking way. Right now it sort of sends information in both the old and new way though the former is not always accurate depending on the build setup. This is going to change soon and this will break IJ until they can get bundle the Studio plugin 3.0.
>
> https://www.reddit.com/r/androiddev/comments/6c71av/using_android_gradle_plugin_300alpha1_with/dhssvlk/

https://stackoverflow.com/questions/46634835/cant-use-android-gradle-plugin-3-0-with-intellij-idea

Reference Custom Java Library from Android Project

Setup Referencing

MyJavaLibrary module ("mjl") is a library previously created in IntelliJ IDEA and converted to gradle (see "Using Intellij IDEA (with custom Libraries)" > Create Custom Java Library and Convert Custom Java Library).

To reference the MyJavaLibrary project from the MyAndroidApp project:

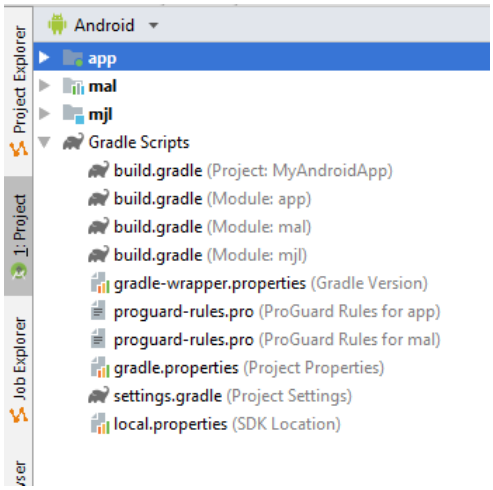
- {Android Studio} > File > Open Recent > [MyAndroidApp]. Open in [This Window].
- In the Android Project ("MyAndroidApp") `settings.gradle (Project Settings)` add mjl info:

```
include ':app'

include ':mal'
// Despite the name "project" we actually need to reference the module directory
// of a project. It doesn't matter whether there is a trailing slash '/' or not.
project(':mal').projectDir =
        new File(settingsDir, '../Libraries/MyAndroidLibrary/mal/')

include ':mjl'
// Despite the name "project" we actually need to reference the module directory
// of a project. It doesn't matter whether there is a trailing slash '/' or not.
project(':mjl').projectDir =
        new File(settingsDir, '../Libraries/MyJavaLibrary/')
// Not '../Libraries/MyJavaLibrary/mjl/'. We need the parent directory of 'src'.

// or
// project(':mjl').projectDir = new File(settingsDir,
'../../../../Java/Examples/LibraryRefDemo/Libraries/MyJavaLibrary/')
```

- In MyAndroidApp's root module `build.gradle (Module:app)`, add mjmodule info:

```
dependencies {
...
    implementation project(':mal')
    implementation project(':mjl')
...
}
```

- Tools > Android > Sync Project with Gradle files (or click on the yellow tip bar "Sync Now").
- Observe the mjl is now referenced in th Project Pane (Android View).



Verify MyJavaLibrary (mjl) module can run independently ...

Create a run configuration for MyJavaLibrary:

1. Using a gradle run configuration:
    a. Create a Gradle based run configuration.
        i. Gradle Pane > MyAndroidApp > :mjl > Tasks > other > runApp
        ii. Right click > Create ' C:/Users/John/Documents/Sda/Code/Android/Examples/LibraryRefDemo/Libraries/MyJavaLibrary [runApp]'
            1. Name: **'MyJavaLibrary [run]'**.
            2. Otherwise Accept Defaults.
            3. [OK]
        iii. Observe in Intellij's toolbar the run configuration 'MyJavaApp [run]' now exist.

2. (Deprecated) Using native run configuration:
    - Add a Run Configuration as specified in Create Java Project/Module above.

      Note we can use Android's Embedded JDK as we can run our Java module OK (presumably because we have a Project with an intact root **android** module).

    - On the toolbar near the run triangle ensure the " MyJavaRun" configuration is selected. Click on the run green triangle.

Verify MyJavaLibrary Run. Main Menu > Run > 'MyJavaLibrary [RunApp]'. Observe in the "Run" pane the string output:

```
12:53:49: Executing task 'runApp'...

Executing tasks: [runApp]

> Task :compileJava
```

```
> Task :processResources NO-SOURCE
> Task :classes

> Task :runApp
MyJavaLibrary Main Class: From MyJavaLibrary ZZZ.

BUILD SUCCESSFUL in 6s
2 actionable tasks: 2 executed
12:54:02: Task execution finished 'runApp'.
```

Verify MyJavaLibrary reference from MyAndroidApp by running

Verify you can reference a string resource in MyJavaLibrary from MyAndroidApp:

- Verify In ..LibraryRefDemo\Libraries\MyJavaLibrary\mjl\src\main\java\au\com\example\mjl\MyStrings.java

```
package au.com.example.mjl;

public class MyStrings {
    public static String GetStringFromLibrary() {
        return "From My Java Library ZZZ";
    }
}
```

- Verify In MyAndroidApp app/res/layout/activity_main.xml

```
<TextView
    android:id="@+id/myandroidapp_output"
    android:layout_width="wrap_content"
...
```

- Add some code in MyAndroidApp to pickup code from MyJavaLibary. In MyAndroidApp/app/java/au.com.example.myandroidapp/MainActivity add a reference to the mjl string ...

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = (TextView) findViewById(R.id.myandroidapp_output);
        String someString =
getResources().getString(R.string.myandroidapp_cool_string) + "\r\n";
        someString +=
getResources().getString(au.com.example.mal.R.string.myandroidlibrary_cool_string) +
"\r\n";
        someString += au.com.example.mjl.MyStrings.GetStringFromLibrary() + "\r\n";
        myText.setText(someString);

    }
}
```

- Run MyAndroidApp. On the toolbar select the "app" configuration. Click the Green arrow.

  If you get an error "you can't install" then Build > Clean Project.

  If you get an error
  > Error: **Default interface methods** are only supported starting with Android N (--min-api 24): java.util.Collection com.google.common.collect.ListMultimap.get(java.lang.Object)

... or ...

If you get an error
> Caused by: com.android.tools.r8.utils.AbortException: Error: **Default interface methods** are only supported starting wit ...

Then add the following in `build.gradle (Module: app)`:

```
android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.myandroidapp"
        minSdkVersion 15
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

...
```

https://stackoverflow.com/a/49525685/872154

- Observe the string coming from your library is reflected in your emulator (or connected device).

```
From MyAndroidApp.
From MyAndroidLibrary XXX.
From MyJavaLibrary ZZZ.
```

- Modify a string in MyJavaLibrary.
...\LibraryRefDemo\Libraries\MyJavaLibrary\mjl\src\main\java\au\com\example\mjl\MyStrings.java

```
public static String GetStringFromLibrary() {
        return "From MyJavaLibrary YYY";
}
```

- Run > Apply changes. Observe that the string is reflected in your emulator (or connected device).

```
From MyAndroidApp
From MyAndroidLibrary XXX
From MyJavaLibrary YYY
```

References (Word)

Bentley, J., n.d. Experiment.

Bentley, J., n.d. *Experimentally Verified.*

Google, (n.d.). *Gradle Plugin User Guide*. [Online]
Available at: http://tools.android.com/tech-docs/new-build-system/user-guide.

Goolge Android Tools, (2014-Jan-11). *Develop > Tools*. [Online]
Available at: http://developer.android.com/tools/index.html.

Katz, D., (2012-Aug-22). *Avoiding 'Source not found' when I debug in Eclipse*. [Online]
Available at: http://www.jayway.com/2012/08/22/avoiding-source-not-found-when-i-debug-in-eclipse/
[Accessed 2013-Jul-01].

Microsoft, (2014-Aug-04). *Scripting with Windows PowerShell*. [Online]
Available at: https://technet.microsoft.com/library/bb978526.aspx
[Accessed 2015-Jul-02].

Document Licence