

For all exercises the complete project should be your submission (excluding the .gradle, .idea, build folders). Add a single file "details.txt" alongwith submissions and add your explanation of what you did for the exercises - any docs, URLs that you referenced so others can understand.

1) Look up java plugin documentation. Make changes in manifest to make it executable with correct class. When run using `java -jar JAR_NAME_HERE` the output should be text "Hello World" on the console.

Initializing a gradle project

```
[15:49:28] gaurav@gaurav:Gradle_Assignment (gradle)$ gradle init --type java-library  
  
BUILD SUCCESSFUL in 0s  
2 actionable tasks: 2 executed  
[15:50:19] gaurav@gaurav:Gradle_Assignment (gradle)$
```

Ref: <https://guides.gradle.org/building-java-libraries/>

```
[15:50:19] gaurav@gaurav:Gradle_Assignment (gradle)$ ll  
total 40  
drwxr-xr-x 5 gaurav ttn 4096 Mar  5 15:50 ./  
drwxr-xr-x 5 gaurav ttn 4096 Mar  5 15:49 ../  
-rw-r--r-- 1 gaurav ttn 1034 Mar  5 15:50 build.gradle  
drwxr-xr-x 3 gaurav ttn 4096 Mar  5 15:50 gradle/  
drwxr-xr-x 4 gaurav ttn 4096 Mar  5 15:50 .gradle/  
-rwxr-xr-x 1 gaurav ttn 5296 Mar  5 15:50 gradlew*  
-rw-r--r-- 1 gaurav ttn 2260 Mar  5 15:50 gradlew.bat  
-rw-r--r-- 1 gaurav ttn   367 Mar  5 15:50 settings.gradle  
drwxr-xr-x 4 gaurav ttn 4096 Mar  5 15:50 src/
```

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

Library.java

```

sourceSets{
    main{
        java{
            srcDirs = ['src/main/java']
        }
    }
    test{
        java{
            srcDirs = ['src/test/java']
        }
    }
}
task runMe(type: JavaExec, dependsOn:'classes'){
    main='Library'
    classpath = sourceSets.main.runtimeClasspath
}

```

Build.gradle. Code responsible for executing the main class in the project.

```

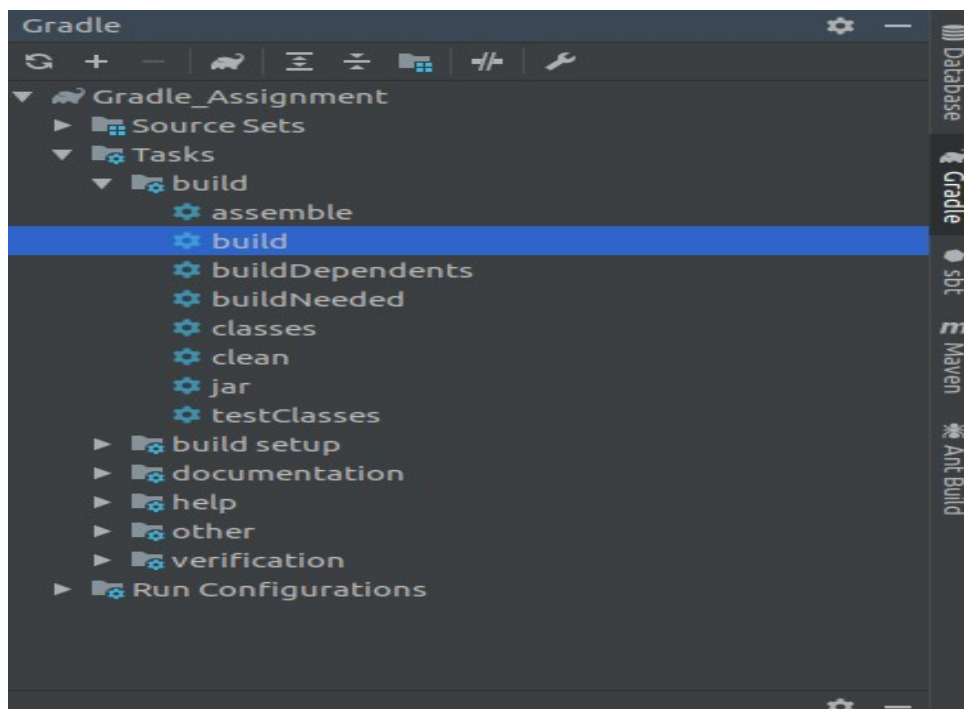
jar {
    manifest {
        attributes "Main-Class": "Library"
    }

    from {
        configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
    }
}

```

Ref: <https://www.baeldung.com/gradle-fat-jar>

Build gradle project:



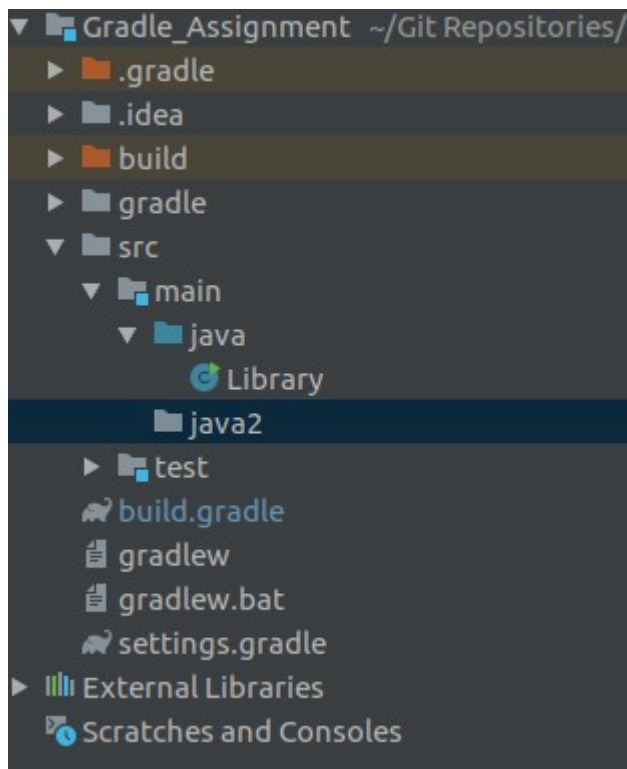
```

[00:23:08] gaurav@gaurav:Gradle_Assignment (master)$ cd build/
classes/      libs/      reports/      test-results/ tmp/
[00:23:08] gaurav@gaurav:Gradle_Assignment (master)$ cd build/libs/
[00:23:16] gaurav@gaurav:libs (master)$ ll
total 12
drwxr-xr-x 2 gaurav ttn 4096 Mar  6 00:18 ./
drwxr-xr-x 7 gaurav ttn 4096 Mar  6 00:19 ../
-rw-r--r-- 1 gaurav ttn  716 Mar  6 00:18 Library.jar
[00:23:17] gaurav@gaurav:libs (master)$ java -jar Library.jar
Hello World
[00:23:23] gaurav@gaurav:libs (master)$

```

2) look up idea plugin. make changes in build.gradle so that the sources of src/main/java as well as src/main/java2 are taken as sources. Ensure that when you make JAR file class files in both are added to the JAR. This will teach you how projects with non-conventional structure can be used with gradle.

Directory Created :



```

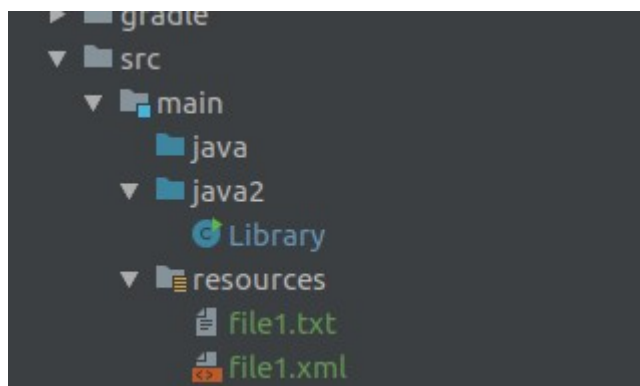
sourceSets{
    main{
        java{
            srcDirs 'src/main/java'
            srcDirs 'src/main/java2'
        }
    }
    test{
        java{
            srcDirs = ['src/test/java']
        }
    }
}

task runMe(type: JavaExec, dependsOn:'classes'){
    main='Library'
    classpath = sourceSets.main.runtimeClasspath
}

jar {
    manifest {
        attributes "Main-Class": "Library"
    }
    baseName = "Library"
    from {
        configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
    }
}

```

3) add 2 files file1.xml and file1.txt in src/main/resources manually. make changes so that when creating jar only file1.xml is added to the jar.



Files added.

```

sourceSets{
    main{
        java{
            srcDirs 'src/main/java'
            srcDirs 'src/main/java2'
        }
        resources{
            srcDirs 'src/main/resources'
        }
    }
    test{
        java{
            srcDirs = ['src/test/java']
        }
    }
}

```

Resources mentioned in source sets.

```

jar {
    // include('file1.xml')
    exclude('file1.txt')
    manifest {
        attributes "Main-Class": "Library"
    }
    baseName = "Library"
    from {
        configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
    }
}

```

Including and excluding files in jar.

4) find how to what is an uberjar. Make changes so you can use commons lang3 StringUtil in your jar. Make this uber jar executable. The output should be text but that should be using the StringUtils class of commons lang3

An **uber-JAR**—also known as a fat JAR or JAR with dependencies—is a JAR file that contains not only a Java program, but embeds its dependencies as well. This means that the JAR functions as an "all-in-one" distribution of the software, without needing any other Java code.

```
// https://mvnrepository.com/artifact/org.apache.commons/commons-io
compile group: 'org.apache.commons', name: 'commons-lang3', version: '3.0'

// https://mvnrepository.com/artifact/org.apache.commons/commons-io
runtime group: 'org.apache.commons', name: 'commons-lang3', version: '3.0'
```

Dependency Added.

```
jar {
    // include('file1.xml')
    exclude('file1.txt')
    manifest {
        // attributes "Main-Class": "Library"
        attributes "Main-Class": "Question4"
    }
    // baseName = "Library"
    baseName = "Question4"
    from {
        configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
    }
}
```

Changed the Main-Class.

5) Find a maven repository and add it as a repository. You can use bintray, jcenter

Dependency URL (<https://mvnrepository.com/artifact/org.apache.commons/commons-lang3/3.0>)

```
dependencies {
    // This dependency is exported to consumers, that is to say found on their compile classpath.
    api 'org.apache.commons:commons-math3:3.6.1'

    // This dependency is used internally, and not exposed to consumers on their own compile classpath.
    implementation 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testImplementation 'junit:junit:4.12'

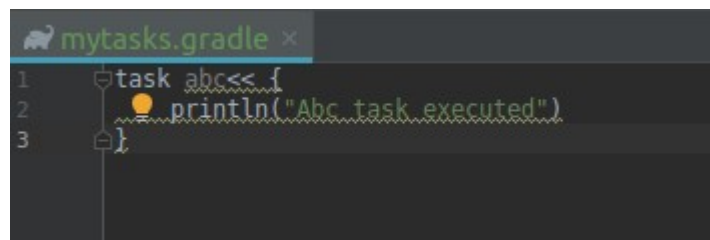
    // https://mvnrepository.com/artifact/org.apache.commons/commons-io
    compile group: 'org.apache.commons', name: 'commons-lang3', version: '3.0'

    // https://mvnrepository.com/artifact/org.apache.commons/commons-io
    runtime group: 'org.apache.commons', name: 'commons-lang3', version: '3.0'
}

// 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()
}
```

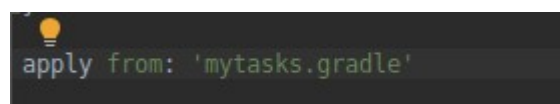
Dependency added.

6) Write a task in file "mytasks.gradle" and use it in your build.gradle.



```
mytasks.gradle x
1 task abc<<{
2     println("Abc task executed")
3 }
```

Task created



```
apply from: 'mytasks.gradle'
```

File applied to build.gradle


```
[19:25:12] gaurav@gaurav:Gradle_Assignment (master)$ gradle abc
```

```
> Task :abc
```

```
Abc task executed
```

```
Deprecated Gradle features were used in this build, making it incompatible with Gradle 5.0.
```

```
Use '--warning-mode all' to show the individual deprecation warnings.
```

```
See https://docs.gradle.org/4.10.2/userguide/command\_line\_interface.html#sec:command\_line\_warnings
```