Prepared By - Nirmala Kumar Sahu

INDEX

Gradle ----------

Gradle

Introduction

Q. What is the meaning of building the application/ project?

Ans. Keep application or project ready for execution is called building the application/ project. This involves,

- Developing source files (1 task)
- Compilations (20 task)
- Adding dependencies/ libraries (jars) to CLASSPATH or BUILDPATH (15)
- Creating project standard directories (10 task)
- Packing the app (creating jars or war file) (1 task)
- Unit Testing (Programmer testing) (1 task)
- Generating API documentation (Javadoc tool) (1 task)
- Deployments (1 task)
- Sending mails/ notifications (optional) (1 task)

We need to repeat the above build processes for multiple times until satisfactory results comes.

Note: Instead of doing this build process activities manually. It's better to automate the complicated, repetitive operations of build process by using build automation tools. or by using batch (.bat) file (bad practice).

build.bat

```
cd E:\ Project1
javac *.java
cd E:\Project1\Dao
javac *.java
mkdir WEB-INF
cd WEB-INF
mkdir classes
cd classes
set classpath=E:\Tomcat9.0\lib\servlet-api.jar;<other jars>;
…….
……..
```

Limitations with .bat file

a. It is procedural we need do everything.
b. Not self-intelligent.
c. We can't create dependency among the tasks.
d. We can't place conditional tasks.

e. Can't download jars dynamically from internet.
and etc.

Note: To overcome the problems of manual process and .bat file-based automation we need to use advanced build automation tools like Ant, Maven, Gradle and etc.

Advantages and limitations with Ant

a. Ant Procedural (Another Neat Tool).
b. Ant can have dependency among tasks.
c. Ant is not self-intelligent.
d. Can't download jars from internet dynamically.
e. can create tasks as conditional tasks.

Looks better than .bat file but overall, not great build tool.

Note: To overcome these problems, we need to work either Maven or Gradle (Gradle is bit ahead).

Gradle Main Features

- Gradle is an open-source build automation tool that is designed to be flexible enough to build almost any type of software.

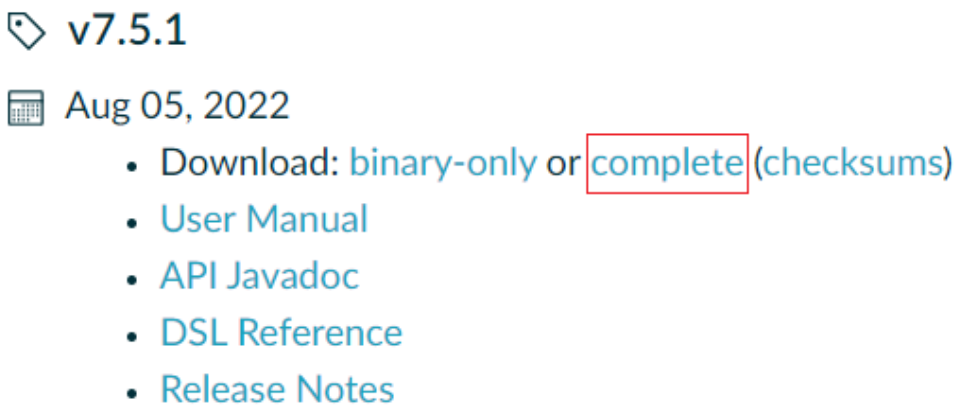
a. Customizable Build tool
   - Allows to write build scripts in Groovy, Kotlin and etc. languages.
   - Can be used for Java, Scala, and etc. JVM based languages.
b. Fast
   - Gives better performance compare to maven.
   - Uses incremental build process (only modified .java file will compile for next time build).
   - Gives Gradle caching concept to reuse jars, plugging and etc.
c. Powerful
   - So many facilities to make build process simple like gradlew (gradle wrapper) to use gradle without installing it.
   - Ability to work with multiple repositories to get jars, plugins and etc. from internet like maven, ivy, google, local file system repositories and etc.

Note: Gradle, we can learn in two ways,
   - CLI Gradle (To understand basics)
   - Eclipse Gradle

CLI Gradle

To keep gradle setup ready in our system follow the below steps,

a. Download gradle as zip file (complete one) from internet and extract it. [Gradle Releases], [Direct download Gradle v7.5.1 complete]



b. Make sure JAVA_HOME environment variable is pointing to JDK installation folder (ells to gradle to use specified version of Java internally).

   Variable name: JAVA_HOME
   Variable value: C:\Program Files\Java\jdk1.8.0_311

c. Make sure that PATH environment variable is pointing to gradle installation related bin directory (to use gradle command from any location of computer).

   Variable name: PATH
   Variable value: D:\gradle-7.5.1-all\bin;<old values>;

d. To check gradle installed properly or not
   CMD> gradle -version

The Technical features of Gradle

1. Gradle is declarative (self-intelligent).
2. Gradle is having life cycle.
3. Gradle work with multiple repositories like maven, ivy, google and etc.
4. Gradle is having incremental build Process.
5. Gradle gives standard directory structures to develop different types projects.
6. Gradle provides lots plugging having tasks to simplify the build process.

7. Gradle Caching feature given lot of reusability of plugins, jars and etc.
8. Easy to develop multi module projects.
9. Allows to convert to gradle projects to maven projects.
10. Integration with multiple IDEs.
11. No need of writing builds script in XML we can use DSL (Domain Specific Language like Groovy, Kotlin.)

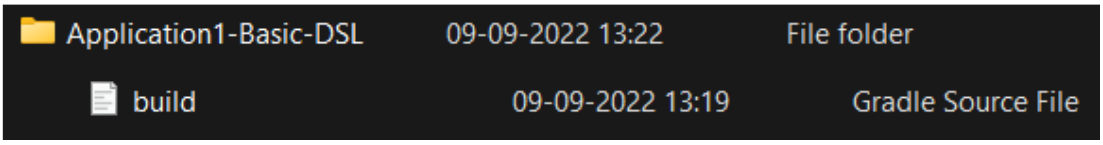
DSL Groovy basics

- Input file is build.gradle (alternate to pom.xml of maven).
- Gradle gives lots built in plugins having tasks with dependency.
  Plugins
  |--> tasks
  |--> actions
  To complete these actions, we can call methods.
- {.....} – encloser
- setDir('hello') = setDir:'hello' = Dir:'hello'

For gradle plugins info – [Link]

Some groovy exercise using CLI

1. Create a Folder and inside that folder create a file called build.gradle like bellow.



2. Inside that build file write the below code.

build.gradle

```
task t1 {
        doLast {
                println "Welcome to gradle"
        }
}
```

3. After that code CMD and run "gradle task -all" command to download and ready the default tasks and in 'Other tasks' section we can see our task.

```
D:\GradleApps\Application1-Basic-DSL>gradle task --all
Starting a Gradle Daemon (subsequent builds will be faster)
<-------------> 0% INITIALIZING [564ms]
> IDLE
```

4. To execute our task run "gradle t1"

```
D:\GradleApps\Application1-Basic-DSL>gradle t1

> Task :t1
Welcome to gradle

BUILD SUCCESSFUL in 883ms
1 actionable task: 1 executed
```

Examples

build.gradle

```
//Syntax 1
task t1 << {
	println "Welcome to gradle 1"
}

//Syntax 2
task t2 {
	println "Welcome to gradle 2"
}

//Syntax 3
task t3
t3 {
	doLast {
		println "Welcome to gradle 3"
	}
}

//Syntax 4
task t4
t4 {
	println "Welcome to gradle 4"
}

//Single line comment
/*Multiline comment*/
```

build.gradle

```
task "t1"
task ("t2")
task t3
task t4

t1 {
        doLast {println "Task 1"}
}

t2 {
        doLast {println "Task 2"}
}

t2 {
        doLast {println "Task 2 2 - Last"}
        doFirst {println "Task 2 2 - First"}
}

t3 {
        dependsOn t2, t1
        doLast {println "Task 3"}
}

t4 {
        doLast {println "Task 4"}
}

t4.dependsOn t3
defaultTasks 't4'
```

Gradle standard directory structure for Standalone apps

D:\Gradle Apps (Work folder, can be anything)
|--> Application2-Java
|--> src
|--> main
|--> java
|--> com.nt.basics
|--> Arithmetic.java

|--> test
|--> java
|--> build.gradle

- src/main/java for source code
- src/test/java for unit testing code

- Develop the above directory structure in your system and also create the java and build file.
- Then place the following code with in their respective files.

Arithmetic.java

```
package com.sahu.basics;

public class Arithmetic {

	public int sum(int a, int b) {
		return a+b;
	}

	public static void main(String[] args) {
		System.out.println("Arithmetic.main()");
		Arithmetic arithmetic = new Arithmetic();
		System.out.println("Result is - "+arithmetic.sum(10, 20));
	}

}
```

build.gradle

```
apply plugin: 'java'
```

After that we have run the following commands
CMD> gradle build

```
D:\GradleApps\Application2-Java>gradle build
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 8s
2 actionable tasks: 2 executed
```

CMD> java -cp build/libs/Application2-Java.jar com.sahu.basics.Arithmetic

```
D:\GradleApps\Application2-Java>java -cp build/libs/Application2-Java.jar com.sahu.basics.Arithmetic
Arithmetic.main()
Result is - 30
```

If you modify the code then you have to go for the following steps

- Modification source code
- gradle clean build
- java -cp build/libs/Application2-Java.jar com.sahu.basics.Arithmetic

Note: To make jar file as standalone executable jar file we should add manifest file having main class name.

manifest.mf

Manifest-Version: 1.0
Main-Class: com.sahu.basics.Arithmetic

CMD> jar cmf manifest.mf Application2-Java.jar

We need not to all these things in gradle.

To add entries to manifest file add the following content in build.gradle then the required content will add to manifest file of jar automatically.

build.gradle

```
apply plugin: 'java'
version= '1.0'

jar {
        manifest {
                attributes 'Main-Class': 'com.sahu.basics.Arithmetic'
        }
}
```

Then run the below command in CMD
CMD> gradle clean build
CMD> java -jar build/libs/Application2-Java-1.0.jar

Note: There is no "run" task in "java" plugin, but we have "run" task in "application" plugin.

- Create another application name Application3-Java, copy the src folder with content and build.gradle file from Application2-Java.
- Then modify the build.gradle file as shown below.

build.gradle

```
apply plugin: 'application'
version= '1.0'
mainClassName= 'com.sahu.basics.Arithmetic'
```

Then run the below command in CMD
CMD> gradle run

```
D:\GradleApps\Application3-Java>gradle run

> Task :run
Arithmetic.main()
Result is - 30

BUILD SUCCESSFUL in 940ms
2 actionable tasks: 2 executed
```

Gradle Wrapper

- By disabling gradle software in our computer still we can execute gradle application, that is known as Gradle Wrapper.

- Create another application name Application4-GradleWrapper, copy the src folder with content and build.gradle file from Application3-Java.
- Then modify the build.gradle file as shown below.

build.gradle

```
apply plugin: 'application'
version= '1.0'
mainClassName= 'com.sahu.basics.Arithmetic'
task myTask(type:Wrapper)
```

Then follow the below steps to feel the gradle wrapper features.

- CMD> gradle run
- CMD> gradle myTask
- Deactivate gradle from your computer by removing gradle bin directory from PATH environment variable.
- Now if you run CMD> gradle run then we will get error.
- So, now run CMD> gradlew run
- After this it will download from internet and execute our application and give the output.

```
D:\GradleApps\Application4-GradleWrapper>gradlew run
Downloading https://services.gradle.org/distributions/gradle-7.5.1-bin.zip
..........10%.........20%.........30%.........40%.........50%.........60%.....
.....90%.........100%

> Task :run
Arithmetic.main()
Result is - 30

BUILD SUCCESSFUL in 44s
2 actionable tasks: 1 executed, 1 up-to-date
```

Note: We can send this application folder having gradlew file to another computers using Git or SVN and they can run the Gradle app without installing gradle in their computer by just using gradlew.

- To use different version of gradle with gradlew
  CMD> gradlew wrapper --gradle-version 6.2.1

Working with Eclipse

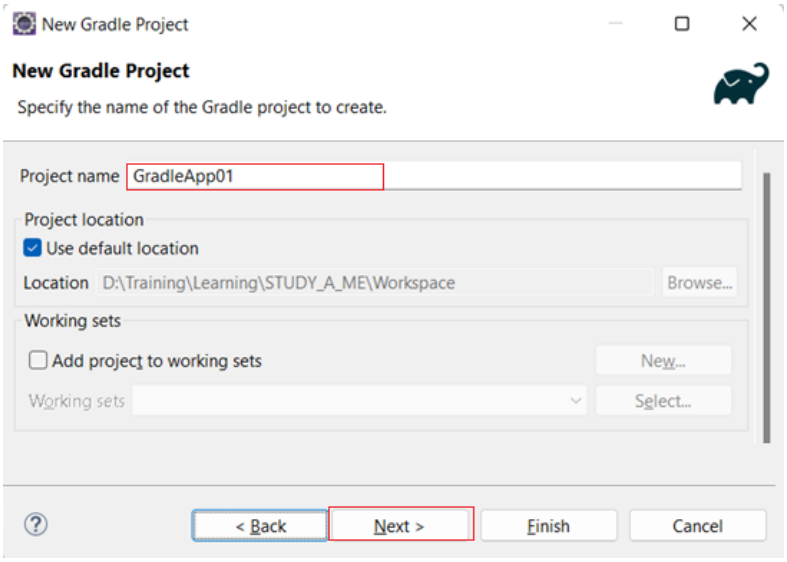
- Old versions of Eclipse (up to 2018), we have to install Buildship plugin manually.
- But in new versions of Eclipse Buildship plugin is built-in plugin.

To confirm/ install Build ship plugin

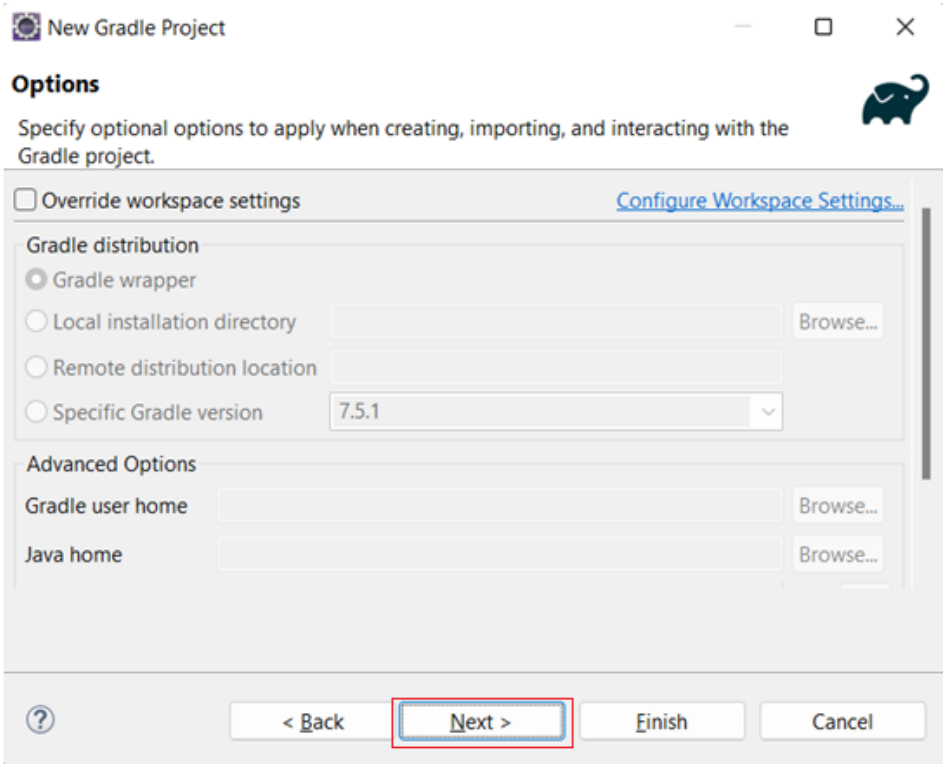
Go to Help > Eclipse Marketplace… > search for Buildship and observe for "installed" message (or) install (not already installed).

To create Gradle project in Eclipse follow the below steps:

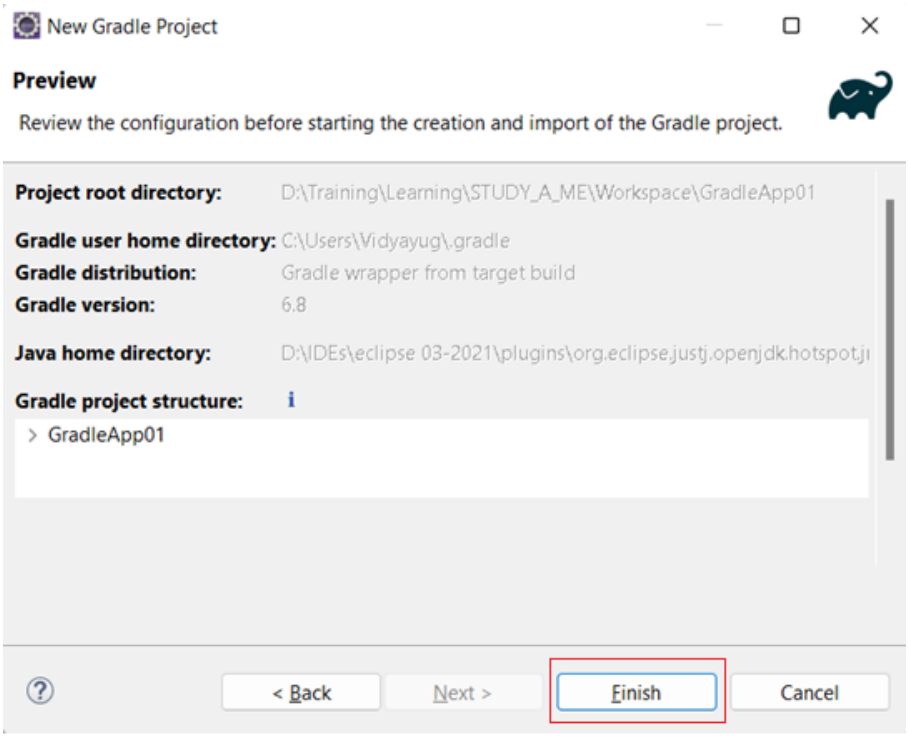
Step 1: File > New > Project… > Gradle > Gradle Project > Give the Project name and click on Next >.



Step 2: Leave it default because by default the "Gradle wrapper" option is choose show it can download from internet otherwise you can specify the bin location of gradle extraction of your computer by choosing "Local installation directory" then click on Next >.

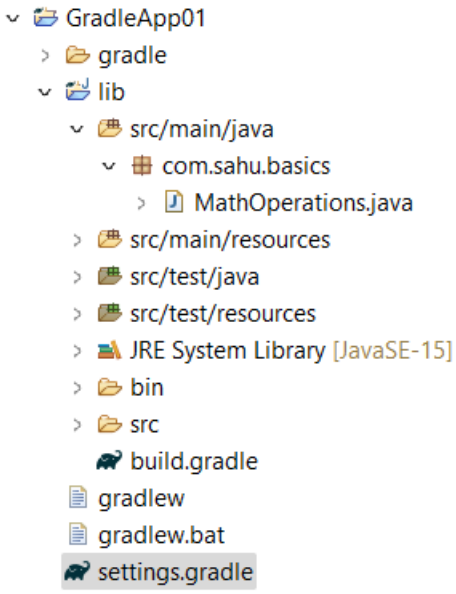


Step 3: Wait for downloading once it done click on the Finish button.



Note: In latest version of gradle the directory structure is changed everything comes inside "lib".

Directory Structure of GradleApp01:



- Develop the above directory structure using Gradle project, remove the default package and folder and create the new package and java class.
- Then place the following code with in their respective files.

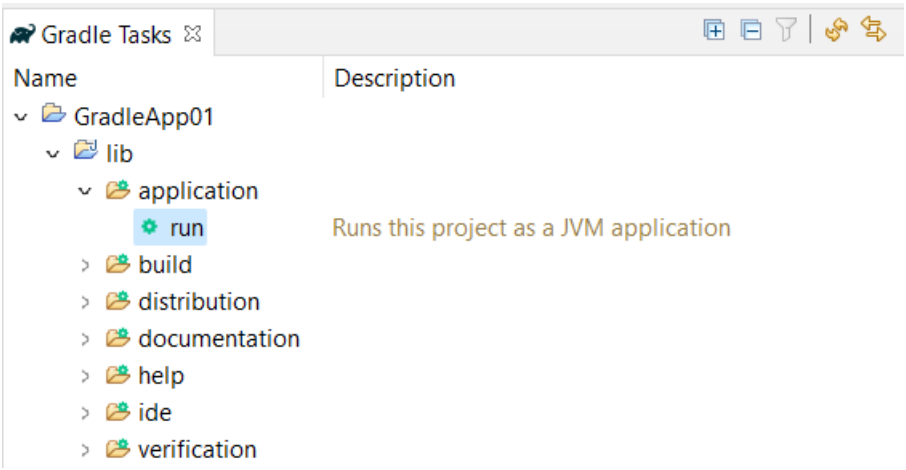
build.gradle

```
plugins {
   id 'application'
}
mainClassName='com.sahu.basics.MathOperations'
```

MathOperations.java

```
package com.sahu.basics;
public class MathOperations {
        public int sum(int x, int y) {
                return x+y;
        }
        public static void main(String[] args) {
                MathOperations operations = new MathOperations();
                System.out.println("Result is  - "+operations.sum(100, 200));
        }
}
```

To run go to Gradle Tasks section expand you project expand the lib go to application click on run, then you will get the output.



Note: After doing any change in build.gradle gradle refresh is required to reflect the changes for that, Right click on the project then Gradle then click on "Refresh Gradle Project" option.

Unit Testing with Gradle

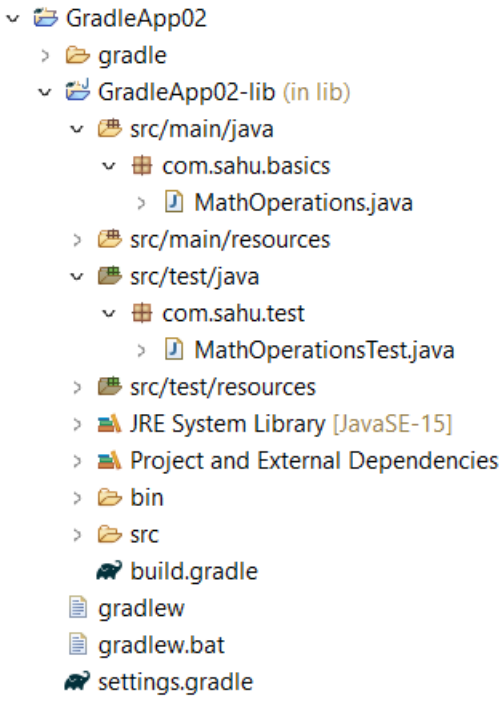
- Programmer Testing on his own piece of code is called Unit testing. Where expected results will be matched with actual results by writing @Test methods. using Junit API.
- Each @Test annotation method gives one test case where expected results will be compared with actual results. Take test cased on 1 per each verity.

- Repository is a small Db s/w (or) storage unit where jars, plugins, readymade projects are available and they are identified with 3 details,
  - groupId: company name/ software name
  - artifactId: jar file name/ plugin name/ project name
  - version: jar file version/ plugin version/ project version

Note:

- ✓ We can get all this dependency info from mvnrepository.com (just a website).
- ✓ In repositories {} encloser we can specify the repository like mavenCentral(), jcenter(), etc.
- ✓ Maven/ Gradle gives dependent jar files automatically when we add main jar file info in the build script, this is call transitive dependency.

Directory Structure of GradleApp02:



- Develop the above directory structure using Gradle project, remove the default package and folder and create the new package and java class.
- Copy the MathOperation.java from previous project.
- Then place the following code with in their respective files.

MathOperationsTest.java

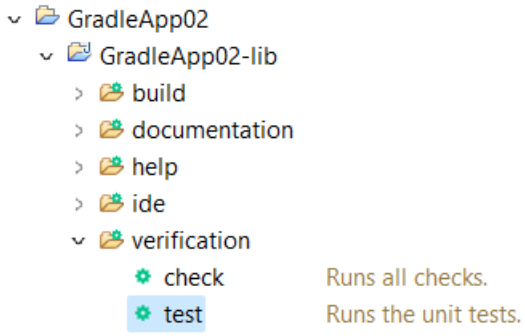
```
package com.sahu.test;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

import com.sahu.basics.MathOperations;

public class MathOperationsTest {

	@Test
	public void testPositives() {
		int expected = 300;
		int actual = new MathOperations().sum(100, 200);
		assertEquals("Test Positive", expected, actual);
	}
```

```
	@Test
	public void testNegatives() {
		int expected = -300;
		int actual = new MathOperations().sum(-100, -200);
		assertEquals("Test Negative", expected, actual);
	}

	@Test
	public void testMixed() {
		int expected = 100;
		int actual = new MathOperations().sum(-100, 200);
		assertEquals("Test Mixed", expected, actual);
	}

	@Test
	public void testZero() {
		int expected = 0;
		int actual = new MathOperations().sum(0, 0);
		assertEquals("Test Zero", expected, actual);
	}

}
```

build.gradle

```
plugins {
    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {
    // https://mvnrepository.com/artifact/junit/junit
	testImplementation group: 'junit', name: 'junit', version: '4.13.2'
}
```

To run the test cases, go to Gradle Tasks section expand you project expand the lib go to verification click on test, then build successful.



Now go to the below location to get the report

<Workspace>\GradleApp02\lib\build\reports\tests\test\index.html

Note:

- ✓ Gradle cache is specific to each gradle installation in a computer and it maintains various dependents gathered from various repositories and uses them across the multiple executions of Gradle build in our system. Gradle Cache Location: (C:\Users\<Uesr name>\.gradle\caches)

Repositories

These are the following repositories we have,

- Maven central – mavenCentral() (Apache), jcenter() (Bintray) (Internet based) [MVN Repository]
- Maven local – mavenLocal() (Specific to each computer)
- Maven third party –

```
maven {
        url ' .....' (Internet based)
}
```

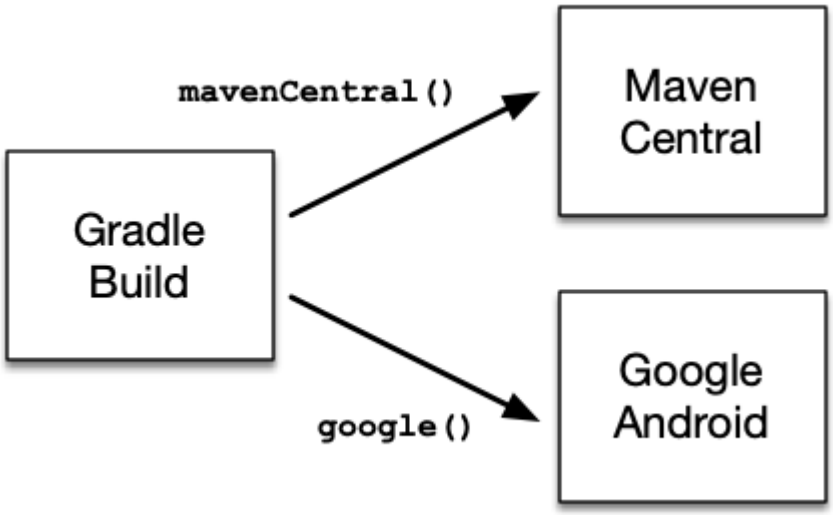
- Google – google() (Internet based) [Google Repository]
- IVY –

```
ivy {
        url '.....' (Internet based)
}
```

- File system – from computer hard driver (specific to one computer)

Note: Maven Local repository is specific to each computer we can see at C:\Users\<User name>\.m2\repository, it will have jar (dependencies). These dependencies are,

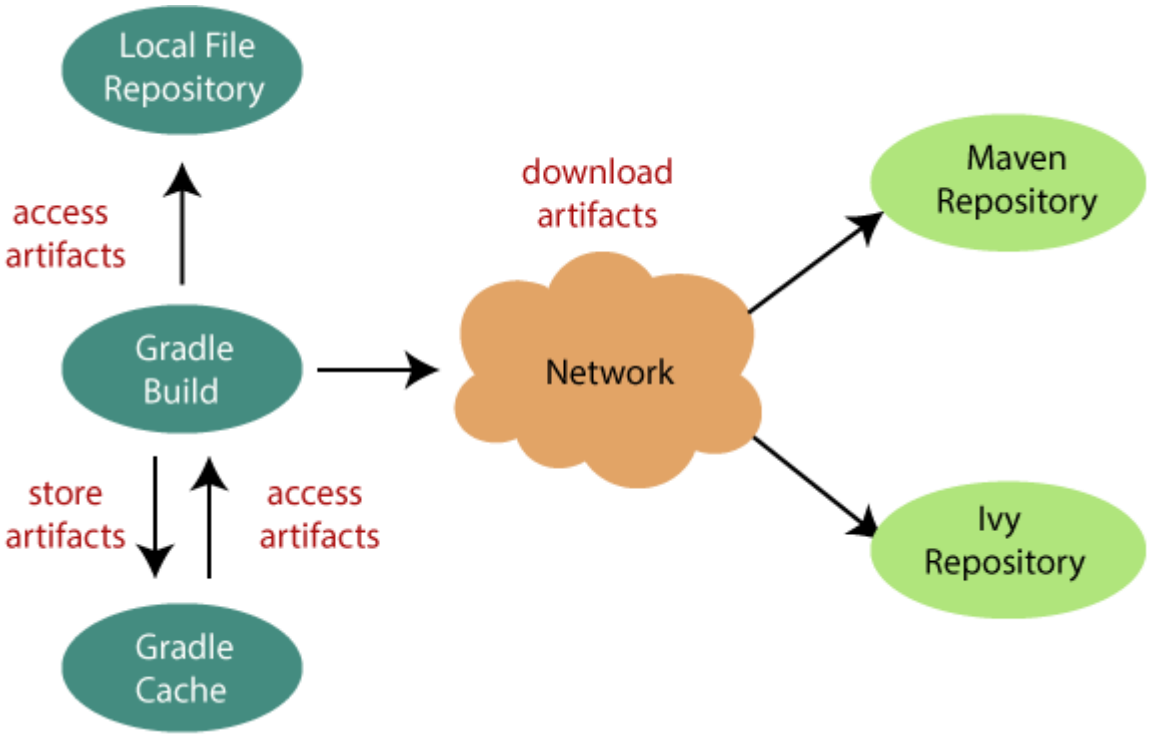
- Collected from maven central repository
- Collected from maven remote (third party) repositories
- Placed by the programmers



build.gradle

```
repositories {
   mavenCentral()
   jcenter()
   maven {
        url 'https://maven.repository.redhat.com/ga/'
   }
   mavenLocal()
   google()
}

dependencies {
        // From Maven central
        testImplementation group: 'junit', name: 'junit', version: '4.13.2'

        // From Third party
        implementation group: 'joda-time', name: 'joda-time', version:
'2.9.7.redhat-1'

        // From Maven local
        implementation group: 'commons-logging', name: 'commons-logging',
version: '1.2'

        // From Google repository
        implementation 'android.arch.core:common:1.1.1'

        // From File System
        implementation files('D:\\JAR\\voter-app.jar')
}
```

To get more about repositories: [Declaring Repositories]



Gradle additional features

ext {}:

- By using ext {} enclosure we can write version for jar files and we can use it for multiple dependencies.

build.gradle

```
ext {
        springVersion='5.2.8.RELEASE'
}

dependencies {
        implementation group: 'org.springframework', name: 'spring-context',
        version: "$springVersion"

        implementation group: 'org.springframework', name: 'spring-core',
        version: "$springVersion"
}
```

- ext variable never allow space and '-' symbol but allow '_' symbol.
- ext value can be under single quotes or double quotes.

- While specifying the jar version with ext variable name, version value must in double quotes and start with '$' symbol.

exclude:

- By using exclude option we can exclude specific dependent jar file from main jar file.

build.gradle

```
dependencies {
        implementation ('org.springframework:spring-
context:5.2.8.RELEASE') {
                exclude group: 'org.springframework', module: 'spring-jcl'
        }
}
```

strictly:

- By using strictly, we can get dependent jar fie from main jar file having its own independent version.

build.gradle

```
dependencies {
        implementation ('org.springframework:spring-
context:5.2.8.RELEASE')

        implementation ('org.springframework:spring-expression') {
                version {
                        strictly '5.2.5.RELEASE'
                }
        }
}
```

sourceSets {}:

- Gradle standard directories name can be changed using this concept.
- We have to do the following steps
  - Remove the standard directory structure.
  - Create your own set of directory structure for source code and testing code.
  - Create package and class according to you requirement then place the below sourceSets {} encloser information in build.gradle.

build.gradle

```
sourceSets {
        main {
                java {
                        srcDir 'source/main/kava'
                }
        }

        test {
                java {
                        srcDir 'source/verify/kava'
                }
        }
}
```

run {}:

- To make Gradle application collecting inputs from keyboard (Standard input device).

build.gradle

```
run {
        standardInput=System.in
}
```

Dependencies configurations/ Scopes

Configuration name	Role	Consumable ?	Resolvable ?	Description
api	Declaring API dependencies	no	no	This is where you should declare dependencies which are transitively exported to consumers, for compile.

implementation	Declaring implementation dependencies	no	no	This is where you should declare dependencies which are purely internal and not meant to be exposed to consumers.
compileOnly	Declaring compile only dependencies	no	no	This is where you should declare dependencies which are only required at compile time, but should not leak into the runtime. This typically includes dependencies which are shaded when found at runtime.
runtimeOnly	Declaring runtime dependencies	no	no	This is where you should declare dependencies which are only required at runtime, and not at compile time.
testImplementation	Test dependencies	no	no	This is where you should declare dependencies which are

				used to compile tests.
testCompileOnly	Declaring test compile only dependencies	no	no	This is where you should declare dependencies which are only required at test compile time, but should not leak into the runtime. This typically includes dependencies which are shaded when found at runtime.
testRuntimeOnly	Declaring test runtime dependencies	no	no	This is where you should declare dependencies which are only required at test runtime, and not at test compile time.

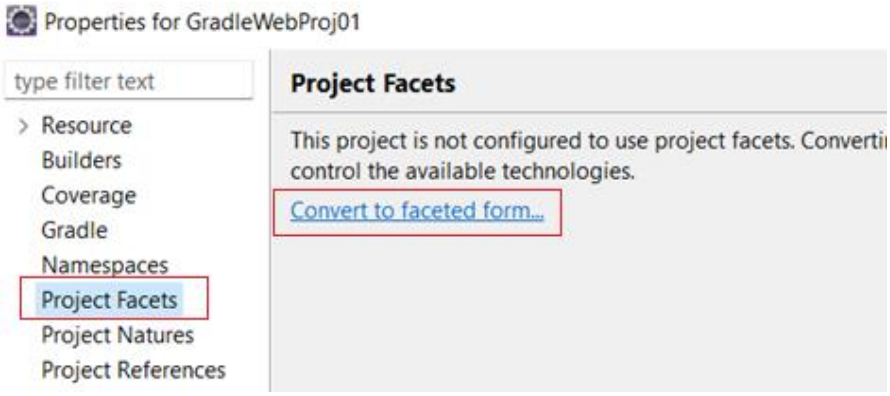
Note:

- ✓ api and implementation are good for source code development time.
- ✓ If it is a web application, webservice application, deployable in the server then prefer api.
- ✓ If it is a non-web application, non-webservice application then prefer implementation.
- ✓ api can be used only by importing a plugin called 'java-library'.
- ✓ testImplementation is good for unit testing.

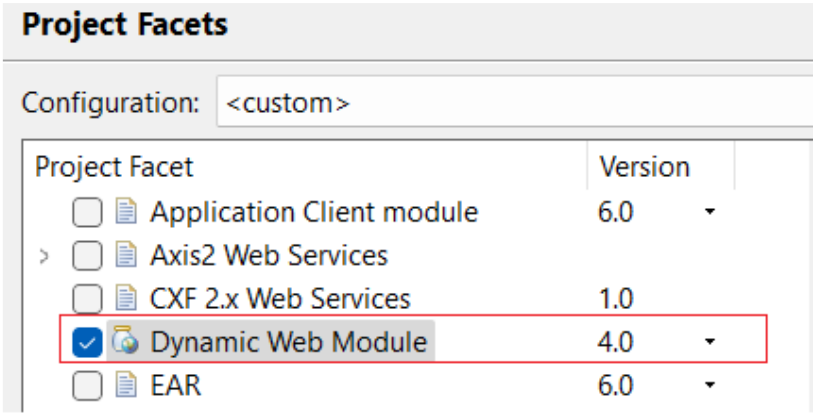
Working with Gradle Web applications

Step 1: Create Gradle Project, to convert into Gradle web application right click on the Project go to properties.

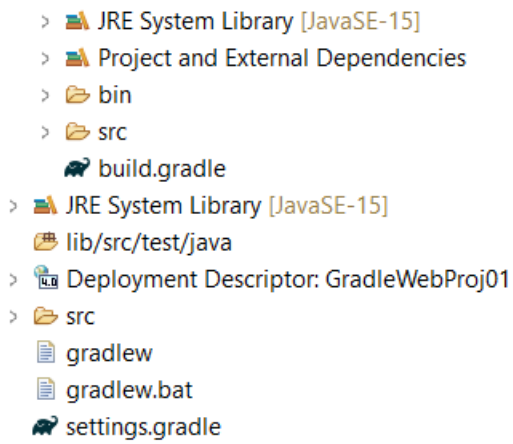
Step 2: Click on Project Facets the click on the Convert to faceted form… link.



Step 3: Now choose the Dynamic Web Module – 4.0 option, then click on Apply, Apply and close.

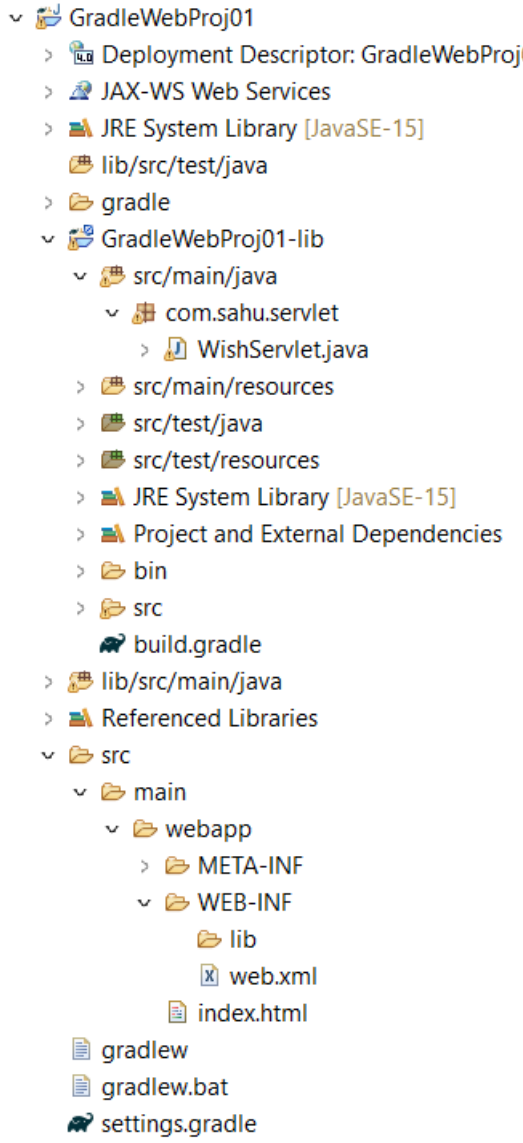


Observe the standard structure of Gradle web application, Project structure they change and they solve all the project related web application also.



Note: Jars added through gradle will be moved to classpath and will also be moved to WEB-INF/lib folder automatically.

Directory Structure of GradleWebProj02:



- Develop the above directory structure using Gradle project, remove the default package and folder and create the new package and java class.
- Convert into web application by going to project fact and choosing Dynamic Web Module option.
- Then place the following code with in their respective files.

build.gradle

```
plugins {
    id 'war'
}

repositories {
    jcenter()
}

dependencies {
    // Use JUnit test framework.
    testImplementation 'junit:junit:4.13'

        // https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api
        implementation group: 'javax.servlet', name: 'javax.servlet-api',
version: '4.0.1'
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee;
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" version="4.0">
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

index.html

```
<a href="wishurl">Get wish message</a>
```

WishServlet.java

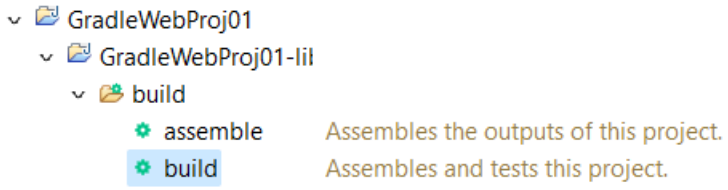
```
package com.sahu.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/wishurl")
public class WishServlet extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		PrintWriter pw = null;
		pw = response.getWriter();
		response.setContentType("text/html");

		pw.println("<h1 style='color:green'>Good morning</h1>");
		//close stream
		pw.close();
	}

	public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		doGet(request, response);
	}

}
```

We can run Gradle web application in 3 ways

a. Using IDE Run As option
   Right click Project > Run As > Run on Server

Note: We need not to add the webcontent folder in Deployment assembly because, Gradle solve the issue by using webapp folder and in current version the webapp folder is already added to deployment assembly.

b. Prepare war file using gradle and deploy the war file into the server manually.
   - Use war plugins.
   - Go to Gradle Tasks, expand your web application then expand the lib folder now go to build folder and run the “build” command.



   - Now go to your actual workspace directory system, and go the following location you will get the war file.
     <Workspace Home>\GradleWebProj01\lib\build\libs
   - Deploy the war file manually in Tomcat server webapps folder.
     D:\Tomcat9x\webapps

Note: In this process, there is a bug, the public files are not coming automatically to war file when you build the jar so, we have to add the public files manually.

c. Configure "tomcat", "jetty" and other server plugins to server as embedded server (very complex, not recommended).

Generate Java documentation using Gradle

- To get the Java documentation first we have to write java documentation comment like below in our code.

index.html

```
package com.sahu.basics;

/**
 * <b>Class performing <i>Arithmetic Operations</i></b>
 * like sum, sub, mul, div and etc..
 *
 * @author NK Sahu
 * @see java.lang.Math
 * @since 1.0v
 */
```

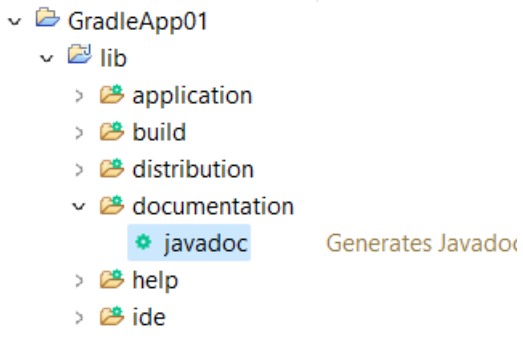
```java
public class MathOperations {

	/**
	 * Method performing <b>addition</b>
	 *
	 * @param x takes value 1
	 * @param y takes value 2
	 * @return give result
	 */
	public int sum(int x, int y) {
		return x+y;
	}

	public static void main(String[] args) {
		MathOperations operations = new MathOperations();
		System.out.println("Result is  - "+operations.sum(100, 200));
	}

}
```

- In your build file you have to plugin like java, application, etc. at least one.

build.gradle

```
plugins {
	id 'java'
	id 'application'
}
```

- Go to Gradle Tasks, expand your application then expand the lib folder now go to documentation folder and run the “javadoc” task.



- Then go the following location you will get your project documentation, special if you click on index.html then you will get everything.
  <Workspace Home>\GradleApp01\lib\build\docs\javadoc

Converting Normal Gradle project to Eclipse Gradle project

- Ready a normal gradle project.
- Add the following things in build.gradle

build.gradle

```
apply plugin: 'application'
apply plugin: 'eclipse'

mainClassName= 'com.sahu.basics.Arithmetic'
```

- Now open CDM on the project structure and run the below command
  CMD> gradle eclipse

```
D:\Training\Learning\Gradle Workshop\Application2-Java>gradle eclipse

BUILD SUCCESSFUL in 1s
3 actionable tasks: 3 executed
D:\Training\Learning\Gradle Workshop\Application2-Java>
```

- Now you can see the project structure change and you can import to Eclipse and run there.

Note: To remove effect of eclipse from project and to maker it normal gradle project we can use "cleanEclipse" plugin.

Converting pom.xml of Maven on Gradle's build.gradle

- Arrange a pom.xml from any Maven project or from internet other you can use the below pom.xml also.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>kms-api-examples</groupId>
```

```
<artifactId>kms-api-examples</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
		<dependency>
			<groupId>org.apache.httpcomponents</groupId>
			<artifactId>httpclient-cache</artifactId>
			<version>4.3.2</version>
			<type>jar</type>
			<scope>compile</scope>
		</dependency>
		<dependency>
			<groupId>org.apache.httpcomponents</groupId>
			<artifactId>httpmime</artifactId>
			<version>4.3.2</version>
			<type>jar</type>
		</dependency>
		<dependency>
			<groupId>com.fasterxml.jackson.core</groupId>
			<artifactId>jackson-core</artifactId>
			<version>2.4.0</version>
		</dependency>
		<dependency>
			<groupId>com.fasterxml.jackson.core</groupId>
			<artifactId>jackson-databind</artifactId>
			<version>2.4.0</version>
		</dependency>
	</dependencies>
<build>
  <plugins>
    <plugin>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.5.1</version>
     <configuration>
       <source>1.7</source>
       <target>1.7</target>
     </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

- Then execute below command in CMD
  CMD> gradle init –type=pom
  Then it will ask "Select build script DSL:", choose 1 and enter again it will asl to Generate type "yes" and hit enter

```
D:\Training\Learning\Gradle Workshop\App2>gradle init --type=pom

Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2] 1

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes
 no]yes


> Task :init
Maven to Gradle conversion is an incubating feature.
Get more help with your project: https://docs.gradle.org/7.5.1/userguide/migrating_from_maven.html

BUILD SUCCESSFUL in 22s
2 actionable tasks: 2 executed
```

- Now you can see the gradle folders and files, build.gradle also.

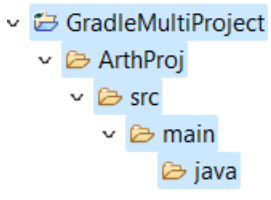
Gradle Multi Module Project

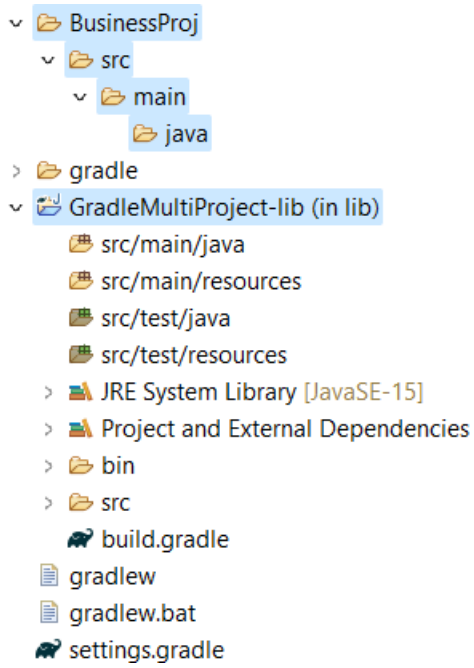
- According to AJILE model or Microservices architecture every task/ module will develop as separate sub project or jar file and will be linked with main application.

University Project
|--> Registrations (sub project – jar 1)
|--> Academics (sub project – jar 2)
|--> Sports (sub project – jar 3)

Note: Every sub project (module) will develop single deployable/ executable component. If interested you can integrate all sub projects into university project.

Step 1: Create a gradle project and under the project create 2 folder ArthProj and LogicalProj then under that create src/main/java folders like below and remove the auto generate the package and codes from lib.



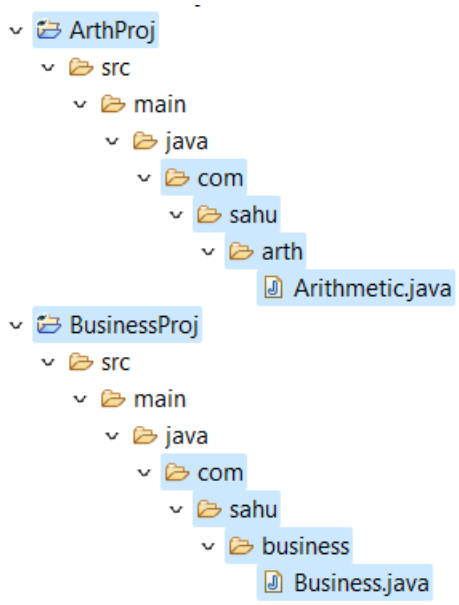


Step 2: Add the following entries in settings.gradle and do Gradle refresh project after that you can see the project.

settrings.gradle

```
rootProject.name = 'GradleMathProject'
include('lib', 'ArthProj', 'BusinessProj')
```

Step 3: Create the folder and java file respective their project and place the following code in their respective file.



Arithmetic.java

```
package com.nt.arth;

public class Arithmetic {

	public int sum(int x, int y) {
		return x+y;
	}

}
```

Business.java

```
package com.nt.business;

public class Business {

	public int findBig(int x, int y) {
		if (x==y)
			return 0;

		return x>y?x:y;
	}

}
```

---------- The END ----------