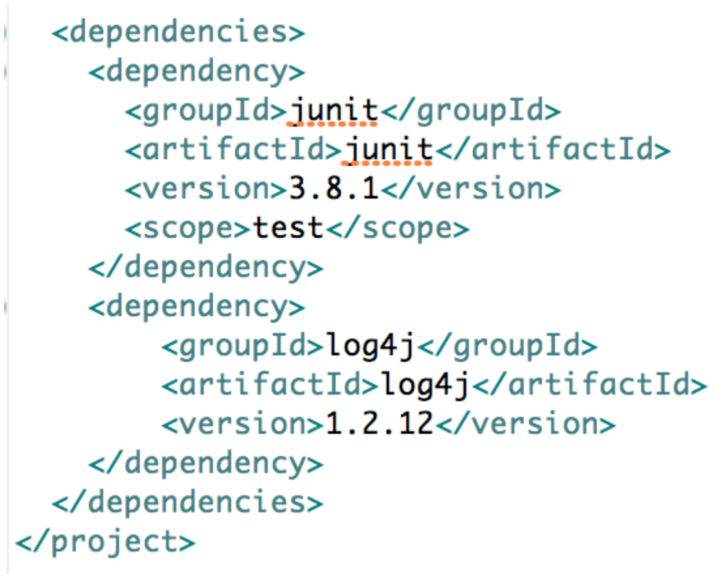
All crap about Maven is here
https://maven.apache.org/guides/getting-started/

To DEBUG maven build process use -X switch -> mvn -X install

As shown below, JUNIT scope if test and default scope for log4j is compile



```
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.12</version>
    </dependency>
  </dependencies>
</project>
```

and repository video here - https://www.youtube.com/watch?v=zA43zyLFk-o&list=PLTgRMOcmRb3OGBIfqPSZFk0Nn0B4xGZqs&index=2

Gradle guide is at
http://www.tutorialspoint.com/gradle/gradle_quick_guide.htm

To set OS path for scala

```
export PATH=/usr/local/share/scala/bin:$PATH
```

To set environment path for gradle in OS

```
export PATH=/opt/gradle/bin:$PATH
```

Gradle builds a script file for handling two things; one is **projects** and another one is **tasks**. Every Gradle build represents one or more projects. A project represents a library JAR or a web application or it might represent a ZIP that assembled from the JARs produced by other projects. In simple words, a project is made up of different tasks. A task means a piece of work which a build performs. A task might be compiling some classes, creating a JAR, generating Javadoc, or publishing some archives to a repository.Gradle uses **Groovy language** for writing scripts.

The Gradle script mainly uses two real Objects; one is Project Object and another one is Script Object.

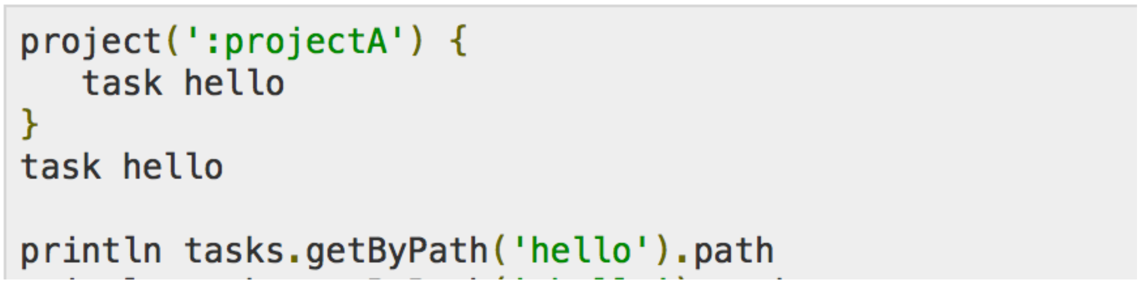
Project Object – Each script describes about one or multiple projects. While in the execution, this script configures the Project Object. You can call some methods and use property in your build script which are delegated to the Project Object.

Script Object – Gradle takes script code into classes, which implements Script Interface and then executes. This means that of all the properties and methods declared by the script interface are available in your script.

You can make a task dependent on another task, which means when one task is done only then the other task will start. Each task is differentiated with a task name. Collection of task names is referred by its tasks collection. To refer to a task in another project, you should use path of the project as a prefix to the respective task name

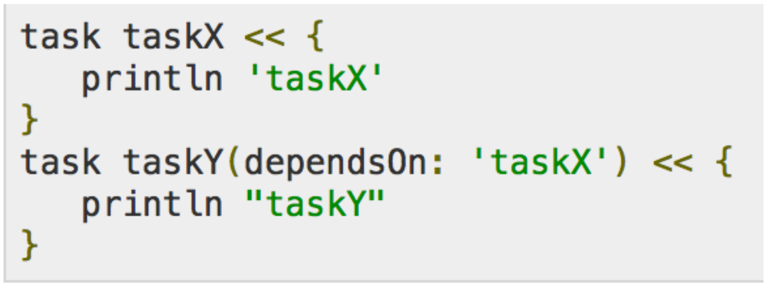
Gradle - Tasks

1. Defining Tasks . Create build.gradle file under bin folder of gradle.
2. Locating Tasks
3. Adding dependencies to Tasks



```
project(':projectA') {
   task hello
}
task hello

println tasks.getByPath('hello').path
```



```
task taskX << {
   println 'taskX'
}
task taskY(dependsOn: 'taskX') << {
   println "taskY"
}
```

There is another way to add dependency to the task, that is, by using closures as shown below

```
task taskX << {
   println 'taskX'
}
taskX.dependsOn {
   tasks.findAll {
      task -> task.name.startsWith('lib')
   }
}
task lib1 << {
   println 'lib1'
}
task lib2 << {
   println 'lib2'
}
task notALib << {
   println 'notALib'
}
```

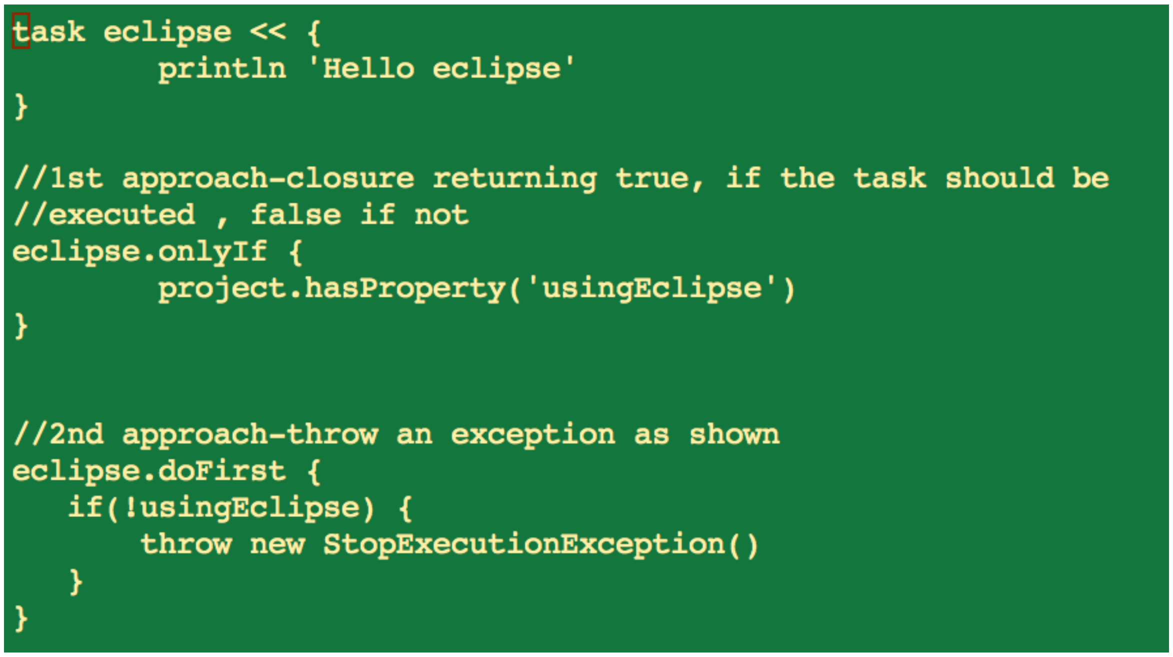
4. Adding a description to task

You can add a description to your task. This description is displayed when executing **Gradle tasks**. This is possible by using the description keyword.

```
task copy(type: Copy) {
   description 'Copies the resource directory to the target directory.'
   from 'resources'
   into 'target'
   include('**/*.txt', '**/*.xml', '**/*.properties')
   println("description applied")
}
```

5. Skipping Tasks -

Skipping tasks can be done by passing a predicate closure. This is possible only if method of a task or a closure throwing a **StopExecutionException** before the actual work of a task is executed.



```
task eclipse << {
        println 'Hello eclipse'
}

//1st approach-closure returning true, if the task should be
//executed , false if not
eclipse.onlyIf {
        project.hasProperty('usingEclipse')
}


//2nd approach-throw an exception as shown
eclipse.doFirst {
   if(!usingEclipse) {
       throw new StopExecutionException()
   }
}
```

Gradle has different phases, when working with tasks. First of all, there is a configuration phase, where the code, which is specified directly in a task's closure, is executed. The configuration block is executed for every available task and not only for those tasks which are later actually executed.

After the configuration phase, the execution phase runs the code inside the **doFirst** or **doLast** closures of those tasks, which are actually executed.

Gradle - Dependency Management

Gradle build script defines a process to build projects; each project contains some dependencies and some publications. Dependencies means the things that support to build your project such as required JAR file from other projects and external JARs like JDBC JAR or Eh-cache JAR in the class path. Publications means the outcomes of the project, such as test class files and build files, like war files.

For example, in order to use Hibernate in the project, you need to include some Hibernate JARs in the classpath. Gradle uses some special script to define the dependencies, which needs to be downloaded.

1. Declaring your dependencies

Gradle follows some special syntax to define dependencies. The following script defines two dependencies, one is Hibernate core 3.6.7 and second one is Junit with the version 4.0 and later.

```
apply plugin: 'java'

repositories {
   mavenCentral()
}

dependencies {
   compile group: 'org.hibernate', name: 'hibernate-core',
version: '3.6.7.Final'
   testCompile group: 'junit', name: 'junit', version: '4.+'
}
```

2. Dependency Configuration

Dependency configuration is nothing but defines a set of dependencies. You can use this feature to declare external dependencies, which you want to download from the web

- **Compile –** The dependencies required to compile the production source of the project.
- **Runtime –** The dependencies required by the production classes at runtime. By default, also includes the compile time dependencies.
- **Test Compile –** The dependencies required to compile the test source of the project. By default, it includes compiled production classes and the compile time dependencies.
- **Test Runtime –** The dependencies required to run the tests. By default, it includes runtime and test compile dependencies.

3. External Dependencies

External dependencies is a type of dependency. This is a dependency on some files that is built outside the current build, and is stored in a repository of some kind, such as Maven central, corporate Maven or lvy repository, or a directory in the local file system.

The following code snippet is to define the external dependency. Use this code in **build.gradle** file

```
dependencies {
   compile group: 'org.hibernate', name: 'hibernate-core',
version: '3.6.7.Final'
}
```

An external dependency is declaring external dependencies and the shortcut form looks like "group: name: version".

4. Repositories

While adding external dependencies. Gradle looks for them in a repository. A repository is just a collection of files, organized by group, name and version. By default, Gradle does not define any repositories. We have to define at least one repository explicitly. The following code snippet defines how to define maven repository. Use this code in **build.gradle** file.

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

```
repositories {
   maven {
      url "http://repo.mycompany.com/maven2"
   }
}
```

5. Publishing Artifacts

Dependency configurations are also used to publish files. These published files are called artifacts. Usually, we use plug-ins to define artifacts. However, you do need to tell Gradle where to publish the artifacts. You can achieve this by attaching repositories to the upload archives task. Take a look at the following syntax for publishing Maven repository. While executing, Gradle will generate and upload a Pom.xml as per the project requirements. Use this code in **build.gradle** file.

```
apply plugin: 'maven'

uploadArchives {
   repositories {
      mavenDeployer {
         repository(url: "file://localhost/tmp/myRepo/")
      }
   }
}
```

————— JAVA - what is SUPER . THIS and POLYMERPHISM … Great video here = https://www.youtube.com/watch?v=nLr9Q7n3c_Y&index=15&list=PLtNErhYMkHnHq4jqUTr17FvGMOveQ3cc_