Table of Contents

[GRADLE 2](#_Toc79659524)

[BASIC GRADLE PROJECT 2](#_Toc79659525)

[GRADLE COMMANDS 2](#_Toc79659526)

[GRADLE PROJECT - FOLDER STRCUTURE 2](#_Toc79659527)

[CREATING JAVA PROJECT 3](#_Toc79659528)

[BUILD.GRADLE – DEPENDENCIES AND RESPOSITORIES 4](#_Toc79659529)

[DEPENDENCIES AND CONFIGURATIONS 4](#_Toc79659530)

[REPOSITORIES 4](#_Toc79659531)

[BUILD PHASES IN GRADLE 4](#_Toc79659532)

[EXAMPLE 5](#_Toc79659533)

[APACHE GROOVY BASICS 5](#_Toc79659534)

[GROOVY SHELL 5](#_Toc79659535)

[GROOVY COMPILATION 5](#_Toc79659536)

[GROOVY CONSOLE 6](#_Toc79659537)

[JAVA VERSUS GROOVY 6](#_Toc79659538)

[SCRIPTS – BEHIND THE SCENE 7](#_Toc79659539)

[CLASSES 8](#_Toc79659540)

[OPERATORS 8](#_Toc79659541)

[DATATYPES 8](#_Toc79659542)

[COLLECTIONS 9](#_Toc79659543)

[CLOSURES 9](#_Toc79659544)

[CONTROL STRUCTURES 11](#_Toc79659545)

[OOPS IN GROOVY 11](#_Toc79659546)

[TRAITS 11](#_Toc79659547)

[GROVVY BEANS 11](#_Toc79659548)

# GRADLE

* Gradle is a build automation tool. The task like compiling, running the test cases , package the artifact as jar/war and deploying it to server or cloud environment can be achieved by gradle.
* Gradle uses programming languages like ***Groovy or Kotlin*** (known as DLS – Domain Specific Languages) in it configuration file (build.gradle)
* **INCREMENTAL BUILD**: Gradle support incremental build. For example - It only compiles the changed the files and used the compiled code of previous build.
* **BUILD CACHE**:
* **PLUGIN SUPPORT :** Gradle never have configuration specific to a particular project type. It us the plugin which brings in specific features like compiling , excuting and running the test case for a particular project type.

**Note : Gradle needs JDK 1.8 or higher**

## BASIC GRADLE PROJECT

|  |  |
| --- | --- |
| Step 1 : INITIALIZE A GRADLE PROJECT | **gradle init** |
| Step 2: Select Project Type  Select “1” for basic project |  |
| Step 3: Select the DSL for the build script  Select “1” for groovy . means we are going to write build,gradle in groovy  This will create a basic gradle project |  |

### GRADLE COMMANDS

|  |  |
| --- | --- |
| **gradle tasks** | * This will list all the tasks in build.gradle file. * If no custom task are added in build.gradle file this command will show the Build Setup and Help tasks. |
| **gradle tasks --all** | * This will list all tasks Build Setup, Help and ***other tasks*** as well. |
| **CREATING A TASK** | task firstTask{      println "Hello From Groovy Task!"  } |
| **RUNNING THE TASK** | * gradle <taskName> * gradle firstTask * gradle fT 🡨 if the task names are written in camel case, it can be abbreviated |

### GRADLE PROJECT - FOLDER STRCUTURE

|  |  |
| --- | --- |
|  | 1. settings.gradle 🡪 This contains the name of the gradle project. We add entry to this file -just in case our project in multi-module project 2. graddlew 🡪 Called gradle wrapper   **GRADLE WRAPPER**   * Gradle Project can be built with its own gradle version. So, when the projects are deployed to any environment, Gradle wrapper downloads the exact version which the project is using – instead of using the version pre-installed gradle version. * ***Hence - jenkin pipeline uses gradlew instead of gradle command to run and build the gradle project.*** |

## CREATING JAVA PROJECT

|  |  |
| --- | --- |
| * It will be a application project * Gradle will create the entire project structure of a java project. * **gradle tasks --all** 🡪 Will list the tasks in build.gradle file. * The task list is are divided into different categories   + Application tasks   + Build tasks   + Build Setup tasks   + Distribution tasks   + Verification tasks   + Other tasks * **APPLICATION TASKS** – This includes the tasks to run the project * **BUILD TASKS** - This includes the tasks to build the project * **VERIFICATION TASKS –** This includes the tasks to run the unit and integration test. * **OTHER TASKS –** Includes the task for very specific items like   + Compiling the Java class of src/main folder only   + Compiling the Java class of test folder only |  |

#### IMPORTANT JAVA PROJECT TASKS

|  |  |
| --- | --- |
|  |  |
|  |  |

#### COMPILING AND EXECUTING

|  |  |
| --- | --- |
| **COMPILING THE JAVA FILE main/src folder** | gradle app:compileJava |
| **COMPILING THE JAVA FILE test folder** | gradle app:compileTestJava |
| **COMPILING BOTH main/src and test folder** | gradle app:build |
| **EXECUTING THE PROJECT**  Gradle know about the main class from build.gradle  (application closure) | gradle app:run |
| **EXECUTING THE TEST** | gradle app:test |
| **CLEANING/DELETE THE BUILD FOLDER** | gradle app:clean |
| **CREATING JAR:** This will create the jar file in /build/lib folder | gradle app:jar |

## BUILD.GRADLE – DEPENDENCIES AND RESPOSITORIES

|  |  |
| --- | --- |
|  | DEPENDENCIES |

### DEPENDENCIES AND CONFIGURATIONS

* We define the dependencies in this section /closure
* Dependencies can be added in several scopes like test or compile time dependencies.

|  |  |
| --- | --- |
| testImplementation | Dependency needed for running Test |
| implementation | Dependency needed for entire app |
|  |  |

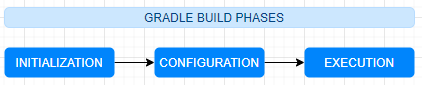
Like scopes – we have configuration in gradle. Gradle configuration are more granular

|  |  |  |
| --- | --- | --- |
| **MAVEN SCOPES** | **GRADLE CONFIGURATION** | **DESCRIPTION** |
| compile | Implementation ,  testImplementation |  |
| provided | compileOnly ,  runtimeOnly |  |
| runtime | testCompileOnly,  testRuntimeOnly |  |
| test | Api(compile) |  |

### REPOSITORIES

|  |  |
| --- | --- |
|  | * All the dependencies are downloaded from a central repository. The repos can be Maven, Ivy or JCenter * It can be a file repository (Folder in local machine). In local machine Gradle stores/caches the dependency or transitive dependency jars in “.gradle/caches” folder in users directory. |
| repositories {  mavenCentral()  jcenter()  mavenLocal()  } | * We can add multiple repositories . They work as a fallback of each other. For ex.. if the dependency is not found in maven central - gradle will look for it in Jcenter , |

## BUILD PHASES IN GRADLE



|  |  |
| --- | --- |
| **BUILD PHASES** | **DESCRIPTION** |
| **INITIALIZATION** | Gradle creates a dedicated “project” object for each module declared in the settings.gradle |
| **CONFIGURATION** | Gradle reads all the tasks needed to run a build and creates ***DAG (Direct Acyclic Graph)*** – To make sure the tasks are not cyclic and avoid gradle to go in infinite loop. |
| **EXECUTION** | Gradle run all the tasks in execution phase. |

### EXAMPLE

|  |  |
| --- | --- |
| println "Start - Configuration Phase"  task firstTask{      println "Hello From Groovy Task - Configuration Phase"      doFirst(){          println "doFirst - Execution Phase"      }      doLast(){          println "doLast - Execution Phase"      }      println "Bye from Groovy Task - Configuration Phase"  }  println "End" | All the statement in doFirst() and doLast() will execute in execution phase – rest get executed on configuration phase |

## UNDERSTANDING BUILD.GRADLE

* Build.gradle is object oriented.

### THE PROJECT OBJECT

|  |  |  |
| --- | --- | --- |
| **SAMPLE BUILD.GRADLE** |  | |
| plugins {  }  repositories {  }  dependencies {  }  application {  } | **ANALOGY**  def calculate(closure){  closure();  }  calculate{  println "hello"  } | * For every project groovy creates a “project” object. * **Repositories, dependencies and application** are the methods defined in the project Class * **These methods take Closures as an input parameter.** * In build.gradle we call those methods on project object (implicit) and pass the closures .See the analogy for understanding. * Note:”brackets” are optional while calling a function in groovy * **Unlike all – plugins is a block of code not a method** |

#### SETTING AND ACCESSING THE VALUES ON PROJECT OBJECT

##### BUILT-IN PROPERTIES

* <https://docs.gradle.org/current/userguide/writing_build_scripts.html>
* <https://docs.gradle.org/current/javadoc/org/gradle/api/Project.html>

We can set and get the properties on project object in ***build.gradle***. Example

|  |  |
| --- | --- |
| plugins {  id 'application'  }  **project.description = "This is app project"**  **project.version = 1.0**  repositories {  // Use Maven Central for resolving dependencies.  println "project name="+ project.name  println "project Description="+ project.description  println "project Version="+ project.version  println "project Version="+ project.getProperty(“version”)  mavenCentral()  }  dependencies {  }  application {  } | * The properties can be set right below the plugin statement. * The project name are read only properties – which can be configured from settings.gradle * The project version helps in versioning the jars in build folder. For example – if project.version=1.0 ; then the Jars will be <***application\_name***>***-1.0.jar*** |

| STANDARD PROJECT PROPERTIES | | |
| --- | --- | --- |
| Name | Type | Default Value |
| project | [Project](https://docs.gradle.org/current/dsl/org.gradle.api.Project.html) | The Project instance |
| name | String | The name of the project directory. |
| path | String | The absolute path of the project. |
| description | String | A description for the project. |
| projectDir | File | The directory containing the build script. |
| buildDir | File | *projectDir*/build |
| group | Object | unspecified |
| version | Object | unspecified |
| ant | [AntBuilder](https://docs.gradle.org/current/javadoc/org/gradle/api/AntBuilder.html) | An AntBuilder instance |

##### CUSTOM PROPERTIES

|  |  |
| --- | --- |
| plugins {  id 'application'  }  project.description = "This is app project"  project.version = 1.0  **project.ext.mycustomProperty="Custom Values"**  repositories {  println "project Custom properties ="+ project.mycustomProperty  mavenCentral()  }  dependencies {  }  application {  } | * Custom properties can be set on “ext” object . Although it does not need “ext” to retrieve it back |

## APACHE GROOVY BASICS

* Groovy can be downloaded as binary or as an installer.
* For windows- Update the “PATH” environment variable to point the “bin” directory of the

### GROOVY SHELL

|  |  |
| --- | --- |
|  | ***groovysh* – to Start Groovy shell**   * <http://groovy-lang.org/groovysh.html> * The Groovy Shell, aka. ***groovysh*** is a command-line application which allows easy access to evaluate Groovy expressions, define classes, and run simple experiments. * For multiline code – groovy will not start the compilation until it not completed. Hence it allow as write the statement after pressing ***Enter Key*** till the statement is completed (highlighted) * This does not create any executable file. |
|  |
| FUNCTIONS IN GROOVY SHELL |

### GROOVY COMPILATION

* The groovy scripts can be compiled using a command line tool ***“groovyc”.* Note – All the groovy scripts are compiled into .class files.**
* <http://groovy-lang.org/groovyc.html>

|  |  |
| --- | --- |
| **groovyc <fileName>.groovy** | Compiling the Groovy Script |
| **groovy <fileName>** | Running the Groovy Script |
| **groovyc \*.groovy** | Compile all groovy script in a folder |
| **groovyc -d classes \*.groovy** | Compile all groovy script and copy all the compiled file in classes folder. It creates the folder too. |

### GROOVY CONSOLE

|  |  |
| --- | --- |
| **COMMANDS**   * OPENS A BLANK FILE IN GROOVY CONSOLE - **groovyconsole** * **OPENS THE GIVEN FILE IN THE GROOVY CONSOLE**   **groovyconsole <filename>.groovy** |  |

### JAVA VERSUS GROOVY

* **METHODS AND CLASSES: All the classes and methods public implicitly, so no need mark them as public explicitly.**
* **PROPERTIES**: All the properties are by default private, so need mark the properties as private explicitly. Although if we want to give protected or public access modifier, we have to do it explicitly.
* **METHOD RETURN STATEMENTS**: No need to have return statement in methods. Groovy consider the last line of the function as return value.
* **SEMICOLON ARE OPTIONAL** – Single statement per line has optional semicolons.
* **GETTER AND SETTERS:**
  + We can define the getters/ setters explicitly also for property if we want to . but ,
  + For private property – groovy compiler generates the getter & setters on it (In the compiled code ). So we need not have to write the getter and setters for a property explicitly– to set and get the values.
* **CONSTRUCTORS:** Groovy beans /POJOs can be initialized by using named parameter. So Groovy beans does not need to have constructors. Although we can declare constructor as we do in java
* **println/print has been added as default method.** No unlike java no need to use “Sysouts”
* **toString()** implementation can be archived by something metaprogramming **[@groovy.transform.ToString()**]
* **PACKAGE IMPORTS**

|  |  |  |
| --- | --- | --- |
| LIST OF IMPLICT PACKAGE IMPORTS  import java.lang.\*  import java.util.\*  import java.io.\*  import java.net.\*  import groovy.lang.\*  import groovy.util.\*  import java.math.BigInteger  import java.math.BigDecimal | * For any groovy file-Groovy does add some implicit package imports in a groovy – so we need have to add them explicitly * This is done because the classes from these packages are most used. By importing this boilerplate code is reduced. * Anything outside these package imports must be explicitly imported. | |
| **JAVA CLASS** | | **EQUIVALENT GROOVY CLASS** |
| **import** java.util.Date;  **public** **class** User {  **private** Long Id;  **private** String firstName;  **private** String lastName;  **private** Date dob;  **public** User() {  **super**();  }  **public** User(String firstName, String lastName) {  **super**();  **this**.firstName = firstName;  **this**.lastName = lastName;  }  **public** User(String firstName, String lastName, Date dob) {  **super**();  **this**.firstName = firstName;  **this**.lastName = lastName;  **this**.dob = dob;  }  **public** Long getId() {  **return** Id;  }  **public** **void** setId(Long id) {  Id = id;  }  **public** String getFirstName() {  **return** firstName;  }  **public** **void** setFirstName(String firstName) {  **this**.firstName = firstName;  }  **public** String getLastName() {  **return** lastName;  }  **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;  }  **public** Date getDob() {  **return** dob;  }  **public** **void** setDob(Date dob) {  **this**.dob = dob;  }  **public** **void** printFullName() {  System.***out***.println("First Name:"+ firstName +"Last Name"+ lastName);  }  @Override  **public** String toString() {  **return** "User [Id=" + Id + ", firstName=" + firstName + ", lastName=" + lastName + ", dob=" + dob + "]";  }  } | | @groovy.transform.ToString()  class User {  Long Id  String firstName  String lastName  Date dob    void printFullName() {  println "First Name: $firstName Last Name: $lastName";  }  }  **INSTATIATING THE CLASS AND CALLING THE METHOD**  //Named Parameters  **User user = new User(Id:"1",firstName:"John",lastName:"Hopkin",dob:new Date(1900,10,19))**  **// toString on an object**  println user  **//Calling a method**  user.printFullName() |

### SCRIPTS – BEHIND THE SCENE

|  |  |
| --- | --- |
|  | A simple groovy script gets compiled into a Java class   * The class has main method * All the groovy script wrapped into “run()” method. |

### CLASSES

* Unlike Java, groovy file name need not to be same as class name.

|  |  |  |
| --- | --- | --- |
| **Source File(AngryBirds.groovy)**  class AngryBirds{ }  class Bird { }  class Pig{ } | If one groovy file having multiple classes. Compiles to individual .class files. |  |

#### COMMUNICATION BETWEEN CLASSES

|  |  |
| --- | --- |
| **DEVELOPER.GROOVY** | **APP.GROOVY** |
| @groovy.transform.ToString()  class Developer{  String firstName  String lastName  def languages=[]    void work(){  println "Developer Name is $firstName $lastName"  }  } | //Create an instance of developer  Developer d = new Developer();  //Initialize the instance variable  d.firstName="John"  d.lastName ="Hopkin"  //Adding a language to list – calls the toString  d.languages << "Groovy"  d.languages << "Java"  // Print the object  println d  // calling the method  d.work() |

#### GOTCHAS IN CLASSES

|  |  |
| --- | --- |
| **User.groovy**  class User {  String userName  String id  void getUserDetails(){  println "The user id= $id and name $userName"  }  }  User user = new User(id:"U001",userName:"John")  user.getUserDetails() | While running this class we get an exception –  “Invalid duplicate class definition of class User : The source User.groovy contains at least two definitions of the class User.”-  WHY? |

### OPERATORS

* <https://groovy-lang.org/operators.html>

|  |  |
| --- | --- |
| **ELVIS OPERATOR :**   * Its a kind of shorthand of Ternary operator. * If “user.name” is truthy – assign it to displayName | displayName = user.name ?:”Anonymous” |
| **SAFE NAVIGATION OPERATOR (?)** |  |

### DATATYPES

|  |  |
| --- | --- |
| Unlike Java primitive type. Every datatype in groovy is an object (Wrapper Objects) |  |

#### OPTIONAL TYPING

|  |  |
| --- | --- |
| def x= 10  x=”John” | * Using “def” we need not have to define the datatype of a variable * The type is inferred using the type of data it is storing |

### COLLECTIONS

#### LIST

|  |  |  |
| --- | --- | --- |
| l = [1,2,3]  println l  println l.class  ll = [4,5] as LinkedList  print ll  println ll.class  /\* Declaring a linked list \*/  LinkedList li =[1,2,4]  println ll.class | **OUTPUT**  [1, 2, 3]  class java.util.ArrayList  [4, 5]class java.util.LinkedList | * The list in groovy is by default an Array List * The default implementation can be overridden using “**as**” keyword. * All the Java function of List works well in groovy |
| **ADDING ITEM TO LIST** | l.add(4);  l << 5  l = l+ [8,9,10] | * Groovy has overloaded operator (**<<**) to add items in a list |
| **REMOVING ITEM FROM LIST** | l = l –[2,3] | This will remove elements from list |
| **ITERATING THE LIST** | l.each{ println it}  l.each{ element -> println element} | * The iteration can be done using each . * It takes a closure as a parameter . * To print the element, we use “it” operator .;y6;; |

#### MAP AND SETS

|  |  |  |
| --- | --- | --- |
| SET | | |
| s = ["Java", "Python","C"] as Set  println s  println s.class | **OUTPUT**  class java.util.LinkedHashSet | * No duplicates * By default sets are LinkedHashSet |
| MAP | | |
| **EMPTY MAP** | m=[:] | |
| **ADDING VALUE TO MAP** | m=[course:"Gradle",rating:"5"] | |
| **ITERATING A MAP** | m.each{k,v ->  println "key=$k : value=$v"  } | |
| **RETRIVEING FROM MAP** | m.course  m["course"]  m.get(“course”) | |
| **ADDING VALUE TO An EXISTING MAP** | m.review="Good"  m['duration']="4 Hours" | |

#### METHODS

|  |  |
| --- | --- |
| int addNumbers(x,y= 10){  x+y;  }  println addNumbers(1,2) | * No need to specify the datatype in function parameter- but it recommended. * Return statement are not mandatory. The last statement is considered as return statement. * Function parameter can take default values. |

### CLOSURES

* Closures are object. They can be passed to function parameters.
* Closure are in java.lang.Closure package

|  |  |
| --- | --- |
| **BASIC CLOSURE:**  **“c” is an instance Closure Class.** | **def c ={}**  c instanceof Closure 🡨 True |
| **EXAMPLE 2: Here closures behave like anonymous function** | def sayHello ={  println "Hello World!"  }  sayHello() 🡨 Calling the closure |
| **EXAMPLE 3: Passing Parameter to Closure** | def sayHello ={ name ->  println "Hello ! $name"  }  sayHello("John") 🡨 O/p = *Hello ! John* |
| **Closure has implicit parameter called “it”. It Is null when not passed.** | FLAVORS OF “it”  def greet ={  println it  }  greet() 🡨 null  greet("Hi") 🡨 It will print ***Hi*** |
| **ITERATING LIST** | // using default parameter  def empList = ["Alex","john","Rob"]  empList.each({  println "Student Name=$it"  })  // using parameter  empList.each({ name ->  println "Student Name=$name"  }) |
| **PASSING CLOSURES**  **EXAMPLE 1:**  def multiplyNums(num, closure){  closure(num)  }  multiplyNums(5, { num -> println num\*2});  multiplyNums(5,{ num -> println num\*3}); | EXAMPLE 2:  def tenTimes(num, clo){  clo(num\*10)  }  tenTimes(2,{ print it })  tenTimes(4,{ print it }) |

#### PARAMETERS IN CLOSURES

|  |  |  |
| --- | --- | --- |
| **NO PARAM CLOSURE:** This type of closure does not take parameters | def noParamClosure = { ->  println "No Param closure"  }  noParamClosure() // OUTPUT = “No Param closure”  noParamClosure(“Some Param’) // OUTPUT = Exception | |
| **PARAMETERS IN CLOSURE –** if the closure are the last parameter of a function parameter or function has just one argument – The braces can be removed. | **EXAMPLE 1**  tenTimes(2),{  print it  } | **EXAMPLE 1**  def empList = ["Alex","john","Rob"]  empList.each { name ->  println "Student Name=$name"  } |
| **DEFAULT VALUE OF CLOSURE PARAMETER-**  If the values are passed it will take the passed parameter otherwise it takes the default value | def greeting ={ name, message ="Hello" ->  println "$message !! $name "  }  greeting("John") 🡨 Hello !! John  greeting("Rob","Bye") 🡨 Bye !! Rob | |
| **VAR ARGS** | def varArgsClosure ={ ...args ->  println args.join(",")  }  varArgsClosure("Alex", "Rob")  varArgsClosure("Max", "Mike","Mac") | |
| **META DATA OF CLOSURES**  **OUTPUT :**  **2**  **[int, int]** | def closureMetadata = { c ->  println c.getMaximumNumberOfParameters()  println c.parameterTypes  }  def add = {int x, int y -> x+y } 🡨 Output is meta data of this closure  closureMetadata(add) | |

#### CLOSURES IN COLLECTIONS

|  |  |
| --- | --- |
| **ITERATION**  List students =["Alex","Rob","Ram"] | students.each{ student ->  println student  }  students.eachWithIndex{ student , index ->  println "$index : $student"  } |
| **USING NAMED PARAMS**  println students.findAll {  student -> student.startsWith("R")}  **USING IMPLICIT PARAM**  println students.findAll { it.startsWith("R")} |
| **COLLECT:** It creates a new list based on some condition or business logic | List nums =[10,20,30,40]  println nums.collect{ it\*2 }  **OUTPUT : [20, 40, 60, 80]** |

## OOPS IN GROOVY

### TRAITS

### GROVVY BEANS

A traditional Java beans has following properties

* All properties must be private
* A public no argument constructor
* Implement Serializable

|  |  |
| --- | --- |
| **JAVA BEANS** | **EQUIVALENT GROOVY BEAN** |
| package com.domain; import java.io.Serializable;  public class EmployeeBean implements Serializable {  String empId;  String company;   public EmployeeBean() {  }   public String getEmpId() {  return empId;  }   public void setEmpId(String empId) {  this.empId = empId;  }   public String getCompany() {  return company;  }   public void setCompany(String company) {  this.company = company;  }   @Override  public String toString() {  return "EmployeeBean{" +  "empId='" + empId + '\'' +  ", company='" + company + '\'' +  '}';  } } | @groovy.transform.ToString()  class EmployeeBean implements Serializable {  String empId;  String company;  } |
| **USING GROOVY BEANS**  **Employee emp = new Employee(empid: 'E001', company: "ABC")**  **println emp**  **CALLING GETTER AND SETTERS**  emp. empid=”E005” 🡨 **Calling Setter**  println emp. empid 🡨 **Calling getter** |