Table of Contents

[GRADLE 2](#_Toc79003813)

[GRADLE WRAPPERS 2](#_Toc79003814)

[APACHE GROOVY BASICS 2](#_Toc79003815)

[GROOVY SHELL 2](#_Toc79003816)

[GROOVY COMPILATION 2](#_Toc79003817)

[GROOVY CONSOLE 2](#_Toc79003818)

[JAVA VERSUS GROOVY 3](#_Toc79003819)

[SCRIPTS – BEHIND THE SCENE 4](#_Toc79003820)

[CLASSES 4](#_Toc79003821)

[ANNOTATIONS AND AST TRANSFORMATION 5](#_Toc79003822)

[OPERATORS 5](#_Toc79003823)

[DATATYPES 5](#_Toc79003824)

[OPERATOR OVERLOADING 6](#_Toc79003825)

[COLLECTIONS 6](#_Toc79003826)

[CLOSURES 6](#_Toc79003827)

[CONTROL STRUCTURES 6](#_Toc79003828)

[OOPS IN GROOVY 6](#_Toc79003829)

[TRAITS 6](#_Toc79003830)

[GROVVY BEANS 6](#_Toc79003831)

# 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 SUP**

## GRADLE WRAPPERS

## 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:**
  + For private property – groovy compiler generates 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? |

### ANNOTATIONS AND AST TRANSFORMATION

### 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 |

### OPERATOR OVERLOADING

### COLLECTIONS

#### RANGE

* **A range is shorthand for specifying a sequence of values**.
* A Range is denoted by the first and last values in the sequence, and Range can be inclusive or exclusive.
* An inclusive Range includes all the values from the first to the last, while an exclusive Range includes all values except the last. Here are some examples of Range literals.
  + 1..10 - An example of an inclusive Range
  + 1..<10 - An example of an exclusive Range
  + ‘a’..’x’ – Ranges can also consist of characters
  + 10..1 – Ranges can also be in descending order
  + ‘x’..’a’ – Ranges can also consist of characters and be in descending order.

#### LIST

#### MAP

### CLOSURES

### CONTROL STRUCTURES

## 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** |