Taller Gradle

Crear una tarea

1. Crear una carpeta con el nombre que desee, en nuestro caso se llamará *gradleTutorial*.
2. Dentro de la carpeta creada abrir la consola y ejecutar el comando *gradle init*.
3. Crear una carpeta llamada *src* en la carpeta donde se realizó el *gradle init*.
4. Dentro de la carpeta *src* crear un archivo llamado *miPrimerTareaConGradle.txt*
5. Editar el archivo de texto creado en el paso anterior con un *"Hola mundo!"* y guardar los cambios.
6. Regresar a la carpeta *gradleTutorial* y abrir el archivo *build.gradle*. En la medida de lo posible utilizar editores de código como Notepad por ejemplo y evitar el uso de word, wordPad, etc.
7. Es posible que el archivo build.gradle se haya creado con algunos comentarios, por favor borrarlos.
8. Dentro del build.gradle crear una tarea copy, la cual realizará una copia del archivo *miPrimerTareaConGradle.txt* de la carpeta *src* a una nueva carpeta llamada *dest*. La tarea es la siguiente:

```
task copy(type: Copy, group: "Custom", description: "Copies
sources to the dest directory") {
    from "src"
    into "dest"
}
```

   Aquí, los parámetros *group* y *description* pueden tener cualquier valor. Incluso se pueden omitir, pero al hacerlo también lo harán en el informe, que se usará más adelante.

   Cabe notar que la tarea *copy* hace parte de la librería de tareas que gradle nos proporciona, además la implementación que se acaba de realizar es un ejercicio sencillo de la función de Copy, si desea más información acerca de ella por favor consultar su documentación (Documentación sobre la tarea Copy).

9. Guardar los cambios realizados en build.gradle y regresar a la consola y ejecutar el comando *gradle copy*.
   Recuerde que al ejecutar el comando debe estar ubicado en la carpeta *gradleTutorial*, puesto que es en esta donde esta nuestro proyecto gradle.

10. Verifique que el proceso fue exitoso, revisando que el archivo *miPrimerTareaConGradle.txt* se copió exitosamente dentro de la carpeta *dest* y que el contenido del archivo coincide con el contenido del que se encuentra en *src*.

Aplicar un plugin

Gradle incluye una variedad de plugins, y muchos, muchos más están disponibles en el portal de plugins de Gradle. Uno de los plugins incluidos en la distribución es *base*. Combinado con un tipo de núcleo llamado Zip, puede crear un archivo zip con un nombre y ubicación configurados.

Se puede seguir en el mismo proyecto de la sección de Crear una tarea

1. Abrir el archivo build.gradle
2. Agregar al inicio del archivo las siguientes líneas:

```
plugins {
    id "base"
}
```

3. Agregar una nueva tarea para crear un zip a partir de la carpeta *src*. Para realizar dicha tarea agrega el siguiente código al archivo build.gradle.

```
task zip(type: Zip, group: "Archive", description:
"Archives sources in a zip file") {
    from "src"
    setArchiveName "basic-demo-1.0.zip"
}
```

   Dada la configuración de la tarea, al ejecutarla se creará un archivo zip llamado *basic-demo-1.0.zip* en la carpeta *build/distributions.*

4. Guardar los cambios y ejecutar la nueva tarea llamada zip
5. Revisar que la tarea se haya ejecutado correctamente, asegurándose de que se creó el archivo zip en la carpeta mencionada anteriormente.

Explorar y depurar la compilación

Para explorar un proyecto gradle será muy importante conocer el comando *tasks*, ya que este nos lista todas las tareas que desde el proyecto se pueden ejecutar. Para comprobar el comando *tasks*, ejecutemos desde la consola del proyecto el comando *gradlew tasks*. Observará en consola todas las tareas que desde el proyecto creado se pueden ejecutar, incluyendo las creadas anteriormente *copy* y *zip*, además podrá observar la utilidad de las opciones anteriormente tratadas, *group* y *description*.

Para realizar un análisis de compilación del proyecto se puede hacer uso de la opción *--scan*, la cual generará un reporte web en *scans.gradle.com*. Para comprobar el uso de la opción *--scan*, se ejecutará nuevamente la tarea zip, pero agregando la opción *--scan*. El comando que se ejecutaría sería el siguiente *gradlew zip --scan*.
Al navegar por el análisis de compilación, se observa qué tareas se ejecutaron y cuánto tiempo tardaron, qué complementos se aplicaron y mucho más. Una buena práctica es compartir una exploración de compilación en StackOverflow, para que los que nos puedan ayudar tengan un informe más detallado de la tarea que se compilo.

Gradle también permite conocer las propiedades de un proyecto, para esto se hace uso del comando *properties.* Para mirar el funcionamiento ejecutar el comando *gradlew properties*.

La parte name del proyecto coincide con el nombre de la carpeta de forma predeterminada. También puede especificar las propiedades *group* y *version*, pero en el momento están tomando sus valores predeterminados, tal como está *description*.

La propiedad *buildFile* es el nombre de ruta para su script de compilación, que reside en el *projectDir* de forma predeterminada.

Se pueden cambiar muchas de las propiedades. Por ejemplo, se puede intentar agregar las siguientes lineas al *build.gradle* y volver a ejecutar *gradlew properties*.

```
description = "Un tutorial de Gradle build"
version = "1.0"
```

Aquí encontrarás una documentación más detallada de la interfaz de línea de comandos. Es importante conocer a detalle todas las posibles combinaciones que podemos realizar.

Ejercicio 1: Entender un build.gradle habitual

La idea de este ejercicio es explicar cuál es la función de cada uno de los elementos que contiene un build.gradle que se utiliza normalmente. Se espera que se haga uso solamente de la documentación oficial de gradle para hacer uso de los conocimientos adquiridos en el taller.

```
repositories {
    maven {
        url "http://artifactory.bancolombia.corp:80/maven-bancolombia/"
    }
}

buildscript {
    repositories {
        maven{
            url "http://artifactory.bancolombia.corp:80/maven-bancolombia/"
        }
    }
    dependencies {
        classpath("net.serenity-bdd:serenity-gradle-plugin:1.9.14")
    }
}

apply plugin: 'java'

sourceCompatibility = 1.8

dependencies {
    Implementation 'net.serenity-bdd:serenity-core:1.9.14'

    testImplementation 'junit:junit:4.12'
}

gradle.startParameter.continueOnFailure = true
```

Ejercicio 2: Entender algunos de los comandos utilizados comúnmente

Porque se puede ejecutar la línea *gradle test* sino se tiene ninguna tarea que se llame de esa manera ?

Porque se puede ejecutar la línea *gradle clean* sino se tiene ninguna tarea que se llame de esa manera ?

Basado en el build.gradle del ejercicio 1, que le hace falta a dicha configuración para poder ejecutar el comando gradle test aggregate ?

Solución ejercicio 1:

El bloque repositories {} configura los repositorios para el proyecto, en este caso se utilizó el de Bancolombia. [1]

El bloque buildscript {} se utiliza para definir los repositorios y las dependencies de Gradle comunes a todos los módulos del proyecto.[2]

El bloque dependencies {} configura todas las dependencias que se utilizaran en el proyecto[3].

La línea *apply plugin: 'java'* permite hacer uso de las herramientas que nos brinda el plugin de java, entre ellas está la posibilidad de generar el .jar del proyecto.

Mediante la línea *gradle.startParameter.continueOnFailure = true* se está indicando que aunque el test falle se debe seguir ejecutando la prueba.

La línea *sourceCompatibility = 1.8* se utiliza para indicar con cual versión de java se va a compilar. Por defecto esta variable tiene el valor que actualmente tiene la JVM [4].

Cibergrafía:

[1]https://docs.gradle.org/4.10-rc-2/dsl/org.gradle.api.Project.html#org.gradle.api.Project:repositories(groovy.lang.Closure)

[2] https://developer.android.com/studio/build/?hl=es-419#top-level

[3]https://docs.gradle.org/4.10-rc-2/dsl/org.gradle.api.Project.html#org.gradle.api.Project:dependencies(groovy.lang.Closure)

[4] https://docs.gradle.org/current/userguide/java_plugin.html#other_convention_properties