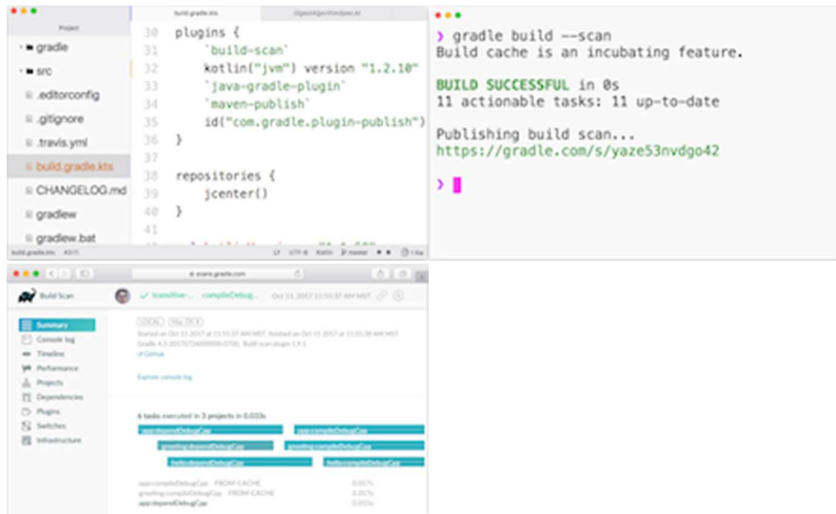


Tutorial sobre uso de Gradle

El primer paso es comprender que es Gradle. Gradle es una herramienta open-source de build automatica enfocada en la flexibilidad y en el rendimiento. Gradle permite trabajar con scripts que son escritos utilizando Groovy o Kotlin DSL (Domain-Specific Language).

Una de sus principales ventajas es que se puede integrar con multiples IDE, como Eclipse, IntelliJ IDEA, Visual Studio 2017, XCode, Netbeans, etc.



Por lo que su alcance en lenguajes no se reduce a Java unicamente, sino que abarca multiples opciones.

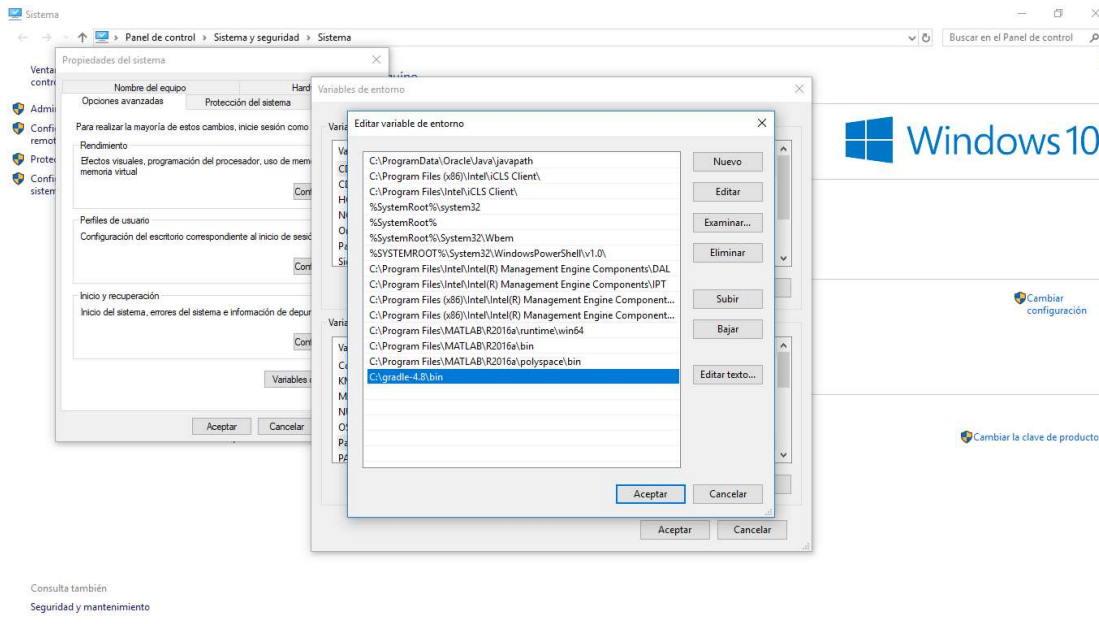
Para utilizar Gradle, debemos primero descargar el archivo. Se puede hacer mediante el siguiente [link](#).

Dependiendo del sistema operativo que se este utilizando, la instalación puede variar. En el caso de Windows, se deberá descomprimir el archivo en una ubicación que sea fija, i.e que no sea de uso frecuente, preferentemente: En la raíz de la particion C:

Una vez hecho esto se debe proceder con las siguiente instrucciones:

Equipo - - > Propiedades - - > Configuración avanzada del sistema - - > Variables de entorno

Aquí en las variable de sistema, seleccionar PATH, editar y añadir la ruta donde descomprimos Gradle-4.8



Para comprobar si hemos instalado correctamente Gradle, procedemos a usar la consola de Windows y ejecutar el comando:

`gradle -v`

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.16299.492]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Gabriel>gradle -v

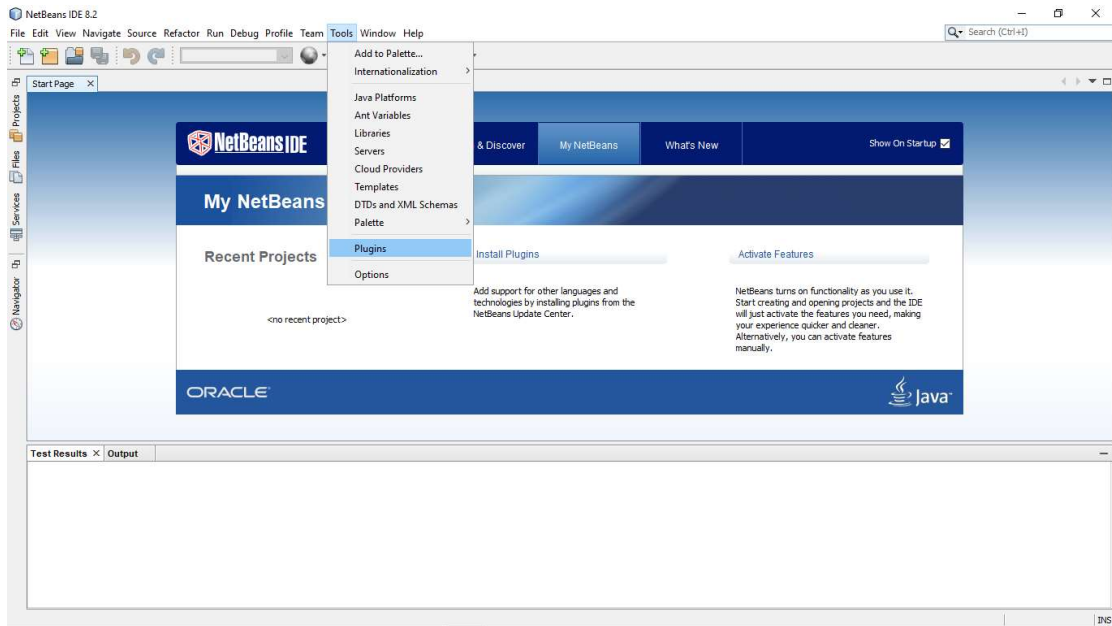
-----
Gradle 4.8
-----

Build time:   2018-06-04 10:39:58 UTC
Revision:     9e1261240e412cbf61a5e3a5ab734f232b2f887d

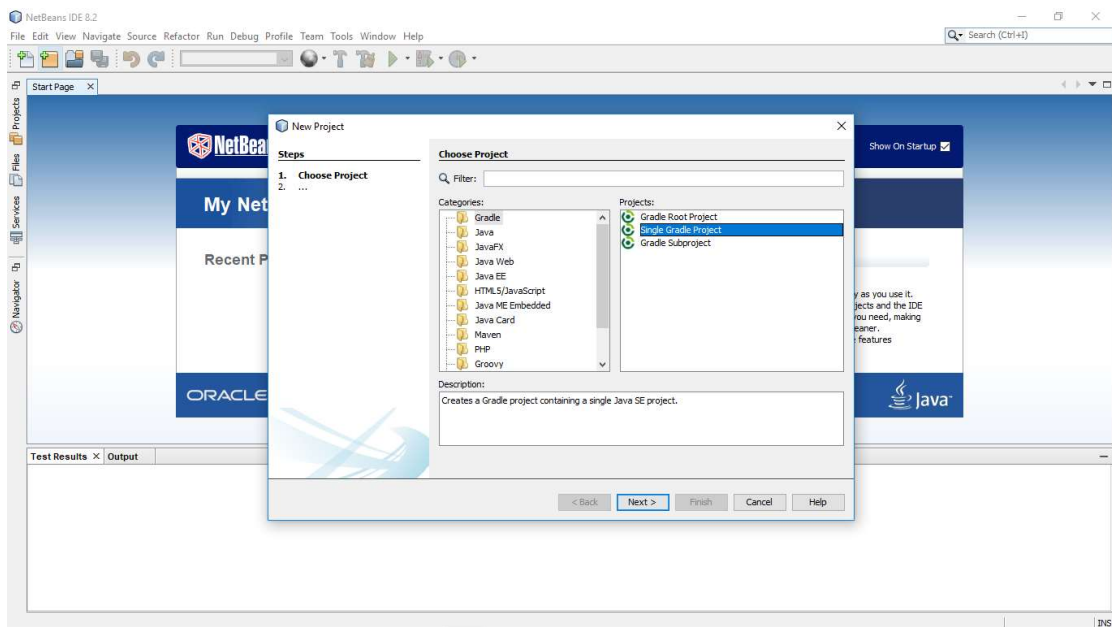
Groovy:       2.4.12
Ant:          Apache Ant(TM) version 1.9.11 compiled on March 23 2018
JVM:          1.8.0_101 (Oracle Corporation 25.101-b12)
OS:           Windows 10 10.0 amd64

C:\Users\Gabriel>
```

Con esto estamos listos para utilizar Gradle, para el uso dentro de la IDE, podemos usarla en Netbeans de la siguiente manera.



Debemos buscar el plugin de Gradle, descargar e instalar.



Teniendo así habilitada la opción de trabajar con Gradle desde Netbeans.

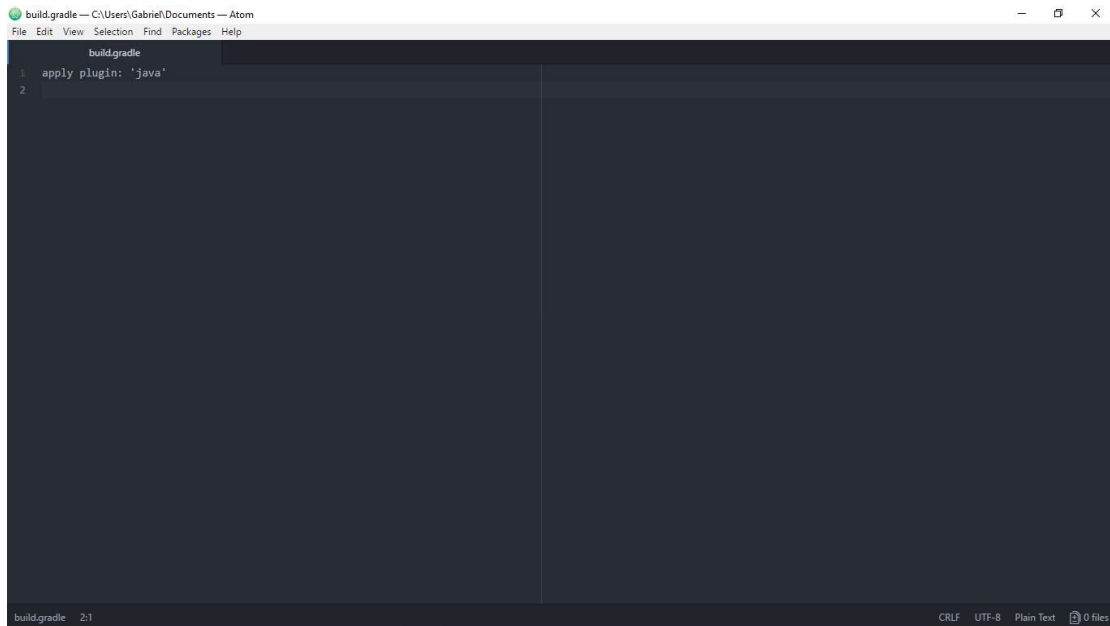
Building Proyectos en Java

Trataremos de djejar el código de Java para enfocarnos en el build del proyecto. Trataremos de desarrollar las mejores prácticas que un proyecto basado en Gradle debería seguir.

Crearemos una simple aplicación en Java, que devuelva el producto y la suma de dos numeros. Primero, debemos crear el directorio. En la raiz del directorio del proyecto creamos el archivo:

Build.gradle

Y añadimos las siguiente linea de codigo:

A screenshot of an Atom text editor window. The title bar shows the file path 'build.gradle' and the location 'C:\Users\Gabriel\Documents'. The menu bar includes 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. The editor area shows the 'build.gradle' file with two lines of code: '1 apply plugin: 'java'' and '2'. The status bar at the bottom indicates 'build.gradle 2:1' and 'CRLF UTF-8 Plain Text 0 files'.

Por defecto, al igual que Maven, los archivos de fuente de Java son leídos desde:

Src/main/java

Podemos configurar esto, por supuesto, pero lo dejaremos para después. Crearemos esta estructura en nuestro proyecto.

Ahora, necesitamos crear una clase de Java que genere las operaciones de la calculadora. También, crearemos una clase *Main* con un método main de tal forma de que los cálculos sean ejecutados desde este método. Los archivos Java deberían mantenerse en una fuente del directorio raíz bajo la apropiada estructura de package. Si utilizamos un ejemplo, tenemos:

```

hello-java
├── build.gradle           // build file
└── src
    ├── main
    │   └── java           // source root
    │       ├── com
    │       │   └── packtpub
    │       │       ├── ge
    │       │       │   └── hello
    │       │       │       ├── GreetingService.java
    │       │       │       └── Main.java

```

El código se puede escribir como:

```

package com.packtpub.ge.hello;

public class GreetingService {
    public String greet(String user) {
        return "Hello " + user;
    }
}

package com.packtpub.ge.hello;

public class Main {
    public static void main(String[] args) {
        GreetingService service = new GreetingService();
        System.out.println(service.greet(args[0]));
    }
}

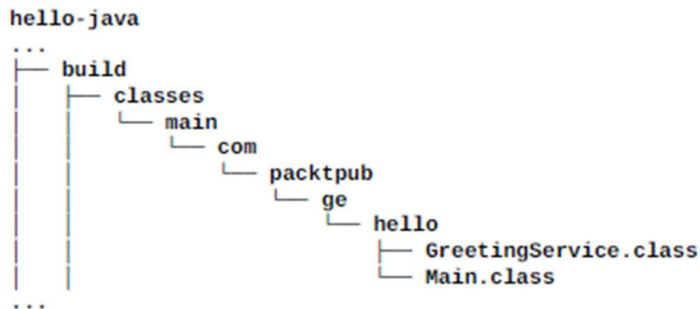
```

Esta clase posee el método main, el cual va a ser invocado cuando el programa inicie.

Después de añadir los archivos Java, queremos ahora compilar el proyecto y producir los archivos de clase. Esto se puede hacer simplemente ejecutando la siguiente task desde el command line:

```
$ gradle compileJava
```

Las clases compiladas se guardaran en *build/classes/main* relativo al proyecto raiz. Podemos confirmarlo chequeando el arbol de proyecto. Ignoraremos otros archivos y directorios por ahora:



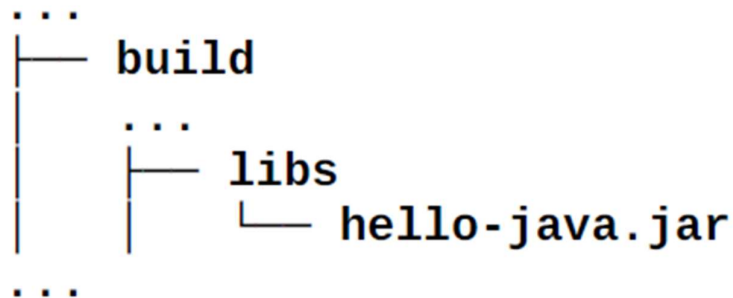
En este punto podemos directamente ejecutar las clases, pero lo haremos de una forma más general, a partir de un archivo .jar

Ejecutemos el siguiente comando:

```
$gradle build
```

El cual produce un Jar en *build/libs* en el directorio:

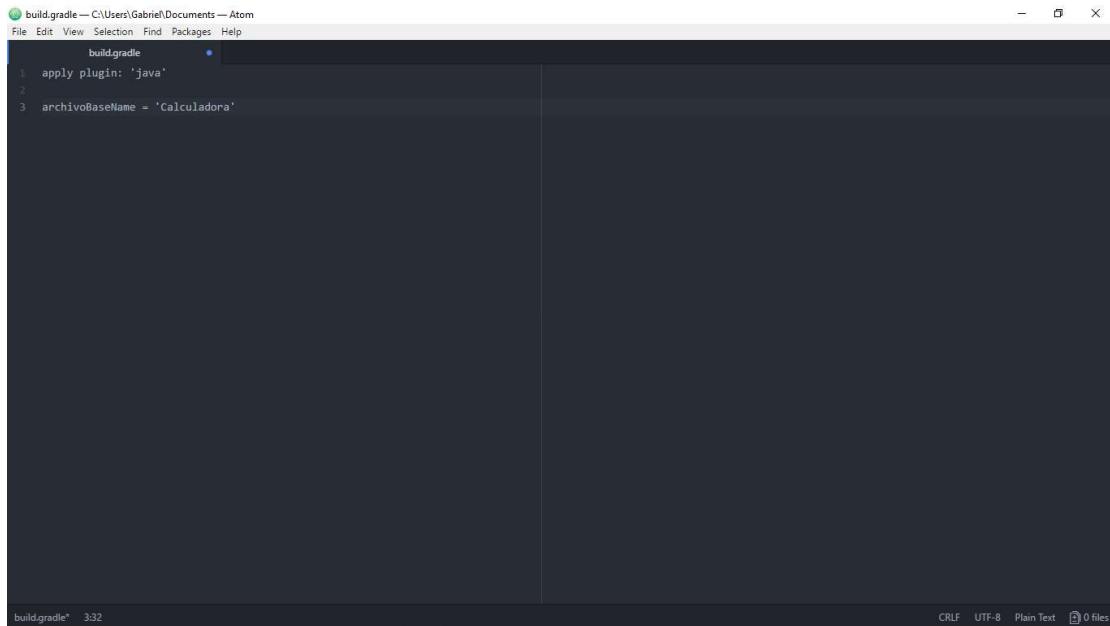
hello-java



Cuando ejecutamos la task build. Gradle tambien invoca el compileJava y otras task dependientes antes de ejecutar build; por lo que no es necesario ejecutar antes compileJava.

El nombre del archivo .jar es el mismo que el proyecto. Esto se puede configurar a traves del archivo build.gradle.

Por ejemplo:

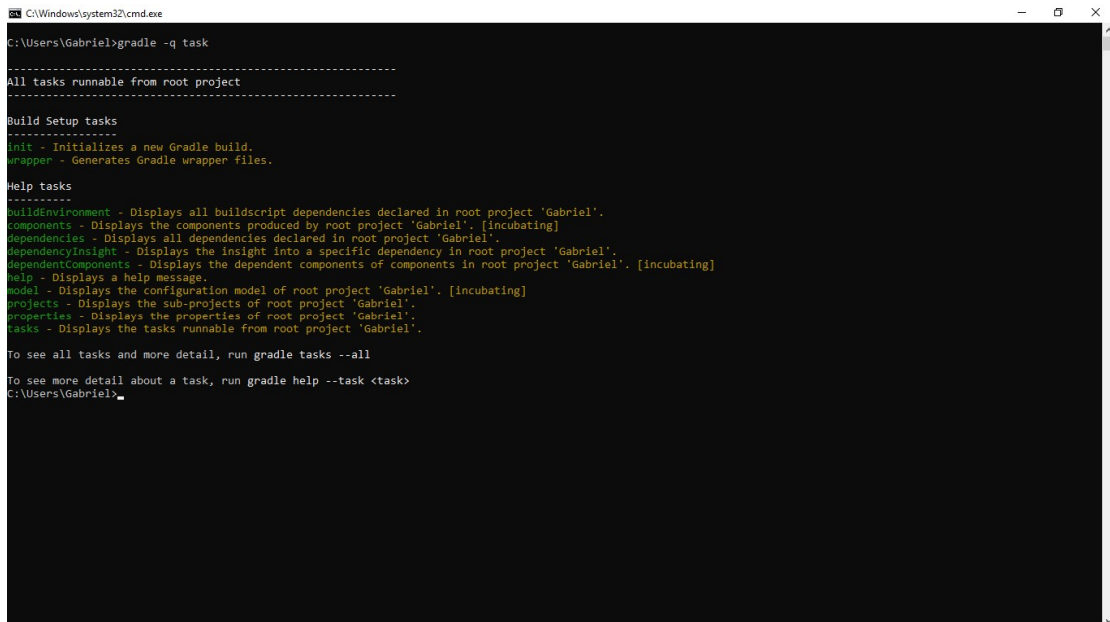


Ahora si ejecutamos:

```
$gradle clean
```

Y chequeamos el arbol de directorio de nuevo. No nos sorprende, que está limpio excepto los archivos fuentes.

Las task de Gradle son muchisimas, podemos ver esto haciendo:



Es interesante ver la cantidad de task utiles disponibles que podemos utilizar con el Java plugin. Claramente, Gradle emplea un poderoso mecanismo de

plugin que puede ser aplicado recordando el principio don't repeat yourself (DRY).

Gradle no es nada más que un task runner. Podemos también ejecutar JUnit test con Gradle. Los JUnit se guardan en `src/test/java` relativo a la raíz del proyecto. Debemos crear este directorio, y utilizando buenas prácticas, la estructura del package test reflejará el siguiente esquema:

```
...
src
├── test
│   ├── java                // test source root
│   │   ├── com
│   │   │   ├── packtpub
│   │   │   │   ├── ge
│   │   │   │   │   ├── hello
│   │   │   │   │   │   └── GreetingServiceTest.java
...

```

Y escribimos el test.

```
package com.packtpub.ge.hello;

import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class GreetingServiceTest {

    GreetingService service;

    @Before
    public void setup() {
        service = new GreetingService();
    }

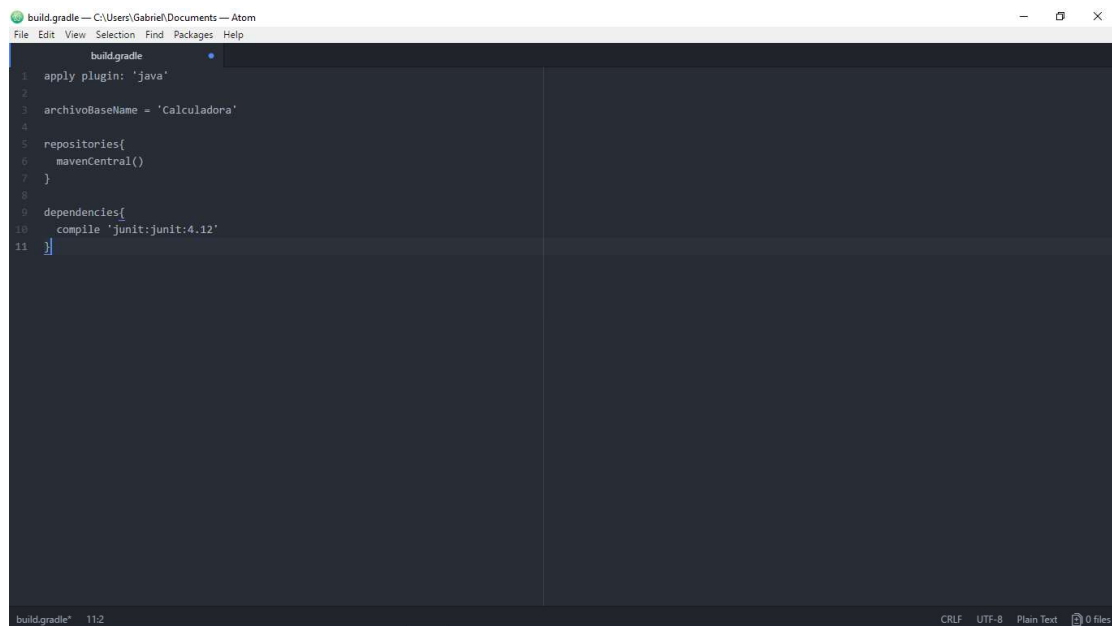
    @Test
    public void testGreet() {
        assertEquals("Hello Test", service.greet("Test"));
    }
}

```

Si intentamos compilar, posiblemente nos surja errores. Nos falta la librería de JUnit, para esto debemos modificar el archivo `build.gradle` como sigue:

Obtenemos las librerías que necesitamos desde:

<https://search.maven.org/>

A screenshot of an Atom text editor window. The title bar shows the file path 'build.gradle' and the location 'C:\Users\Gabriel\Documents'. The menu bar includes 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. The editor area shows a 'build.gradle' file with the following content:

```
1 apply plugin: 'java'
2
3 archivoBaseName = 'Calculadora'
4
5 repositories{
6     mavenCentral()
7 }
8
9 dependencies{
10     compile 'junit:junit:4.12'
11 }
```

The status bar at the bottom indicates 'build.gradle' 11:2, CRLF, UTF-8, Plain Text, and 0 files.

Con

\$gradle test

Ejecutamos todos los Junit.

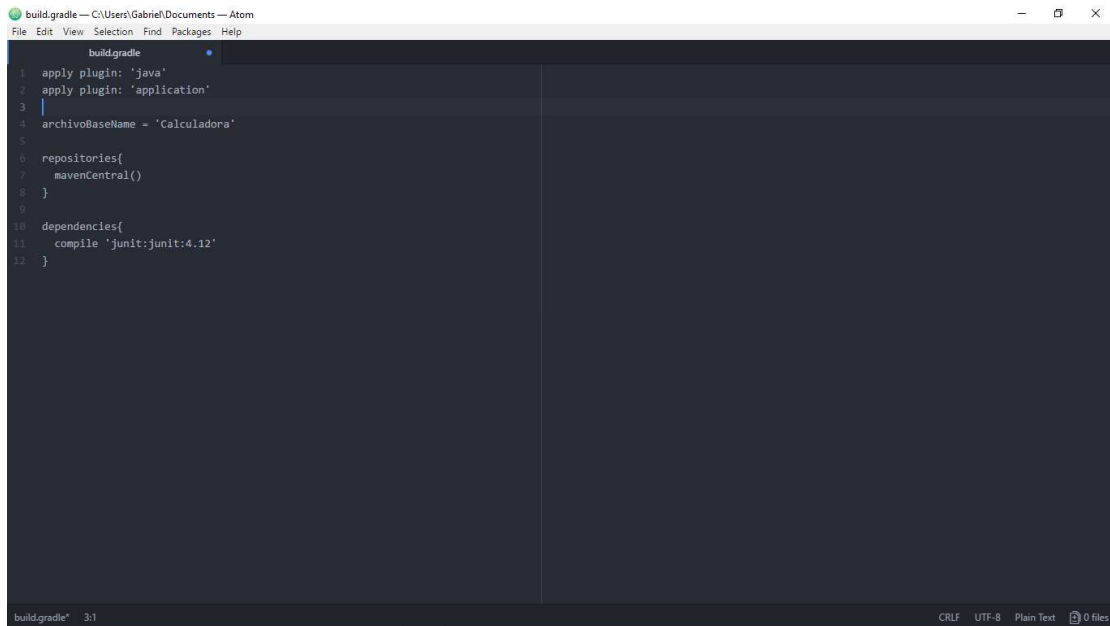
Si los test pasan al hacer:

\$gradle build

Podemos construir los archivos binarios de nuestra app.

Nota: UP-TO-DATE significa que nada a cambiado, y por lo tanto, Gradle no compilará de manera innecesaria los archivos fuentes de nuevo. Podemos forzar a ejecutar los task de nuevo haciendo `--rerun-tasks` en la command line, de tal forma que las acciones de task se ejecuten de nuevo.

Podemos hacer la aplicación de manera que se pueda distribuir, para ello solamente resta añadir una línea de comando.



```
build.gradle
1  apply plugin: 'java'
2  apply plugin: 'application'
3
4  archivoBaseName = 'Calculadora'
5
6  repositories{
7      mavenCentral()
8  }
9
10 dependencies{
11     compile 'junit:junit:4.12'
12 }
```

Para usar el segundo plugin debemos configurar Gradle para usar nuestra clase Main.

Para ello hacemos:

\$ mainClassName = “com.packtpub.ge.hello.Main”

Añadiendolo en nuestro archivo build.gradle

Usando el comando:

\$gradle distZip

Para poder distribuir nuestro programa. Este archivo se guardara en build/distributions. El nombre del ZIP por defecto es del proyecto. Esto se puede modificar mediante la linea:

\$ distributions.main.baseName = ‘NombrePrograma’

En el archivo build.gradle

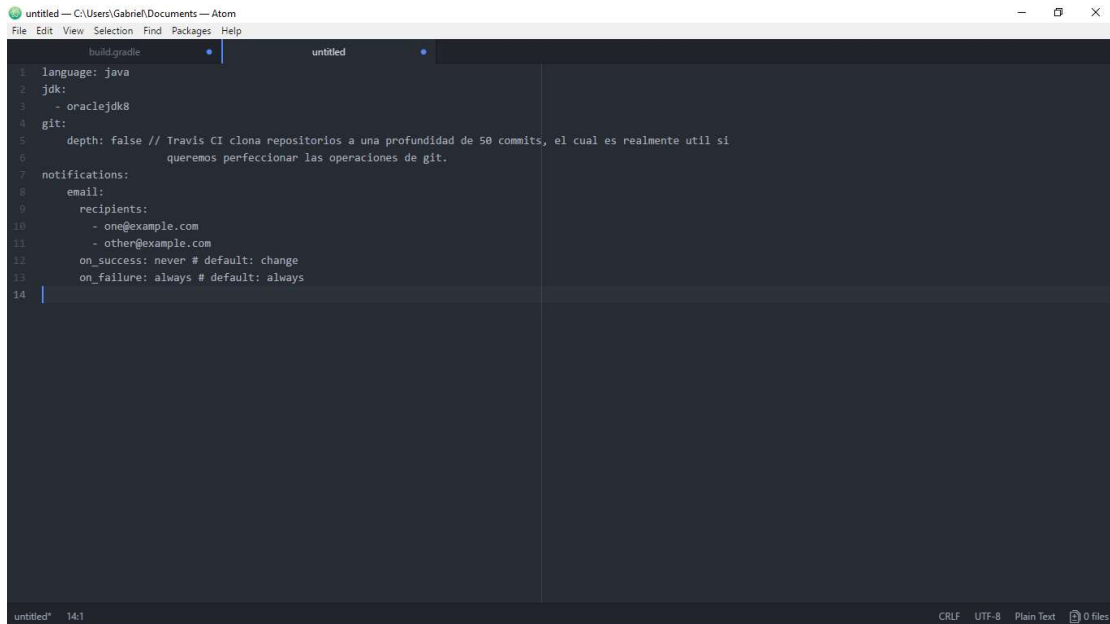
Travis

Con Gradle, nos aseguramos de poder usar Travis CI casi sin ningún problema, es más, si preferimos usar Jenkins existe un plugin especial para Gradle.

Cuando usamos Travis, creamos un archivo cuyo nombre es:

.travis.yml

Y en su contenido escribir, en el caso de que el código sea en Java:



```
1 language: java
2 jdk:
3   - oraclejdk8
4 git:
5   depth: false // Travis CI clona repositorios a una profundidad de 50 commits, el cual es realmente util si
6               queremos perfeccionar las operaciones de git.
7 notifications:
8   email:
9     recipients:
10      - one@example.com
11      - other@example.com
12   on_success: never # default: change
13   on_failure: always # default: always
14
```

Para mas informacion consultar:

<https://docs.travis-ci.com/>