

UD 4. Eines d'automatització

Automatització en Java. Gradle.

Entorns de desenvolupament



Continguts

1. Introducció a Gradle	3
2. Instal·lació de Gradle	3
3. Exemple amb Gradle	4
3.1. Creació d'un projecte	4
Creació del projecte amb una sola ordre	5
Estructura del projecte	5
3.2. Construcció i execució del projecte	7
3.3. Afegint dependències	8
4. Gradle en VSCode i Netbeans	11
4.1. Gradle i VSCode	11
4.2. Projectes Gradle a Netbeans	12

1. Introducció a Gradle

Fins ara hem vist dues eines per tal de crear projectes en Java i gestionar les seues dependències: Ant i Maven. Ant, com hem vist, és una eina molt versàtil, però deixa al desenvolupador la responsabilitat de definir totes les tasques que s'han de fer i com fer-les, el que porta a grans fitxers de construcció (build.xml) que a més poden ser molt complexos. Maven, per la seua banda, ens ho dóna tot predefinit, i els fitxers de configuració (pom.xml) no són excessivament complexos. A més, dóna suport a tot el cicle de vida de construcció. L'inconvenient que suposa és que quan volem realitzar algun projecte que no s'ajuste als arquetipus que té predefinit, pot arribar a ser bastant complex. Per altra banda, al tractar-se d'una eina tan completa, també creix en complexitat, fins al punt de, com hem vist a l'exemple de l'*hola Món*, generar fitxers de construcció pom.xml més llargs que el propi codi de l'aplicació.

En aquest context, el 2008 apareix Gradle que pretén integrar el millor de cada eina: les tasques i la personalització d'Ant, la potència, el cicle de construcció i proves tipus Maven, o la gestió de dependències d'altres eines com Ivy. A més, segueix un paradigma de convenció sobre configuració: Totes les opcions de configuració tenen valors per defecte comuns o útils, de manera que només s'hauran de modificar per a casos especials.

Una altra diferència important és que Gradle no fa ús de fitxers de configuració XML, sinó d'un llenguatge específic de domini (*Domain-Specific Language* o DSL), inspirat en el llenguatge de programació Groovy.

Per altra banda, Gradle disposa d'una arquitectura de plugins variats: compilació de projecte Java, Groovy, Maven, generació de documentació, proves, etc.

Anem a seguir el mateix procediment que per a Maven, i crearem un projecte en Gradle de tipus *Hola Món*.

2. Instal·lació de Gradle

Gradle es troba als dipòsits de programari d'Ubuntu, pel que podem instal·lar-lo amb apt:

```
1 $ sudo apt-get update
2 ...
3 $ sudo apt-get install gradle
```

Com veurem, el paquet arrossega moltes dependències. Diem que sí a tot, i una vegada instal·lat, podem comprovar la versió amb:

```
1 -----
2 Gradle 5.1
```

```
3 -----
4
5 Build time:    2019-01-02 18:57:47 UTC
6 Revision:     d09c2e354576ac41078c322815cc6db2b66d976e
7
8 Kotlin DSL:    1.1.0
9 Kotlin:        1.3.11
10 Groovy:        2.5.4
11 Ant:           Apache Ant(TM) version 1.9.13 compiled on July 10 2018
12 JVM:           11.0.5 (Private Build 11.0.5+10-post-Ubuntu-0ubuntu1
13               .118.04)
14 OS:            Linux 4.15.0-72-generic amd64
```

Com podem veure, se'ns mostra informació sobre la versió de Gradle (5.1), Groovy (2.5.4), Ant (1.9.13), JVM (11.0.5), entre d'altres, així com el sistema operatiu.

3. Exemple amb Gradle

Anem a veure com farem amb Gradle el projecte per a un *Hola Món* típic.

3.1. Creació d'un projecte

Per tal de crear un projecte nou en Gradle, farem ús de l'ordre `gradle init`. Per a això, ens ubiquem en una carpeta buida, invoquem `gradle init` i anem seguint l'assistent:

```
1 $ gradle init
2
3 Select type of project to generate:
4   1: basic
5   2: groovy-application
6   3: groovy-library
7   4: java-application
8   5: java-library
9   6: kotlin-application
10  7: kotlin-library
11  8: scala-library
12 Enter selection (default: basic) [1..8] 4
13
14 Select build script DSL:
15   1: groovy
16   2: kotlin
17 Enter selection (default: groovy) [1..2] 1
18
19 Select test framework:
20   1: junit
21   2: testng
```

```
22 3: spock
23 Enter selection (default: junit) [1..3] 1
24
25 Project name (default: gradle): helloGradle
26 Source package (default: helloGradle): com.ieseljust.edd
27
28 BUILD SUCCESSFUL in 17s
29 2 actionable tasks: 2 executed
```

Com veiem, se'ns executa un assistent que ens demana el tipus de projecte (*java application*), el llenguatge per definir el projecte (*groovy*), el framework de test (*junit*) i el nom del projecte (*helloGradle*).

Creació del projecte amb una sola ordre

Gradle també permet indicar-li mitjançant paràmetres les diferents característiques del projecte a generar. Així, podem fer exactament el mateix que amb l'assistent anterior de la següent forma:

1. En primer lloc, creem el directori del projecte i accedim a ell:

```
1 $ mkdir helloGradle
2 $ cd helloGradle
```

2. I ara, llancem el `gradle init` amb els paràmetres corresponents al tipus d'aplicació, al tipus de test, al tipus de llenguatge per al build.gradle, el nom del projecte i el nom complet del paquet:

```
1 $ gradle init --type java-application \
2               --test-framework junit \
3               --dsl groovy \
4               --project-name helloGradle \
5               --package com.ieseljust.edd
```

Com veurem, si ho fem així, el projecte es genera directament, sense fer-nos cap pregunta.

Estructura del projecte

Tant si hem generat el projecte amb l'assistent o directament, aquest presenta la següent estructura de directoris:

```
1 .
2 |-- build.gradle
3 |-- gradle
4 |-- wrapper
```

```

5 |         |-- gradle-wrapper.jar
6 |         |-- gradle-wrapper.properties
7 | -- gradlew
8 | -- gradlew.bat
9 | -- settings.gradle
10 | -- src
11 |     |-- main
12 |         |-- java
13 |             |-- com
14 |                 |-- ieseljust
15 |                     |-- edd
16 |                         |-- App.java
17 |         |-- resources
18 |     |-- test
19 |         |-- java
20 |             |-- com
21 |                 |-- ieseljust
22 |                     |-- edd
23 |                         |-- AppTest.java
24 |         |-- resources
25 | -- wrapper
26 |     |-- gradle-wrapper.jar
27 |     |-- gradle-wrapper.properties

```

Com que de moment no necessitem generar tests, podem esborrar els fitxers d'aquesta carpeta amb `rm -r src/test`.

El contingut del fitxer `App.java` és una altra variant de l'*Hola Món*:

```

1  /*
2  * This Java source file was generated by the Gradle 'init' task.
3  */
4  package com.ieseljust.edd;
5
6  public class App {
7      public String getGreeting() {
8          return "Hello world.";
9      }
10
11     public static void main(String[] args) {
12         System.out.println(new App().getGreeting());
13     }
14 }

```

Veiem ara el contingut del fitxer de projecte `build.gradle` (llevem els comentaris per a què es llisca millor):

```

1  plugins {
2      id 'java'
3      id 'application'

```

```
4 }
5
6 repositories {
7     jcenter()
8 }
9
10 dependencies {
11     implementation 'com.google.guava:guava:26.0-jre'
12     testImplementation 'junit:junit:4.12'
13 }
14
15 mainClassName = 'com.ieseljust.edd.App'
```

De moment, el que ens interessa és veure la secció de plugins, en la que indiquem que es tracta d'una aplicació java, i que la classe principal és `com.ieseljust.edd.App`. La secció de repositoris i dependències ens servirà quan treballem amb llibreries externes. **Tingueu en compte que si la vostra classe principal -la que conté el mètode main- està implementada a un fitxer diferent a `App.java`, com per exemple `Calcula.java`, caldrà canviar també el `mainClassName` per a que faci referència a aquesta classe, així com si es troba en alguna subcarpeta -`com.ieseljust.edd.calc.Calcula` si es tracta de `Calcula.java` dins la carpeta/subpaquet `calc`.**

3.2. Construcció i execució del projecte

Una vegada hem vist l'esquelet del projecte, podem construir-lo amb l'ordre `gradle build`, des de la carpeta principal del projecte:

```
1 $ gradle build
2
3 BUILD SUCCESSFUL in 20s
4 7 actionable tasks: 7 executed
```

Amb açò, compila, processa els recursos i genera les classes i l'empaquetat jar de l'aplicació. Si ens fixem, tenim una nova carpeta `build` amb el resultat de la construcció del paquet:

```
1 build
2 |-- classes
3 |   |-- java
4 |       |-- main
5 |           |-- com
6 |               |-- ieseljust
7 |                   |-- edd
8 |                       |-- App.class
9 |-- distributions
10 |   |-- helloGradle.tar
11 |   |-- helloGradle.zip
12 |-- libs
```

```
13 |   '-- helloGradle.jar
14 |   -- scripts
15 |   |   '-- helloGradle
16 |   |   '-- helloGradle.bat
17 |   -- tmp
18 |   |   '-- compileJava
19 |   |   '-- jar
20 |   |   '-- MANIFEST.MF
```

Si volem veure les diferents fases per les que passa la construcció de l'aplicació, podem utilitzar l'opció `-i` de Gradle: `gradle build -i`.

Per altra banda, si volem netejar el projecte i tornar a la configuració anterior, farem:

```
1 $ gradle clean
```

Finalment, per tal d'executar l'aplicació, només haure d'invocar `gradle run` per executar aquest:

```
1 $ gradle run
2
3 > Task :run
4 Hello world.
5
6 BUILD SUCCESSFUL in 1s
7 2 actionable tasks: 1 executed, 1 up-to-date
```

3.3. Afegint dependències

Moltes vegades, als nostres projectes necessitem algunes funcionalitats que no proporcionen les pròpies llibreríes de Java, i ho farem mitjançant llibreríes de tercers. Per exemple, quan hàgem de treballar amb fitxers JSON necessitem una llibreria que ens permeti *entendre* aquest tipus de fitxers, o quan hàgem de treballar amb connectors a bases de dades, necessitem llibreríes que s'encarreguen de facilitar-nos les connexions.

Un dels llocs de referència per tal de trobar és el repositori de Maven (<https://mvnrepository.com/>), que a banda de Maven, permet utilitzar les llibreríes en qualsevol tipus de projectes.

Com a exemple, anem a veure com inclouríem una llibreria externa dins les nostres dependències. Anem a utilitzar la llibreria *Apache Commons Math*, amb funcions matemàtiques addicionals. Per a això, busquem mitjançant el quadre de cerca de la web del repositori Maven alguna llibreria que contingui *Math* i veurem que aquesta ens apareix la primera:

Search results for 'math' showing 497 results. The top two results are:

- 1. Apache Commons Math** (1,512 usages)
 - org.apache.commons » commons-math3
 - The Apache Commons Math project is a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language or commons-lang.
 - Last Release on Mar 17, 2016
- 2. Commons Math** (348 usages)
 - org.apache.commons » commons-math
 - The Math project is a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language or commons-lang.
 - Last Release on Feb 26, 2011

Figura 1: Búsqueda de la llibrería math

Si fem clic en ella, ens portarà a una altra pàgina amb les diferents versions d'aquesta llibreria:

Home » org.apache.commons » commons-math3

Apache Commons Math

The Apache Commons Math project is a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language or commons-lang.

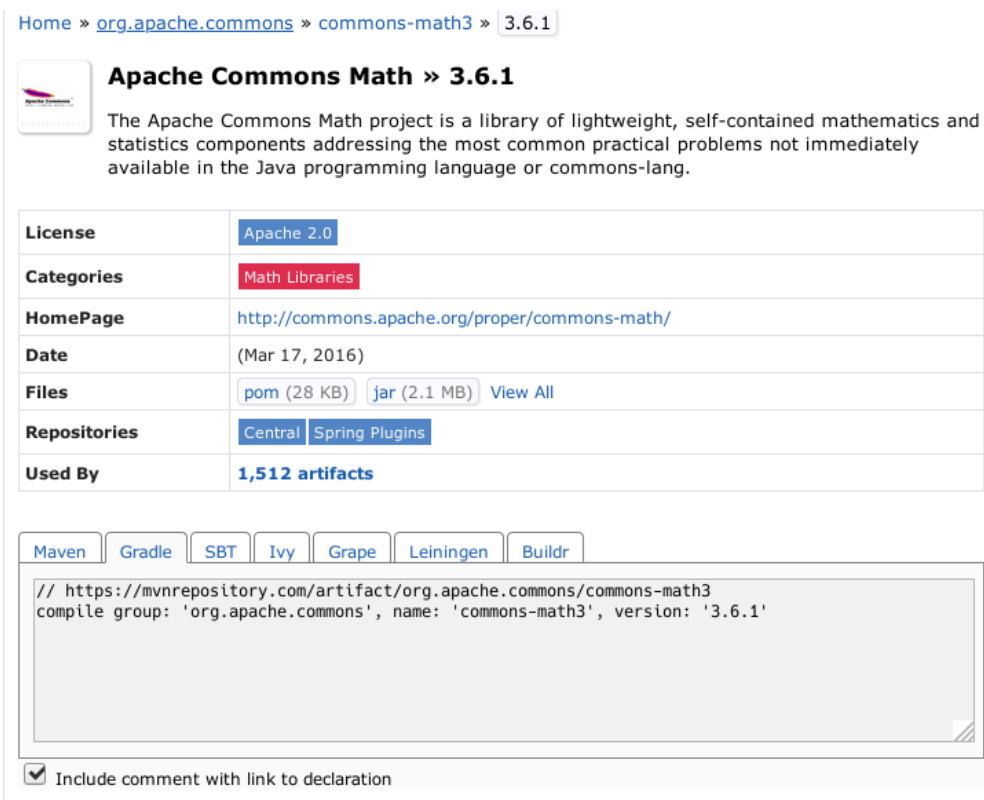
License	Apache 2.0
Categories	Math Libraries
Tags	math apache commons
Used By	1,512 artifacts

Central (10) Redhat GA (1) ICM (2)

	Version	Repository	Usages	Date
3.6.x	3.6.1	Central	771	Mar, 2016
	3.6	Central	70	Jan, 2016
3.5.x	3.5	Central	213	Apr, 2015

Figura 2: Búsqueda de la llibrería math

Fem clic a la versió més recent (3.6.1), i en la següent pàgina, veurem una secció amb diferents pestanyes, per tal d'utilitzar la llibreria en diferents sistemes. Fem clic a la pestanya de *Gradle*, ja que la volem incloure en un projecte d'aquest tipus, i copiem el contingut que ens ofereix al portaretalls:



Home » [org.apache.commons](#) » [commons-math3](#) » 3.6.1

Apache Commons Math » 3.6.1

The Apache Commons Math project is a library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language or commons-lang.

License	Apache 2.0
Categories	Math Libraries
HomePage	http://commons.apache.org/proper/commons-math/
Date	(Mar 17, 2016)
Files	pom (28 KB) jar (2.1 MB) View All
Repositories	Central Spring Plugins
Used By	1,512 artifacts

[Maven](#) [Gradle](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
// https://mvnrepository.com/artifact/org.apache.commons/commons-math3
compile group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
```

☒ Include comment with link to declaration

Figura 3: Búsqueda de la llibrería math

Ara editarem el fitxer `build.gradle` del nostre projecte, i modificarem dues de les seccions que apareixen. En primer lloc, cal modificar la secció de *repositories*, per tal d'afegir el repositori de Maven, per a això, localitzem la secció *repositories*, i afegim la línia `mavenCentral()` a aquest. Ens quedarà de la següent forma:

```
1 repositories {
2     // Use jcenter for resolving your dependencies.
3     // You can declare any Maven/Ivy/file repository here.
4     jcenter()
5     mavenCentral()
6 }
```

Ara haurèm d'afegir el codi proporcionat pel repositori de Gradle a la secció *dependencies*. Aquesta quedarà de la següent manera:

```
1 dependencies {
2     // This dependency is found on compile classpath of this component
3     // and consumers.
4     implementation 'com.google.guava:guava:26.0-jre'
5
6     // Use JUnit test framework
7     testImplementation 'junit:junit:4.12'
8
9     // https://mvnrepository.com/artifact/org.apache.commons/commons-
10    math3
11    compile group: 'org.apache.commons', name: 'commons-math3', version
12    : '3.6.1'
13 }
```

Ara, quan construïm el projecte, automàticament es descarregarà la llibreria `commons-math3` i estarà preparada per utilitzar-se al projecte. Quan fem un `clean`, aquesta s'esborrarà, per tornar a descarregar-se en una nova construcció.

4. Gradle en VSCode i Netbeans

4.1. Gradle i VSCode

VSCode disposa d'una extensió de suport al llenguatge Gradle que ens pot ajudar a construir el fitxer `build.gradle` i a remarcar-ne la sintaxi, però, a diferència de la de Maven, no ens ofereix la possibilitat de llençar objectius directament, cosa que haurem de seguir fent des de la consola.

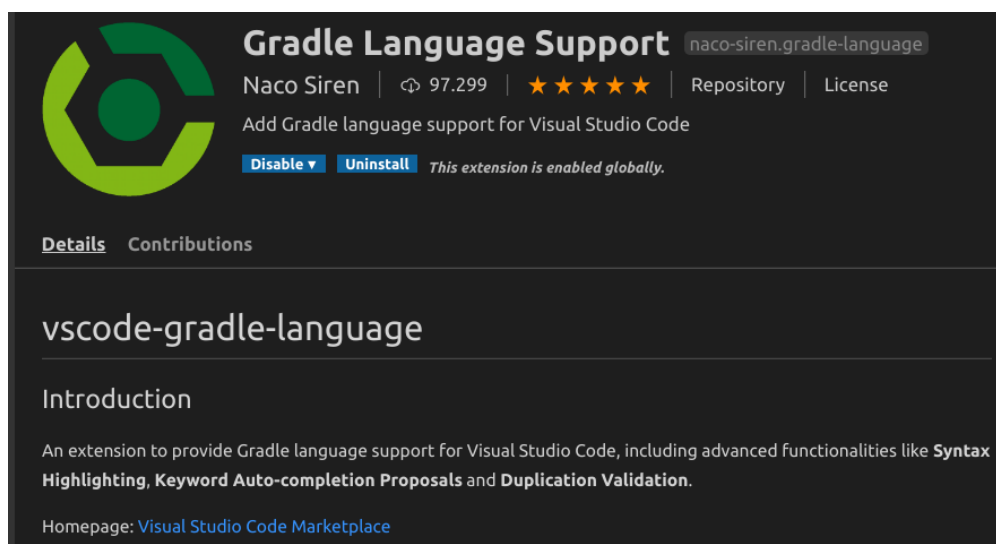


Figura 4: Extensió de Gradle per a VSCode

4.2. Projectes Gradle a Netbeans

A partir de la versió 11 de Netbeans, els projectes Gradle se suporten de forma nativa. Per tal de crear un projecte Gradle en Netbeans, farem com en tot projecte: *File -> New Project*. Ara, haurem de seleccionar que volem generar una aplicació java en Gradle:

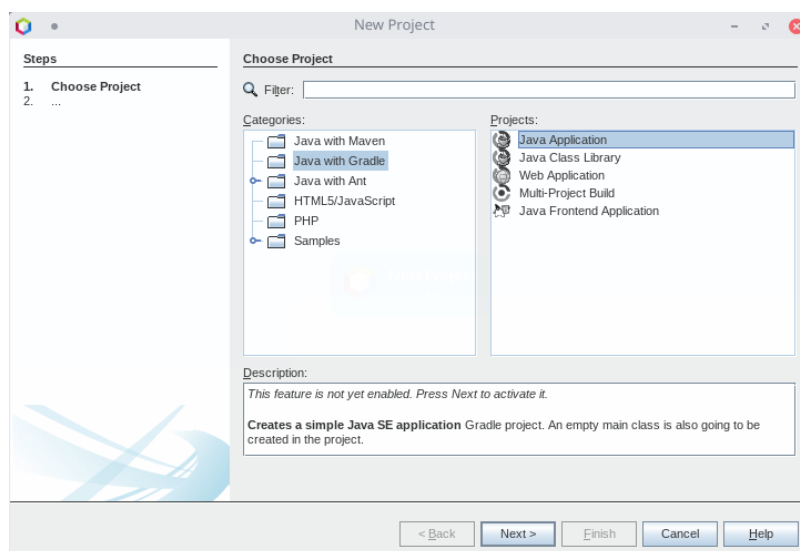


Figura 5: Creació d'un projecte Gradle a Netbeans

I després indicar el nom i la ubicació del projecte:

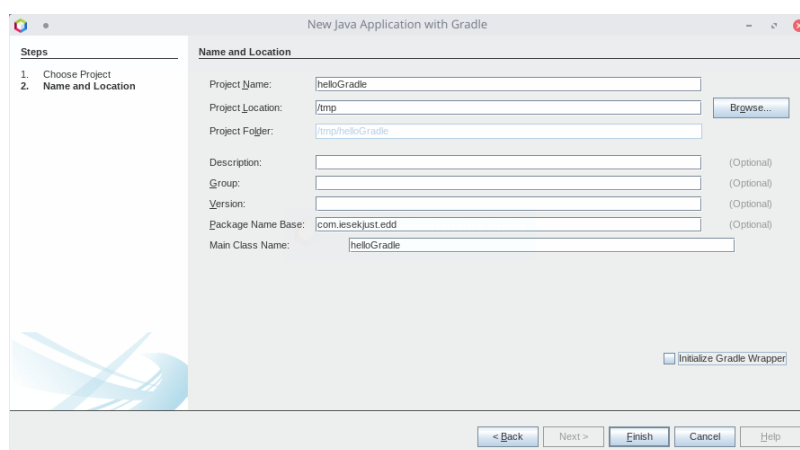


Figura 6: Nom i localització del projecte

Fet açò, fem click en *Finish*, per veure com queda el projecte. Com podem apreciar, amb Netbeans se'ns queda una estructura més senzilla que utilitzant només Gradle, on veiem, des de la vista de

projecte que tenim el paquet creat dins la carpeta *Source Packages*, i l'script *build.gradle* a la carpeta *Build Scripts*.

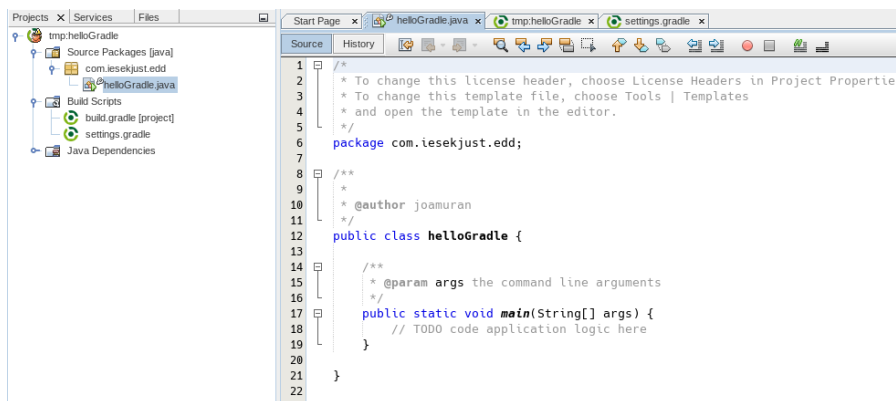


Figura 7: Vista del projecte

Si fem clic a la vista de fitxers, veurem que tenim l'estructura de carpetes que dóna lloc al paquet, i el fitxer *build.gradle* a la carpeta arrel. També veurem que aquest fitxer és basant més senzill que el que ens ha generat Gradle anteriorment.

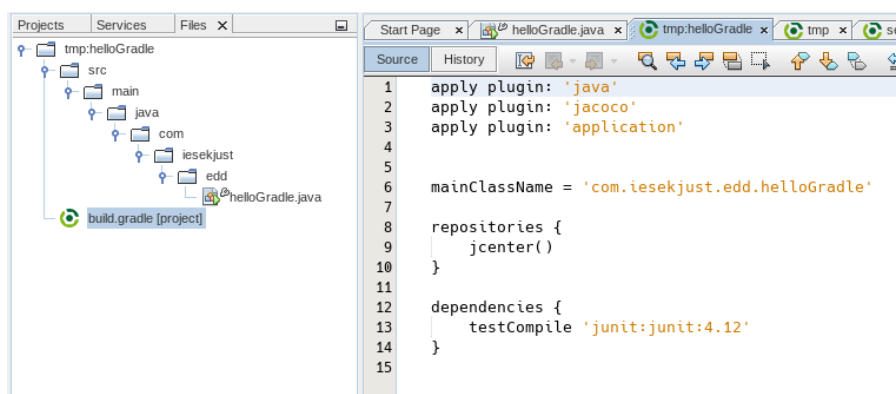


Figura 8: Fitxers del projecte

El principal inconvenient que trobarem amb Gradle al Netbeans és que requereix la versió 4.10 per funcionar, mentre que nosaltres estem treballant amb la versió 5.1, pel que per executar el projecte ha de descarregar aquesta versió. Si volem utilitzar la versió de Gradle que tenim instal·lada, podem utilitzar-la amb el projecte creat en Netbeans des de la línia d'ordres sense problemes.

Per a més informació sobre Gradle, podeu consultar els següents enllaços:

- Construcció de projectes Gradle: <https://spring.io/guides/gs/gradle/>
- Web oficial de Gradle: <https://gradle.org/>

TO-DO

Creeu un projecte amb Gradle per al vostre codi de la calculadora. A aquesta calculadora haureu d'afegir ara a més dos mètodes més:

```
1 public Boolean esPrim(float op1){...}
```

Aquesta funció ens retornarà un valor lògic, indicant si el número és prim o no (prim=divisible només per ell mateix i per 1).

```
1 public Integer proximPrim(float op1){...}
```

Que ens retornarà el següent número prim al que li indiquem (si indiquem un prim ens retornarà el mateix)

Per a estos mètodes haureu d'utilitzar les els mètodes estàtics `isPrime((int)op1)` i `nextPrime((int)op1)` de la classe `Primes`, que haurem d'importar al projecte mitjançant l'import de la llibreria `org.apache.commons.math3.primes.Primes`; que hem inclòs per utilitzar al fitxer `gradle.build`.

Ara, a més, al programa principal, després de mostrar totes les operacions anteriors, haurà d'indicar si el primer argument proporcionat és prim i quin és el seu pròxim número prim.

L'eixida del programa serà una cosa semblant a aquesta:

```
1 $ gradle run --args "4 3"
2
3 Task :run
4 La suma entre 4.0 i 3.0 és 7.0
5 La resta entre 4.0 i 3.0 és 1.0
6 La multiplicació entre 4.0 i 3.0 és 12.0
7 La divisió entre 4.0 i 3.0 és 1.3333334
8 Última operació realitzada: Divideix; Últim resultat: 1.3333334
9 El número 4.0 NO és prim
10 El prim següent de 4.0 es 5
```

Podeu utilitzar qualsevol dels mecanismes i IDEs que hem vist.