Presentació: Ferramentes d'automatització

Una altra de les \*tares d'especial importància en la fase d'implementació, i a la qual generalment no es presta l'atenció que requeriria és l'automatització de tasques de construcció i gestió de dependències. Estes ferramentes ens ajudaran d'una banda a gestionar projectes amb múltiples fitxers font, gestionar les dependències que té el nostre programari respecte a biblioteques de tercers, així com a generar l'empaquetat de les nostres aplicacions.

En el món de Java existixen principalment tres ferramentes d'este tipus: Apatxe \*Ant, Apatxe \*Maven i \*Gradle. Veurem les principals característiques que tenen estes ferramentes i aprendrem a crear i gestionar el cicle de construcció de l'aplicació en cadascuna d'elles.

Apartat: Ferramentes d'automatització

Com hem comentat, les ferramentes d'automatització comencen a ser d'utilitat quan els nostres programes comencen a créixer i a dependre de llibreries de tercers. Per a exemplificar totes les ferramentes, anem a partir d'un senzill programa, a l'estil “Hola Món”, però amb tots els ingredients necessaris.

1. Exemple de partida: un Hola Món amb dos fitxers

Este exemple de base, al costat dels diferents exemples de la resta d'unitat amb \*Ant, \*Maven i \*Gradle els tens disponibles en el repositori de \*Github: [https://git.io/jcari](https://git.io/JcarI)

REPOSITORI

Disposem de dos fitxers \*Main.\*java i Saluda.\*java, organitzats en la següent estructura de directoris en la carpeta *\*saludaBase*:

.

└── \*src

   └── \*com

       └── \*mgh

           └── \*edd

               ├── Hola.\*java

               └── Saluda.\*java

I amb el següent contingut:

|  |  |
| --- | --- |
| Fitxer \*src/\*com/\*mgh/\*edd/Saluda.\*java | Fitxer \*src/\*com/\*mgh/\*edd/Hola.\*java |
| \*package \*com.\*mgh.\*edd;  \*public \*class Saluda {  \*public \*static \*void saluda(\*String nom) {  \*System.out.\*println("Hola "+nom);  }  } | \*package \*com.\*mgh.\*edd;  \*public \*class Hola {  \*public \*static \*void \*main(\*String[] \*args) {  \*String nom=\*args.\*length>0?\*args[0]:"";  Saluda.saluda(nom);  }  } |

Expliquem un poc el contingut. En primer lloc, quan treballem amb diversos fitxers font a Java, podem organitzar el codi en carpetes, que donen lloc al que es coneix com a *paquets (\*package en anglés)*. Els noms de paquets s'escriuen en minúscula, i generalment, les empreses utilitzen la seua de domini d'Internet al revés com a part d'este. Per exemple, si el nostre domini anara *mgh.com*, el paquet de la nostra aplicació podria ser *\*com.\*mgh.\*edd.* Això té dos implicacions en els fitxers font:

* D'una banda, han d'indicar que pertanyen a este paquet, mitjançant la directiva al principi del fitxer: \*package \*com.\*mgh.\*edd,
* D'altra banda, ha d'existir esta correlació entre el nom del paquet i l'estructura de carpetes al fet que es correspon: *\*com/\*mgh/\*edd* amb *\*com.\*mgh.\*edd*. Observeu que hem situat el directori *\*com*, dins d'un altre directori *\*src.* Este directori *\*src* (abreviatura de *\*source*) serà el directori arrel per al nostre codi font.

D'altra banda, \*fijáos que el programa es compon de dos fitxers, el principal (Hola.\*java) que és qui conté el mètode *\*main*, i d'altra banda, el fitxer *Saluda.\*java*, que oferix la funcionalitat *saluda*. A més, el mètode principal *\*Main* arreplega un *argument* opcional de la línia d'ordes per a personalitzar el missatge que mostrarà per pantalla.

Tot això ho veureu en el seu moment en programació, així que no us preocupeu si no enteneu part del codi. De moment, el que ens interessa és que el nostre programa es compon de dos fitxers font, i que un d'ells depén d'una funcionalitat de l'altre.

important

Per a compilar el codi anterior, ens situarem en la carpeta arrel del codi *\*saludaBase/\*src* , i farem:

$ \*javac \*com/\*mgh/\*edd/Saluda.\*java

$ \*javac \*com/\*mgh/\*edd/Hola.\*java

Amb això generem en la carpeta *\*com/\*mgh/\*edd* els dos fitxers en \*btyecode *Saluda.\*class* i *Hola.\*class*.

Realment, només amb la segona orde, és a dir, compilant el fitxer *Hola.\*java* hauria sigut suficient, ja que el compilador detecta que s'utilitza una funcionalitat que està implementada en un altre fitxer font del mateix paquet i compilarà també est.

Finalment, per a executar el codi anterior, utilitzarem:

$ \*java \*com.\*mgh.\*edd.Hola Jose

Hola Jose

Fixeu-vos que ara per a executar el nostre programa hem utilitzat el nom de la *classe* Hola precedida del nom del paquet. A més, podem passar-li qualsevol argument al nostre programa perquè emeta una salutació personalitzada.

A partir d'este exemple de base, veurem com automatitzaríem este procés.

1. Apatxe \*ant

Apatxe \*Ant és una biblioteca de Java que ens permet automatitzar el procés de construcció d'aplicacions. Encara que s'utilitza principalment a Java, també suporta altres llenguatges. Inicialment, va formar part del projecte Apatxe \*Tomcat, però l'any 2000 es va llançar com a projecte independent.

1. Instal·lació en Linux

Apatxe \*Ant està disponible en els repositoris de totes les distribucions GNU/Linux. Per a la seua instal·lació en \*Ubuntu o derivades, utilitzarem la ferramenta de línia de comandos *\*Advanced \*Package \*Tool*, més coneguda com *\*apt*. En primer lloc, refresquem la llista de paquets amb:

$ sue \*apt \*update

I després instal·lem \*ant amb:

sue \*apt \*install \*ant

**La \*instaiación en Windows es veurà en cas pràctic apart.**

1. El fitxer \*build.\*xml

Tot projecte en \*Ant es basa en un *fitxer de construcció*, anomenat generalment *\*build.\*xml*, situat en l'arrel del projecte, i que conté els diferents objectius o *\*targets*, que no són més que les diferents fases de construcció del projecte.

Vegem estos *\*targets* mitjançant el següent exemple de fitxer per al nostre projecte *\*saludaAnt* del repositori d'exemples de la unitat:

<\*project \*name="\*saludaAnt">

<\*target \*name="\*clean">

<\*delete \*dir="\*classes" />

</\*target>

<\*target \*name="compile" \*depends="\*clean">

<\*mkdir \*dir="\*classes" />

<\*javac \*includeantruntime="\*false"

\*srcdir="\*src/\*com/\*mgh/\*edd" \*destdir="\*classes" />

</\*target>

<\*target \*name="\*run" \*depends="compile">

<\*property \*name="\*arg0" \*value=""/>

<\*java \*classpath="\*classes" \*classname="\*com.\*mgh.\*edd.Hola">

<\*arg \*value="${\*arg0}"/>

</\*java>

</\*target>

</\*project>

Com podem veure, este fitxer XML conté un element arrel *<\*project>* amb l'atribut *\*name=”\*saludaAnt ”*, i que conté tres elements *\*target* amb diferent valor per a *\*name*. Com hem comentat, estos *\*targets* es correspondran a diferents fases del projecte de construcció:

* **\*clean:** S'encarrega de *netejar* el projecte. Com podem veure, conté un element *\*delete* amb un atribut *\*dir*, amb valor *\*classes*. Amb això, el que indiquem és que en la fase de *\*limipeza* del projecte, el que es farà serà esborrar la carpeta anomenada *classes*.
* **compile**: S'encarrega de la compilació del nostre projecte. En primer lloc, observeu que té l'atribut *\*depends=”\*clean”*, que indica que la fase de *compilació*, depén de la fase de *neteja* Això és, per a compilar el nostre projecte, necessitem netejar-ho abans. A més, hem d'incloure l'atribut *i\*ncludeantruntime="\*false"* per a no incloure les biblioteques de \*ant en la carpeta de classes i evitar un missatge d'alerta posterior. D'altra banda, inclou dos etiquetes *\*mkdir* i *\*javac.* Amb la primera, el que fa és crear una nova carpeta anomenada *\*classes*, en la qual situarem els arxius *.\*class*, per a així separar-los del codi font. Amb l'etiqueta *\*javac*, el que fa és indicar que s'invocarà al compilador proporcionant-li els arxius situats en la carpeta *\*srcdir (\*src/\*com/\*mgh/\*edd)*, i deixant els fonts en la carpeta *\*classes*.
* **\*run**: S'encarrega de l'execució de l'aplicació. Com veiem, depén de la fase de compilació, ja que, per a executar nostre *\*bytecode* haurem de compilar-ho prèviament. Este \*target conté l'etiqueta *<\*java>* que invoca a la màquina virtual de Java per a llançar la *classe* *\*com.\*mgh.\*edd.Hola*, on se situa el nostre mètode *\*Main*. A més, establix l'atribut *\*classpath* al valor *\*classes*, per a indicar que la resta de recursos del programa estaran en esta carpeta *\*classes* que hem creat. Dins d'esta etiqueta, també hem afegit una altra etiqueta *<\*arg>*, amb valor "${\*arg0}", amb el que indiquem que es passarà al programa principal el primer argument que es proporcione en la invocació de *\*ant.* A més, s'ha afegit un valor per defecte per a este argument que serà la cadena buida, mitjançant l'etiqueta <\*property \*name="\*arg0" \*value=""/>.

1. Ús de *\*ant* i el fitxer \*build.\*xml

Una vegada generat el fitxer *\*build.\*xml* en el nostre projecte, podem invocar a *\*ant* proporcionant-li els diferents *\*targets*:

* Si desitgem compilar el projecte, des d'una terminal, farem:

**$ \*ant compile**

\*Buildfile: /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*build.\*xml

\*clean:

[\*delete] \*Deleting \*directory /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*classes

compile:

[\*mkdir] \*Created \*dir: /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*classes

[\*javac] \*Compiling 2 \*source \*files \*to /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*classes

\*BUILD \*SUCCESSFUL

Total estafe: 1 \*second

Com veiem en els missatges que ens retorna *\*ant*, en primer lloc es llança la tasca *\*clean*. Si és la primera vegada que ho executem, esta tasca no fa res, ja que no tenim res a netejar. Si ja existira la tasca *\*classes* d'altres compilacions, en este pas esborraria esta carpeta.

Una vegada feta la neteja, es fa la tasca de compilació, amb el que es crearà la carpeta *\*classes*, i després s'invoca al compilador de *\*java* per a generar els fitxers en \*bytecode (*.\*class*).

Després de la compilació, l'estructura del nostre projecte queda com seguix:

.

├── \*build.\*xml

├── \*classes

│ └── \*com

│ └── \*mgh

│ └── \*edd

│ ├── Hola.\*class

│ └── Saluda.\*class

└── \*src

└── \*com

└── \*mgh

└── \*edd

├── Hola.\*java

└── Saluda.\*java

Com podem veure, s'ha generat la carpeta *\*classes* indicada en el \*target *compile* del fitxer \*build.\*xml. Dins d'esta carpeta, s'ha replicat tota l'estructura de carpetes corresponent al paquet, però que conté els fitxers *.\*class*.

Si ara volem executar el projecte, farem:

**$ \*ant \*run -\*Darg0=Jose**

\*Buildfile: /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*build.\*xml

\*clean:

[\*delete] \*Deleting \*directory /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*classes

compile:

[\*mkdir] \*Created \*dir: /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*classes

[\*javac] \*Compiling 2 \*source \*files \*to /\*home/\*dam/\*edd/\*scv/\*ud3-ferramentes-\*automatizacion/\*saludaAnt/\*classes

\*run:

[\*java] Hola Jose

\*BUILD \*SUCCESSFUL

Total estafe: 1 \*second

Observeu que al comando \*ant \*run hem d'oferir-li els arguments mitjançant -\*Darg0=Jose . És a dir, afegim l'opció -D, seguida del nom d'argument usat en el *\*build.\*xml*.

A més, en este cas, com no hem llançat \*ant \*clean prèviament, la tasca *\*clean* si que ha esborrat la carpeta de classes, posteriorment realitza la compilació i finalment la \*ejeución. Si desitjàrem netejar tot el projecte, sí que realitzaríem ara el \*ant \*clean.

1. Integració de \*ant amb \*VSCode i Eclipsi

Com hem vist, \*ant pot usar-se de forma molt pràctica des de la línia de comandos. De totes maneres, podem usar esta ferramenta al costat d'editors com \*VSCode o \*IDEs com a Eclipsi.

1. \*Ant i \*VSCode

La forma més senzilla d'utilitzar \*Ant amb Visual \*Studio \*Code és mitjançant la terminal integrada. No obstant això, disposem de l'extensió *\*Ant \*Target \*Runner[[1]](#footnote-2),* que ens afig una nova vista en la barra lateral on se'ns mostren els diferents *\*targets*, i de forma niada, les dependències entre estos. A més, en esta vista, si polsem el botó dret del ratolí sobre un \*target d'esta vista, ens dona la possibilitat d'executar el \*target (*\*Run \*Ant \*Target*) o bé d'anar a la seua definició (*\*Reveal \*Definition).*

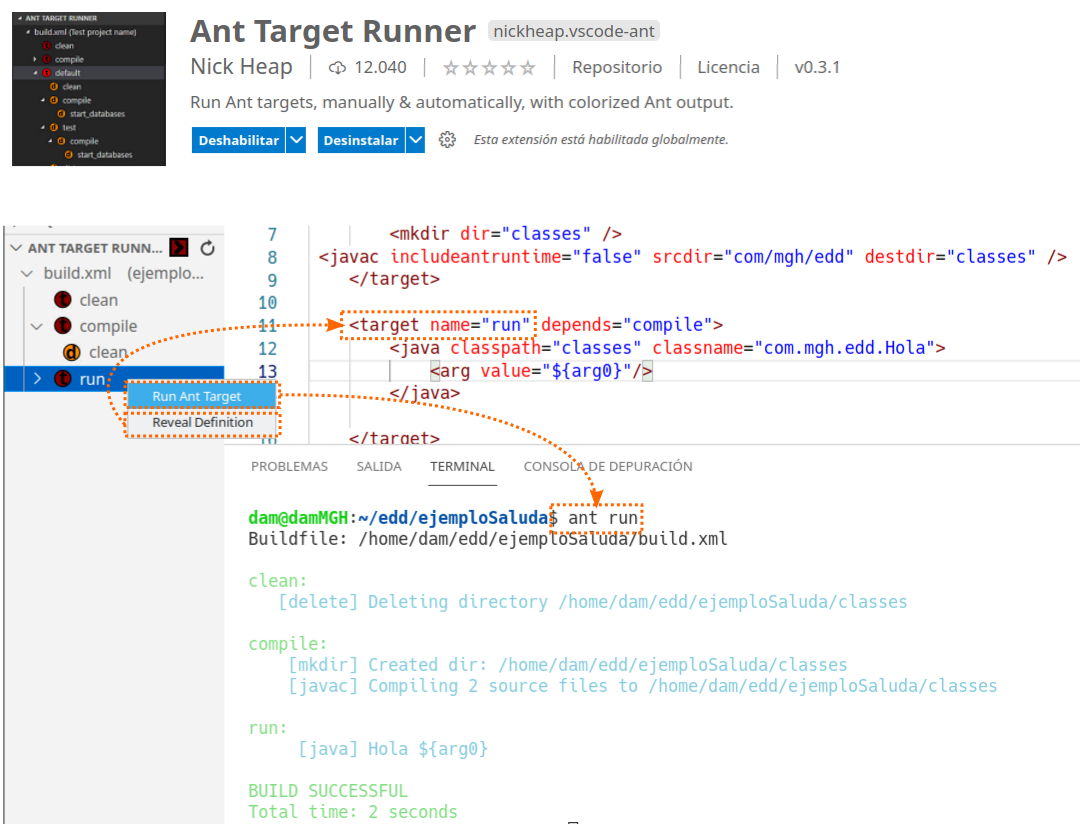


Fig. 3.22. Extensió \*Ant \*Target \*Runner. ( \*ant\_\*target\_\*runner.\*png)

1. \*Ant i Eclipsi

Eclipsi porta \*Ant integrat, i no requerix de la instal·lació de cap extensió ni configuració especial. Únicament, si desitgem canviar la versió de \*Ant per la qual tenim en l'equip podríem fer-lo mitjançant la finestra de preferències. No obstant això, la \*versió de \*Ant integrada en la versió d'Eclipsi que estem usant (1.10.9) és més recent que per exemple la que es troba per defecte en els repositoris de \*Ubuntu 20.04 (1.10.7).

Així doncs, quan estem treballant amb un projecte en \*Ant, Eclipsi ens permet navegar per ell mitjançant l'explorador de projectes. Si en este accedim al fitxer *\*build.\*xml*, en la finestra de resum (\*Outline) se'ns mostraran també els diferents *\*targets* especificats en ell.



Figure 3.23. Projecte \*Ant en Eclipsi. (\*ant\_eclipsi.\*png)

\*Dese l'explorador de projectes, podrem executar el nostre projecte amb \*Ant. Per a això, fem clic amb el botó dret del ratolí sobre el fitxer *\*build.\*xml*  per a obrir el menú contextual. Busquem l'opció *\*Run As*, i dins d'ella ens apareixen dos \*subopciones: “*\*Run As*” i “*\*Run As...*”. La primera executa \*directamete el projecte amb l'última configuració utilitzada, mentres que la segona, ens permet realitzar esta configuració abans de l'execució.

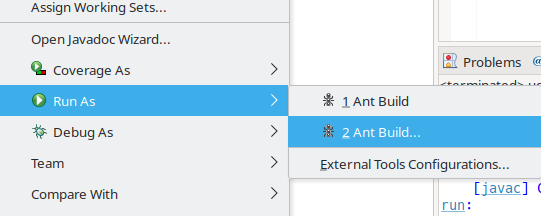


Figure 3.24. Execució mitjançant \*Ant (\*ant\_\*eclipse2.\*png)

Si triem “*\*Run As...”,* ens \*muesta una finestra des d'on es podrà triar tant el \*target a llançar com els arguments a utilitzar, entre moltes altres opcions.

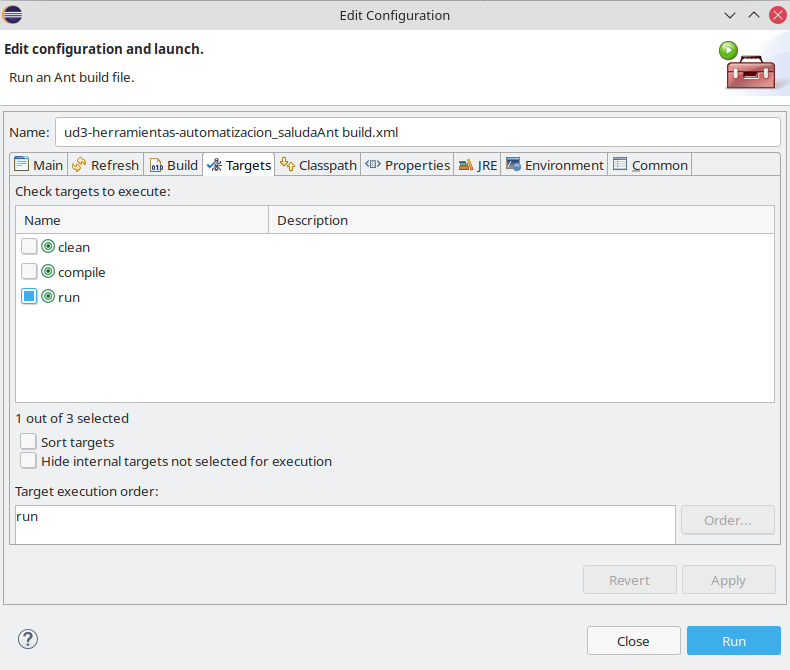


Figura 3.25. Execució mitjançant \*Ant (2). (\*ant\_\*eclipse3.\*png)



CAS PRÀCTIC 4.

\*Ant i Eclipsi

A partir del projecte *\*saludaAnt* del repositori amb els exemples, importa'l en Eclipsi i executa els diferents \*Targets. Afig-li també un argument a l'aplicació (per exemple, el teu nom), i observa com apareix en els resultats.

Solució

Per a importar el projecte utilitzem l'opció de menú *\*File* > *\*Open \*Pojects \*form \*File \*System , i* seleccionem la carpeta *\*saludaAnt* del repositori.

Una vegada obert, ho veurem com es descriu en la figura 3.23, i per a executar els diferents \*targes, farem clic amb el botó dret sobre el fitxer \*build.\*xml i seleccionarem *\*Run As* > *\*Run As...* En la finestra que se'ns obri, tenim diferents pestanyes. Les que ens interessen són *\*Targets*, on especifiquem què \*target llançarem, i la pestanya \*Main, que ens permet afegir arguments a l'execució.

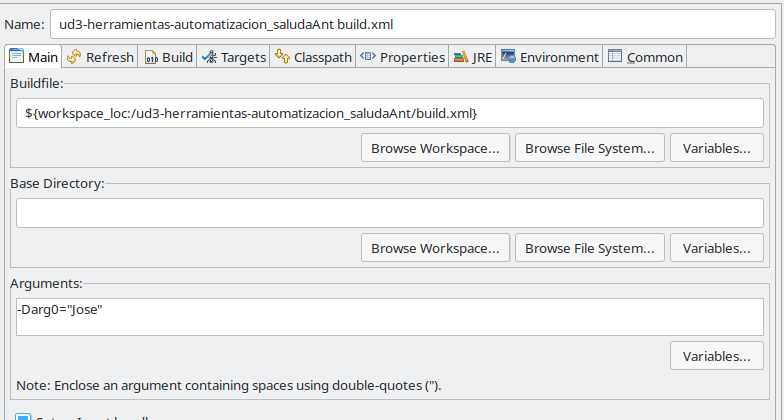


Figura 3.26. Afegint arguments a l'execució. (\*ant\_\*eclipse4.\*png)

POSA'T A PROVA 4

Generació del \*build.\*xml Eclipsi

Importa ara el projecte \*saludaBase, i explora les possibilitats que oferix Eclipsi per a la generació del fitxer *\*build.\*xml*. Genera este, observa el resultat i executa algun dels \*targets.

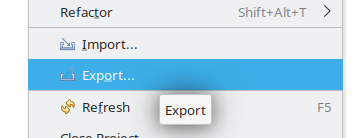
Claus de resolució

La importació es realitzarà de la mateixa manera que en el cas practique anterior. D'altra banda, per a generar el fitxer \*build.\*xml hauràs d'explorar les opcions d'Exportació que ens oferix Eclipsi.

Solució

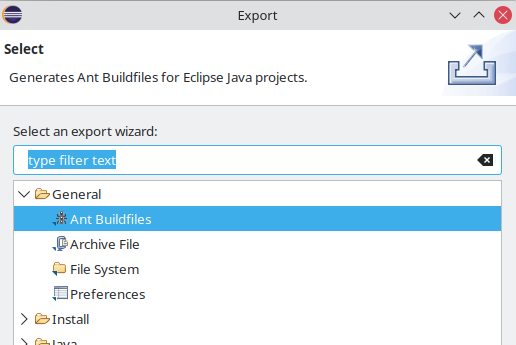
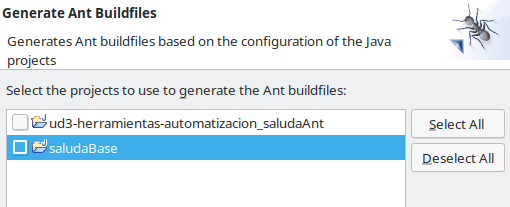
Importem el projecte amb *\*File > \*Open \*Pojects \*form \*File \*System.* Evidentment, este projecte no compta amb el fitxer \*build.\*xml, ja que únicament conté els fonts.

Per a generar el \*xml, fem clic amb el botó dret sobre el projecte (entrada \*saludaBase en l'arbre de projectes), i busquem l'opció *\*Export.*



(\*ant\_\*eclipse5.\*png)

Esta opció ens mostrarà un diàleg a partir del qual seleccionar l'assistent d'exportació. Seleccionarem “\*Ant \*Buildfiles” de la secció General, i en la següent finestra, triem el projecte \*saludaBase:

(\*ant\_\*eclipse6.\*png i \*ant\_\*eclipse7.\*png)

Això ens generarà un fitxer \*build.\*xml per al projecte, que podrem utilitzar com en el cas \*prático anterior. De totes maneres, si explorem este fitxer veurem que diferix bastant del que havíem generat manualment. Com podreu observar, ha generat este a partir d'una plantilla amb diversos \*targets més, i diverses opcions que no hem utilitzat. Haurem de tindre això en compte quan ens plantegem generar fitxers de construcció de manera automàtica, ja que poden incloure opcions que no utilitzem, o fins i tot que no desitgem en els nostres projectes.

Exercici 1:

**COMPROVA EL TEU APRENENTATGE**

Si tenim el fitxer Hola.\*java en el directori \*src/\*com/\*mgh/\*edd, i compilem des de la carpeta \*src, quina afirmació és correcta?

1. **La classe Hola estaria en el paquet \*com.\*mgh.\*edd, i els \*bytecodes es generarien en \*src/\*com/\*mgh/\*edd.**
2. La classe Hola estaria en el paquet src.com.\*mgh.\*edd, i els \*bytecodes es generarien en \*src/\*com/\*mgh/\*edd.
3. La classe Hola estaria en el paquet \*com.\*mgh.\*edd, i els \*bytecodes es generarien en \*classes/\*com/\*mgh/\*edd.
4. Donaria error de compilació error: \*cannot \*find \*symbol

Exercici 2

**COMPROVA EL TEU APRENENTATGE**

Donat el següent codi XML en el fitxer \*build.\*gradle:

<\*target \*name="\*run" \*depends="compile">

<\*property \*name="\*arg0" \*value=""/>

<\*java \*classpath="\*classes" \*classname="\*com.\*mgh.\*edd.Hola">

<\*arg \*value="${\*arg0}"/>

</\*java>

</\*target>

Indica quines afirmacions són veritables o falses:

1. Quan executem \*ant \*run s'executarà en primer lloc l'objectiu compile **(V)**
2. Si llancem \*ant \*run sense arguments, ens llançarà un error, ja que necessita un argument **(F)**
3. Els fitxers e \*bytecode es generaran en el mateix directori \*com/\*mgh/\*edd/ **(F)**
4. Els fitxers font es buscaran en el \*direcorio "\*classes" **(F)**

Exercici 3

**COMPROVA EL TEU APRENENTATGE**

Indica quines de les següents afirmacions són \*verdareas respecte a la integració de \*Ant amb \*VScode i Eclipsi:

a) \*VSCode únicament permet executar \*ant des de la terminal integrada **(F)**

b) Eclipsi permet generar automàticament el fitxer \*build.\*xml **(V)**

c) Per defecte, Eclipsi no utilitza el \*Ant del sistema, sinó un integrat en el propi \*IDE **(V)**

d) Els fitxers \*build.\*xml generats de manera automàtica poden introduir variables d'entorn i opcions que no anem a utilitzar **(V)**

Presentació: \*Maven i \*Gradle

\*Maven va sorgir en el si d'Apatxe, amb la finalitat de simplificar els processos de construcció en el Projecte Jakarta \*Turbine. Este projecte estava compost de diversos subprojectes, cadascun amb una estructura particular, i els seus respectius arxius de compilació de \*Ant, diferents entre ells. En este context, es va decidir desenrotllar una forma estàndard de construcció de projectes que oferira a més una forma senzilla de publicar les aplicacions. El resultat va ser \*Maven, que es va integrar inicialment a Jakarta, i posteriorment es va alliberar com a projecte de nivell superior de l'Apatxe \*Softwre \*Foundation.

Mitjançant \*Maven, podem doncs generar aplicacions que seguixen una estructura i un procés de construcció ben definits i coneguts, seguint a més bones pràctiques. No obstant això, a vegades este mecanisme és massa rígid, i resulta bastant complicat abordar el desenrotllament d'aplicacions que no seguisquen un model \*estándard. És #ací on apareix \*Gradle, aprofitant les bondats de \*Ant i \*Maven i evitant els seus defectes.

En el següent apartat veurem una xicoteta introducció als conceptes més rellevants de cadascuna d'estes ferramentes, i es plantejaran un parell de casos pràctics on es posaran en pràctica.

Apartat: \*Maven i \*Gradle

1. Apatxe \*Maven

Apatxe \*Maven és una ferramenta que ens permet crear i gestionar projectes Java, de manera similar a com féiem amb \*Ant, però des d'un punt de partida diferent. La idea subjacent d'Apatxe \*Maven és oferir suport al cicle de vida de construcció segons estàndards, de manera que les nostres aplicacions siguen més més homogènies i tinguen en compte bones pràctiques. Seguix el principi de *convenció sobre configuració (\*CoC),* segons el qual, s'establixen unes certes convencions que simplifiquen la presa de decisions per part del programador.

D'esta manera, es promou la construcció de solucions de programari homogènies, i que reduïxen la variabilitat.

Entre les diferents ajudes que oferix \*Maven trobem la construcció d'aplicacions, el manteniment de la documentació, la gestió de dependències, de versions i la distribució de l'aplicació.

Un dels conceptes que treballa \*Maven, i amb el qual ens trobarem llegint la seua documentació és el de *\*artifact*, la traducció literal del qual seria *artefacte*. Este concepte fa referència al que anomenarem una *solució*, o un producte programari reutilitzable, que és gestionat per \*Maven. Això implicarà que tinga una estructura concreta i seguisca unes certes convencions.

1. Arquetips

Com hem comentat, el principal objectiu de \*Maven és homogeneïtzar l'estructura de pràcticament qualsevol tipus d'aplicació. Per a això introduïx el concepte d'arquetip.

Un arquetip (\*archetype) en \*Maven és un model original a partir del qual creguem l'esquelet de solucions que seguixen el mateix patró, o són del mateix tipus.

Els arquetips ens brinden una forma consistent d'aplicar bones pràctiques. Mitjançant ells, podem tindre un projecte en funcionament de manera ràpida i amb una estructura ben definida i coneguda, amb el que a més es facilitarà el manteniment posterior i el treball en equip.

Entre els arquetips més rellevants podríem trobar *\*maven-\*archetype-\*quickstart*, per a generar un projecte de tipus *Hola Món*, *\*maven-\*archetype-simple*, per a generar un projecte simple, o bé *\*maven-\*archetype-\*webapp*, per a una aplicació web.

La creació d'un projecte basada en un arquetip la veurem de manera pràctica en el cas pràctic ampliat 5.

1. El cicle de vida de construcció

\*Maven introduïx el concepte de cicle de vida de construcció (que no hem de confondre amb el cicle de vida de desenrotllament). El cicle de vida de construcció es podria definir com:

Un procés ben definit per a la construcció i distribució de solucions (\*artifacts).

\*Maven distingix tres cicles de vida en el procés de construcció:

* **\*Default**: El cicle de vida per defecte que gestiona la construcció en si de la solució,
* **\*Clean**: El cicle de vida que s'encarrega de la neteja del projecte,
* **\*Site**: El cicle de vida per a la generació de documentació.

Cadascun d'estos cicle de vida es compon de diferents etapes, i en cadascuna d'elles es realitzen diferents accions. El cicle de vida més important és el *\*Default*, que es compon de 22 fases, encara que les més importants són:

|  |  |
| --- | --- |
| **Fase** | **Tasques** |
| \*validate | Comprova que l'estructura del projecte és correcta i tota la informació necessària es troba en el seu lloc. |
| compile | Compila els fonts, generant el \*bytecode corresponent. |
| test | Realitza proves unitàries sobre els fonts generats amb \*JUnit. Parlarem d'estes proves en la següent unitat. |
| \*package | Empaqueta tots els *.\*class* obtinguts en un format \*distribuible , mitjançant un arxiu \*JAR. |
| \*verify | Realitza les comprovacions corresponents a les proves d'integració, per a garantir criteris de qualitat. |
| \*install | Instal·la el paquet en un directori local que fa de repositori, de manera que puga usar-se en altres projectes de manera local. |
| \*deploy | Desplega el paquet generat en un repositori remot, de manera que puga usar-se amb altres desenrotlladors i projectes. |

Estes fases podran llançar-se de manera individual, però tenint en consideració que per a llançar una fase s'ha de passar per les anteriors. És a dir, si volem compilar el projecte (sense empaquetar ni desplegar), deurem primer passar per la fase de validació i després per la fase de compilació.

Quan desenrotllem aplicacions d'escriptori que no requerisquen la seua instal·lació en un servidor, serà suficient amb llançar la fase *\*verify*, per a realitzar la compilació, empaquetat i verificar la integració.

1. Complements (\*plugins) i objectius (\*goals)

Com hem comentat, cadascuna de les fases del cicle de vida fa diverses tasques, més conegudes com a *objectius*  o *\*goals*.

Els objectius (\*goals) representen perquè una tasca específica, que contribuïx a la construcció o gestió d'una solució, i s'organitzen en complements (\*plugins).

Encara que cada fase del cicle de vida es *componga*, de diversos objectius, pot ser que existisquen objectius no vinculats a cap fase del cicle de vida o vinculats a més d'una fase. Un objectiu no vinculat a cap fase pot invocar-se directament fora del cicle de vida, de manera que no passe per cap fase prèviament.

Vegem en la següent taula la relació d'algunes de les fases del cicle de vida per defecte i alguns objectius que es llancen en ella, al costat del complement en què estos es troben.

|  |  |  |
| --- | --- | --- |
| **Fase** | **Objectiu (\*Goal)** | **Complement (\*plugin)** |
| **compile** | \*compiler:compile | \*maven-\*compiler-\*plugin |
| ***test-compile*** | *\*compiler:\*testCompile* | *\*maven-\*compiler-\*plugin* |
| **test** | \*surefire:test | \*maven-\*surefire-\*plugin |
| **\*package** | \*jar:\*jar | \*maven-\*jar-\*plugin |
| **\*verify** | \*verifier:\*verify | \*maven-\*verifier-\*plugin |
| **\*install** | \*install:\*install | \*maven-\*install-\*plugin |
| **\*deploy** | \*deploy:\*deploy | \*maven-\*deploy-\*plugin |

Com podem apreciar, s'ha inclòs, a tall d'exemple, la fase test-compile, de la qual no hem parlat abans. Esta fase s'encarrega de compilar el codi font de les proves. Com veiem, l'objectiu *\*testCompile* que llança es troba en el complement *\*maven-\*compiler-\*plugin*, que també inclou l'objectiu *compile* de la fase de compilació. Així doncs, tenim un \*plugin que conté objectius que es llancen en dos fases diferents.

1. El \*POM

El Model d'Objectes del Projecte (\*Project \*Object \*Model, \*POM) és un fitxer XML que descriu la configuració del projecte en \*Maven, i proporciona la major part d'informació necessària per a la seua construcció. Pot arribar a ser un arxiu llarg i complex, però no és necessari entendre tot el seu contingut per a traure tota l'efectivitat de \*Maven.

El model conté valors predeterminats per a la majoria de projectes, com poden ser el directori de compilació (*\*target*), el directori dels fonts (*\*src/\*main/\*java*), el directori dels fonts de les \*pruegas (*\*src/test/\*java*), etc.

Quan es llança un objectiu, \*Maven llig este fitxer per a obtindre la informació necessària i després executar l'objectiu.

A més dels valors per defecte, podem especificar les dependències del projecte, especificar objectius, perfils de compilació, o informació com el nom, l'organització, la descripció o la versió entre altres.

\*Maven proporciona molts valors per defecte a través del que es coneix com el *\*Super \*POM*. Estos valors per defecte es combinaran amb els valors definits en el fitxer en l'arrel del projecte *\*pom.\*xml* per a obtindre el que es coneix com el \*POM efectiu, que és el que s'aplicarà al projecte en qüestió.

En este punt pots fer el cas pràctic ampliat 5, on es tractarà \*Maven de manera pràctica, i s'aprofundirà en estos continguts.

Cas pràctic ampliat

1. \*Gradle

Com hem vist, \*Ant és una ferramenta molt versàtil, però que descarrega molta responsabilitat en el desenrotllador. En canvi, \*Maven oferix mitjançant els arquetips una estructura pràcticament predefinida d'aplicació, donant suport a tot el cicle de vida de construcció. Això pot ser que siga un inconvenient quan el nostre projecte no s'ajusta a cap arquetip.

En este context, en 2008 va aparéixer \*Gradle, que pretén integrar el millor de cada ferramenta: les tasques i facilitat de personalització de \*Ant, i la potència, el suport al cicle vida de construcció i les proves de \*Maven.

Igual que \*Maven, \*Gradle també seguix el paradigma de convenció sobre configuració, on totes les opcions de configuració tindran valors per defecte comuns o útils, de manera que únicament hàgem de modificar-los en casos especials.

Una altra diferència important respecte a \*Ant i \*Maven és que \*Gradle no utilitza el format XML en la seua configuració, sinó que els \*scripts de compilació s'expressen mitjançant un llenguatge específic de domini (*\*Domain \*Specific \*Language, o \*Select \*build \*script \*DSL:*

*1: \*Groovy*

*2: \*Kotlin*

*\*Enter \*selection (\*default: \*Groovy) [1..2] 1*) basat en \*Groovy o \*Kotlin.

A més, \*Gradle permet utilitzar altres llenguatges a part de Java, com a C++, Scala, \*Swift o \*Kotlin, i és el sistema de compilació utilitzat en projectes Android.

1. Projectes en \*Gradle

Un projecte es compon d'un conjunt de recursos ben organitzats. Estos recursos inclouran, entre altres el codi font i la lògica per a la seua construcció.

\*Gradle a més, permet que un \*poyecto continga subprojectes (multiprojecte), de manera que tots ells formen part en conjunt de l'aplicació. Per exemple, un projecte d'una aplicació de missatgeria que estiguera composta per dos projectes: un per a l'aplicació client, i un altre per al servidor.

A més, els projectes es registraran en un fitxer anomenat \*settings.\*gradle (o \*settings.\*gradle.\*kts), i els subprojectes o submòduls tindran els seus respectius fitxers \*build.\*gradle (o \*build.\*gradle.\*kts).

1. Tasques en \*Gradle

Una tasca representa una acció, bé siga predeterminada o personalitzada, que s'executarà durant el procés de construcció dels nostres projectes.

El \*script de construcció del projecte (\*build.\*gradle) permet definir tasques, així com dependències entre elles. Una dependència entre tasques significa que per a fer una tasca, és necessari que s'haja executat una altra prèviament. Per exemple, per a poder llançar la tasca d'executar l'aplicació, prèviament ha d'haver-se llançat la tasca de compilació.

Durant la construcció, \*Gradle garantirà que estes tasques s'executen de manera única i ordenada. És a dir, cada tasca s'executarà una vegada, i totes les tasques de les quals esta depén s'hauran executat prèviament.

1. Cicle de vida en \*Gradle

El cicle de vida de construcció descriu les fases per les quals passem per a \*constuir un projecte. En \*Gradle, este cicle consta únicament de tres fases: inicialització, configuració execució.

* **Inicialització:** \*Gradle determina quins projectes estaran involucrats en el procés de construcció. Per a això, examina el fitxer del projecte *\*settings.\*gradle*, a la recerca dels subprojectes i dels seus respectius fitxers de construcció *\*build.\*gradle.* En finalitzar esta fase, es crea una instància de projecte (\*org.\*gradle.\*api.\*Project) per a cadascun d'ells.
* **Configuració**: S'analitzen els \*scripts de construcció dels projectes identificats en la inicialització, i les diferents tasques que componen cadascun. A partir d'esta anàlisi, \*Gradle genera un Graf Acíclic Dirigit (\*GAD) de tasques. Un *graf* és la representació d'un conjunt d'elements (en esta case les tasques), i la relació entre elles (en este cas la relació de dependència). El fet que siga dirigit significa que la relació és unidireccional, en el sentit de la dependència, i el que siga acíclic indica que no es produïxen bucles en el graf. Amb això assegurem que les tasques s'executaran en l'orde corresponent i solament una vegada.

Vegem, a tall d'exemple el següent grup de tasques i les dependències entre elles, i el grau acíclic dirigit que formen. Com podeu observar, les dependències van en una única direcció, i les tasques \*sólamente s'executen una vegada, encara que depenguen de diverses tasques posteriors.

|  |  |
| --- | --- |
| **tasca** *Compilar* { }  **tasca** *\*TestUnitario*{  **\*dependeDe** Compilar  }  **tasca** *Empaquetar*{  **\*dependeDe** Compilar  }  **tasca** *Construir*{  **\*dependeDe** \*TestUnitario, Empaquetar  } |  |

* **Execució**: En esta fase es llancen les diferents tasques segons l'orde de dependència generat en el graf de tasques que va obtindre en la fase anterior. En este punt és on realment es realitza tot el treball de construcció: compilació, còpia de recursos, neteja del projecte, generar empaquetats, etcètera.

En este punt pots fer el cas pràctic ampliat 6, on es tractarà \*Gradle de manera pràctica, i s'aprofundirà en estos continguts.

Cas pràctic ampliat



\*saludaMaven

Seguint els passos descrits en el cas pràctic ampliat 5, crea un nou projecte \*Maven utilitzant els fitxers proporcionats en l'exemple \*saludaBase (*Hola.\*java* i *Saluda.\*java*), que tens disponible en \*Github ([https://git.io/jcari](https://git.io/JcarI)).

\*Raliza també la compilació i l'empaquetat, comprovant que tot funciona correctament.

Solució

Per a generar el projecte ens situem en la carpeta pare on desitgem crear-lo, i utilitzem la instrucció \*mvn \*archetype:\*generate del següent mode:

$ **\*mvn \*archetype:\*generate -\*DgroupId=\*com.\*mgh.\*edd -\*DartifactId=\*saludaMaven \ -\*DarchetypeArtifactId=\*maven-\*archetype-\*quickstart       \  
 -\*DarchetypeVersion=1.4 -\*DinteractiveMode=\*false** 

Una vegada creat, incorporem els fitxers Hola.\*java i Saluda.\*java en la carpeta dels fonts:

$ **\*cp {Carpeta de \*SaludaBase}/\*saludaBase/\*src/\*com/\*mgh/\*edd/\*.\*java**

**\*saludaMaven/\*src/\*main/\*java/\*com/\*mgh/\*edd/** 

I eliminem el fitxer App.\*java que ja no necessitem:

$ **\*rm \*saludaMaven/\*src/\*main/\*java/\*com/\*mgh/\*edd/App.\*java** 

El contingut del directori de fonts quedaria doncs:

\*saludaMaven/\*src/\*main/\*java/

└── \*com

   └── \*mgh

       └── \*edd

           ├── Hola.\*java

           └── Saluda.\*java

**[Continua fora del requadre]**

CAS PRACTIQUE 5

Ara ens quedaria modificar el \*pom.\*xml perquè quan es genere l'empaquetat sàpia quina classe ha d'executar. Per a això, busquem la configuració del \*plugin \*maven-\*jar-\*plugin, i afegim la configuració:

<\*plugin>

<\*artifactId>\*maven-\*jar-\*plugin</\*artifactId>

<\*version>3.0.2</\*version>

**<\*configuration>**

**<arxive>**

**<\*manifest>**

**<\*addClasspath>\*true</\*addClasspath>**

**<\*mainClass>\*com.\*mgh.\*edd.App</\*mainClass>**

**</\*manifest>**

**</arxive>**

**</\*configuration>**

</\*plugin>

Amb això ja podrem realitzar la compilació :

$ **\*mvn compile** 

Executar-ho, amb o sense arguments:

$ **\*java -\*cp \*target/\*classes \*com.\*mgh.\*edd.Hola Jose** 

Hola Jose

Empaquetar-ho:

$ **\*mvn \*package** 

I llançar l'empaquetat:

$ **\*java -\*jar \*target/\*saludaMaven-1.0-\*SNAPSHOT.\*jar Jose** 

\*saludaGradle

Seguint els passos descrits en el cas pràctic ampliat 6, crea un nou projecte \*Gradle utilitzant els fitxers proporcionats en l'exemple \*saludaBase (*Hola.\*java* i *Saluda.\*java*), que tens disponible en \*Github ([https://git.io/jcari](https://git.io/JcarI)).

Claus de resolució

Per a resoldre amb èxit l'exercici, seguix pas a pas el cas pràctic ampliat 6, i una vegada finalitzat, afig els nous fonts i comprova que tot estiga correcte en el fitxer *\*build.\*gradle*.

Solució

[S'Incorpora després del requadre]

POSA'T A PROVA 5

Per a crear el projecte, el farem mitjançant \*gradle \*init i seguirem l'assistent:

$ **\*gradle \*init** 

\*Starting a \*Gradle \*Daemon, 2 incompatible \*and 1 \*stopped \*Daemons \*could \*not be \*reused, use --estatus \*for \*details

\*Select \*type \*of \*project \*to \*generate:

1: \*basic

2: \*application

3: \*library

4: \*Gradle \*plugin

\*Enter \*selection (\*default: \*basic) [1..4] **2** 

\*Select \*implementation \*language:

1: C++

2: \*Groovy

3: Java

4: \*Kotlin

5: Scala

6: \*Swift

\*Enter \*selection (\*default: Java) [1..6] **3** 

\*Split \*functionality \*across \*multiple \*subprojects?:

1: no - \*only \*one \*application \*project

2: \*yes - \*application \*and \*library \*projects

\*Enter \*selection (\*default: no - \*only \*one \*application \*project) [1..2] **1** 

\*Select \*build \*script \*DSL:

1: \*Groovy

2: \*Kotlin

\*Enter \*selection (\*default: \*Groovy) [1..2] **1** 

\*Select test \*framework:

1: \*JUnit 4

2: \*TestNG

3: \*Spock

4: \*JUnit \*Jupiter

\*Enter \*selection (\*default: \*JUnit \*Jupiter) [1..4] **1** 

\*Project \*name (\*default: \*saludaGradle): 

\*Source \*package (\*default: \*saludaGradle): **\*com.\*mgh.\*edd** 

> \*Task :\*init

...

Amb això ja tenim el projecte creat. Ara copiarem en ell els fonts de \*SaludaBase, i a eliminar el fitxer App.\*java creat per defecte:

$ **\*cp ../\*saludaBase/\*src/\*com/\*mgh/\*edd/\*.\*java app/\*src/\*main/\*java/\*com/\*mgh/\*edd/** 

$ **\*rm app/\*src/\*main/\*java/\*com/\*mgh/\*edd/App.\*java** 

Com el mètode \*main ha canviat d'ubicació (estava en *App.\*java* i ara està en *Hola.\*java*), haurem de modificar el valor de *\*mainClass*, dins de *\*application* del fitxer *app/\*build.\*gradle*:

\*application {

// Definix \*the \*main \*class \*for \*the \*application.

   \*mainClass = '**\*com.\*mgh.\*edd.Hola**'

}

Finalment, si volem construir-ho o executar-ho, ho faríem mitjançant *\*gradle \*build* i *\*gradle \*run* respectivament, o bé utilitzant el \*wrapper de \*Graddle *./\*gradlew \*build* i *./\*gradlew \*run*.

$ **\*gradle \*run** 

> \*Task :app:\*run

Hola

...

**Afegint arguments**

Si intentem incorporar arguments a la línia d'execució del \*gradle  *\*run*, veurem que ens llança un error, ja que el comando que estem executant és *\*gradle*, i no *\*java*. Per a passar els arguments a la nostra aplicació \*java, a través de \*Gradle haurem d'indicar-los amb l'argument *—\*args* de \*Gradle:

$ **\*gradle \*run --\*args="Jose"** 

> \*Task :app:\*run

Hola Jose

...

**Nota sobre els tests**

És possible que durant la construcció amb *\*gradle \*build* ens done error en les classes de tests, ja que ara, la classe *App* no existix. Atés que encara no hem vist la unitat dedicada a això, podem directament esborrar la carpeta *app/\*src/test/*, i evitar estes fallades.

Exercici 1:

**COMPROVA EL TEU APRENENTATGE**

Si llancem en \*Maven la fase *\*verify*, indica quines fases del cicle de vida de construcció es realitzaran.

1. \*Verify
2. compile, \*package, \*verify
3. **\*validate, compile, test, \*package, \*verify**
4. \*validate, compile test, \*package, \*verify, \*install, \*deploy

Exercici 2:

Indica si les següents afirmacions són unes certes o falses sobre \*Maven.

1. Un arquetip és un model a partir del qual generem solucions que seguixen el mateix patró o són del mateix tipus. **(V)**
2. Un artefacte en \*Maven és una solució o projecte, que seguix una estructura i cicle de vida de construcció definit pel programador. **(F)**
3. El cicle de vida de construcció es definix com un procés ben definit per a la construcció i distribució de solucions **(V)**
4. Un complement en \*Maven conté exclusivament objectius corresponents a una fase del cicle de vida **(F)**

Exercici 3:

Indica quines tasques es realitzen en cadascuna de les següents fases del cicle de vida de \*Gradle:

1. Realitza la compilació, còpia de recursos, neteja o generació d'empaquetats.
2. Examina les dependències entre tasques.
3. Llança les tasques especificades en els subprojectes
4. Genera una instància de \*Project per a cada projecte que participa en la construcció.
5. Genera un \*GAD de tasques.
6. Determina els projectes que participaran en la construcció.

|  |  |  |
| --- | --- | --- |
| Inicialització | Configuració | Execució |
| F, D | B, E | C, A |

1. Quina de les següents afirmacions respecte als sistemes de control de versions és **FALSA**?

**AVALUACIÓ FINAL**

Seleccione una:

* 1. Són sistemes que registren els canvis produïts en un conjunt de fitxers al llarg del temps
  2. Permeten en qualsevol moment tornar a una versió anterior
  3. Permeten veure qui ha fet cada canvi
  4. **Són especialment potents quan els fitxers s'emmagatzemen en mode binari.**

1. Quins tipus de sistemes de control de versions existixen?

Seleccione una:

* 1. **\*SCV Local, \*SCV Centralitzat i \*SCV Distribuït**
  2. \*SCV Local, \*SCV en xarxa, \*SCV \*distirbuido
  3. \*SCV Local, \*SCV client-servidor, \*SCV en xarxa
  4. \*RCS, \*Subversion i \*Git

1. Quina de les següents característiques **NO** és una certa respecte al \*SCV \*git?

Seleccione una:

* 1. Utilitza una còpia completa de l'historial de desenrotllament,
  2. Permet treballar de manera local
  3. Permet treballar amb branques
  4. **Per a les diferents versions, es guarden les diferències existents entre els fitxers.**

1. En quins estats pot estar un fitxer que es troba sota seguiment (\*tracked) en \*git?

Seleccione una:

* 1. **Confirmat, Modificat i Preparat**
  2. Confirmat, Denegat i Llest
  3. Amb canvis, Sense canvis i \*unstagged.
  4. Confirmat, modificat i sense \*segumiento.

1. Indica quina de les següents afirmacions sobre els repositoris remots amb \*Git és certa.

Seleccione una:

* 1. El repositori complet únicament es troba en un servidor.
  2. **Existix un servidor central, de tipus descobert (\*bare) que manté el repositori principal**
  3. No es necessita cap mena de servidor per a treballar en remot
  4. El servidor de tipus \*bare disposa d'un directori ocult anomenat .\*git amb tota la informació del repositori

1. Quines diferències hi ha entre les operacions \*git \*FETCH i \*PULL?

Seleccione una:

* 1. \*FETCH \*envia els canvis al servidor, \*mienstras que \*PULL els rep del servidor per a sincronitzar-los en local
  2. \*PULL \*envia els canvis al servidor, \*mienstras que \*FETCH els rep del servidor per a sincronitzar-los en local
  3. **Tots dos sincronitzen el repositori local amb el del servidor, però únicament \*PULL actualitza el contingut del directori de treball.**
  4. Són operacions equivalents.

1. Indica quina relació hi ha a Java entre un paquet (\*package) i l'estructura de directoris.

Seleccione una:

* 1. **L'estructura de directoris en què s'emmagatzema un paquet ha de ser exactament igual que el nom del paquet.**
  2. Un paquet és una organització lògica del codi, servix per a poder tindre classes en diferents paquets amb el mateix nom, i no té relació amb l'estructura de directoris
  3. Una directori pot contindre diversos paquets.
  4. Un mateix paquet pot estar en diversos directoris.

1. Indica quina afirmació respecte a \*ant és **FALSA**::

Seleccione una:

* 1. Un projecte en \*Ant es basa en un fitxer de construcció (habitualment anomenat \*build.\*xml)
  2. El fitxer de construcció conté diferents objectius, corresponents a les diferents fases del projecte.
  3. Els objectius o \*targets poden dependre d'altres objectius
  4. **Quan es llança \*ant, es genera un graf acíclic dirigit per a determinar l'orde d'execució de cada objectiu.**

1. Quina de les següents afirmacions respecte al \*POM és **FALSA**?

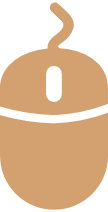
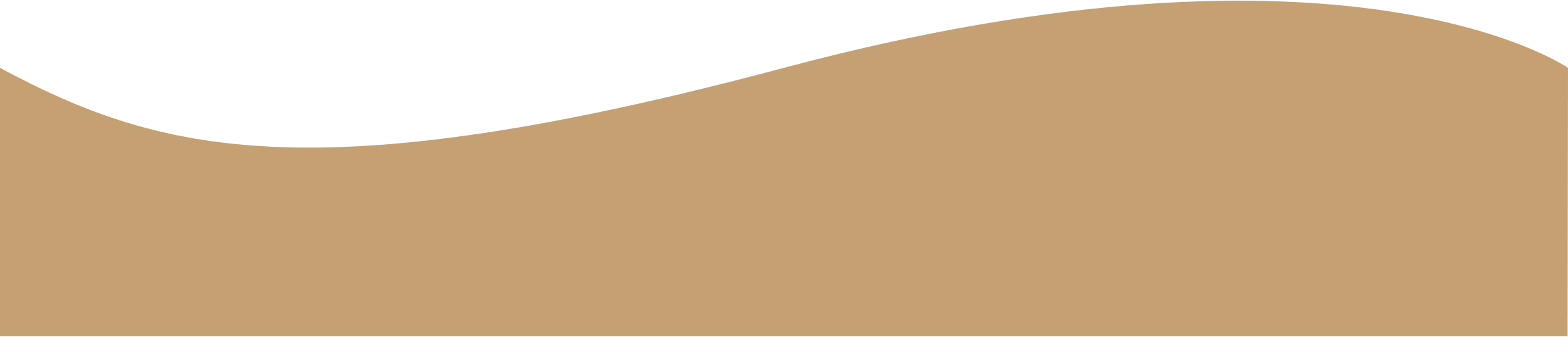
Seleccione una:

* 1. El \*POM és un fitxer XML que descriu la configuració d'un projecte en \*Maven
  2. **El programador ha d'incloure obligatòriament alguns valors necessaris com el directori de compilació o el dels fonts**
  3. El \*POM efectiu és el resultat de combinar els valors del \*Super \*Pom amb el \*POM del projecte
  4. \*Maven utilitza el \*POM cada vegada que es llança un objectiu per a obtindre informació necessària per a l'execució.

1. Per a què genera \*Gradle un \*Grafi Acíclic Dirigit (\*GAD)?

Seleccione una:

* 1. Per a comprovar que no existisquen dependències recíproques entre tasques i ens permeta corregir-les.
  2. **Per a assegurar-se que les tasques s'executen en l'orde corresponent segons les seues dependències i que estes s'executen \*sólamente una vegada.**
  3. Per a assegurar-se que totes les tasques s'executen tantes vegades com siguen invocades com dependències.
  4. Per a determinar durant la fase d'inicialització quins projectes estaran involucrats en el procés de construcció.

**Cas pràctic ampliat 5.**

3

**UNITAT**

**\*Maven.**

\*Maven

En l'apartat sobre \*Maven i \*Gradle de la unitat ja hem comentat els aspectes més rellevants d'estes ferramentes. En este cas pràctic, desenrotllarem de manera pràctica els conceptes relacionats amb \*Maven que es tracten.

Recordem que \*Maven és una ferramenta per a la construcció d'aplicacions, la seua documentació, gestió de dependències, versions i distribució de l'aplicació, donant suport a tot el cicle de desenrotllament, concorde a estàndards i bones pràctiques.

* Instal·lació en \*Ubuntu

En este cas pràctic utilitzarem \*Ubuntu 20.04 (\*LTS), o una distribució derivada per a treballar amb \*Maven.

El primer que haurem de fer és instal·lar la ferramenta, ja que per defecte, no ve instal·lada en el sistema. Per a això, des de la línia de comandos, actualitzem la llista de paquets, i instal·lem el paquet *\*maven*.

**Pas 1. Actualitzar la llista de paquets**

$ sue \*apt \*update 

**Pas 2. Instal·lar el paquet \*maven**

$ sue \*apt \*install \*maven 

...

S'instal·laran els següents paquets NOUS:

 \*maven

...

Preparant per a desempaquetar .../arxives/\*maven\_3.6.3-1\_\*all.\*deb ...

Desempaquetant \*maven (3.6.3-1) ...

Configurant \*maven (3.6.3-1) ...

Com podem veure, en este cas, s'instal·la la versió 3.6.3-1. És possible que en el moment en què realitzes esta activitat esta versió siga superior. També és possible que arrossegue algunes dependències que no estigueren prèviament instal·lades.

**Pas 3. Comprovació**

Per a comprovar si la instal·lació s'ha realitzat correctament, podem consultar la versió actual de \*maven, amb la següent orde:

\*mvn --\*version

**Apatxe \*Maven 3.6.3**

\*Maven \*home: /\*usr/\*share/\*maven

Java \*version: 11.0.11, \*vendor: \*Ubuntu, \*runtime: /\*usr/\*lib/\*jvm/\*java-11-\*openjdk-\*amd64

\*Default \*locale: és\_ÉS, \*platform \*encoding: \*UTF-8

US \*name: "\*linux", \*version: "5.4.0-73-\*generic", \*arch: "\*amd64", \*family: "\*unix"

Com veiem, ens oferix, a part d'informació sobre la versió, la ubicació de les instal·lacions de \*Maven i la màquina virtual de Java, així com informació genèrica sobre el sistema i la configuració de l'idioma.

**Instal·lació de \*Maven en Windows**

En l'Annex I d'este document trobaràs els passos per a realitzar la instal·lació i configuració de \*Maven en Windows 10. Una vegada instal·lat, podràs continuar amb ell a partir del següent punt.

important

* Creació del projecte

Per a crear un projecte en \*Maven hem d'utilitzar el mecanisme d'arquetips que ens oferix. Per a això, utilitzarem l'orde *\*mvn \*archetype:\*generate*, la qual admet un mode interactiu i un altre no interactiu. El mode interactiu el tractarem posteriorment. En este apartat veurem el mode no interactiu, en el qual li oferirem al comando de generació del projecte algunes opcions, i deixarem unes altres amb les seues opcions per defecte.

**Pas 1. Creació del projecte a partir de l'arquetip**

Per a crear el projecte amb \*Maven, a partir d'un arquetip, ens situem en la carpeta on desitgem crear la carpeta amb el projecte, i llancem la següent orde:

\*mvn \*archetype:\*generate -\*DgroupId=\*com.\*mgh.\*edd -\*DartifactId=\*saludaMaven

-\*DarchetypeArtifactId=\*maven-\*archetype-\*quickstart

-\*DarchetypeVersion=1.4 -\*DinteractiveMode=\*false 

Vegem les diferents opcions:

* **\*Archetype: \*generate**: Indiquem a \*maven que volem executar l'objectiu (*\*goal*) ***\*generate*** del \*plugin ***\*Archetype***. Recordem que en \*Maven, els *\*goals*  són equiparables als *\*tasks* de \*Ant. Este objectiu en concret, ens genera un projecte simple basat en un arquetip. Podem dir que un *\*plugin* és una col·lecció d'objectius amb un propòsit comú.
* **\*DgroupId:** Indiquem l'identificador únic de l'organització que crea el projecte, basat normalment en el domini completament qualificat de l'organització (*\*fully \*qualified \*domain*), en este cas *\*com.\*mgh.\*edd*.
* **\*DartifactID**: Indica el nom del recurs (*\*artifact*) que generarem. També ha de ser un nom únic per al projecte. En este cas, *\*saludaMaven.*
* **\*DarchetypeArtifactId**: Indica el nom del recurs d'arquetip a partir del qual triar la plantilla per al nostre projecte. En este cas, indiquem el recurs *\*maven-\*archetipe-\*quickstart*, que generarà una aplicació senzilla i ràpida, del tipus *Hola Món*.
* **\*DinteractiveMode**: Indica si volem utilitzar o no el mode interactiu. En este cas, hem indicat *\*false*, per a agilitzar la tasca, i que uns certs paràmetres prenguen els seus valors per defecte.

Amb tot això, l'eixida de l'orde anterior (i després de descarregar algun programari addicional ...) és la següent:

[\*INFO] \*Generating \*project \*in \*Batch \*mode

[\*INFO] ----------------------------------------------------------------------------

[\*INFO] \*Using \*following \*parameters \*for \*creating \*project \*from \*Archetype: **\*maven-\*archetype-\*quickstart:1.4**

[\*INFO] ----------------------------------------------------------------------------

[\*INFO] \*Parameter: **\*groupId**, \*Value: **\*com.\*mgh.\*edd**

[\*INFO] \*Parameter: **\*artifactId**, \*Value: **\*saludaMaven**

[\*INFO] \*Parameter: **\*version**, \*Value: **1.0-\*SNAPSHOT**

[\*INFO] \*Parameter: **\*package**, \*Value: **\*com.\*mgh.\*edd**

[\*INFO] \*Parameter: **\*packageInPathFormat**, \*Value: **\*com/\*mgh/\*edd**

[\*INFO] \*Parameter: **\*package**, \*Value: **\*com.\*mgh.\*edd**

[\*INFO] \*Parameter: **\*groupId**, \*Value: **\*com.\*mgh.\*edd**

[\*INFO] \*Parameter: **\*artifactId**, \*Value: **\*saludaMaven**

[\*INFO] \*Parameter: **\*version**, \*Value: **1.0-\*SNAPSHOT**

[\*INFO] \*Project \*created \*from \*Archetype \*in \*dir: .../\*saludaMaven

[\*INFO] ------------------------------------------------------------------------

[\*INFO] \*BUILD \*SUCCESS

[\*INFO] ------------------------------------------------------------------------

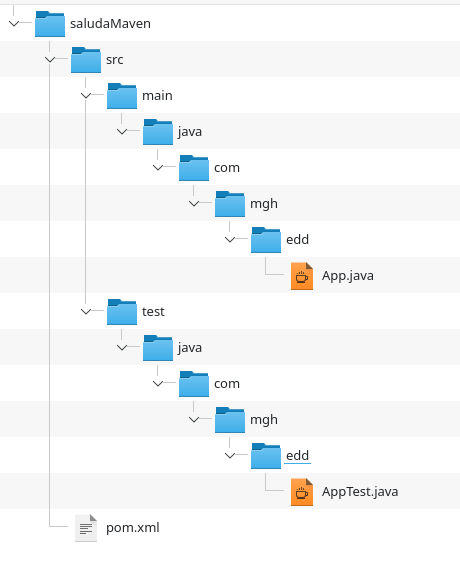
[\*INFO] Total estafe:  5.150 s

[\*INFO] \*Finished \*at: 2021-08-18T08:46:20+02.00

[\*INFO] ------------------------------------------------------------------------

Com podrem veure, ens mostra algunes alertes i missatges informatius, amb els valors que ha establit per a alguns paràmetres. A més, es realitzen algunes descàrregues des de *repo.maven.apache.org*. Esta direcció és el repositori de \*Maven, i conté totes les plantilles i ferramentes que podem utilitzar amb ella.

Vegem què ens ha generat esta orde:



Com veiem, s'ha creat la carpeta del projecte *\*saludaMaven* amb el fitxer *\*pom.\*xml*, que descriu el projecte segons el *\*Project \*Ojbect Model (\*POM)*. Dins d'esta carpeta tenim la carpeta *\*src*, amb els arxius font i de proves, degudament organitzats en carpetes segons el nom de domini completament qualificat.

* Analitzant el \*POM

Com ja s'ha comentat en la part teòrica, el fitxer *\*pom.\*xml* descriu la configuració del projecte en \*Maven, i proporciona la major part d'informació necessària per a la seua construcció. Pot arribar a ser un arxiu llarg i complex, però no és necessari entendre tot el seu contingut per a traure tota l'efectivitat de *\*Maven*.

Vegem el contingut del nostre arxiu *\*pom.\*xml*:

<?\*xml \*version="1.0" \*encoding="\*UTF-8"?>

<\*project \*xmlns="http://maven.apache.org/pom/4.0.0" \*xmlns:\*xsi="http://www.w3.org/2001/xmlschema-instance"

\*xsi:\*schemaLocation="http://maven.apache.org/pom/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<\*modelVersion>4.0.0</\*modelVersion>

<\*groupId>\*com.\*mgh.\*edd</\*groupId>

<\*artifactId>\*saludaMaven</\*artifactId>

<\*version>1.0-\*SNAPSHOT</\*version>

<\*name>\*saludaMaven</\*name>

<!-- \*FIXME \*change \*it \*to \*the \*project's website -->

<\*url>http://www.example.com</\*url>

<\*properties>

<\*project.\*build.\*sourceEncoding>\*UTF-8</\*project.\*build.\*sourceEncoding>

<\*maven.\*compiler.\*source>1.7</\*maven.\*compiler.\*source>

<\*maven.\*compiler.\*target>1.7</\*maven.\*compiler.\*target>

</\*properties>

<\*dependencies>

<\*dependency>

<\*groupId>\*junit</\*groupId>

<\*artifactId>\*junit</\*artifactId>

<\*version>4.11</\*version>

<\*scope>test</\*scope>

</\*dependency>

</\*dependencies>

<\*build>

<\*pluginManagement><!-- \*lock \*down \*plugins \*versions \*to \*avoid \*using \*Maven \*defaults (\*may be moveu \*to \*parent \*pom) -->

<\*plugins>

<\*plugin>...</\*plugin>

...

</\*plugins>

</\*pluginManagement>

</\*build>

</\*project>

I l a descripció d'alguns elements:

* **\*project**: És l'etiqueta arrel del document. Consistix en una capçalera XML, on es definix el tipus de document a través del seu espai de noms.
* **\*modelVersion**: Indica la versió de \*POM que es descriu el document.
* **\*groupId, \*artifactId i \*name**: fan referència a l'identificador de l'organització (*\*groupId*), al nom de la solució (*\*artifactId*) i al de l'aplicació (*\*name*)
* **\*version**: Indica la versió de l'aplicació. El sufix \*SNAPSHOT, indica que es tracta de l'últim codi en una branca de desenrotllament, la qual cosa no garantix l'estabilitat d'este. Generalment, s'usa quan estem en fase de desenrotllament i proves, fins que alliberem una *\*release* estable.
* **\*Properties**: Establix algunes propietats per al projecte, com ara:
  + ***\*project.\*build.\*sourceEncoding:*** Especifica la codificació dels arxius font, en este cas \*UTF-8.
  + ***\*maven.\*compiler.\*source*** i ***\*maven.\*compiler.\*target***: Especifica les propietats parell \*ael compilador, indicant que tant els fonts com el \*bytecode es correspondrà com a mínim amb la plataforma Java 1.7. Això es deu al fet que \*Maven ja no suporta versions de Java anteriors a la 1.6.
* **\*Dependencies**: Indica les dependències que té la nostra aplicació. Com veiem, ho fa indicant per a cadascuna el seu identificador d'organització (*\*groupId*), de recurs (*\*artifactId*), i la versió. A més, especifica l'àmbit (*\*scope*) en què s'aplica la dependència. En este cas, l'arquetip a partir del que hem generat la nostra aplicació incorpora la dependència de \*JUnit per a realitzar els tests.
* Secció **\*build** i **\*plugins**: En esta secció es definixen alguns paràmetres de la construcció i els \*plugins utilitzats per a això. Podem veure els \*plugins utilitzats en els diferents cicles de vida (\*clean, \*default, \*site, etcètera)

En principi, si no necessitem afegir cap dependència o alguna característica específica per al nostre projecte, no serà necessari fer canvis en el fitxer.

* Compilar i executar el projecte generat

L'arquetip que hem utilitzat per a generar l'aplicació, *\*maven-\*Archetype-\*quickstart*, ens crea directament un esquelet d'aplicació del tipus *Hola Món*. Si accedim a l'arxiu \*src/\*main/\*java /\*com/\*mgh/\*edd/App.\*java, veurem un codi bastant familiar:

\*package \*com.\*mgh.\*edd;

/\*\*

\* \*Hello \*world!

\*

\*/

\*public \*class App

{

   \*public \*static \*void \*main( \*String[] \*args )

   {

       \*System.out.\*println( "\*Hello \*World!" );

   }

}

Per a compilar i executar este codi utilitzarem el comando de \*maven *compile*, i realitzarem la invocació a la màquina virtual de \*java. Vegem-ho per parts.

**Pas 1. Compilació**

Ens situem en l'arrel del projecte, i executem:

**\*mvn compile** 

[\*INFO] \*Scanning \*for \*projects...

[\*INFO]

[\*INFO] ----------------------< \*com.\*mgh.\*edd:\*saludaMaven >-----------------------

[\*INFO] **\*Building \*saludaMaven 1.0-\*SNAPSHOT**

[\*INFO] --------------------------------[ \*jar ]---------------------------------

[\*INFO]

[\*INFO] --- \*maven-\*resources-\*plugin:3.0.2:\*resources(\*default-\*resources) @ \*saludaMaven ---

[\*INFO] \*Using '\*UTF-8' \*encoding \*to \*copy \*filtered \*resources.

[\*INFO] \*skip senar \*existing \*resourceDirectory ../\*saludaMaven/\*src/\*main/\*resources

[\*INFO]

[\*INFO] --- \*maven-\*compiler-\*plugin:3.8.0:compile(\*default-compile) @ \*saludaMaven --

[\*INFO] \*Changes \*detected - \*recompiling \*the module!

[\*INFO] \*Compiling 1 \*source \*file \*to /\*tmp/mes\_\*proves/\*saludaMaven/\*target/\*classes

[\*INFO] ------------------------------------------------------------------------

[\*INFO] \*BUILD \*SUCCESS

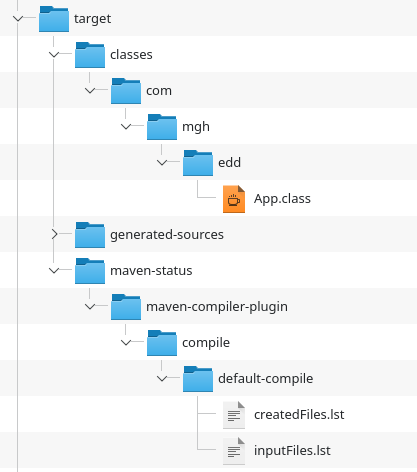
[\*INFO] ------------------------------------------------------------------------

[\*INFO] Total estafe:  2.278 s

[\*INFO] \*Finished \*at: 2021-08-18T17:26:53+02.00

[\*INFO] ------------------------------------------------------------------------

Amb això es genera una nova carpeta *\*target* en l'arrel del projecte, amb les classes generades, seguint la següent estructura:



Com veiem, es genera la carpeta \*classes, amb els fitxers .\*class en \*bytecode, segons l'estructura del paquet, i a més un parell de carpetes més. Una amb recursos, que contindrà una carpeta *\*annotations* buida, i una altra *\*maven-estatus*, amb l'estat o resultats dels \*plugins. En este cas, el resum del \*plugin de compilació n us indica quins fitxers es van crear i quins van ser els fitxers d'entrada.

**Pas 2. Execució**

Per a executar l'aplicació, \*debermos indicar el *\*classpath* a *\*java*, mitjançant la \*opcion *-\*cp*, i executar esta, indicant el nom de la classe completament qualificat:

**$ \*java -\*cp \*target/\*classes \*com.\*mgh.\*edd.App**

\*Hello \*World!

**\*Classpath**

El \*classpath li indica a la màquina virtual de Java on es troben els paquets i classes definides per l'usuari, necessàries per a executar un programa.

El \*classpath es pot indicar bé mitjançant l'opció *-\*cp* des de la línia de comandos o mitjançant la variable d'entorn.\*CLASSPATH.

important

* Neteja del projecte

Per a netejar el projecte, utilitzarem l'orde *\*mvn \*clean*, invocant-la des de l'arrel del propi projecte.

**$ \*mvn \*clean** 

[\*INFO] \*Scanning \*for \*projects...

[\*INFO]

[\*INFO] ----------------------< \*com.\*mgh.\*edd:\*saludaMaven >-----------------------

[\*INFO] \*Building \*saludaMaven 1.0-\*SNAPSHOT

[\*INFO] --------------------------------[ \*jar ]---------------------------------

[\*INFO]

[\*INFO] --- \*maven-\*clean-\*plugin:3.1.0:\*clean(\*default-\*clean) @ \*saludaMaven ---

[\*INFO] \*Deleting .../\*saludaMaven/\*target

[\*INFO] ------------------------------------------------------------------------

[\*INFO] \*BUILD \*SUCCESS

[\*INFO] ------------------------------------------------------------------------

[\*INFO] Total estafe:  0.971 s

[\*INFO] \*Finished \*at: 2021-08-18T23:59:33+02.00

[\*INFO] ------------------------------------------------------------------------

Com veiem, la qual cosa ha fet *\*mvn \*clean*, ha sigut esborrar la carpeta \*target.

* Empaquetat

Les aplicacions Java solen distribuir-se empaquetades en un fitxer de tipus \*JAR (Java Arxive), que continga tots els .\*class de l'aplicació.

Per a crear este fitxer, utilitzarem el comando *\*mvn \*package*. Vegem com fer-ho:

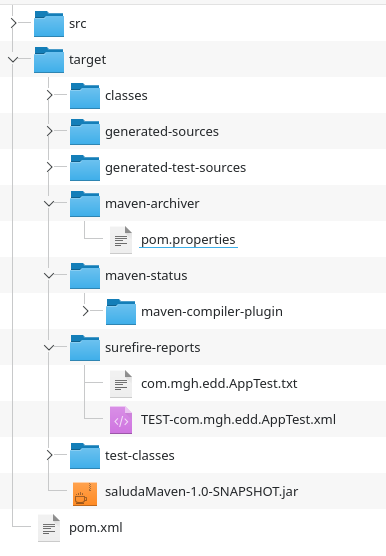
**Pas 1. Creació del \*JAR**

**$ \*mvn \*package**

O bé, si desitgem netejar abans el projecte:

**$ \*mvn \*clean \*package**

Per a completar el cicle de construcció de l'empaquetat, es passa prèviament per la compilació i els tests, generant la carpeta *\*target* amb una mica més de contingut que únicament per a la compilació:



A part de les carpetes *\*classes*, amb els .\*class resultat de la compilació, i les carpetes que ja vam veure que es generen quan es realitza la compilació, s'han generat altres carpetes com *\*maven-\*archiver*, que conté un fitxer *\*pom.\*properties*, amb el *\*groupid*, el *\*artifactid* i la *versió* del projecte, així com la carpeta *\*surefire-\*reports*, amb informes sobre els tests. De totes maneres, el que ens interessa és el fitxer generat *\*saludaMaven-1.0-\*SNAPSHOT.\*jar*, amb l'arxiu de l'aplicació.

**Pas 2. Executant el \*JAR**

Per a executar-ho, podem fer-ho amb:

\*java -\*cp \*target/\*saludaMaven-1.0-\*SNAPSHOT.\*jar \*com.\*mgh.\*edd.App 

Amb això, indiquem com *\*classpath* el \*JAR, ja que és ell qui conté les classes, i a més, la classe principal.

No obstant això, la forma habitual d'executar en \*JAR serà amb l'opció *-\*jar* de *\*java*:

**\*java -\*jar \*target/\*saludaMaven-1.0-\*SNAPSHOT.\*jar** 

no hi ha cap atribut de manifest principal en \*target/\*saludaMaven-1.0-\*SNAPSHOT.\*jar

Encara que com veiem, ens mostra un error, indicant que no existix un *atribut en el manifest principal*. El Manifest, en un fitxer \*JAR és un fitxer situat dins de l'arxiu \*JAR que definix algunes característiques d'este. Si obrim el \*JAR generat amb una aplicació de compressió i descompressió d'arxius, ho trobarem en *META-\*INF/\*MANIFEST.\*MF*, i el seu contingut és el següent:

\*Manifest-\*Version: 1.0

\*Created-\*By: Apatxe \*Maven 3.6.3

\*Built-\*By: \*dam

\*Build-\*Jdk: 11.0.11

Com veiem, conté informació sobre la versió del manifest, qui el va generar i el \*JDK, però no s'indica quina és la classe principal de l'aplicació.

Perquè \*Maven afija #ací la classe, haurem d'indicar-li-ho en el *\*pom.\*xml.* Per a això, editem este fitxer i busquem el \*plugin *\*maven-\*jar-\*plugin:*

<\*plugin>

<\*artifactId>\*maven-\*jar-\*plugin</\*artifactId>

<\*version>3.0.2</\*version>

</\*plugin>

Ara, el que farem serà afegir una nova etiqueta de configuració, per a indicar-li que s'incloga en el manifest el \*Classpath i especificar la classe principal de l'aplicació. Així doncs, ens quedaria:

<\*plugin>

<\*artifactId>\*maven-\*jar-\*plugin</\*artifactId>

<\*version>3.0.2</\*version>

**<\*configuration>**

**<arxive>**

**<\*manifest>**

**<\*addClasspath>\*true</\*addClasspath>**

**<\*mainClass>\*com.\*mgh.\*edd.App</\*mainClass>**

**</\*manifest>**

**</arxive>**

**</\*configuration>**

</\*plugin>

Amb això ja podem generar de nou el \*JAR:

**$ \*mvn \*clean \*package**

I executar-ho amb el paràmetre *-\*jar*:

**$ \*java -\*jar \*target/\*saludaMaven-1.0-\*SNAPSHOT.\*jar**  

\*Hello \*World!

Si tornem a consultar el fitxer de Manifest, veurem que ara sí que conté el nom de la classe principal:

\*Manifest-\*Version: 1.0

\*Created-\*By: Apatxe \*Maven 3.6.3

\*Built-\*By: \*dam

\*Build-\*Jdk: 11.0.11

**\*Main-\*Class: \*com.\*mgh.\*edd.App**

* El mode interactiu de \*Maven

Com ja comentem, a part del mode *\*batch* que hem vist, \*Maven oferix un mode de funcionament interactiu, que ens va guiant en la creació dels nostres projectes.

Per a crear un projecte \*Maven des del mode interactiu, \*sólamente haurem d'indicar que desitgem executar l'objectiu *\*generate* del \*plugin *\*archerype*, sense res més. Així doncs, per a generar un altre projecte, ens situem fora de l'anterior, i executem:

\*mvn \*archetype:\*generate 

Després d'alguns possibles avisos i missatges informatius ens oferix una llista de totes les plantilles disponibles en \*Maven (prop de 3000). Per defecte proposa la creació d'un projecte de tipus *\*maven-\*archetype-\*quickstart (1808),* que seria el mateix que hem utilitzat, amb una estructura d'Hola  *Món*. Haureu de tindre en compte que este número pot variar en diferents versions de \*Maven.\*mvn \*archetype:\*generate.

\*Choose a \*number \*or \*apply \*filter (\*format: [\*groupId:]\*artifactId, case \*sensitive \*contains): 1808: 

Després ens demana el número de versió de l'aplicació. Seleccionem el *5* (la 1.0):

\*Choose \*org.apatxe.\*maven.\*archetypes:\*maven-\*archetype-\*quickstart \*version:

1: 1.0-\*alpha-1

2: 1.0-\*alpha-2

3: 1.0-\*alpha-3

4: 1.0-\*alpha-4

5: 1.0

6: 1.1

7: 1.3

8: 1.4

\*Choose a \*number: 8: **5** 

Després ens pregunta altres dades, com el *\*groupId, \*artifact*, etcètera. Per als quals ja ens oferix un valor vàlid per defecte, no farà falta reescriure'ls, i podrem polsar *Intro* per a acceptar estos valors.

Definix \*value \*for \*property '\*groupId': **\*com.\*mgh.\*edd** 

Definix \*value \*for \*property '\*artifactId': **\*Saluda2** 

Definix \*value \*for \*property '\*version' 1.0-\*SNAPSHOT: :  

Definix \*value \*for \*property '\*package' \*com.\*mgh.\*edd: :  

\*Confirm \*properties \*configuration:

\*groupId: \*com.\*mgh.\*edd

\*artifactId: \*Saluda2

\*version: 1.0-\*SNAPSHOT

\*package: \*com.\*mgh.\*edd

I: :  

Després de demanar la confirmació de tot, ja ens mostra el resultat de la creació.

[\*INFO] ----------------------------------------------------------------------------

[\*INFO] \*Using \*following \*parameters \*for \*creating \*project \*from \*Old (1.x) \*Archetype: \*maven-\*archetype-\*quickstart:1.0

[\*INFO] ----------------------------------------------------------------------------

[\*INFO] \*Parameter: \*basedir, \*Value: /\*tmp/mes\_\*proves

[\*INFO] \*Parameter: \*package, \*Value: \*com.\*mgh.\*edd

[\*INFO] \*Parameter: \*groupId, \*Value: \*com.\*mgh.\*edd

[\*INFO] \*Parameter: \*artifactId, \*Value: \*Saluda2

[\*INFO] \*Parameter: \*packageName, \*Value: \*com.\*mgh.\*edd

[\*INFO] \*Parameter: \*version, \*Value: 1.0-\*SNAPSHOT

[\*INFO] \*project \*created \*from \*Old (1.x) \*Archetype \*in \*dir: /\*tmp/mes\_\*proves/\*Saluda2

[\*INFO] ------------------------------------------------------------------------

[\*INFO] \*BUILD \*SUCCESS

[\*INFO] ------------------------------------------------------------------------

[\*INFO] Total estafe:  07.23 min

[\*INFO] \*Finished \*at: 2021-08-19T08:37:44+02.00

[\*INFO] ------------------------------------------------------------------------

En este cas, el *\*pom.\*xml* generat és possible que es genere d'una plantilla diferent, i diferisca del que hem generat anteriorment. Les principals diferències poden ser que no s'especifique les opcions *\*maven.\*compiler.\*source* i *\*maven.\*compiler.\*target*:, així com la configuració dels *\*plugins*.

Així doncs, en primer lloc, haurem d'afegir les opcions relatives al compilador. I en segon, solament aquells \*plugins que vulguem configurar, ja que la resta prendrà les opcions per defecte. En el nostre cas, es tractaria d'afegir la configuració a l'empaquetat.

En resum, hauríem de modificar el nou \*pom.\*xml amb el següent contingut (s'afigen en diferent format i color les modificacions):

<\*project \*xmlns="http://maven.apache.org/pom/4.0.0" \*xmlns:\*xsi="http://www.w3.org/2001/xmlschema-instance"

\*xsi:\*schemaLocation="http://maven.apache.org/pom/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<\*modelVersion>4.0.0</\*modelVersion>

<\*groupId>\*com.\*mgh.\*edd</\*groupId>

<\*artifactId>\*Saluda2</\*artifactId>

<embalatge>\*jar</embalatge>

<\*version>1.0-\*SNAPSHOT</\*version>

<\*name>\*Saluda2</\*name>

<\*url>http://maven.apache.org</\*url>

**<\*properties>**

**<\*project.\*build.\*sourceEncoding>\*UTF-8</\*project.\*build.\*sourceEncoding>**

**<\*maven.\*compiler.\*source>1.7</\*maven.\*compiler.\*source>**

**<\*maven.\*compiler.\*target>1.7</\*maven.\*compiler.\*target>**

**</\*properties>**

<\*dependencies>

<\*dependency>

<\*groupId>\*junit</\*groupId>

<\*artifactId>\*junit</\*artifactId>

<\*version>3.8.1</\*version>

<\*scope>test</\*scope>

</\*dependency>

</\*dependencies>

**<\*build>**

**<\*pluginManagement>**

**<\*plugins>**

**<\*plugin>**

**<\*artifactId>\*maven-\*jar-\*plugin</\*artifactId>**

**<\*version>3.0.2</\*version>**

**<\*configuration>**

**<arxive>**

**<\*manifest>**

**<\*addClasspath>\*true</\*addClasspath>**

**<\*mainClass>\*com.\*mgh.\*edd.App</\*mainClass>**

**</\*manifest>**

**</arxive>**

**</\*configuration>**

**</\*plugin>**

**</\*plugins>**

**</\*pluginManagement>**

**</\*build>**

</\*project>

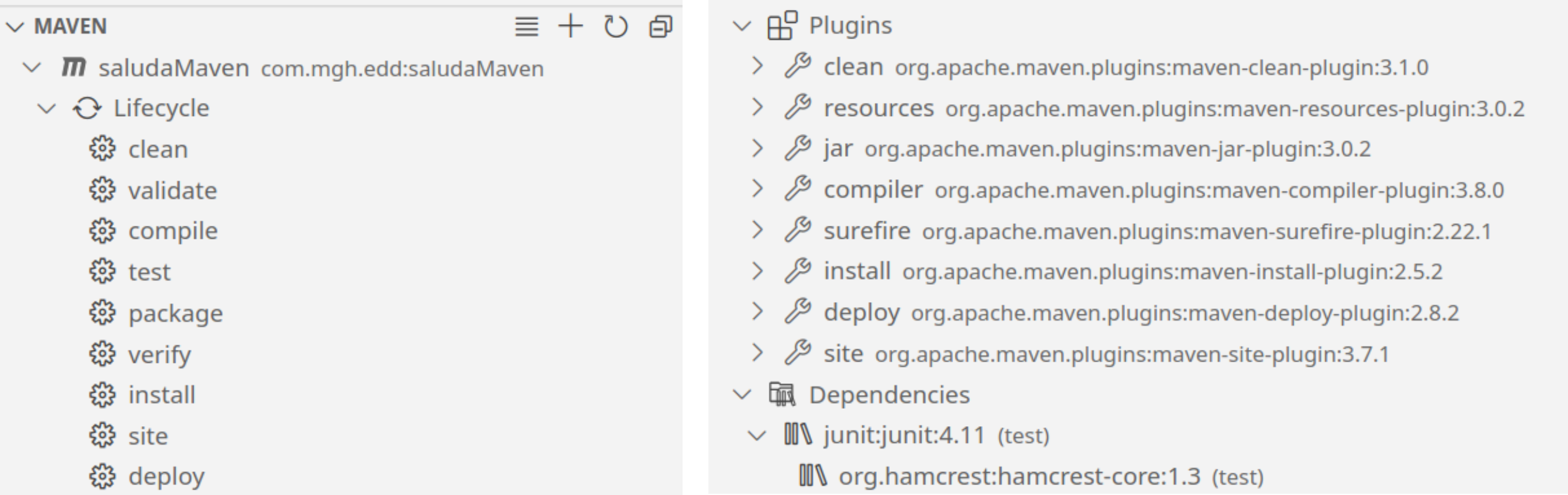
Amb això ja podrem compilar i empaquetar el projecte tal com vam fer en el projecte anterior.

El \*plugin de \*Maven per a \*VSCode

Com ja sabem, en instal·lar el *Java \*Extension \*Pack*, una de les extensions que ens arrossega és l'extensió *\*Maven \*for Java*. Esta extensió ens oferix un explorador de projectes \*Maven i afig les següent funcionalitats:

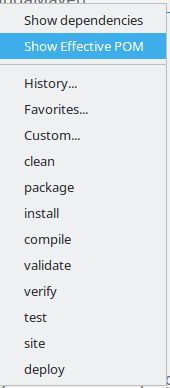
1. Suport per a la generació de projectes a partir d'arquetips \*Maven,
2. Suport per a generar \*POM de manera eficient,
3. Oferix dreceres de teclat per a llançar objectius de \*Maven comuns: *\*clean, \*validate, compile test, \*package, \*verify, \*install, \*site i \*deploy* ,
4. Manteniment d'un historial d'ordes per a tornar a llançar ordes recents.
5. L'explorador de projectes \*Maven

Quan obrim una carpeta o espai de treball que conté un fitxer *\*pom.\*xml*, \*VSCode ens mostra una nova secció en la barra lateral anomenada *\*Maven \*Projects*, amb els projectes \*Maven del \*worksapce i els seus mòduls.



Com veiem en la figura anterior, després d'obrir la carpeta on tenim el projecte de \*saludaMaven, este se'ns obrirà en la pestanya *\*MAVEN*. Si fem clic en el nom del projecte, se'ns despleguen els diferents cicles de vida, que podrem executar directament mitjançant la icona *\*Run* () que ens apareix al costat de cada cicle, així \*com accedir al fitxer *\*pom.\*xml* des de la icona *\*Open \*POM \*File (**)*

Si ara fem clic amb el botó dret sobre el projecte *\*saludaMaven*, veurem diferents opcions, algunes d'elles relacionades amb els objectius del \*LifeCycle (\*clean, \*validate, etcètera), a més d'accés a l'historial (*\*History*), entre altres opcions. Una de les més interessants és *Xou \*Effective \*POM*.



**Què és el *\*Efective \*POM*?**

Quan creguem un projecte \*Maven a partir d'un arquetip, definim per al mateix un conjunt de propietats com ara el *\*groupID*, el *\*artifactID* i la *versió*. Estos tres camps són els que es requerixen per a tot projecte, i són el que conformen el \*POM més simple que podríem generar. El \*POM d'un projecte, a més, hereta unes certes propietats del que es coneix com el *\*Super \*POM*, que vindria a ser com la classe base de la que hereten tots els \*POM. És a dir, un \*POM es compon dels atributs que definim explícitament en funció de l'arquetip que hàgem utilitzat, i a més, posseïx un conjunt de valors per defecte compartits per tots els projectes, i que no apareixen explícitament en el fitxer *\*pom.\*xml*.

Tenint tot això en compte, el \*POM Efectiu (*\*Effective \*POM)* s'entén com la combinació de tots dos \*POM, el \*Super \*POM i el \*POM més simple. A efectes pràctics, si consultem el \*POM efectiu del nostre projecte, veurem que es tracta d'un fitxer XML bastant més extens que l'original, i en el qual tindrem totes les característiques que teníem definides explícitament més totes aquelles que hi havia implícitament i que són comunes a tots els projectes.

IMPORTANT

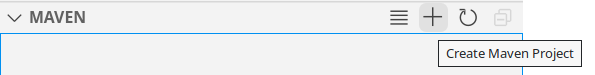
1. Generació d'un projecte \*Maven des de \*VSCode

Visual \*Studio ens permet crear projectes \*Maven a partir d'arquetips de diferents repositoris. Simplement, el que fa és llançar per nosaltres les ordes que hem vist anteriorment en la terminal de \*VSCode.

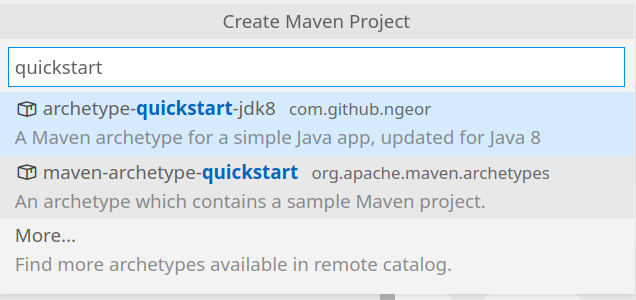
Vegem com fer-ho. Partint d'una finestra nova de \*VSCode (*Arxiu > Nova Finestra*):

**Pas 1.** En l'activitat de l'Explorador, busquem la pestanya desplegable *\*MAVEN*, i el símbol "+" que apareix a la seua dreta.

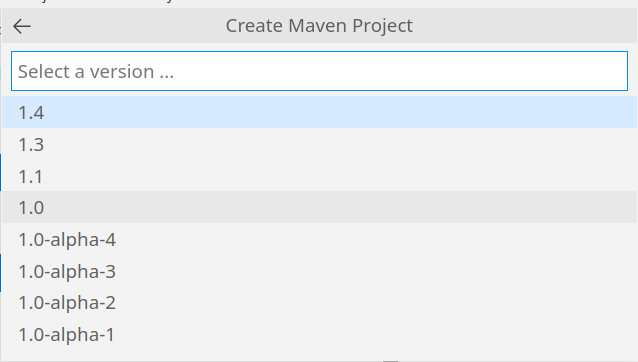
**Pas 2.** Fem clic en este símbol "+" per a crear un projecte \* \*Maven \* nou.

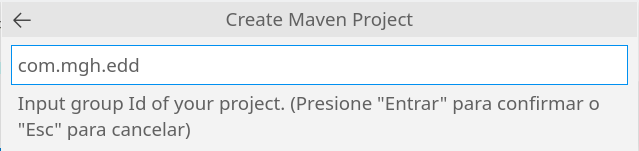


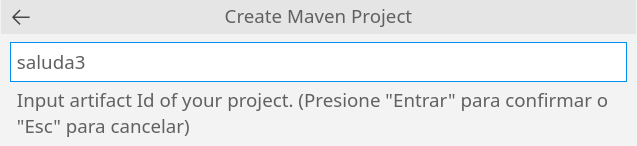
**Pas 3**. En la paleta d'ordes ens apareixerà els diferents arquetips de què disposem. Busquem *\*quickstart*, i seleccionem *\*maven-\*Archetype-\*quickstart*.



**Pas 4.** Indiquem la versió, el *\*groupId*, i el nom del projecte:







**Pas 5.** Fet això, se'ns obrirà la terminal integrada de \*VSCode, i ens demanarà la propietat *\*version* que falta per configurar. Deixem el valor predeterminat 1.0-\*SNAPSHOT, i polsem *Intro* per a crear el projecte:

Definix \*value \*for \*property '\*version' 1.0-\*SNAPSHOT: : 

[\*INFO] \*Using \*property: \*package = \*com.\*mgh.\*edd

\*Confirm \*properties \*configuration:

\*groupId: \*com.\*mgh.\*edd

\*artifactId: \*saluda3

\*version: 1.0-\*SNAPSHOT

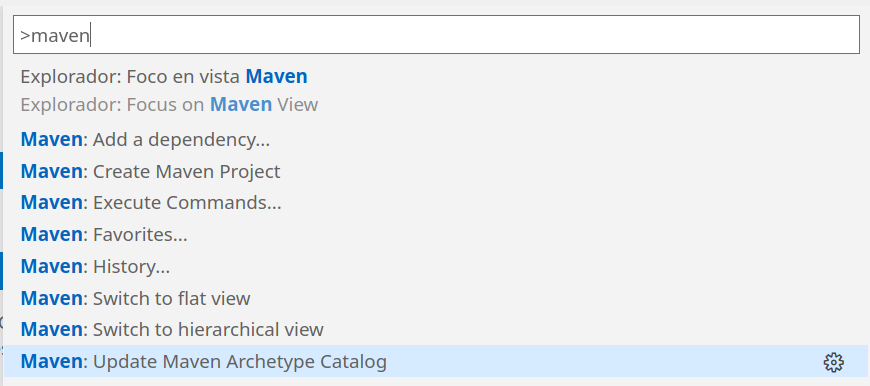
\*package: \*com.\*mgh.\*edd

I: : 

Una vegada generat el projecte, si donem una ullada a l'estructura de carpetes generada i al \*pom.\*xml , veurem que és pràcticament el mateix que havíem definit per consola.

1. Actualització de la caixet d'arquetips

Una altra opció que podem realitzar \*VSCode, és actualitzar la caixet local dels diferents arquetips des del repositori central de *\*Maven*. Per a això, accedirem a la *Consola d'Ordes mitjançant \*Ctrl* +*\*Shift* + *P*, i quan se'ns òbriga esta, començarem a escriure *\*Maven:*... per a filtrar les diferents opcions, fins a tindre a la vista l'opció de *\*Update \*Maven \*Archetype \*Catalog*.

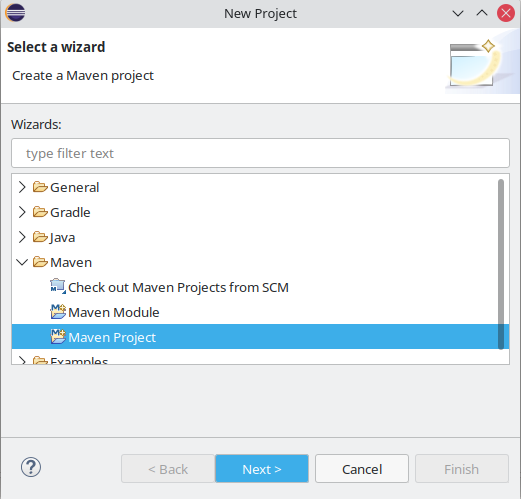


\*Maven en Eclipsi

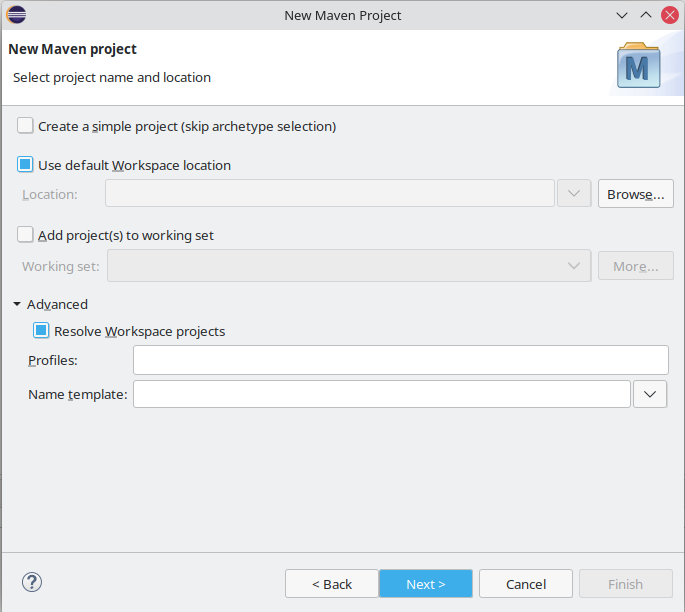
El \*IDE Eclipsi gestiona de manera nativa projecte de tipus \*Maven. Per a generar un nou projecte d'este tipus, seguirem els següents passos:

**Pas 1.** Obrim l'assistent per a la gestió de projectes, mitjançant l'opció de menú *\*File > \*New > \*Project.*

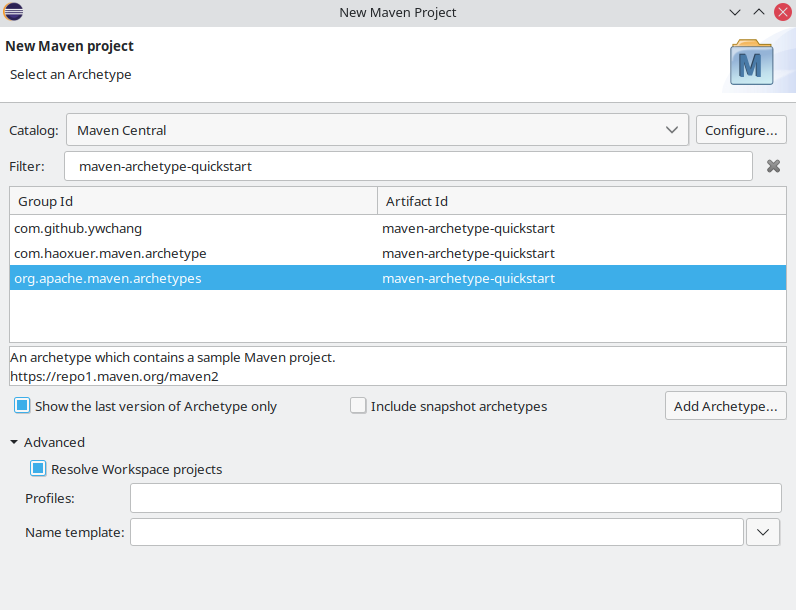
**Pas 2.** En l'assistent (\*Wizard) despleguem la carpeta de tipus de projectes *\*Maven,*  seleccionem *\*Maven \*Project*, i fem clic en *\*Next.*



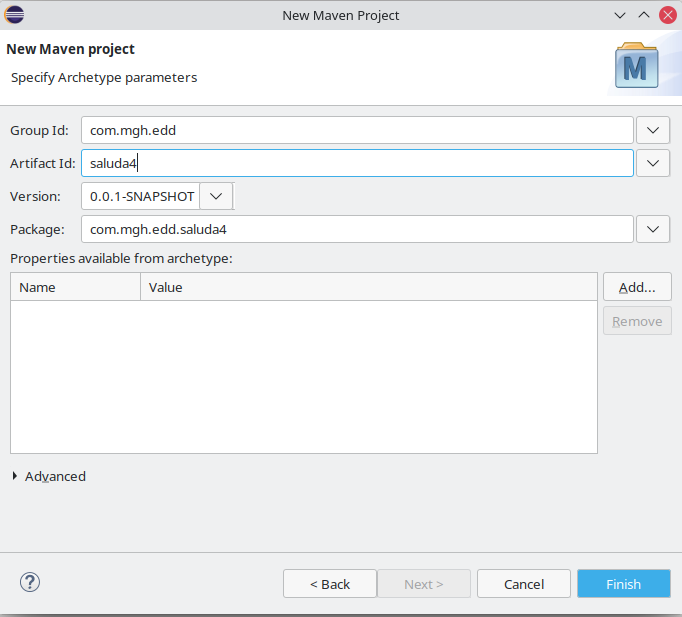
**Pas 3.** En la següent finestra de l'assistent, seleccionem la ubicació del projecte (deixem la ubicació del \*Workspace per defecte) i fem clic en *\*Next.*



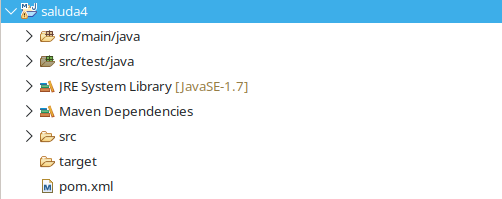
**Pas 4.** En la següent finestra seleccionarem l'arquetip. Per a això, buscarem en el catàleg de \*Maven Central (*\*Catalog*) i filtrem per *\*maven-\*archetype-\*quickstart*. Seleccionarem el *\*Quickstart* de *\*org.apatxe.\*maven.\*archetypes*.



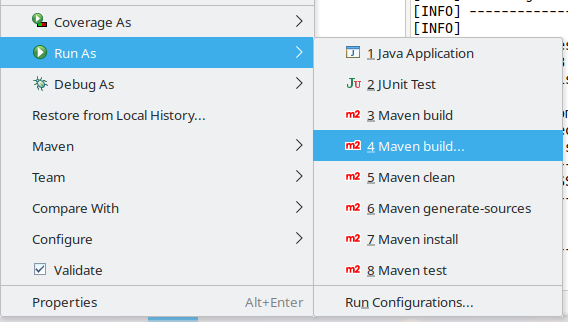
**Pas 5.** L'última finestra ja ens demana els diferents paràmetres del projecte, com el *\*GroupId*, el \*Artifact Aneu, el *paquet* o la versió:



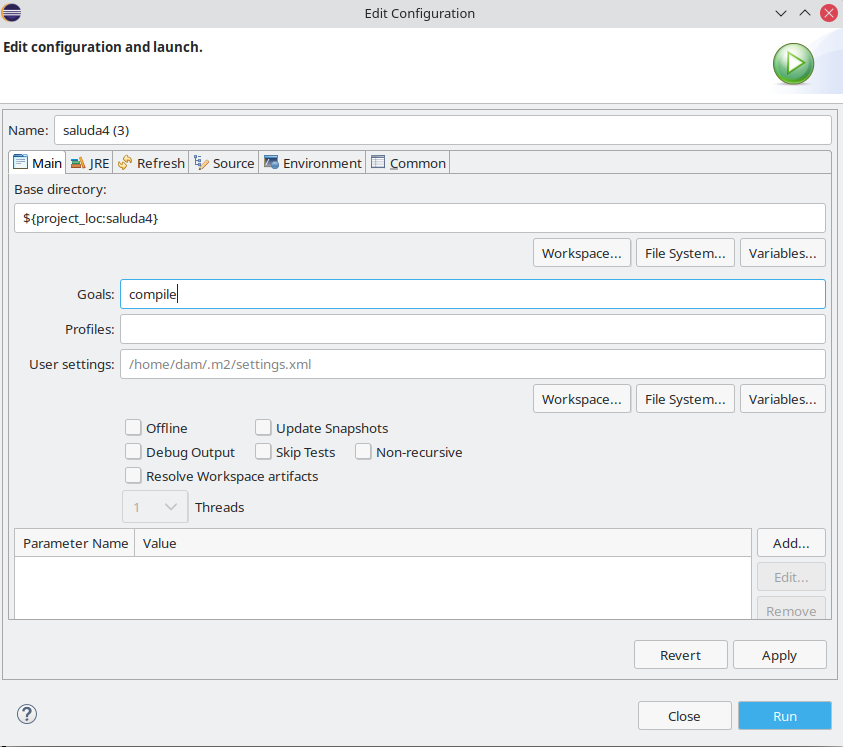
**Pas 6.** Finalment, fem clic en \*Finish per a generar el projecte. En l'Explorador, veurem que tenim la mateixa estructura que generem des de la línia de comandos:



**Pas 7.** Per a llançar un objectiu, fem clic amb el botó dret sobre el projecte, i busquem l'opció *\*Run As > \*Maven \*build...*:



**Pas 8.** En la configuració seleccionarem l'objectiu que desitgem, per exemple *compile*, i executem:



Finalment, ens mostrarà el resultat de la compilació per la Consola:

[\*INFO] \*Scanning \*for \*projects...

[\*INFO]

[\*INFO] ------------------------< \*com.\*mgh.\*edd:\*saluda4 >-------------------------

[\*INFO] \*Building \*saluda4 0.0.1-\*SNAPSHOT

[\*INFO] --------------------------------[ \*jar ]---------------------------------

[\*INFO]

[\*INFO] --- \*maven-\*resources-\*plugin:3.0.2:\*resources (\*default-\*resources) @ \*saluda4 ---

[\*INFO] \*Using '\*UTF-8' \*encoding \*to \*copy \*filtered \*resources.

[\*INFO] \*skip senar \*existing \*resourceDirectory /\*home/\*dam/eclipsi-\*workspace/\*saluda4/\*src/\*main/\*resources

[\*INFO]

[\*INFO] --- \*maven-\*compiler-\*plugin:3.8.0:compile (\*default-compile) @ \*saluda4 ---

[\*INFO] \*Changes \*detected - \*recompiling \*the module!

[\*INFO] \*Compiling 1 \*source \*file \*to /\*home/\*dam/eclipsi-\*workspace/\*saluda4/\*target/\*classes

[\*INFO] ------------------------------------------------------------------------

[\*INFO] \*BUILD \*SUCCESS

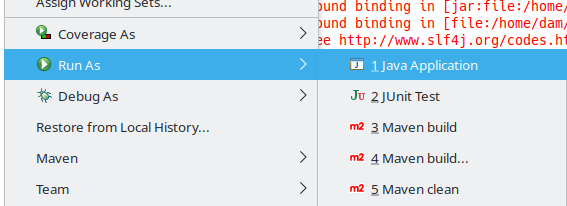
[\*INFO] ------------------------------------------------------------------------

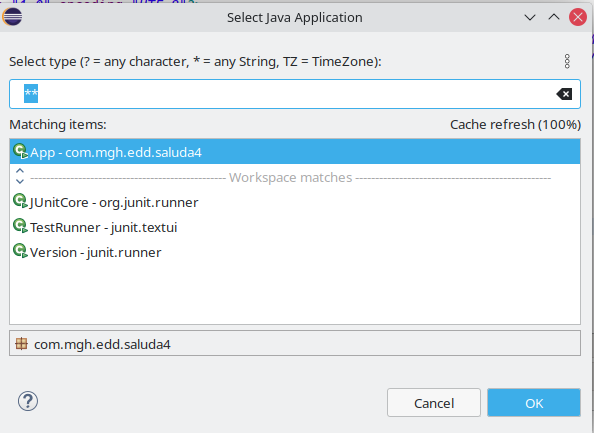
[\*INFO] Total estafe: 3.109 s

[\*INFO] \*Finished \*at: 2021-08-19T16:51:21+02.00

[\*INFO] ------------------------------------------------------------------------

**Pas 8.** Per a executar el projecte, finalment, seleccionem \*bon el botó dret sobre este l'opció *\*Run As > Java \*Application*, i el nom de la classe a executar:





Annex I. Instal·lació de \*Maven en Windows 10

Apatxe \*Maven no proporciona un instal·lador com a tal per a Windows, sinó que oferix tant els arxius binaris com els fonts en diferents formats de compressió. Així doncs, la instal·lació consistirà a descarregar estos binaris, descomprimir-los en una carpeta del nostre equip i configurar el \*path per a poder accedir des de qualsevol part del sistema als executables de \*Maven.

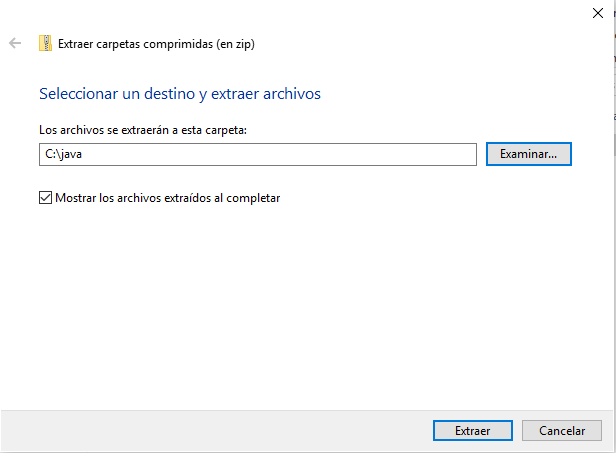
**Pas 1. Descàrrega**

En primer lloc, haurem d'accedir a la web de descàrregues d'Apatxe \*Maven, concretament a la secció *\*files*, on se situen els fonts i els binaris: [http://maven.apache.org/download.cgi#\*files](http://maven.apache.org/download.cgi#files)

Des d'esta secció de \*Files, descarreguem el fitxer *apatxe-\*maven-3.8.2-\*bin.\*zip* amb els binaris comprimits en \*zip.

**Pas 2. Descompressió**

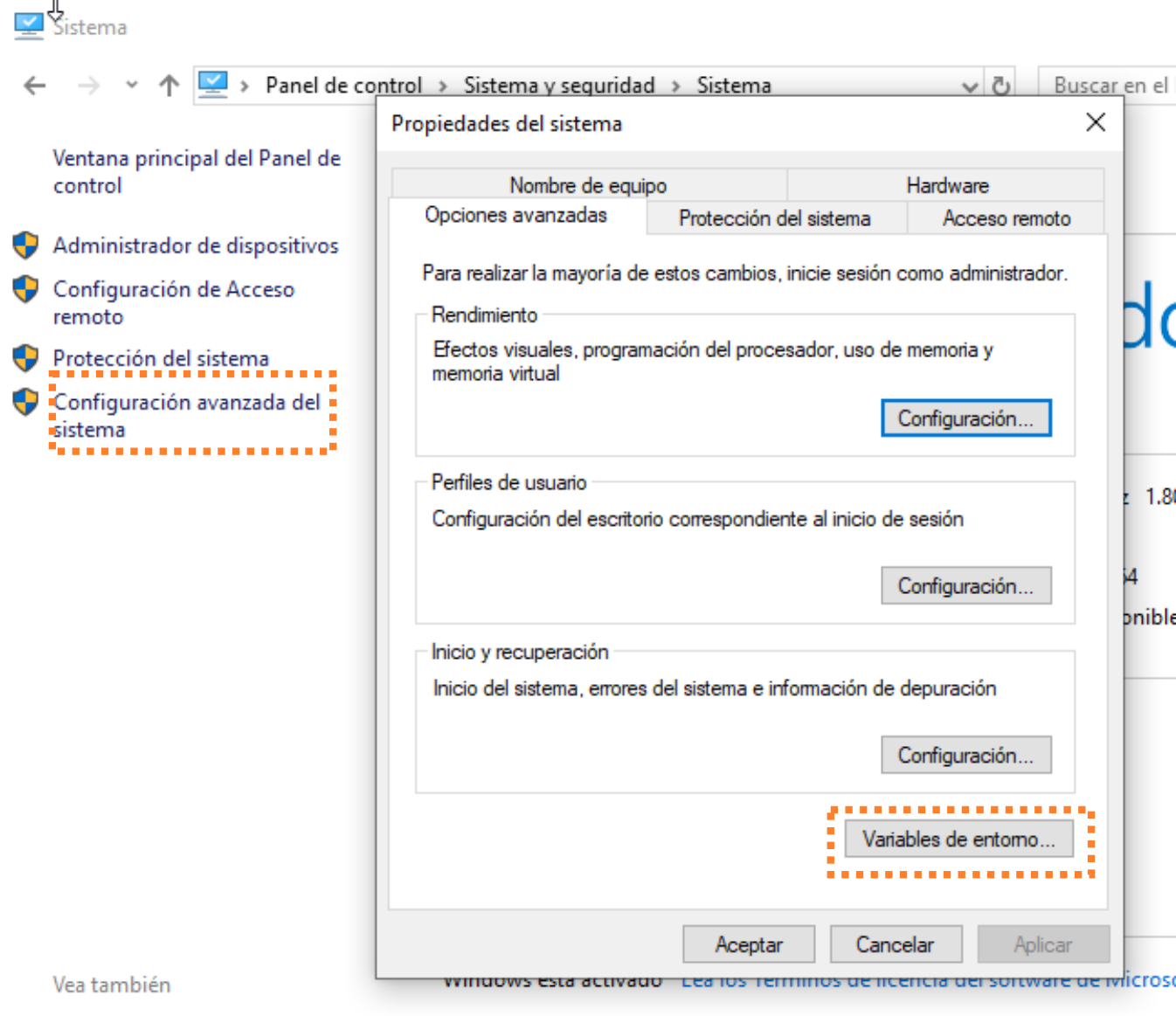
Descomprimim el fitxer descarregat en qualsevol carpeta del sistema. En el nostre cas, l'hem descomprimida en la carpeta C:/\*java, on ja es troba el \*JDK, però pots fer-ho en la carpeta que desitges.



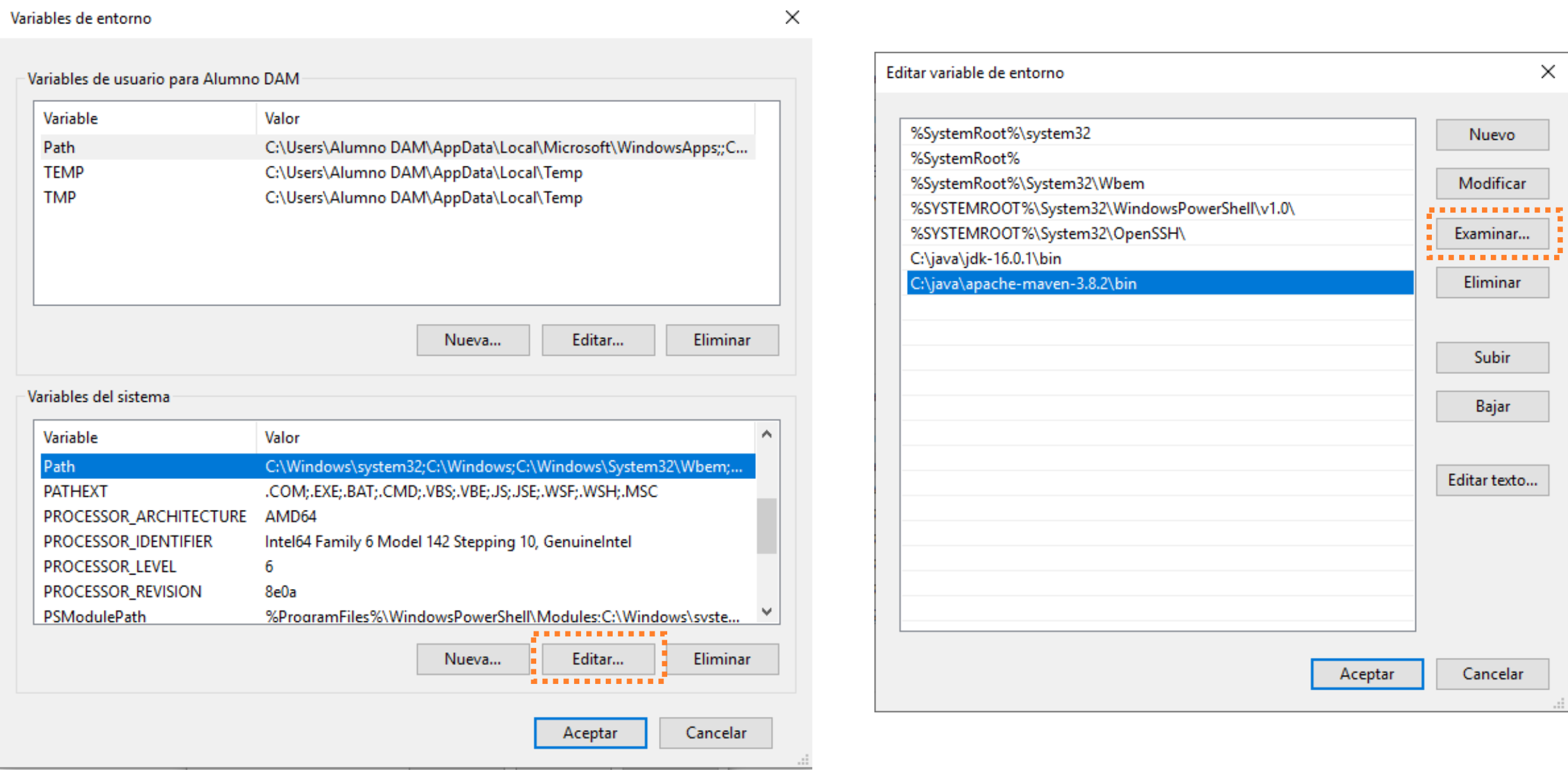
Això ens generarà la \*carpata C:\\*java\apatxe-\*maven-3.8.2 (o el número de versió que hages descarregat), amb els diferents fitxers i carpetes de \*Maven. Ara tocarà establir algunes variables d'entorn per a facilitar el seu accés.

**Pas 3. Afegint \*maven al \*PATH**

Per a afegir la ruta dels binaris de \*Maven al \*path haurem d'accedir a la configuració de les variables d'entorn, tal com vam veure en la unitat anterior. Des del menú d'Inici, busquem *Sistema de Windows > Panell de Control*. I des d'este panell de Control, *Sistema i Seguretat* > *Sistema* > *Configuració Avançada del Sistema.* En la finestra de configuració avançada, disposàvem d'un botó per a configurar les variables d'entorn. Recordeu que també podeu buscar esta finestra des del propi cercador de la barra de tasques de Windows.



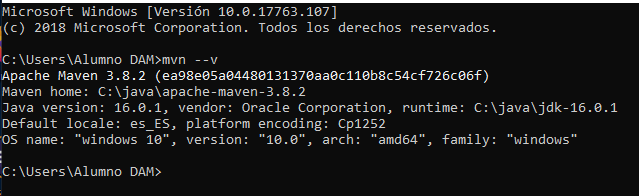
En la configuració de les variables d'entorn, editarem la variable del sistema \*Path. Per a això, la seleccionem i polsem sobre el botó *Editar.* Veurem una nova finestra amb tots els directoris inclosos en el \*path, i seleccionarem *Examinar* per a seleccionar el directori *\*bin* dins de la carpeta que hem descomprimit, en el nostre cas, *C:\\*java\apatxe-\*maven-3.8.2\\*bin*, que és on es troben els executables de \*Maven.



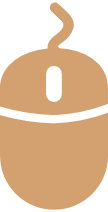
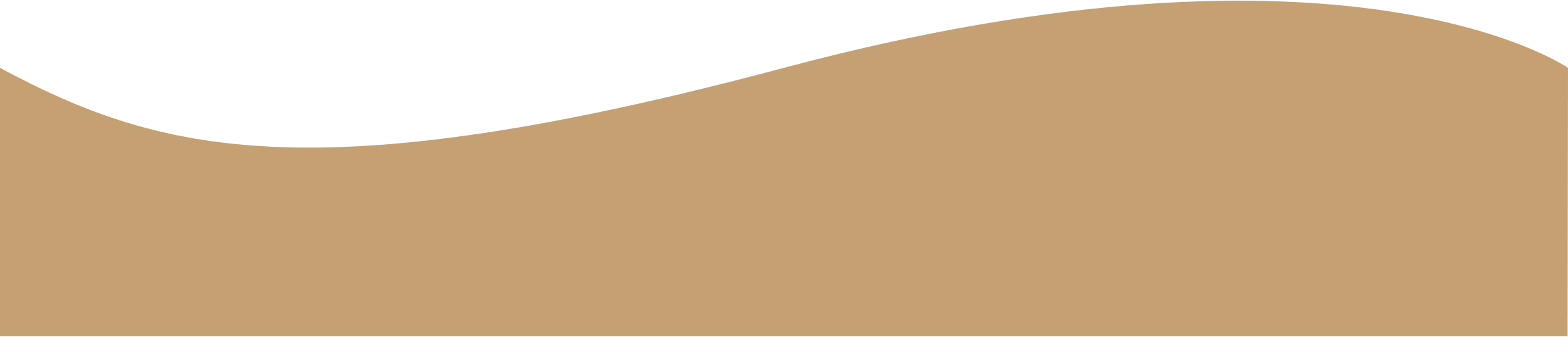
Una vegada fet això, fem clic a Acceptar i continuem amb el següent pas.

**Pas 4. Comprovació**

Finalment, per a comprovar si la instal·lació ha sigut correcta, consultarem la versió instal·lada de \*Maven mitjançant el comando *\*mvn --\*v*.



A partir d'este punt, ja teniu instal·lat i configurat \*Maven en el vostre equip Windows, per la qual cosa podeu continuar amb el cas pràctic.

**Cas pràctic ampliat 6.**

3

**UNITAT**

**\*Gradle.**

* \*Gradle

Després d'aprendre a generar projectes mitjançant \*Maven, aprendrem ara a fer-ho amb la ferramenta \*Gradle. Recordeu que es tracta d'una ferramenta que combina les tasques i facilitat de personalització de \*Ant, i la potència, el suport al cicle vida de construcció i les proves de \*Maven.

* Instal·lació

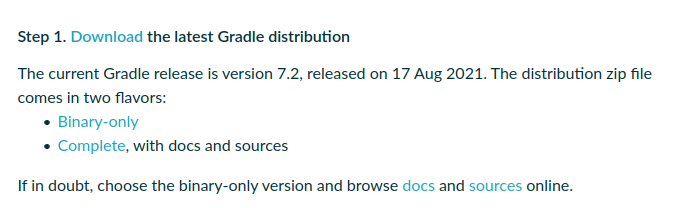
\*Gradle es troba en els repositoris de \*Ubuntu, però amb una versió bastant \*desactualizada

(4.4.1-10), si es compara amb la versió actual (7.2 a agost de 2021).

Així doncs, realitzarem la instal·lació de \*Gradle de manera manual. Per a això, seguirem les instruccions sobre la instal·lació indicades en la web de \*Gradle: <https://gradle.org/install/>, Podeu consultar l'enllaç per a unes instruccions més detallades.

**Pas 1. Descàrrega dels binaris**

En primer lloc, accedim a la secció d'Instal·lació  *Manual* de la pàgina d'instal·lació de \*Gradle: [https://gradle.org/install/#\*manually](https://gradle.org/install/#manually), i descarreguem únicament els binaris:



Amb això ens descarregarà el fitxer *\*gradle-7.2-\*bin.\*zip*. En el vostre cas, és possible que es descarregue una versió posterior.

**Pas 2. Descompressió**

En segon lloc, descomprimim el fitxer i el copiem en el directori on volem que estiga instal·lat.

En el nostre cas, per a \*Ubuntu el deixarem en el directori */\*opt*. Per a fer-ho per la línia de comandos, obrim una terminal i ens situem en la carpeta on hàgem descarregat el \*zip (per exemple la carpeta Descàrregues), i ho descomprimim amb *\*unzip*:

$ \*cd ~/Descàrregues 

Descàrregues$ \*unzip \*gradle-7.2-\*bin.\*zip 

Una vegada descomprimit ho movem com a usuaris administradors a */\*opt*:

$ sue \*mv \*gradle-7.2 /\*opt/ 

Per a **Windows**, seguirem els mateixos passos, descarregarem els binaris, i els descomprimirem, per exemple en la carpeta *C:\\*java*, que ja usem per al \*JDK i per a \*Maven.

**Pas 3. Afegint la ruta al \*Path**

Ara ens falta afegir la ruta on tenim instal·lat \*Gradle al \*PATH. Per a això, des de la terminal, segons la documentació escriuríem el següent:

\*export \*PATH=$\*PATH:/\*opt/\*gradle-7.2/\*bin

Ara bé, la línia anterior, exporta la variable \*PATH modificada a la terminal actual, però no persistix si obrim una altra terminal o si reiniciem l'ordinador.

Així doncs, per a fer este canvi persistent, farem el mateix que quan vam establir la variable JAVA\_\*HOME i modifiquem el \*PATH: Afegir un \*script al directori /etc/\*profile.d perquè ho carregue sempre que iniciem sessió.

Per a generar este fitxer amb el contingut que desitgem, des de la terminal escriurem el següent:

tire '\*export \*PATH=$\*PATH:/\*opt/\*gradle-7.2/\*bin' | sue \*tee /etc/\*profile.d/gradle.sh > /\*dev/\*null 

Amb el que tindrem el fitxer /*etc/\*profile.d/gradle.sh* creat amb el contingut *\*export \*PATH=$\*PATH:/\*opt/\*gradle-7.2/\*bin*. Ara només ens quedarà recarregar el directori *\*profile*:

\*source /etc/\*profile 

Amb això tindrem el \*PATH actualitzat en la terminal en què estiguem, però no en altres terminals. Per a això, haurem de tancar sessió i obrir una nova.

En el cas de **Windows**, accedirem a la configuració de les variables d'entorn i editarem el \*PATH per a afegir-li esta ruta, de la mateixa manera que vam fer amb la ruta de la instal·lació de \*Maven en el cas pràctic ampliat 5.

**Pas 4. Comprovació**

Una vegada recarregat el \*profile, comprovarem la versió de \*Gradle amb *\*gradle -\*v:*

$ **\*gradle -\*v** 

------------------------------------------------------------

\*Gradle 7.2

------------------------------------------------------------

\*Build estafe: 2021-08-17 09.59:03 \*UTC

\*Revision: \*a773786b58bb28710e3dc96c4d1a7063628952ad

\*Kotlin: 1.5.21

\*Groovy: 3.0.8

\*Ant: Apatxe \*Ant(TM) \*version 1.10.9 \*compiled \*on \*September 27 2020

\*JVM: 11.0.11 (\*Ubuntu 11.0.11+9-\*Ubuntu-\*0ubuntu2.20.04)

US: Linux 5.4.0-73-\*generic \*amd64

Com podem veure, ens mostra informació sobre les versions tant de \*Gradle (7.2), com de \*Kotlin (1.5.21), \*Groovy (3.0.8), \*Ant (1.10.9), la \*JVM (11.0.1), o el sistema operatiu.

* Creació d'un projecte en \*Gradle

Veurem com generar un projecte de tipus *Hola Món* típic en \*Gradle. Per a això, farem ara al revés que en el cas pràctic ampliat anterior, i veurem primer el mode interactiu per a veure després com crearíem el projecte amb una sola orde.

Per a crear un projecte en \*Gradle, ens situem en la nostra carpeta de projectes i crearem una carpeta per al projecte, ja que \*Gradle generarà tota l'estructura d'este en la carpeta on ens trobem:

$ **\*mkdir \*HelloGradle** 

$ **\*cd \*HelloGradle** 

I ja dins d'esta, utilitzem l'orde *\*gradle \*init* per a iniciar l'assistent (en Windows serà *\*gradle.\*bat)*:

$ **\*gradle \*init** 

Veurem pas a pas este assistent:

**Pas 1. Tipus de projecte**

El primer que se'ns pregunta és el tipus de projecte a generar: Un projecte bàsic, sense codi, una aplicació, una llibreria o un \*plugin de \*Gradle. Seleccionarem la segona opció (*\*application*):

\*Select \*type \*of \*project \*to \*generate:

 1: \*basic

 2: \*application

 3: \*library

 4: \*Gradle \*plugin

\*Enter \*selection (\*default: \*basic) [1..4] **2** 

**Pas 2. Llenguatge de programació**

En segon lloc, ens pregunta pel llenguatge de programació del projecte. Seleccionarem l'opció corresponent a Java:

\*Select \*implementation \*language:

 1: C++

 2: \*Groovy

 3: Java

 4: \*Kotlin

 5: Scala

 6: \*Swift

\*Enter \*selection (\*default: Java) [1..6] **3** 

**Pas 3. Subprojectes**

En tercer lloc ens pregunta si volem dividir l'aplicació en diversos subprojectes. En el nostre cas indicarem que no, ja que únicament generarem una aplicació.

\*Split \*functionality \*across \*multiple \*subprojects?:

 1: no - \*only \*one \*application \*project

 2: \*yes - \*application \*and \*library \*projects

\*Enter \*selection (\*default: no - \*only \*one \*application \*project) [1..2] **1** 

**Pas 4. \*DSL per al \*script de compilació**

Ara haurem de seleccionar què \*DSL (Llenguatge \*Específicao de Domini) s'utilitzarà en el \*script de compilació. Podem seleccionar \*Groovy o \*Kotlin. Utilitzarem esta primera opció:

\*Select \*build \*script \*DSL:

 1: \*Groovy

 2: \*Kotlin

\*Enter \*selection (\*default: \*Groovy) [1..2] **1** 

**Pas 5. \*Framework per als tests**

El següent pas, ens demana que indiquem el *\*framework* que utilitzarem per als tests unitaris. Aprofundireu més en ells en la següent unitat. De moment, seleccionarem el 1.

\*Select test \*framework:

1: \*JUnit 4

2: \*TestNG

3: \*Spock

4: \*JUnit \*Jupiter

\*Enter \*selection (\*default: \*JUnit \*Jupiter) [1..4] **1** 

**Pas 6. Nom del \*proyeto**

En sext lloc, haurem d'indicar el nom del projecte. \*Gradle ens suggerix el nom de la carpeta en la qual estem (i que hem generat \*especíicamente per a este projecte). Podem utilitzar este (confirmant-ho o tornant-ho a escriure) o utilitzar un altre qualsevol:

\*Project \*name (\*default: \*HelloGradle): **\*HelloGradle** 

**Pas 7. Nom del paquet**

Finalment ens demana el nom del paquet. En principi, suggerix el mateix que l'aplicació. En el nostre cas utilitzarem este, però completament qualificat i en minúscula:

\*Source \*package (\*default: \*HelloGradle): **\*com.\*mgh.\*edd.\*hellogradle** 

**\*Cración del projecte sense l'assistent**

Este projecte podríem haver-lo generat amb una \*sóla orde, sense necessitat d'utilitzar l'assistent, especificant els diferents paràmetres com a arguments. L'orde corresponent seria la següent, a executar dins del directori creat per al projecte:

$ **\*gradle \*init --\*type \*java-\*application \**

**--test-\*framework \*JUnit \**

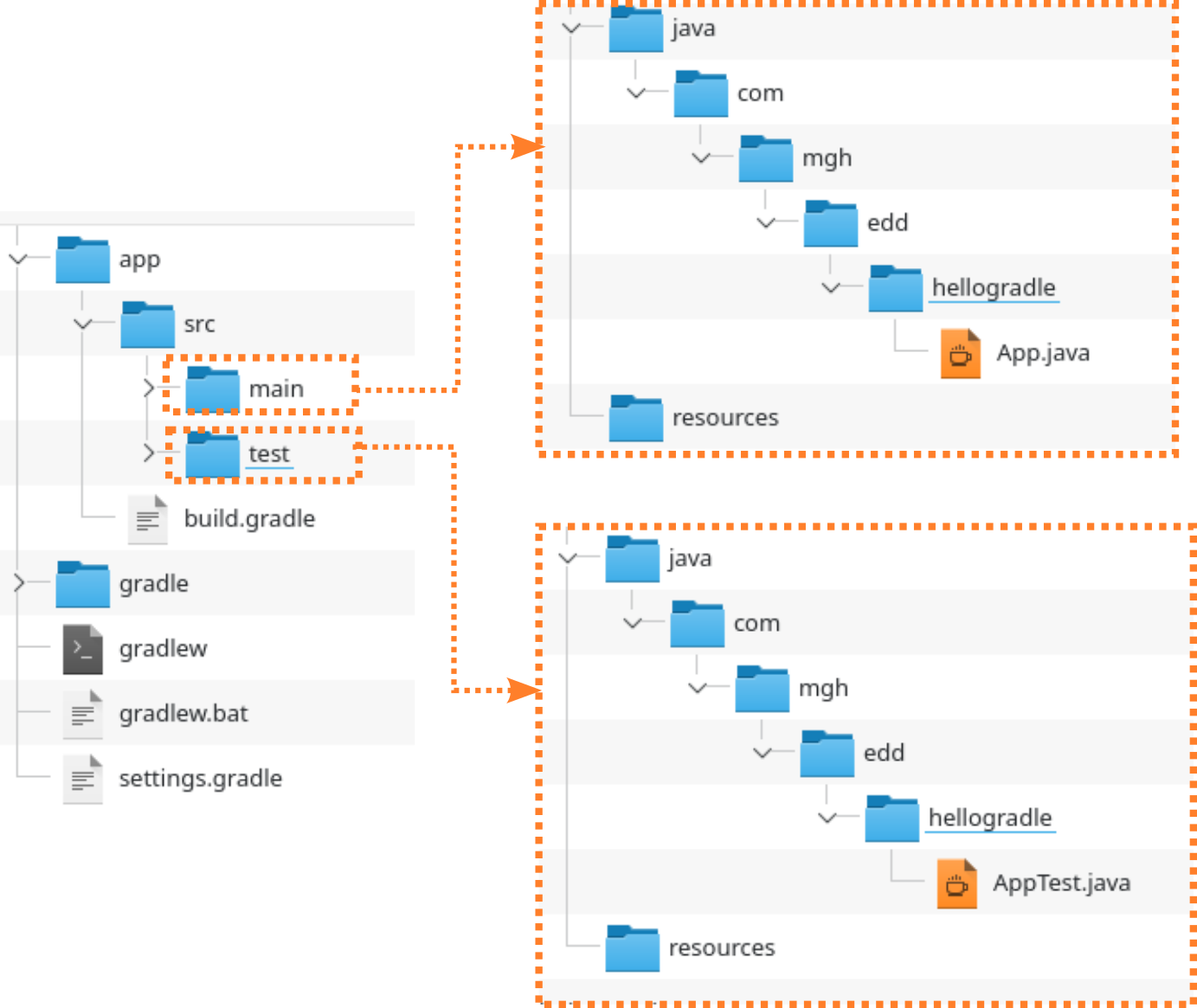
**--\*dsl \*groovy \**

**--\*project-\*name \*helloGradle \**

**--\*package \*com.\*mgh.\*edd.\*hellogradle** 

* Analitzant l'estructura del projecte

Hàgem procedit bé amb l'assistent o bé generant directament el projecte, este presenta la següent estructura de fitxers i directoris:



Com veiem, tenim la carpeta ***app*** amb la carpeta de codi de l'aplicació (*\*src*), que al seu torn conté les carpetes *\*main* amb el codi principal, i *test* amb el codi dels tests. Com podeu veure, dins d'eixes carpetes tenim la carpeta \*java amb l'estructura de carpetes corresponent al paquet de l'aplicació, i amb el fitxer font *App.\*java* per al codi i *\*AppTest.\*java* per als tests.

Centrant-nos en la part del codi, si observem el contingut del fitxer App.\*java, veurem que es tracta, encara que expressat utilitzant funcions, del codi per a mostrar un missatge d'Hola  *Món* per pantalla.

/\*

\* \*This Java \*source \*file \*was \*generated \*by \*the \*Gradle '\*init' \*task.

\*/

\*package \*com.\*mgh.\*edd.\*hellogradle;

\*public \*class App {

\*public \*String \*getGreeting() {

\*return "\*Hello \*World!";

}

\*public \*static \*void \*main(\*String[] \*args) {

\*System.out.\*println(\*new App().\*getGreeting());

}

}

L'altre fitxer important dins d'esta carpeta ***app*** és el fitxer *\*build.\*gradle*, que és el \*script de construcció de l'aplicació:

/\*

\* \*This \*file \*was \*generated \*by \*the \*Gradle '\*init' \*task.

\*

\* \*This \*generated \*file \*contains a \*sample Java \*application

\* \*project \*to \*get \*you \*started.

\* \*For habite \*details \*take a look \*at \*the '\*Building Java & \*JVM

\* \*projects' \*chapter \*in \*the \*Gradle

\* \*User Manual \*available \*at

\* https://docs.gradle.org/7.2/userguide/building\_java\_projects.html

\*/

\*plugins {

// \*Apply \*the \*application \*plugin \*to \*add \*support \*for \*building a

// \*CLI \*application \*in Java.

aneu '\*application'

}

\*repositories {

// Use \*Maven Central \*for \*resolving \*dependencies.

\*mavenCentral()

}

\*dependencies {

// Use \*JUnit test \*framework.

\*testImplementation '\*junit:\*junit:4.13.2'

// \*This \*dependency \*is \*used \*by \*the \*application.

\*implementation '\*com.\*google.\*guava:\*guava:30.1.1-\*jre'

}

\*application {

// Definix \*the \*main \*class \*for \*the \*application.

\*mainClass = '\*com.\*mgh.\*edd.\*hellogradle.App'

}

Com podem veure, es tracta d'una sintaxi relativament intuïtiva, en la qual definim uns certs paràmetres dins de diferents seccions:

1. **\*plugins**: \*Gradle en si proporciona molt poques funcionalitats d'automatització, i estes s'oferixen mitjançant \*plugins. En este cas, s'inclou el \*plugin *\*application*, per a afegir suport a la construcció d'aplicacions JAVA en mode \*CLI.
2. **\*repositories**: En esta secció es carreguen els diferents repositoris per a descarregar llibreries i dependències. Per defecte, veiem que carrega *\*mavenCentral()*, ja que \*Gradle és compatible amb totes les llibreries de \*Maven.
3. **\*dependencies**: #Ací especificarem les llibreries de les quals depén el nostre projecte. En este cas, únicament depén d'un parell de llibreries necessàries per als tests i l'aplicació, però podem afegir tantes dependències com desitgem. Quan busquem una llibreria en \*Maven Central, ens indicarà com afegir-la a este fitxer.
4. **\*application**: Definix alguns aspectes sobre l'aplicació. En este cas, indica el valor *\*mainClass*, amb el nom completament qualificat de la classe principal. D'esta manera, \*Gradle sap ja quina classe ha de carregar quan executem l'aplicació.

A més del contingut en la carpeta ***app***, en l'arrel del projecte tenim alguns fitxers més d'interés:

1. **\*settings.\*gradle**: Conté la configuració del projecte o projectes dels quals consta l'aplicació. En este cas, únicament s'inclou l'aplicació *\*HelloGradle*, que es troba e la carpeta *app*.
2. **\*gradlew**, **\*gradlew.\*bat**, i carpeta **\*gradle/\*wrapper**: El \*wrapper de \*Gradle és una instal·lació portable de \*Gradle dins del projecte, de manera que no es requerix, per exemple, tindre \*Gradle instal·lat en el nostre equip. A més, això proporciona també una versió fixa de \*Gradle per al projecte, de manera que no hi haja \*incomptibilidades entre versions.

**Què és més recomanable usar el comando \*gradle o el \*wrapper?**

Per a la creació de projectes (*\*gradle \*init*), així com per a la generació d'un nou \*wrapper (*\*gradle \*wrapper*) \*utilzaremos el comando *\*gradle (*o *\*gradle.\*bat* en Windows), mentres que per a la construcció i execució del projecte, seria preferible utilitzar el \*wrapper (*\*gradlew i \*gradlew.\*bat)*

* Construcció i execució del projecte

Una vegada hem vist l'estructura d'un projecte, podem construir-lo amb el comando *\*gradle \*build*, des de la carpeta principal del projecte:

$ **\*gradle \*build** 

\*BUILD \*SUCCESSFUL \*in \*10s

7 \*actionable \*tasks: 7 \*executed

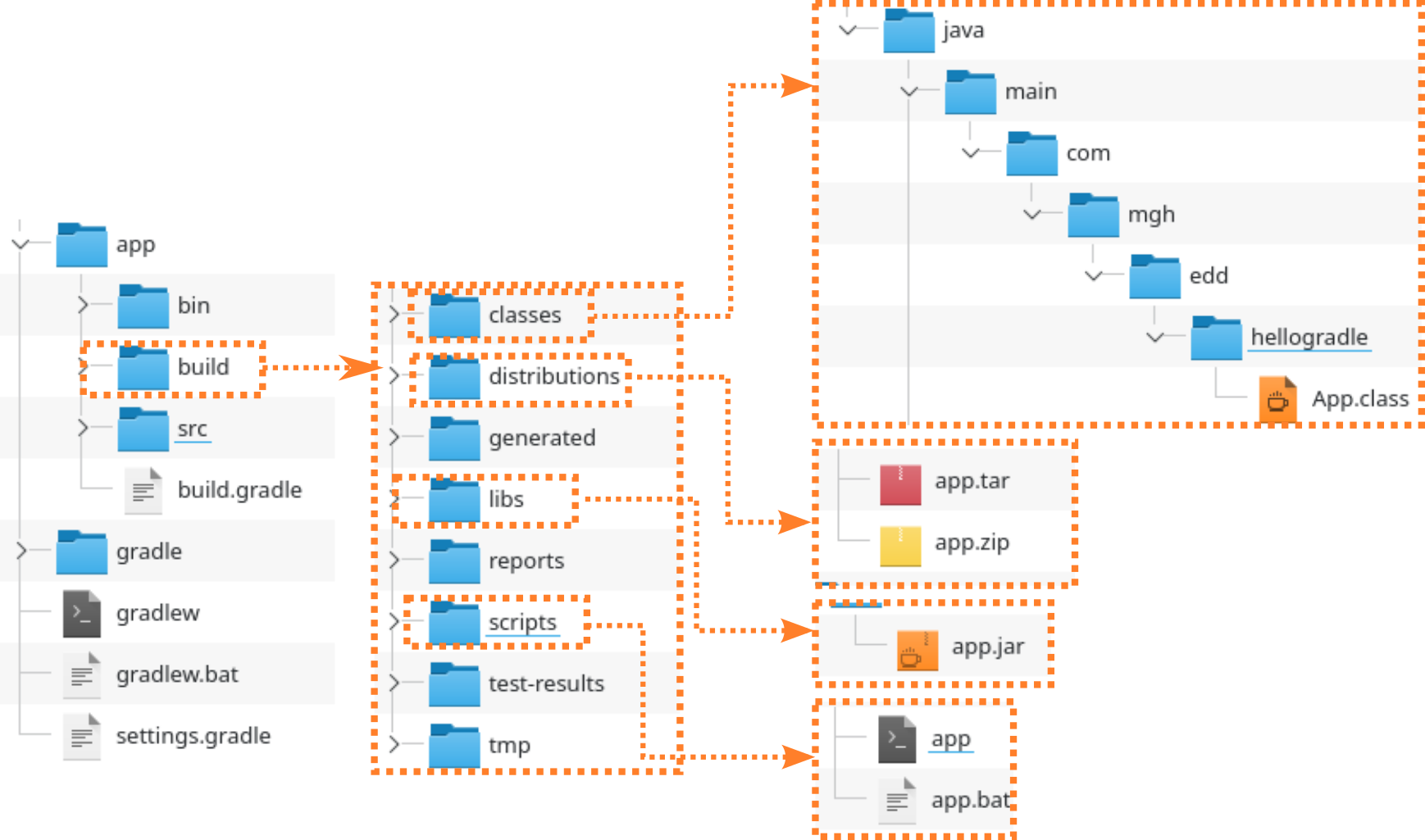
O bé mitjançant el \*wrapper de \*Gradle:

$ **./\*gradlew \*build** 

\*BUILD \*SUCCESSFUL \*in \*3s

7 \*actionable \*tasks: 7 \*up-\*to-dona't

Amb això, 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:



En esta carpeta *\*build,* \*deestacamos:

1. La carpeta *\*classes*, amb els .\*class generats,
2. La carpeta *\*distributions*, amb l'aplicació comprimida en \*tar i en \*zip,
3. La carpeta *\*libs* amb l'empaquetat en *\*jar*,
4. La carpeta *\*scripts*, amb els \*scripts per a llançar l'aplicació.

Si volem veure les diferents fases per les quals passa la construcció de l'aplicació, podem utilitzar l'opció `-i` de \*Gradle:

$ **\*gradle \*build -i** 

o bé, amb el \*wrapper:

**./\*gradlew \*build -i** 

L'eixida d'este comando serà bastant extensa, i mostrarà el resultat de l'execució de cada fase. A mode il·lustratiu vam mostrar esta eixida retallada, indicant únicament les fases per les passa el procés:

...

> Configure \*project :

...

> Configure \*project :app

...

> \*Task :app:\*compileJava

...

> \*Task :app:\*processResources NO-\*SOURCE

...

> \*Task :app:\*classes

...

> \*Task :app:\*jar

...

> \*Task :app:\*startScripts

...

> \*Task :app:\*distTar

...

> \*Task :app:\*distZip

...

> \*Task :app:\*assemble

...

> \*Task :app:\*compileTestJava

...

> \*Task :app:\*processTestResources NO-\*SOURCE

...

> \*Task :app:\*testClasses

...

> \*Task :app:test

...

> \*Task :app:\*check

...

> \*Task :app:\*build

Com veiem, en primer lloc es compilen els fonts, després s'incorporen els recursos (que en este cas no existixen), es creen les classes, l'empaquetat \*jar, els \*scripts, els arxius \*distribuibles, i posteriorment es compilen els tests i s'apliquen.

**Neteja del projecte**

D'altra banda, si volem netejar el projecte, eliminant esta carpeta *\*build*, utilitzarem el comando *\*clean* de \*Gradle:

$ **\*gradle \*clean** 

o bé mitjançant el \*wrapper:

$ **./\*gradlew \*clean** 

**Execució**

Finalment, amb la finalitat d'executar l'aplicació, haurem d'invocar el comando *\*run* de \*Gradle:

$ **\*gradle \*run** 

> \*Task :app:\*run

\*Hello \*World!

\*BUILD \*SUCCESSFUL \*in \*3s

2 \*actionable \*tasks: 1 \*executed, 1 \*up-\*to-dona't

o bé, mitjançant el \*wrapper:

$ **./\*gradlew \*run** 

> \*Task :app:\*run

\*Hello \*World!

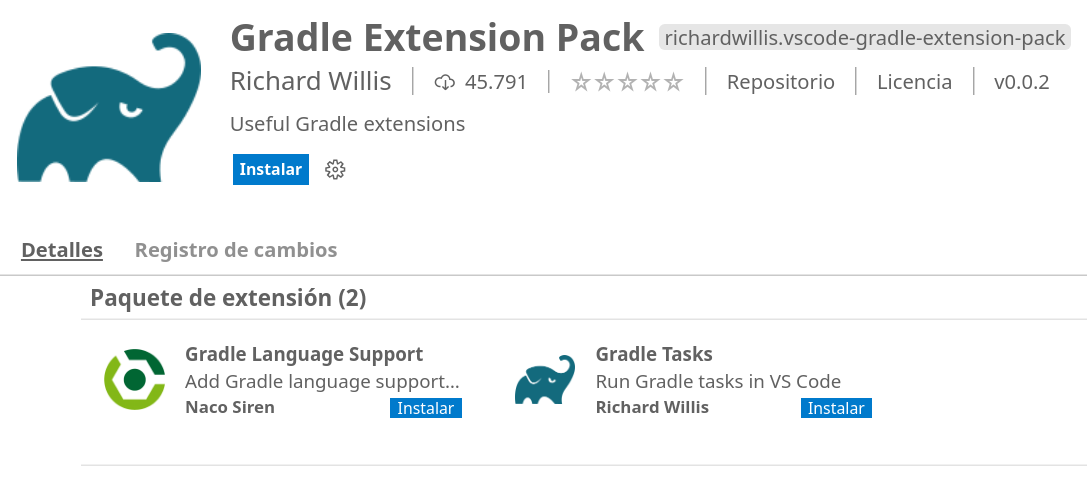
\*BUILD \*SUCCESSFUL \*in \*2s

2 \*actionable \*tasks: 1 \*executed, 1 \*up-\*to-dona't

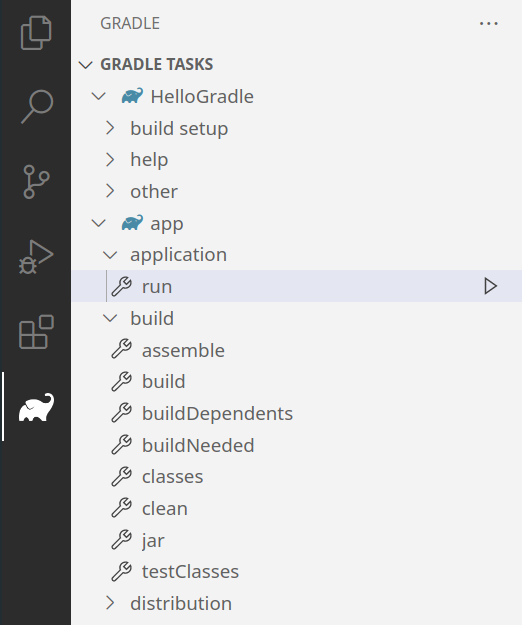
* \*Gradle i Visual \*Studio \*Code

En Visual \*Studio \*Code disposem del complement *\*Gradle \*Extension \*Pack*, que es compon de les extensions \*Gradle \*Language \*support, amb suport a la sintaxi de \*Gradle i ajuda a la generació de l'arxiu, així com de \*Gradle \*Tasks, que ens permet executar les diferents tasques.

Per a instal·lar la \*exstensión \*sólamente haurem de buscar-la en l'activitat d'extensions i fer clic en el botó de \*Install.

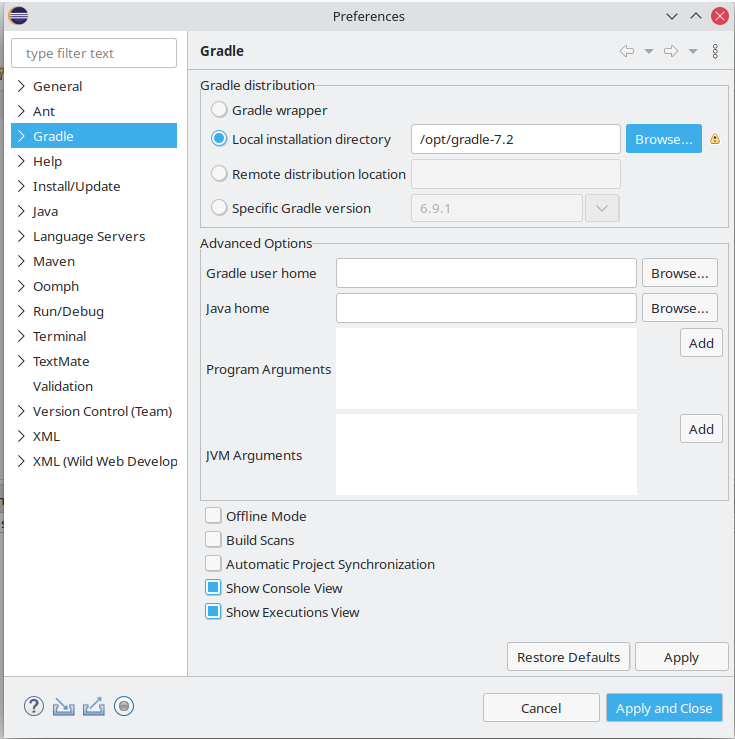


Una vegada instal·lades, l'extensió \*Gradle \*Language \*Support ens permetrà ressaltar la sintaxi dels \*scripts de construcció, a més d'aportar suggeriments i detectar duplicats. Per part seua, \*Gradle \*Tasks ens mostrarà una nova activitat en la interfície de Visual \*Studio \*Code, des de la qual podrem veure en forma d'arbre i llançar totes les tasques relacionades amb el projecte:

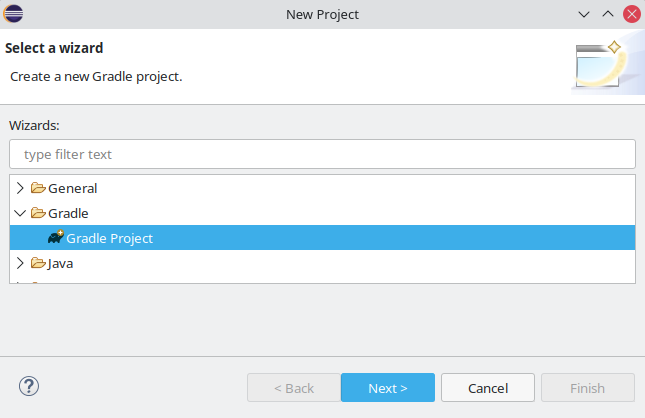


* \*Gradle i Eclipsi

Eclipsi suporta també projectes \*Gradle, i a més ens permet seleccionar, mitjançant la finestra de preferències (*\*Window > \*Preferences*), si desitgem utilitzar el \*Wrapper de \*Gradle, una versió específica, o indicar una instal·lació local:

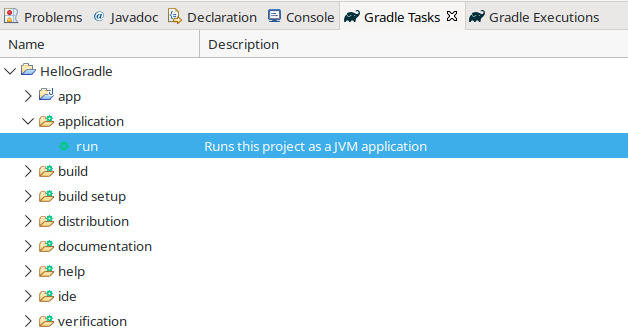


A més, quan creguem un nou projecte, tenim la possibilitat d'indicar que \*seá un projecte de tipus \*Gradle, encara que #ací l'estructura generada per defecte serà la d'una llibreria (*aneu ‘\*java-\*library’*).



També tenim la possibilitat d'importar un projecte \*Gradle mitjançant *\*File > \*Import*.

Una vegada tenim un projecte \*Gradle carregat, tindrem disponibles les pestanyes de \*tares i execució de \*Gradle, des de les quals podrem llançar les diferents tasques i veure el resultat de la seua execució:



1. Pàgina oficial: [https://marketplace.visualstudio.com/items?itemname=nickheap.vscode-ant](https://marketplace.visualstudio.com/items?itemName=nickheap.vscode-ant) .

   Pàgina en \*gitHub: [https://github.com/nickheap2/vscode-ant](https://github.com/NickHeap2/vscode-ant) [↑](#footnote-ref-2)