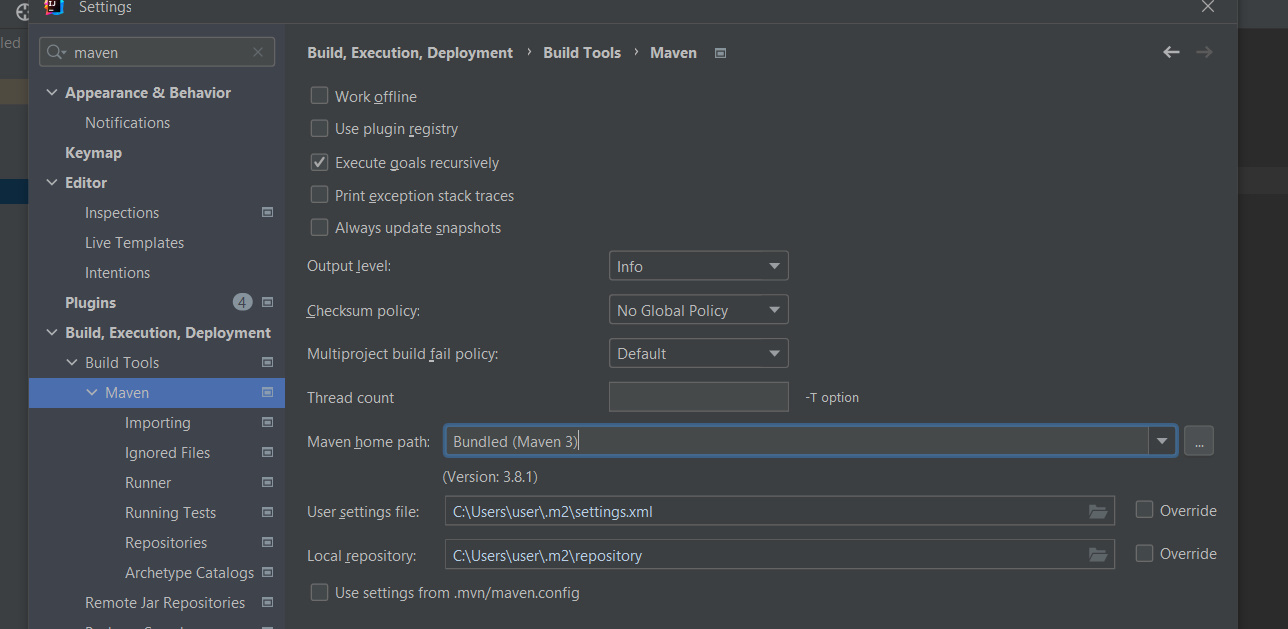


* Maven ofera o structura a proiectului standarta
* target – contine rezultatele procesului de executie a lui Maven, care compileaza codul.
* Maven este si el un fel de plugin ce foloseste multe alturi pluginuri pentru orice actiune ce o face.
* **Plugin** – modul .jar ce poate fi folosit de Maven. E o bucata de cod ce nu e standarta, ci face ceva concret si poate fi inclus.
* **Archetype** – structura la un anumit proiect.

**Instalare**

* Daca punem Maven in Path in variables, comenzile vor fi disponibile in consola
* Mai e necesar sa cream chiar o variabila M2\_HOME ca alte aplicatii sa il poata gasi
* In proiecte este adesea necesar sa modificam Maven pe care Intellij il foloseste. Intellij foloseste propriul maven, dar daca vrem sa il setam pe altul, ne ducem aici:





Anume din M2\_HOME din variables ia Intellij Maven



**Coordinates**

* **groupId** – e ca o familie, sau nume la firma, intreprindere, proiect etc. Se foloseste org.nume de ex.
* **ArtifactId** – numele la proiect concret,
* **Version** – versiunea proiectului. Numele la versiuni, ca SNAPSHOT, sunt importante pentru maven. RELEASE e versiune stabila, testata, dar SNAPSHOT inseamna ca e inca in dezvoltare.

**Cum functioneaza Maven**

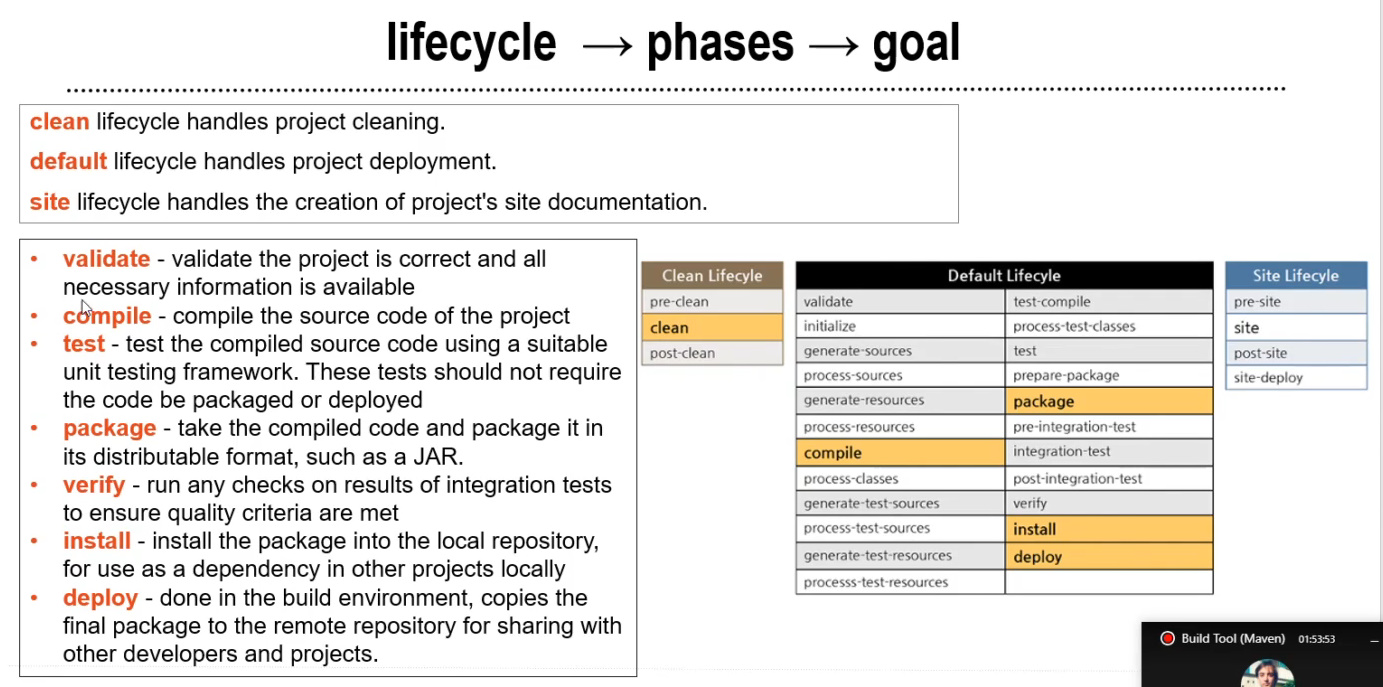
Atunci cand maven da compile sau build la un proiect, el trece 3 etape(**LifeCycle**):

1. **clean** – curata toata informatia inutila. Adica, curata totul din target, sterge fisiere generate nenecesare.
2. **default** – maven face actiuni pentru a crea proiectul
3. **site** – genereaza informatiile, ca documentatia, ca site

* Orice pom creat de noi va mosteni de la parent pom. Acest parent pom are toate configuratiile necesare ce ar trebui puse in pom.xml. In mod default, pom.xml al nostru mosteneste de la el.

**Plugins**

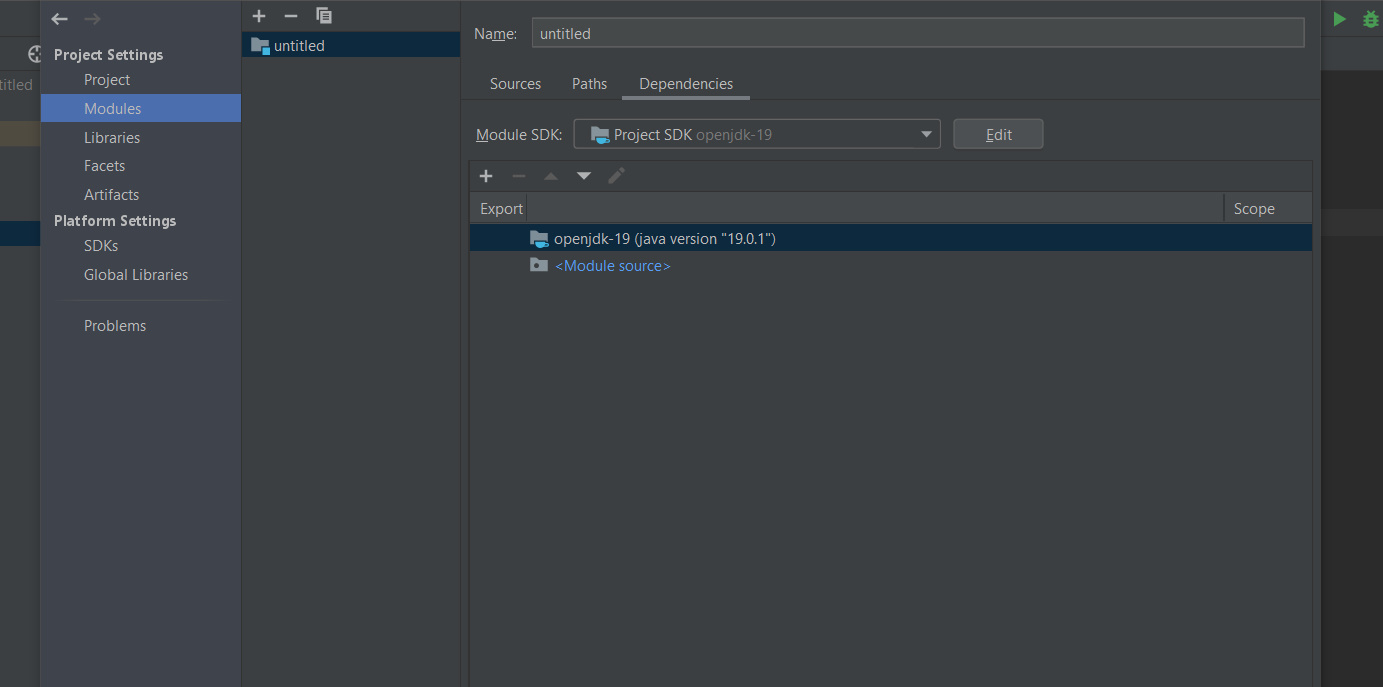
* Maven are o arhitectura orientata pe plugin. Orice maven face, o face prin pluginuri
* Maven cand compileaza codul, pune in target totul despre el.Pe baza la ce este in target se face totul in jar, si daca nu vom da mvn clean test, cand vom crea jarul pot sa ramana chestii vechi de care nu avem nevoie de la compilarile anterioare. Daca unele deja exista, maven va da override, dar nu le va sterge pe cele inutile.
* plugin = dependency
* Maven plugins repo:
* **packaging** – war jar pom rar
* **integration** – IDEA, Eclipse etc.
* **tools**
* **reporting** – JavaDoc reports etc.



Validate verifica structura proiectului. Verify integration tests

**Maven cu Intellij**

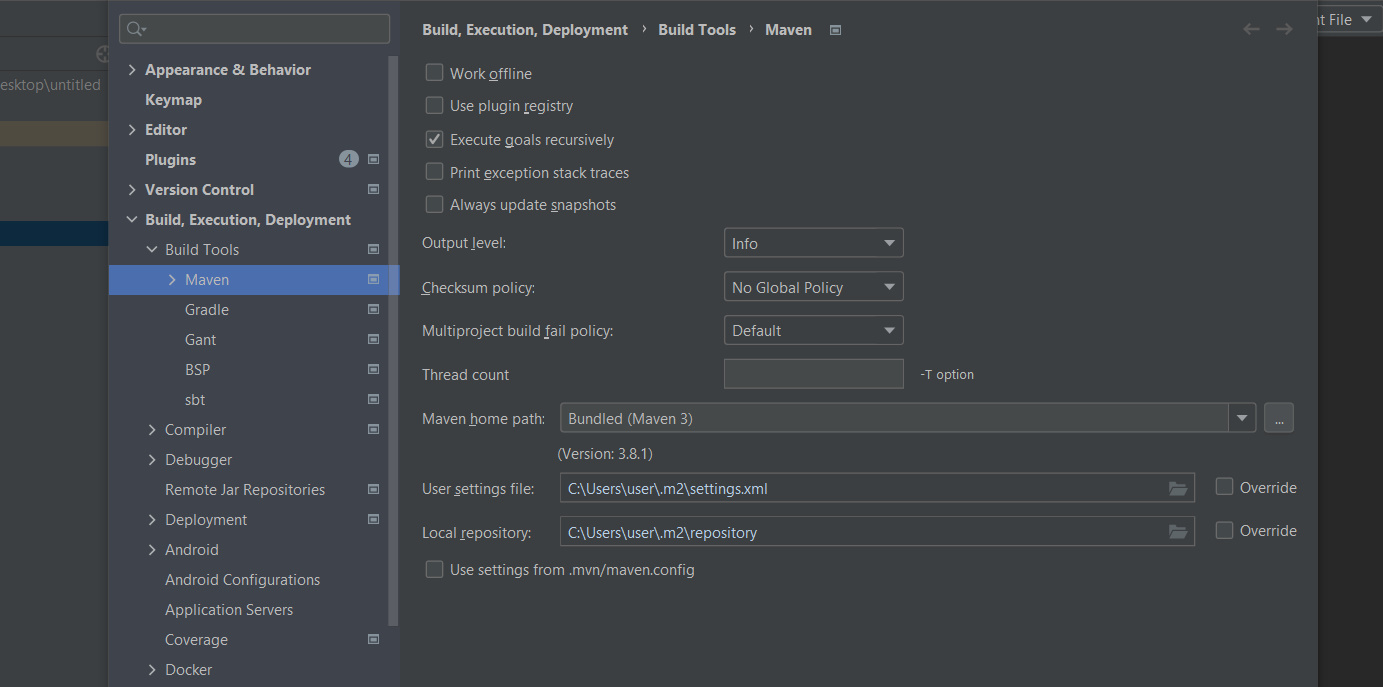
* Maven nu depinde de IDE
* In .gitignore se pun mere fisierele ce tin de configurarea la IDE sau Maven
* Intellij poate lucra si fara Maven, dar se lucreaza oricum si cu Maven
* Intellij nu poate importa dependentele automatizat, si asta nu e deloc comod.
* E logic ca Intellij nu poate intelege tot ce e in pom.xml, dar anumite chestii le poate intelege si face actiuni, dar nu cunoaste tot ce e in pom.xml
* Totusi, Intellij lucreaza mai bine cand avem mai multe JDK in PC. Putem noi alege usor ce JDK vrem si de unde, dar maven mereu se uita la JAVA\_HOME din variables properties.
* In Project Settings, la Modules, putem da click pe Import Module si asa sa lucram cu mai multe proiecte in parallel



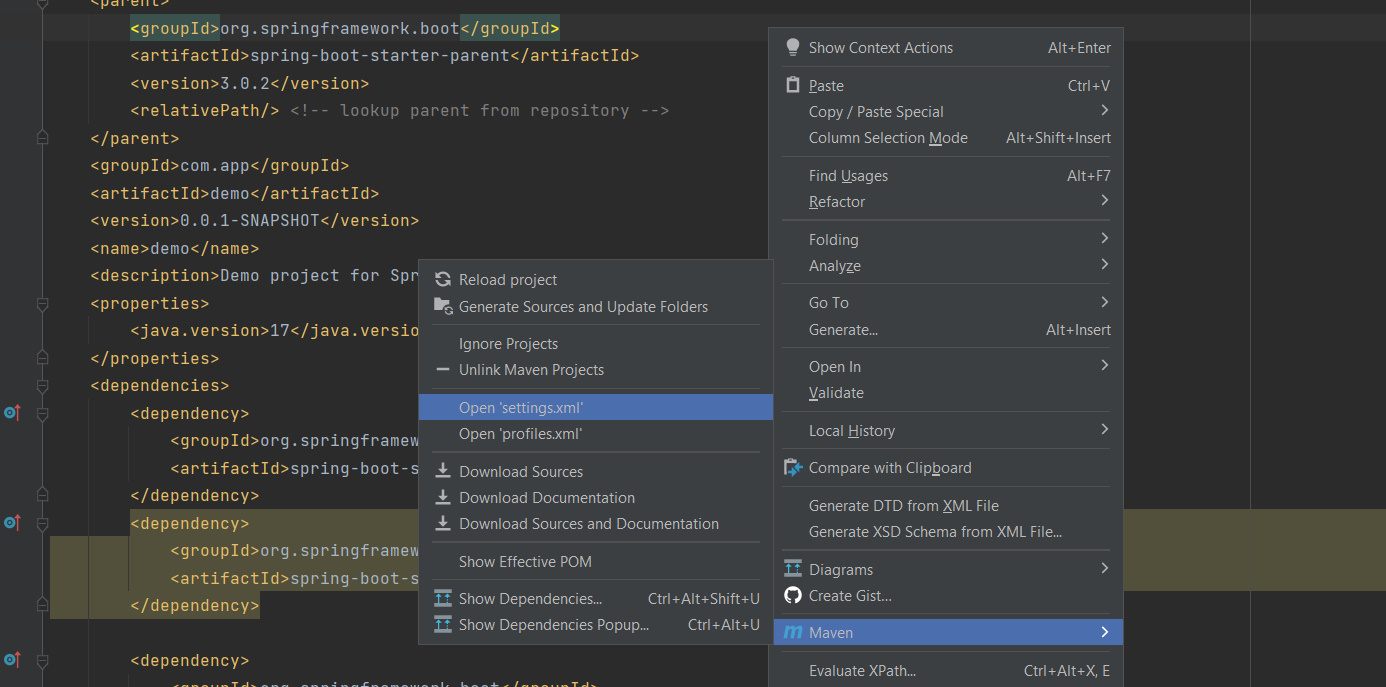


Apoi putem alege daca importam ca Maven project sau sa lasam Intellij sa importe cumva proiectul si apoi vom alege unde se afla Source,Resource etc.

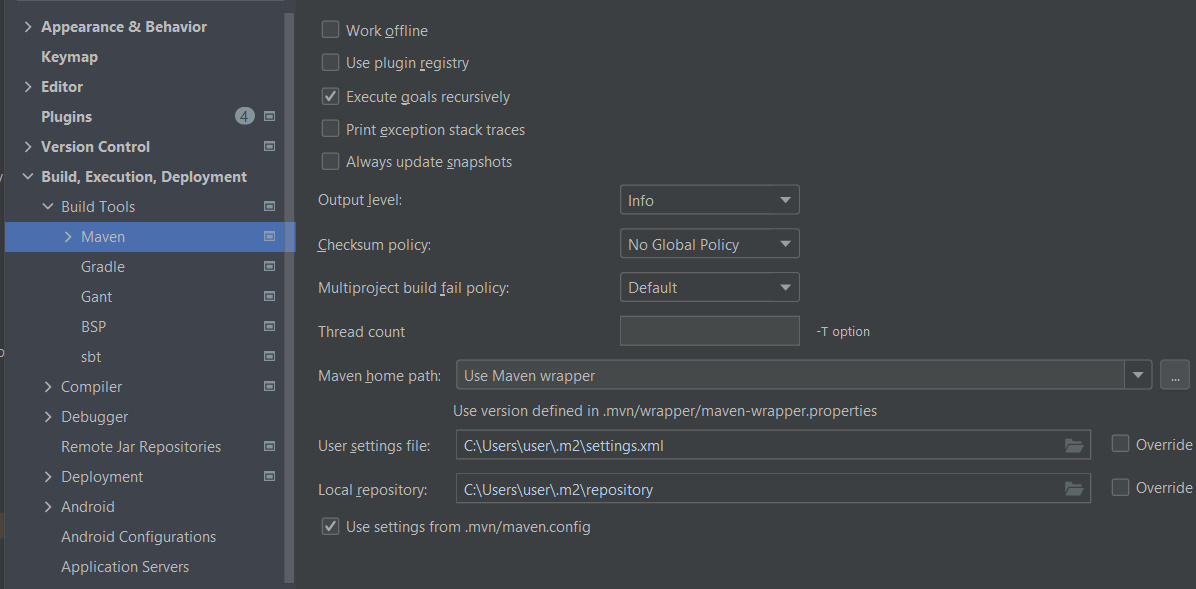
* O sectiune importanta din Intellij este cea a lui Maven:



* settings.xml reprezinta un fisier unde se pastreaza difertie configurari generale. De multe ori va fi necesar sa il modificam in proiecte. Acest fisier contine setari si ele pot fi folosite pentru mai multe proiecte, deci reprezinta niste setari generale, ce pot fi folosite in mai multe poriecte.Pentru a-l deschide, intram in pom.xml, click dreaptam,Maven si :

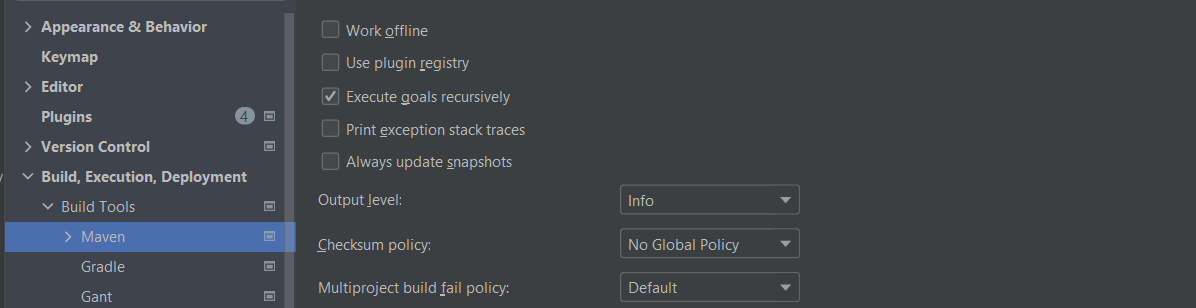


Putem modifica insa fisierul folosit:



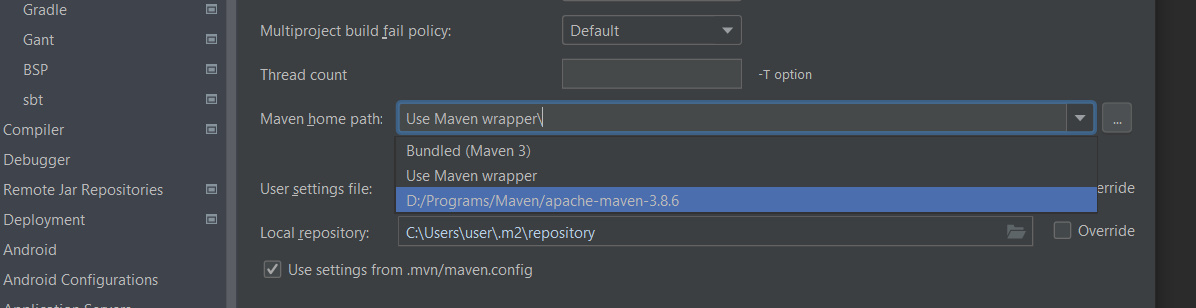


Si asa un fisier settings.xml poate fi folosit de mai multe proiecte, spre deosebire de pom.xml care e doar pentru un proiect. Nu ar avea sens sa tot punem aceste setari in toate pom.xml, mai bine avem un fisier cu setari pentru mai multe poriecte

* 

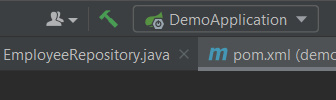
Work offline – nu cauta dependente in remote repository

* In intellij putem seta cu care Maven lucram:



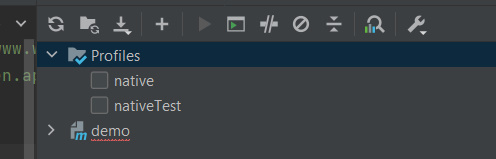
Dar atentie! Cand lucram cu command line, chiar si cel din Intellij, se va deschide Maven setat in variables, nu cel ales in Intellij!Deci atentie ca comenzile cu Maven din Intellij executate pot sa se comporte diferit de cele executate din command line. Asa dar, settings.xml setat de noi in Intellij sunt doar valabile cand lucram cu Mavem in Intellij, nu in command line!

* Tot ce scriem in pom.xml se refera la cum va fi creat artefactul
* Cand cream un Spring Boot project, compilarea, injectarea de dependente(din pom.xml) nu e facuta de Mavem, ci de Intellij. Deci, cand rulam acel SpringBoot starte, adica acesta:



nu Mavem va crea .jar project, si nici Intellij nu o va face, dar Intellij va rula aplicatia asa pe baza la toate jar-urile din ea, fara a crea acel .jar file. El se va consulta cu Maven, dar nu Maven se va ocupa de tot. Asa dar, si Intellij citeste pom.xml

* Din cauza asta, comportamentul aplicatie rulata de Intellij poate diferentia de cel rulat de Maven
* **Profile** - A profile in Maven is an alternative set of configuration values which set or override default values. Using a profile, you can customize a build for different environments. Profiles are configured in the pom. xml and are given an identifier.Deci, cu profile, putem modifica setarile default ale proiectului, dar fara a crea alt pom.xml, ci tot in acelasi
* **Deci, cand vom acele un profile:**



Acest profile, care de ex sa zicem ca s-ar numi oracle, ca mai jos, ar face ca toate dependentele si configuratiile ce sunt in pom.xml, mai sus de el, sa fie suprascrise de cele intre <profile>

