



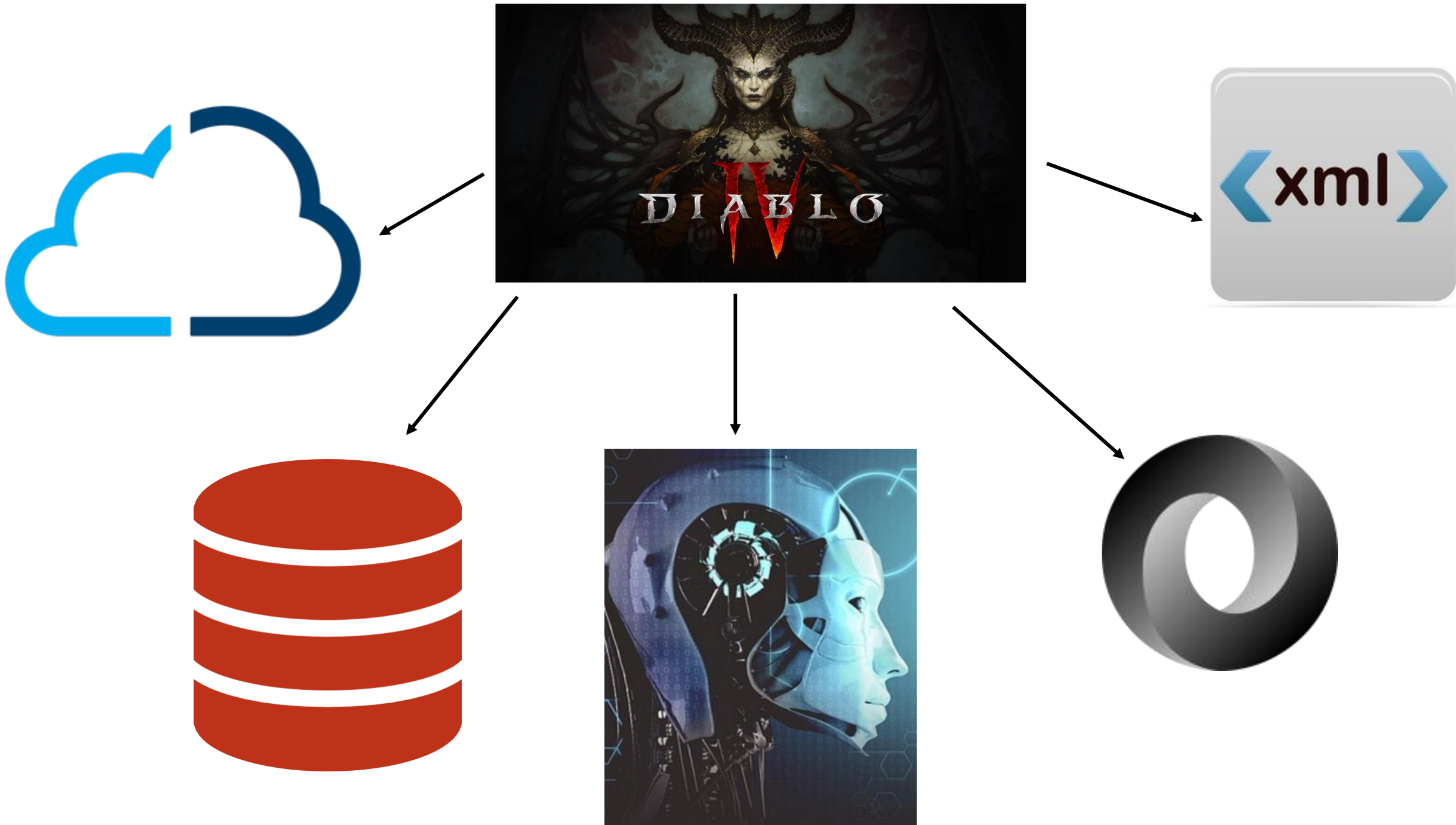
C125/C206 – Programação Orientada a Objetos  
com Java

# Automatização da Build com Maven

Prof. Phyllipe Lima  
phyllipe@inatel.br

🎮 Considere o desenvolvimento de um jogo (poderia ser qualquer software)





🎮 Você vai fazer o download dessas bibliotecas uma por uma?

🎮 E se alguma é atualizada? Uma nova versão com melhorias. Será necessário fazer o download novamente.

A Google search bar with the text "baixar dependências na mão" entered. It includes a magnifying glass icon on the left, a close button (X) on the right, and a keyboard icon on the far right.

Pesquisa Google

Estou com sorte

🎮 E sua equipe? Alguém vai baixar a dependência nova e enviar por e-mail para os demais?





🎮 Ou pior.....vai usar pen-drive?



🎮 E pra fazer o *build*? Com todas essas dependências?



BUILDER

🎮 Build é o processo de geração do software como produto (compilação?). Envolve, geralmente, as etapas de: teste automatizado, compilação e empacotamento (jar , por exemplo)

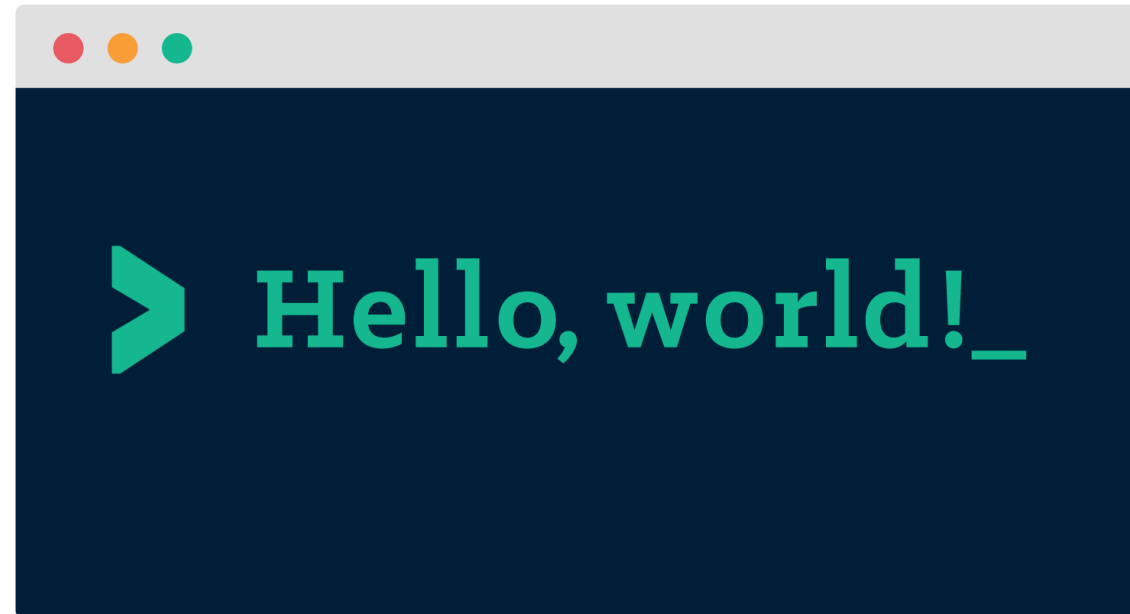


BUILDER





- 🎮 Build – Fases para a construção do software
- 🎮 Programas pequenos, apenas compilar é o suficiente!
- 🎮 Exemplo: “Hello World”

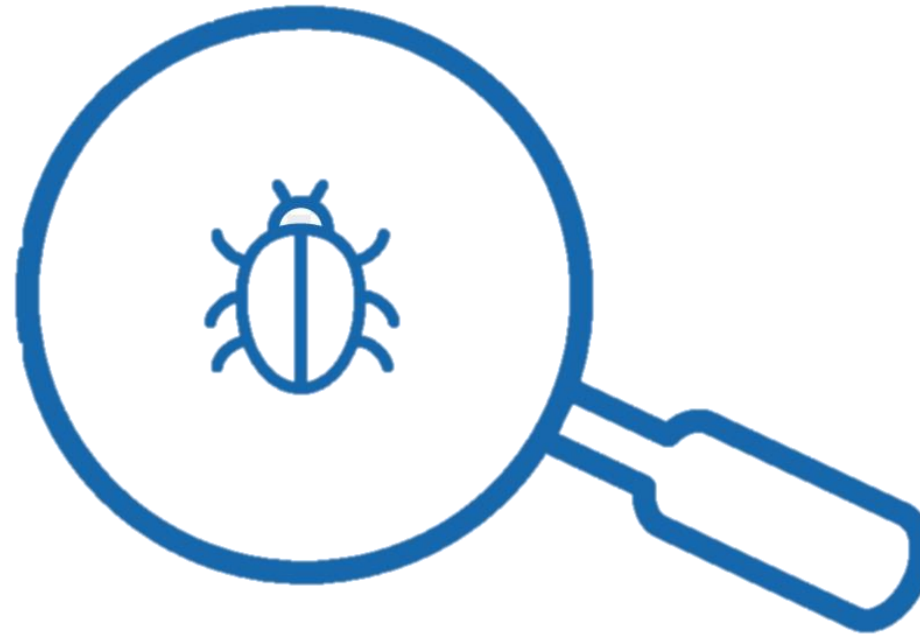




BUILDER

🎮 Software real requer mais etapas para “construir” com qualidade!

🎮 Testes!





BUILDER



Empacotar!

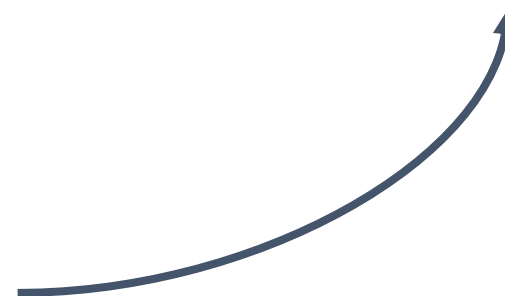
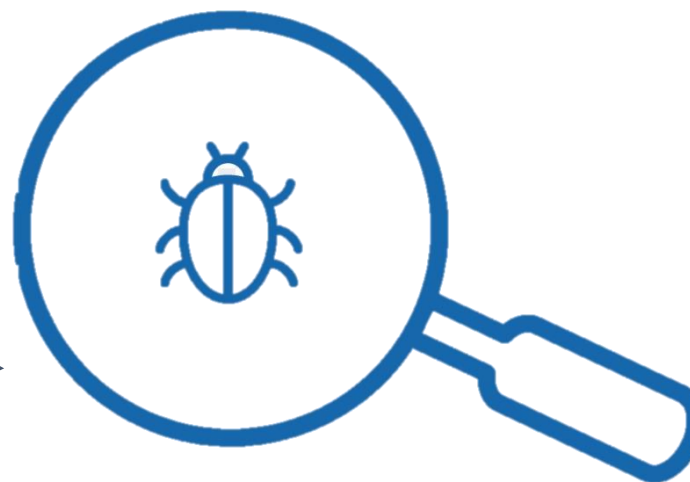
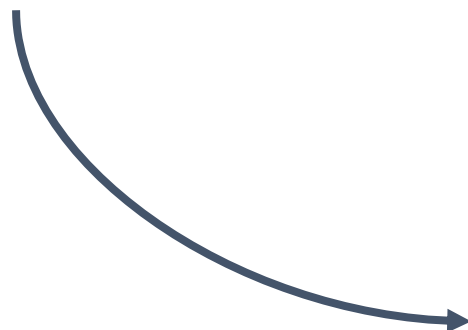


Gerar um pacote com tudo necessário para instalação do software.

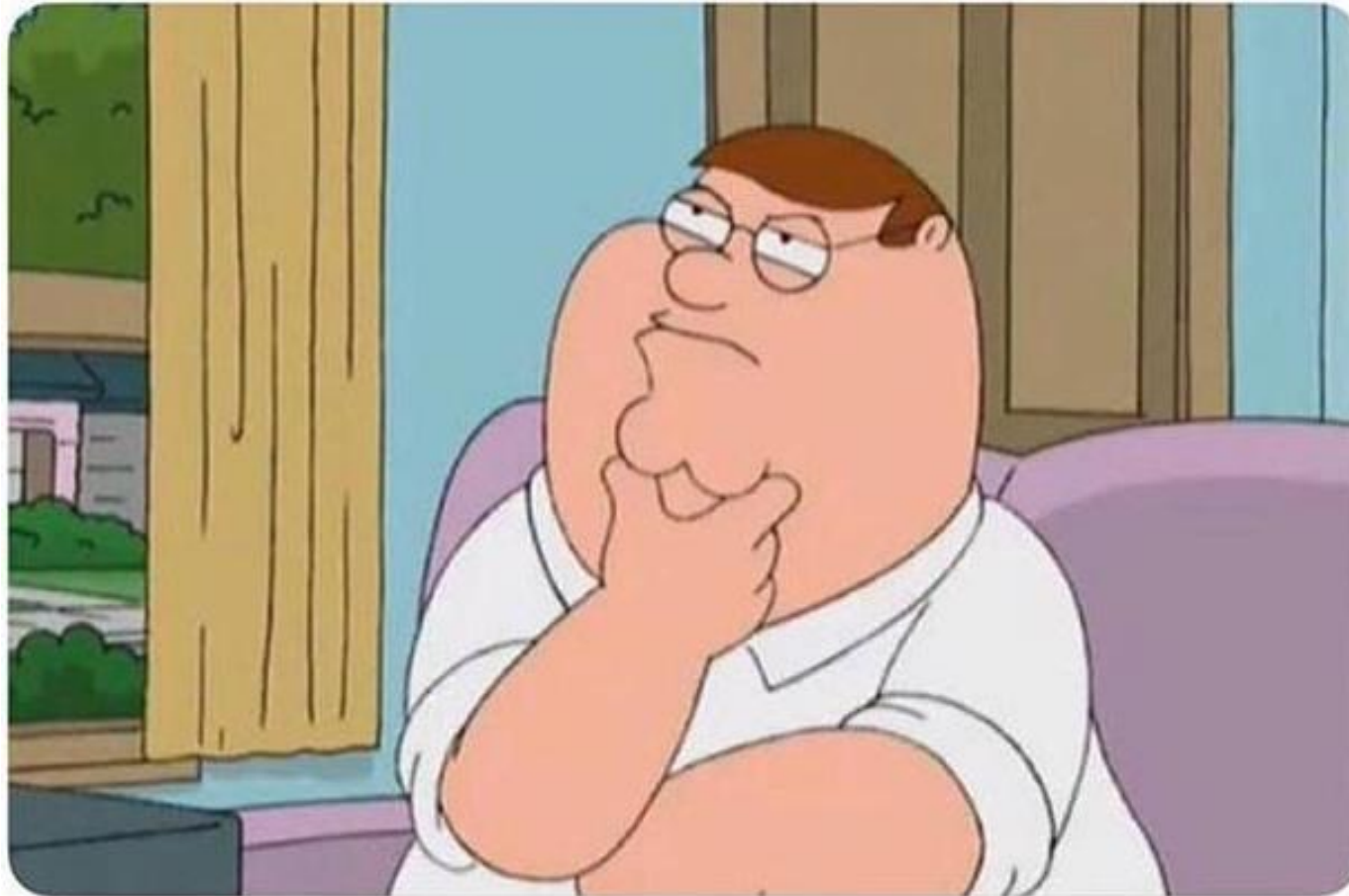


Exemplo: Jar





🎮 Tem como automatizar esse processo?







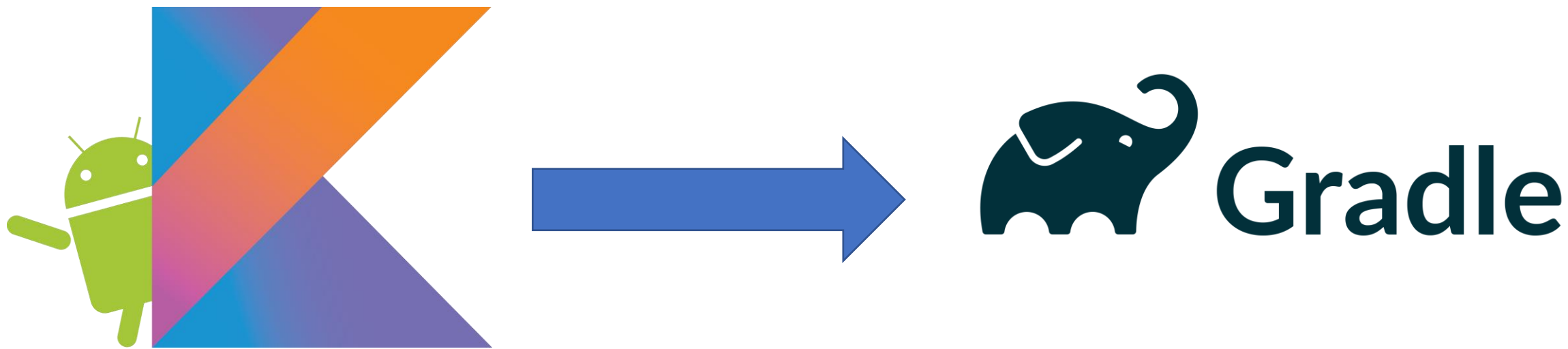
 Automação da Build



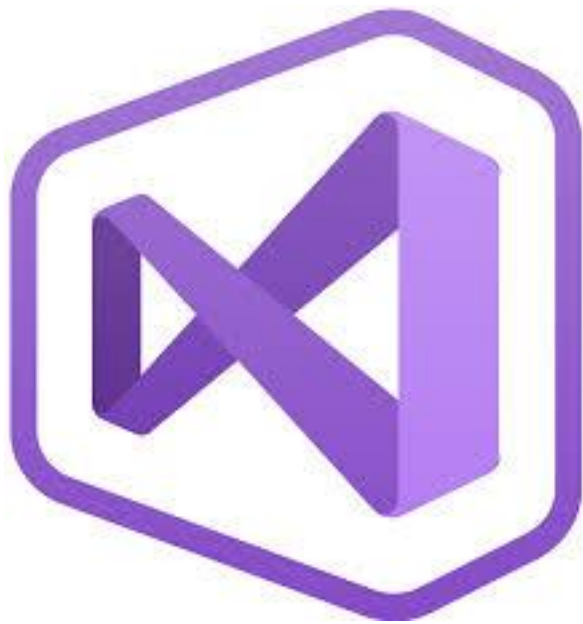
***Maven***<sup>TM</sup>



🎮 OBS: O Nuget faz apenas a  
gerência de dependências!



- 🎮 O Gradle é *cross-platform* e pode ser usado com diversas linguagens, inclusive Java.
- 🎮 Ele é totalmente baseado no Maven, e compartilham várias características!
- 🎮 É o padrão em projetos Android



MSBuild

🎮 O Visual Studio já está totalmente integrado com o Nuget (gerenciador de pacotes) e o MSBuild (automatização da build)



Para nosso curso usaremos...





🎮 Vamos explorar dois aspectos do Maven!

🎮 Dependências!

🎮 Fazer a build!

🎮 Começando pelas dependências.

🎮 Onde elas ficam?

🎮 O que fazem?


🎮 Como achamos?

🎮 Como usamos?



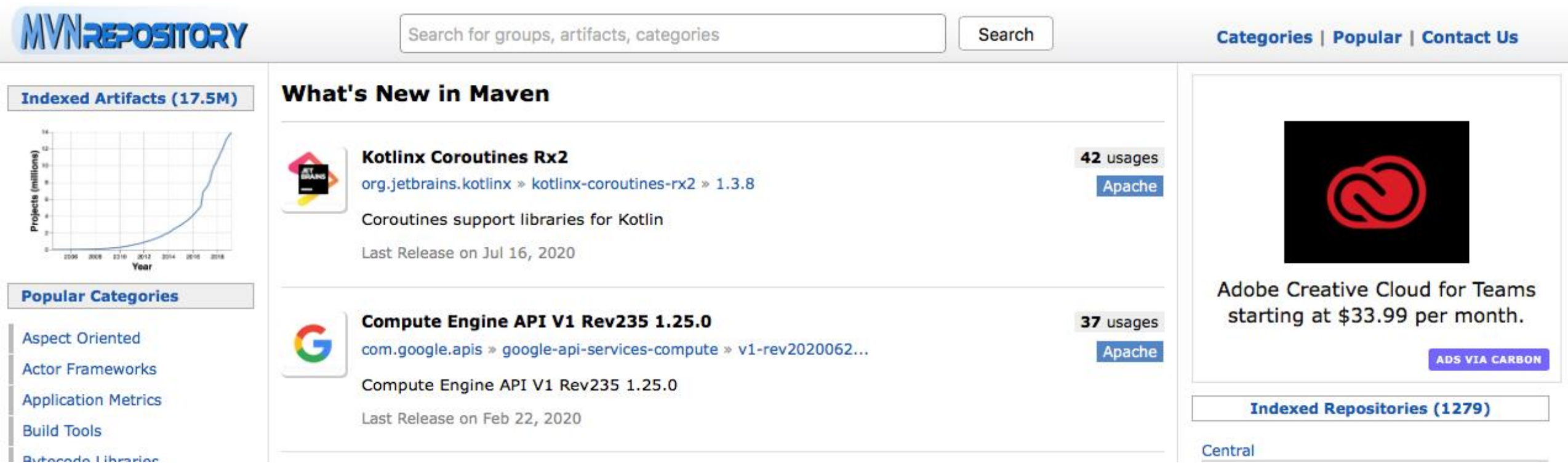
 Bem vindo ao repositório central!

# The Central Repository

 O repositório central é um local onde estão armazenados todos os artefatos (jar) de software disponibilizados pelo Maven

🎮 É onde se encontram as bibliotecas, frameworks, ferramentas e qualquer outro software que desejamos utilizar em nossas próprias soluções.

🎮 Acessamos por <https://mvnrepository.com>



The screenshot shows the Maven Repository website. At the top, there's a search bar with the text "Search for groups, artifacts, categories" and a "Search" button. To the right of the search bar are links for "Categories", "Popular", and "Contact Us". Below the search bar, the page is divided into three main sections. On the left, there's a section titled "Indexed Artifacts (17.5M)" which includes a line graph showing the growth of projects from 2009 to 2018. Below the graph is a list of "Popular Categories" including Aspect Oriented, Actor Frameworks, Application Metrics, Build Tools, and Runtime Libraries. The middle section is titled "What's New in Maven" and lists two recent updates: "Kotlinx Coroutines Rx2" and "Compute Engine API V1 Rev235 1.25.0". Each update includes its version number, a link to the source, a brief description, and the last release date. On the right, there's a large advertisement for "Adobe Creative Cloud for Teams" starting at \$33.99 per month, with a "ADS VIA CARBON" button. At the bottom right, there's a section titled "Indexed Repositories (1279)" with a link to "Central".

**Indexed Artifacts (17.5M)**

Projects (millions)

Year

**Popular Categories**

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Runtime Libraries

**What's New in Maven**

**Kotlinx Coroutines Rx2** 42 usages Apache  
org.jetbrains.kotlinx » kotlinx-coroutines-rx2 » 1.3.8  
Coroutines support libraries for Kotlin  
Last Release on Jul 16, 2020

**Compute Engine API V1 Rev235 1.25.0** 37 usages Apache  
com.google.apis » google-api-services-compute » v1-rev2020062...  
Compute Engine API V1 Rev235 1.25.0  
Last Release on Feb 22, 2020

**Adobe Creative Cloud for Teams** starting at \$33.99 per month.  
ADS VIA CARBON

**Indexed Repositories (1279)**

Central



E como podemos buscar algum artefato nesse repositório?



Considere que queremos utilizar uma biblioteca capaz de converter instâncias Java para objetos JSON.

🎮 Já existe uma biblioteca para isso, chamada GSON. Vamos usar o Maven para gerenciar essa dependência!

🎮 Vamos acessar o repositório e buscar por GSON

The screenshot shows the Maven Repository website. At the top, the 'MVNREPOSITORY' logo is on the left. A search bar in the center contains the text 'gson', and a 'Search' button is to its right. Below the search bar, the text 'Found 419 results' is displayed. On the left side, there is a sidebar titled 'Repository' with a list of repository categories and their counts: Central (279), JCenter (99), Sonatype (95), Spring Plugins (50), Spring Lib M (49), ICM (12), and Spring Lib Release (10). The main content area shows the search results, sorted by 'relevance'. The first result is '1. Gson' by 'com.google.code.gson', with a link to 'gson'. It includes a Google logo icon, the text 'Gson', and 'Last Release on Oct 4, 2019'. To the right of the result, it shows '13,037 usages' and an 'Apache' license badge.

**MVNREPOSITORY**

gson Search

**Repository**

- Central 279
- JCenter 99
- Sonatype 95
- Spring Plugins 50
- Spring Lib M 49
- ICM 12
- Spring Lib Release 10

**Found 419 results**

Sort: **relevance** | popular | newest

 **1. Gson** **13,037 usages**

com.google.code.gson » gson **Apache**


Gson

Last Release on Oct 4, 2019



🎮 Normalmente escolhemos a última versão. A menos que exista alguma restrição de compatibilidade com o seu projeto

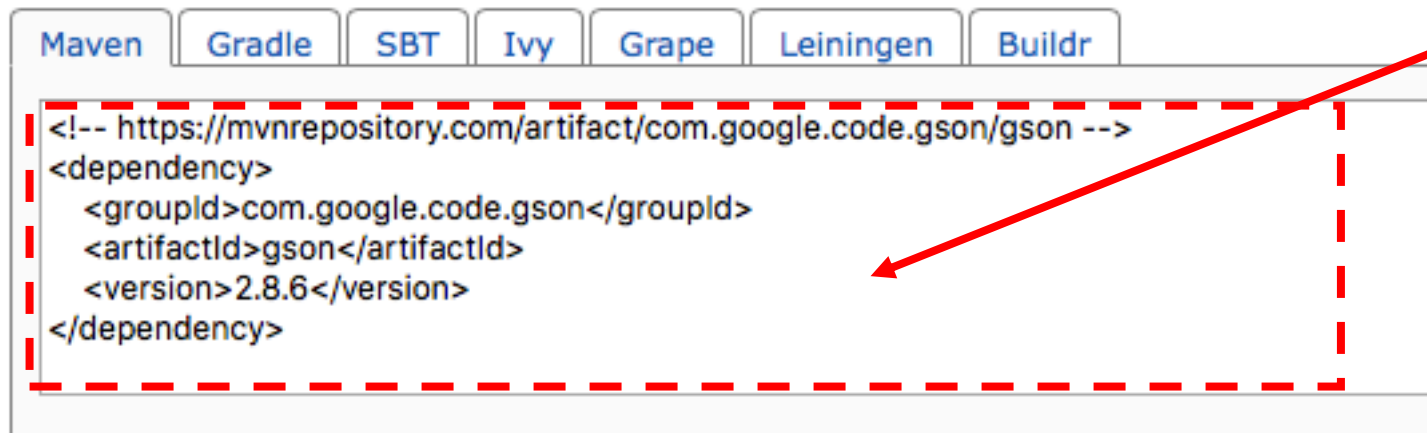
Nesse exemplo vamos usar a versão 2.8.6 da biblioteca GSON

 <b>Gson</b> Gson	
License	Apache 2.0
Categories	JSON Libraries
Tags	google json
Used By	13,037 artifacts

Central (29)	Atlassian 3rd-P Old (4)	Spring Plugins (4)	WSO2 Dist (1)	Redhat GA (3)
Redhat EA (1)	ICM (2)			
Version		Repository	Usages	Date
2.8.x	2.8.6	Central	1,216	Oct, 2019
	2.8.5	Central	3,453	May, 2018
	2.8.4	Central	234	May, 2018
	2.8.3	Central	23	Apr, 2018
	2.8.2	Central	1,580	Sep, 2017
	2.8.1	Central	708	May, 2017
	2.8.0	Central	1,551	Oct, 2016

🎮 Assim que clicarmos na versão desejada, aparecem as opções para incluirmos a biblioteca no nosso projeto! Observe que podemos clicar na aba Gradle, e também outras ferramentas de **build** automatizada.



The image shows a screenshot of an IDE's build tool selection interface. At the top, there are tabs for Maven, Gradle, SBT, Ivy, Grape, Leiningen, and Buildr. The Maven tab is currently selected. Below the tabs, a dashed red box highlights a snippet of XML code representing a Maven dependency. A red arrow points from the text 'Para o Maven, ele nos mostra um XML.' to this XML snippet.

```
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.6</version>
</dependency>
```

🎮 Para o Maven, ele nos mostra um XML.

🎮 Mas o que faremos com isso?

🎮 O que é esse XML?

**PARA!** PARA!  
PARA! PARA! **PARA!**  
**PARA!** PARA!



🎮 Vamos entender como o Maven funciona!



🎮 Toda a configuração do Maven é feita através de um único arquivo chamado ***pom.xml***

🎮 ***POM -> Project Object Model***

🎮 Nesse arquivo iremos definir ***tudo*** que precisamos que o Maven cuide para a build automatizada do nosso projeto. Inclusive a gerência das dependências



🎮 Podemos configurar:

🎮 As nossas dependências (outros software)

🎮 O nome do *jar*

🎮 O tipo de empacotamento (jar, war)

🎮 Escopo das dependências (algumas podem ser necessárias apenas para teste)

🎮 Executar apenas os testes

🎮 .....



🎮 O arquivo ***pom.xml*** mais simples possível!

🎮 Possui no mínimo 3 informações

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.inatel.cdg</groupId>
  <artifactId>aula1-maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

O groupId + artifactId  
deve fornecer um  
nome global único  
para seu projeto!

🎮 groupId -> Identificação da empresa, ou grupo  
de projetos. Segue a convenção para nomear  
pacotes em Java

🎮 artifactId -> Identificação do projeto

🎮 version -> Versão do projeto



# Onde iremos colocar as nossas dependências nesse arquivo ***pom***?

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.inatel.cdg</groupId>
  <artifactId>aula1-maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>
```

```
<dependencies>
```

```
</dependencies>
```

```
<build>
```

```
</build>
```

Colocaremos nossas dependências

Colocaremos as instruções para o build (compilação, empacotamento, etc..)

```
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->  
<dependency>  
  <groupId>com.google.code.gson</groupId>  
  <artifactId>gson</artifactId>  
  <version>2.8.6</version>  
</dependency>
```

🎮 Vamos resgatar a dependência  
do GSON que vimos no  
repositório central do Maven

🎮 Vamos coloca-la no ***pom***



🎮 Onde iremos colocar as nossas dependências nesse arquivo ***pom***?

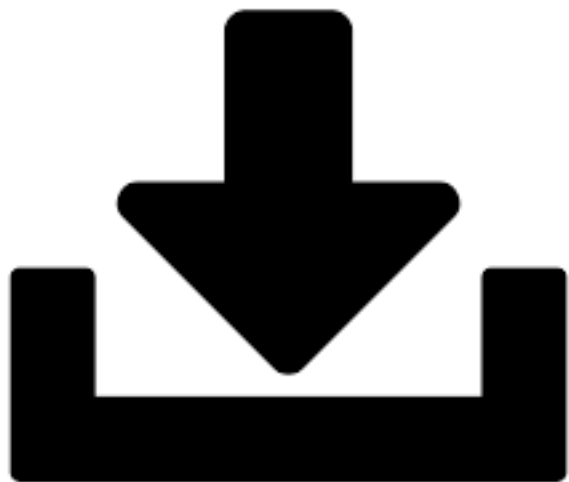
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.inatel.cdg</groupId>
  <artifactId>aula1-maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.8.6</version>
    </dependency>
  </dependencies>
```

Colocamos dentro da tag **<dependencies>** (plural).  
Cada dependência ficará na sua própria tag **<dependency>**

🎮 E agora? O que o Maven faz?

🎮 Com a dependência no ***pom***, o Maven sabe que necessita baixar a biblioteca GSON para nosso projeto. Ele irá buscar essa dependência (Jar do GSON) no repositório central do maven!



GSON


Google


JSON

🎮 Observe que o XML que colocamos não é a biblioteca GSON propriamente dita. O que ele representa são instruções para o Maven, de fato, fazer o download da biblioteca para nós!

🎮 Isso é a gerência de dependências!

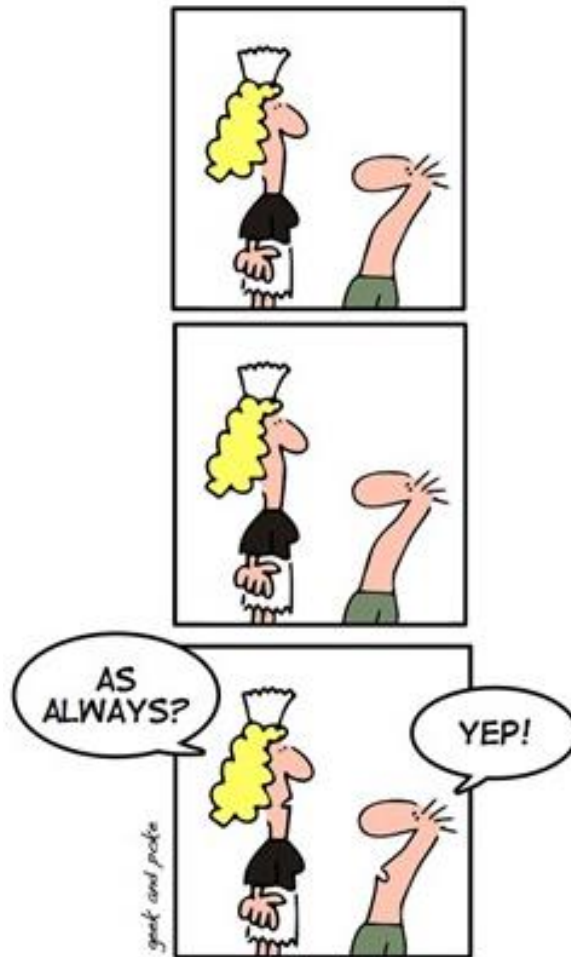


 O Maven cria um repositório local e coloca todas as dependências antes de ir buscar no repositório central.

 Assim, o Maven busca a dependência primeiro no repositório local. Caso não encontre, então ele busca no repositório central (remoto)



🎮 O Maven utiliza o conceito de ***convention over configuration***. Isto é, se seguirmos a convenção não precisamos de muitas configurações



🎮 Como convenção, o Maven possui uma estrutura de diretórios para colocarmos nossas classes Java

🎮 src/main/java

🎮 src/main/resources

🎮 src/test/java

🎮 src/test/resources

Colocaremos nosso código principal

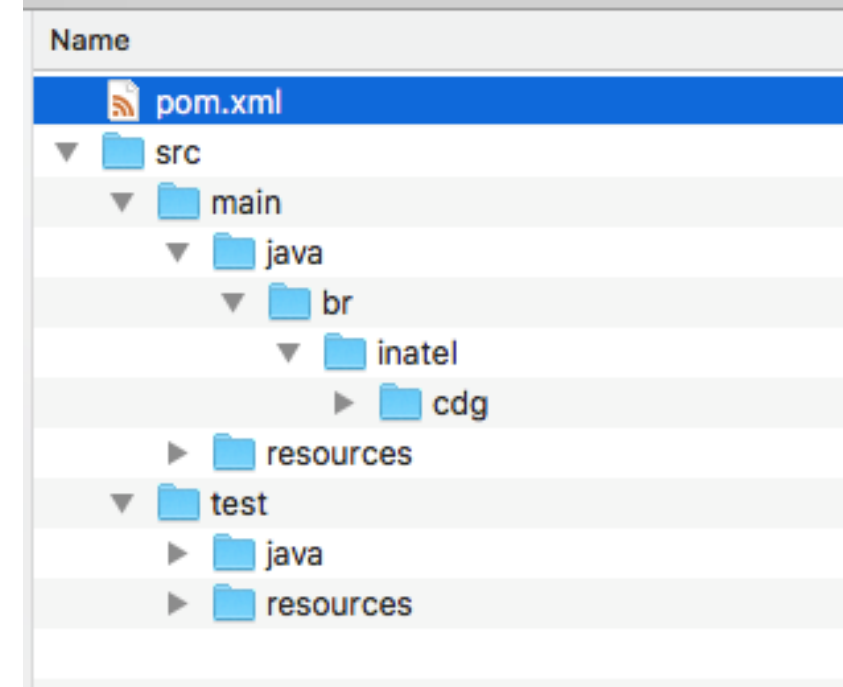
Colocaremos nosso código de teste

🎮 Os “resources” podem ser arquivos de configurações:

🎮 logj4 (Logger)

🎮 persistence (JPA)

🎮 ...



🎮 Podemos usar o Maven apenas para gerenciar as dependências. Nesse caso os testes, compilação e empacotamento precisarão ser feitos usando os recursos da IDE.

🎮 E a build?



🎮 Sem nenhuma configuração adicional, com o comando: ***mvn package***, o Maven irá:

🎮 Compilar o projeto

🎮 Executar os testes de unidade (que estiverem em src/main/test)

🎮 Gerar o jar (e colocar no diretório *target*, mas não será um executável)

🎮 O comando deve ser executado na raiz



🎮 Podemos customizar as nossas ***builds*** inserindo mais informações no ***pom.xml***

🎮 Elas devem ficar dentro das tags <build></build>

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.inatel.cdg</groupId>
  <artifactId>aula1-maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>

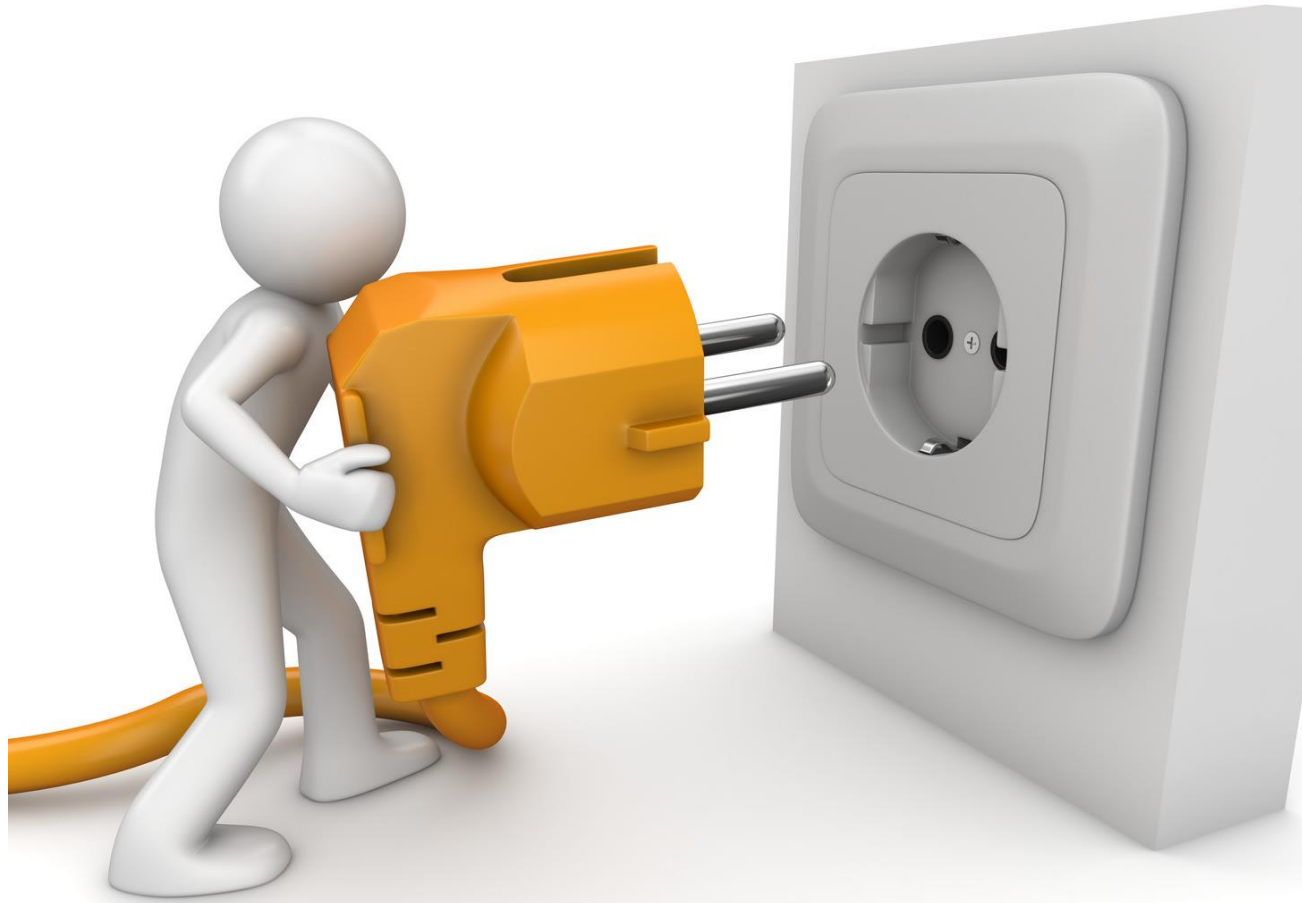
  <dependencies>

  </dependencies>
```

```
<build>
</build>
```

Instruções para o build (compilação, empacotamento, etc..)

🎮 O Maven trabalha com o conceito de *plugin*. Isso significa que outros módulos são “plugados” para indicar um novo comportamento.





Para indicarmos que desejamos utilizar o compilador do Java 8, por exemplo, fazemos:

**<plugins>** indica que podemos ter outros plugin no pom

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

**<plugin>** é a tag onde ficará o plugin propriamente dito





🎮 Cada <plugin> tem, **obrigatoriamente**, três campos:

🎮 groupId -> o grupo do plugin

🎮 artifactId -> o nome do plugin

🎮 version -> a versão

🎮 No **pom** abaixo estamos configurando o plugin que pertence ao grupo “org.apache.maven.plugin” (groupId). Esse é o grupo padrão do Maven. Portanto podemos omitir (Nos próximos slides ele estará omitido)

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
```

- 🎮 O plugin que estamos configurando é o “maven-compiler-plugin” (artifactId), e sua versão é a 3.8.1 (mais recente disponível).
- 🎮 Essas informações podem ser obtidas diretamente da página do Maven.

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

🎮 Cada plugin pode ser configurado através da tag <configuration> ou podemos dizer o que executar através da tag <execution> (será visto no lab).

🎮 O conteúdo dentro da tag <configuration> é específico de cada plugin. Para o “maven-compiler-plugin”, queremos dizer qual a versão da plataforma alvo da nossa aplicação (<target>) e qual versão do Java usaremos para a build (<source>)

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

🎮 Podemos também, dentro do pom, definir algumas propriedades.

Por exemplo a codificação de caracteres que iremos utilizar.

🎮 É boa prática utilizarmos o UTF-8

🎮 Essa configuração poderia ser feita via <plugin> dentro de <build> caso quiséssemos especificar outros detalhes.

```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
</properties>
```

# 🎮 O pom completo!



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.inatel.cdg</groupId>
  <artifactId>aula1-maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.8.6</version>
    </dependency>
  </dependencies>

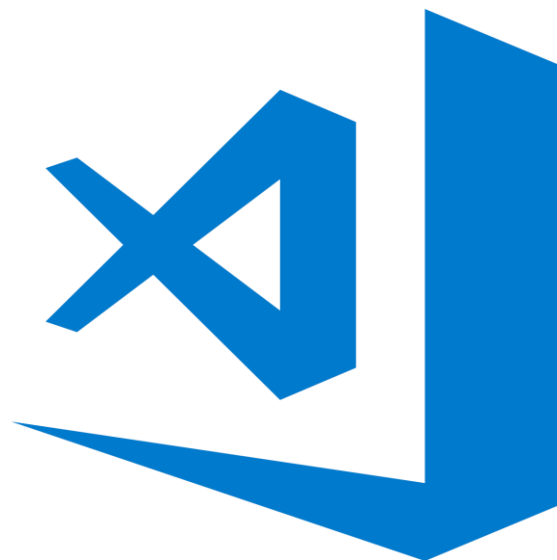
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

🎮 O Maven possui integração com diversas IDEs e editor de texto para facilitar ainda mais seu uso.



eclipse



🎮 Mas nada impede de utilizar o terminal



🎮 Raiz!

🎮 Durante o curso faremos uma abordagem híbrida



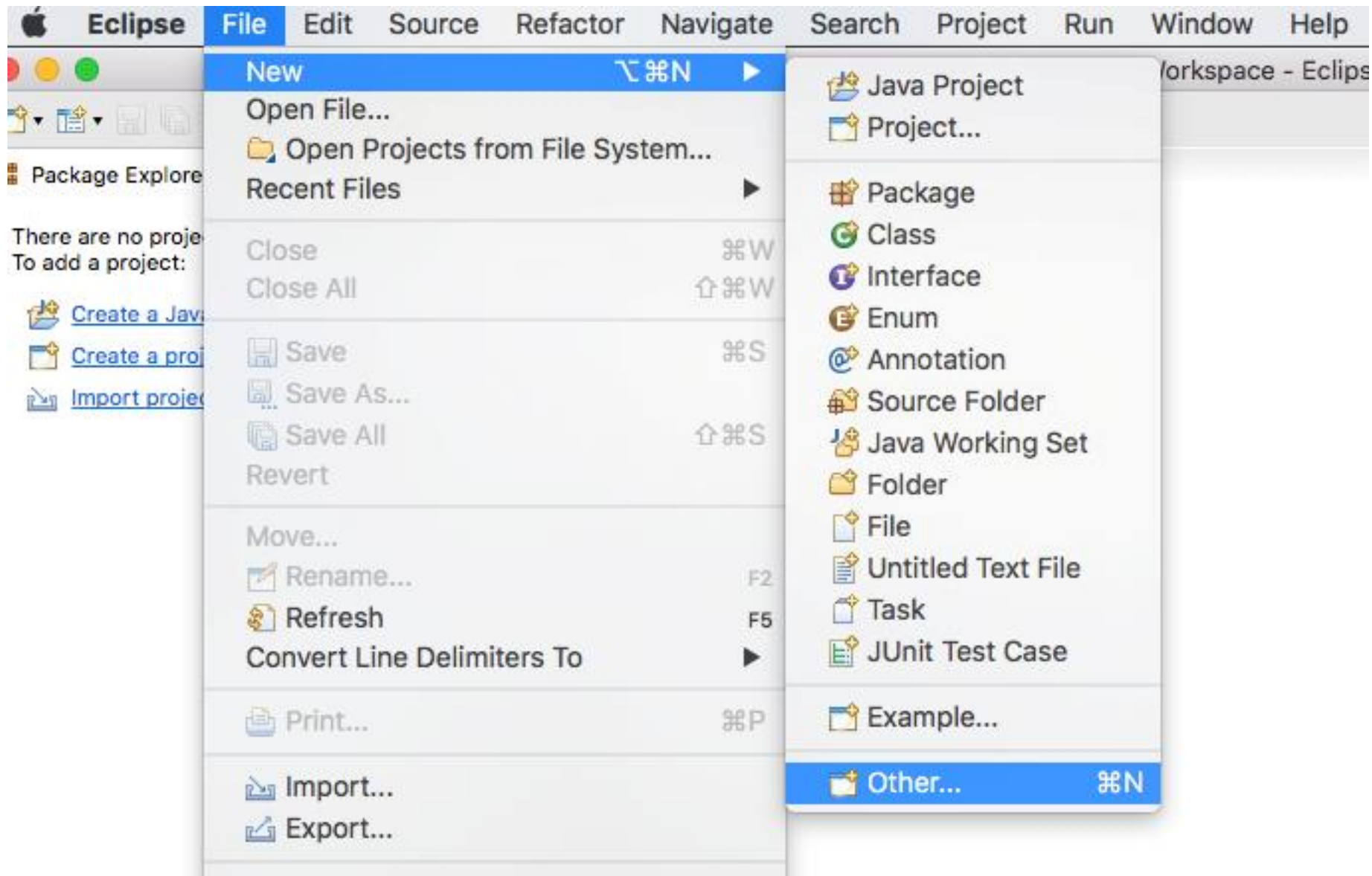




***Maven***<sup>TM</sup>

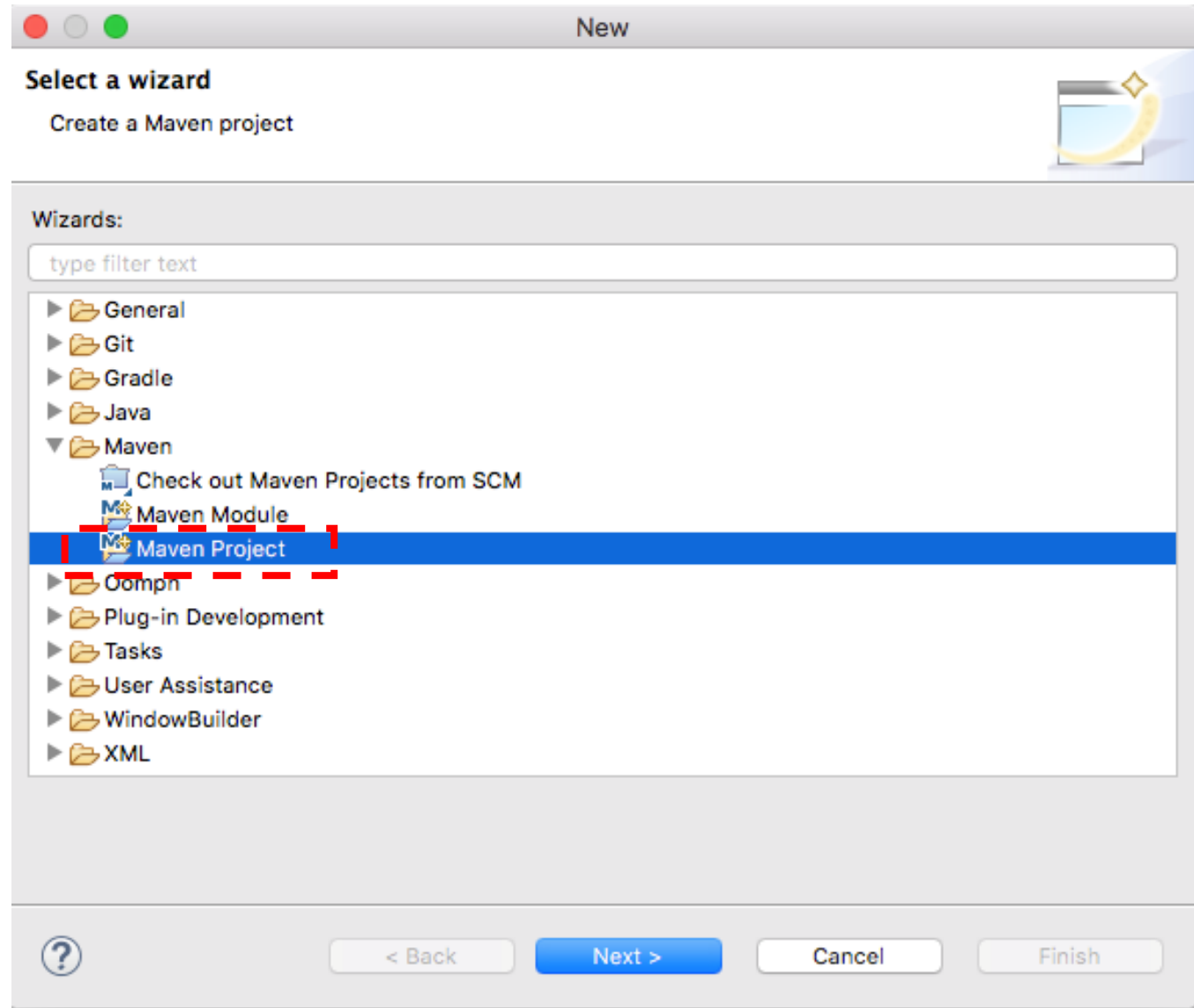
🎮 Vamos criar um programa que emita relatório JSON representando inimigos do ***Dark Souls***

# Vamos criar um projeto Maven no Eclipse

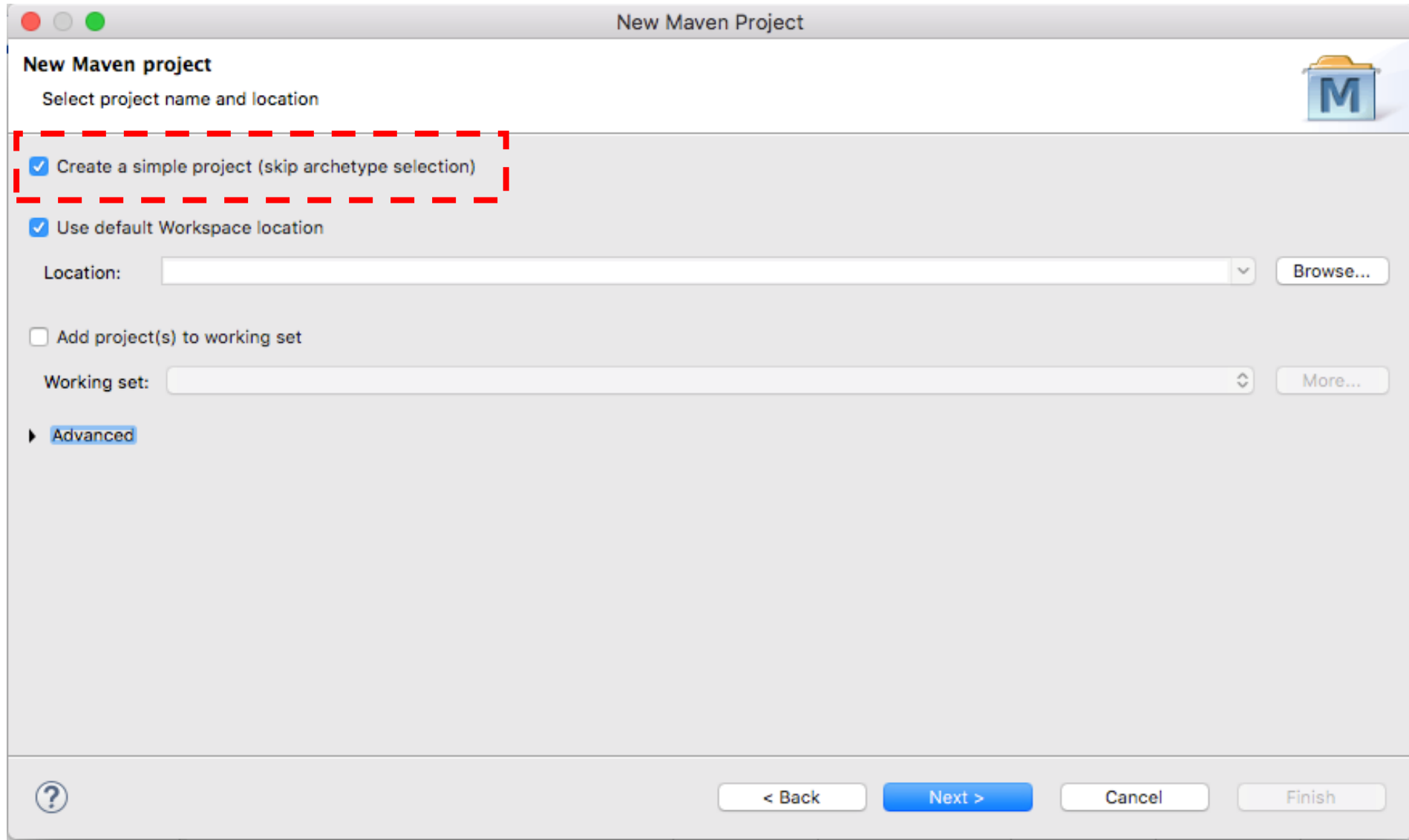




# Escolha “Maven Project”



# Marque “Simple Project”



New Maven Project

New Maven project

Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location:  Browse...

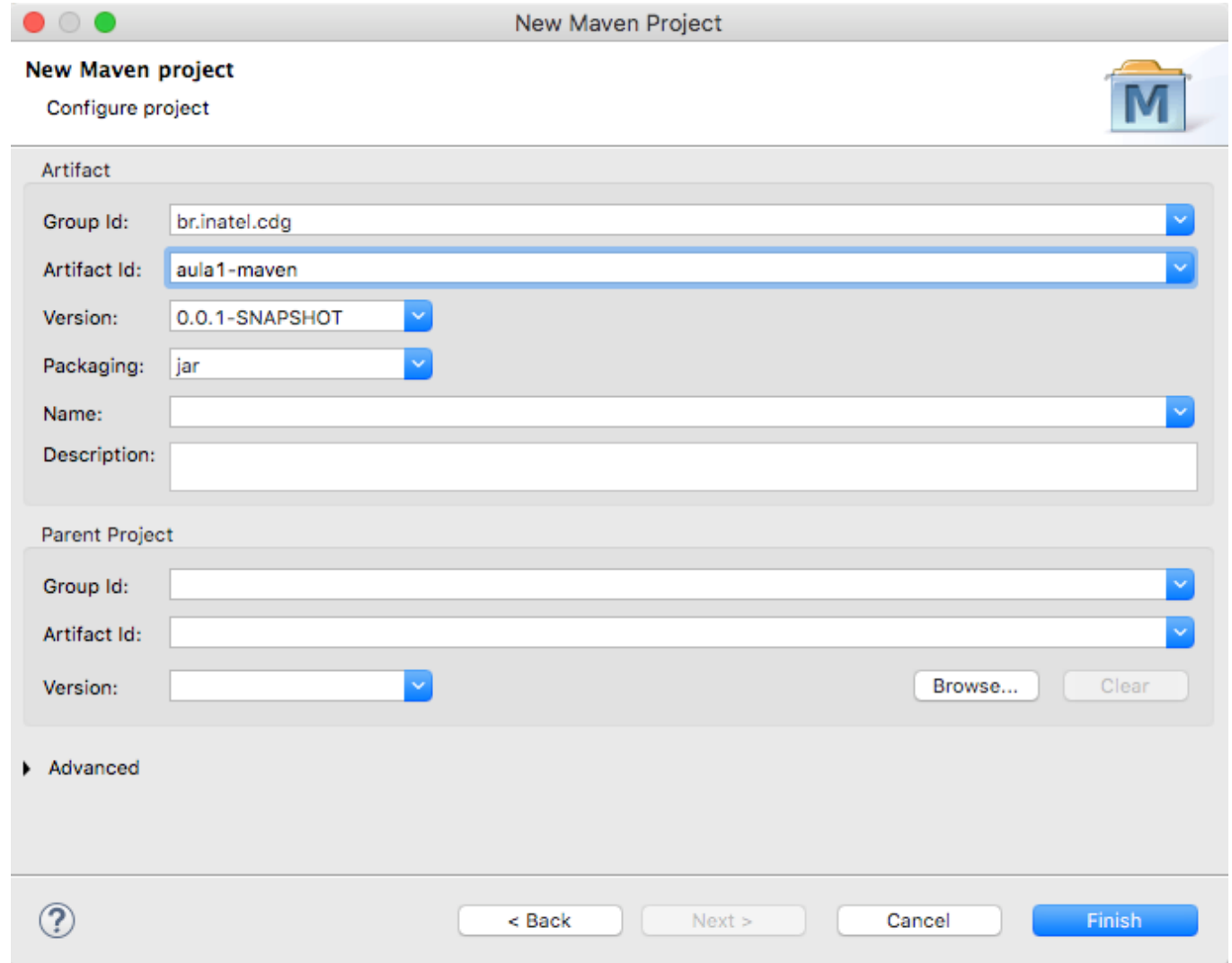
☐ Add project(s) to working set

Working set:  More...

Advanced

? < Back Next > Cancel Finish

🎮 Preencha o groupId (convecção de pacote Java) e artifactId (projeto)



New Maven Project

New Maven project  
Configure project

Artifact

Group Id: br.inatel.cdg

Artifact Id: aula1-maven

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

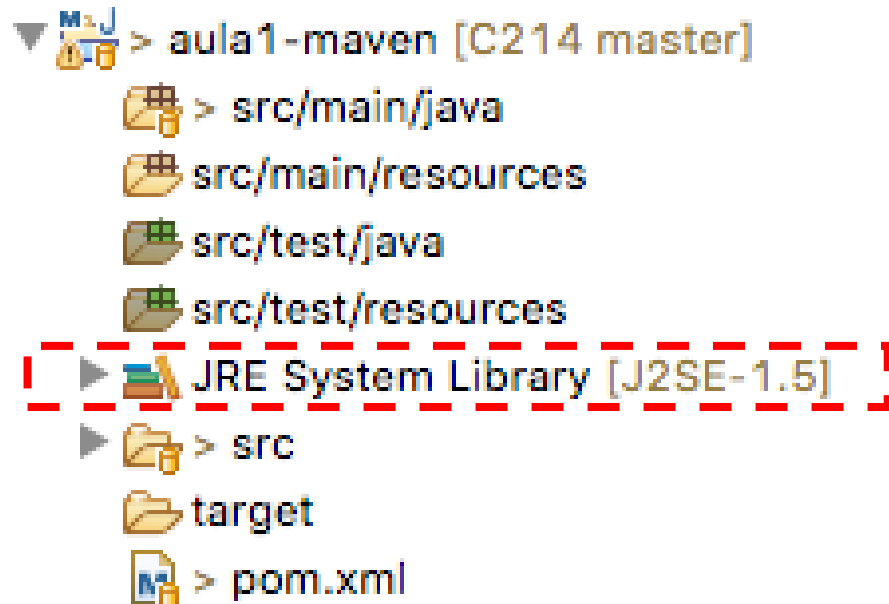
Version:

Browse... Clear


Advanced

? < Back Next > Cancel Finish

- 🎮 Observe a estrutura que é criada para o projeto.
- 🎮 Veja também que a versão do Java está como 5





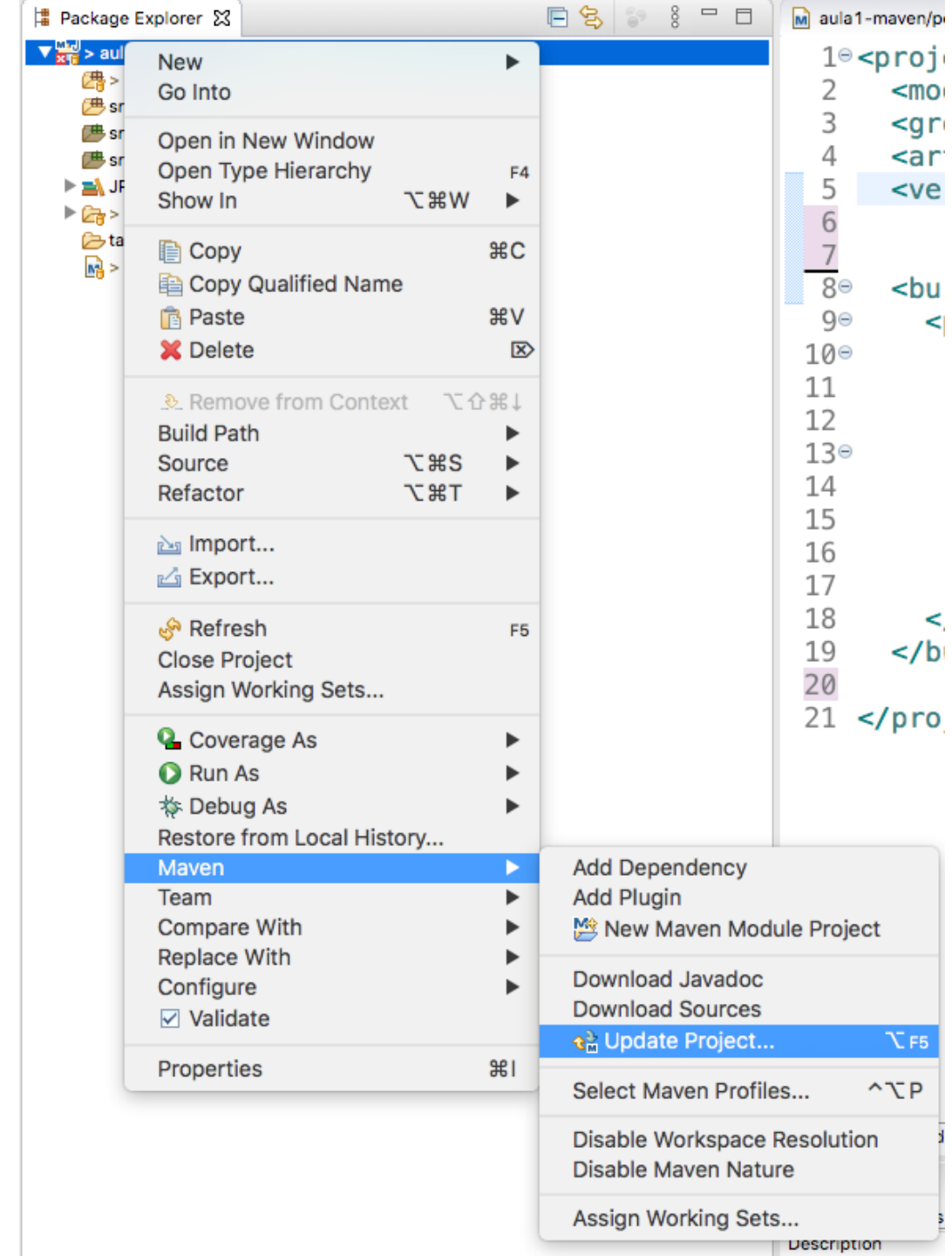
 Vamos resolver a questão da versão do Java. Para isso usamos o plugin “maven-compiler-plugin”

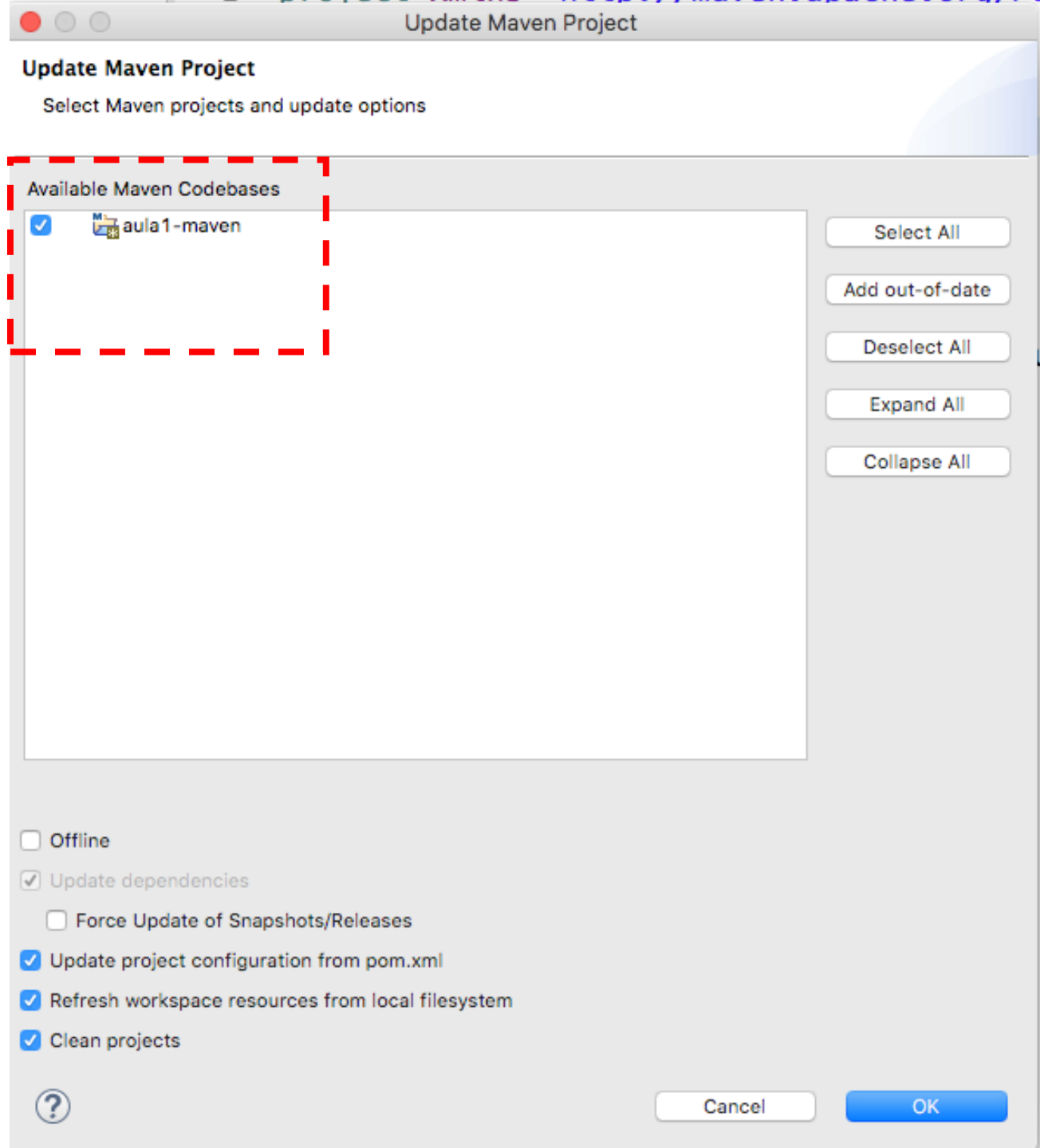
src/main/java  
src/main/resources  
src/test/java  
src/test/resources  
JRE System Library [J2SE-1.5]  
src  
target  
pom.xml

```
1 <!-- Project name -->  
2 <modelVersion>4.0.0</modelVersion>  
3 <groupId>br.inatel.cdg</groupId>  
4 <artifactId>aula1-maven</artifactId>  
5 <version>0.0.1-SNAPSHOT</version>  
6  
7  
8 <build>  
9   <plugins>  
10     <plugin>  
11       <artifactId>maven-compiler-plugin</artifactId>  
12       <version>3.8.1</version>  
13       <configuration>  
14         <source>1.8</source>  
15         <target>1.8</target>  
16       </configuration>  
17     </plugin>  
18   </plugins>  
19 </build>  
20  
21 </project>
```












# Vamos pedir para o Maven “atualizar” o projeto





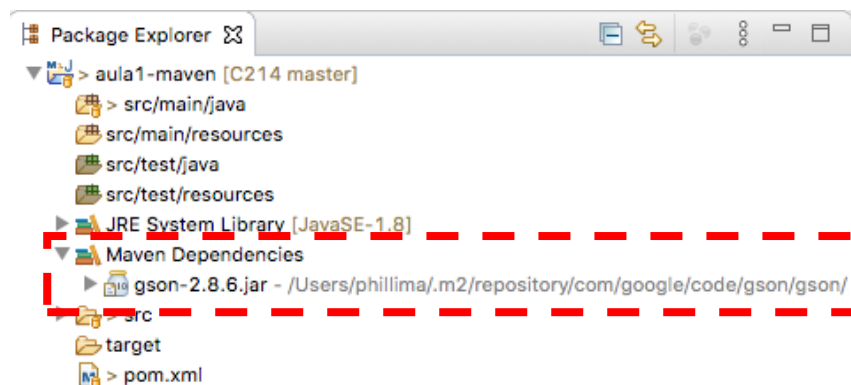
# Marque o projeto


# 🎮 Praise the Sun!


▼  > aula1-maven [C214 master]  
     > src/main/java  
     src/main/resources  
     src/test/java  
     src/test/resources  
    ▶  JRE System Library [JavaSE-1.8]  
    ▶  > src  
     target  
     > pom.xml



# Vamos adicionar a dependência do GSON



 Observe que o Maven criou um diretório “Maven Dependencies” e armazenou o jar do GSON.

 Caso não veja essa pasta, peça para o Maven “atualizar” igual fizemos para o Java 8.

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xs
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>br.inatel.cdg</groupId>
4   <artifactId>aula1-maven</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6
7
8 <dependencies>
9   <dependency>
10     <groupId>com.google.code.gson</groupId>
11     <artifactId>gson</artifactId>
12     <version>2.8.6</version>
13   </dependency>
14 </dependencies>
15
16 <build>
17   <plugins>
18     <plugin>
19       <artifactId>maven-compiler-plugin</artifactId>
20       <version>3.8.1</version>
21       <configuration>
22         <source>1.8</source>
23         <target>1.8</target>
24       </configuration>
25     </plugin>
26   </plugins>
27 </build>
28
29 </project>
```

 Agora é só programar

 Crie uma classe Inimigo com nome e vida

 Crie uma classe Main que irá criar o JSON


```
package br.inatel.cdg.inimigo;

public class Inimigo {

    private String nome;
    private double vida;

    public Inimigo(String nome, double vida) {
        this.nome = nome;
        this.vida = vida;
    }
}
```

# Classe Main

 Esses métodos pertencem a biblioteca GSON. E conseguimos usar pelo fato do Maven ter baixado a dependência.

```
package br.inatel.cdg;

import java.io.FileWriter;

public class Main {

    public static void main(String[] args) {

        List<Inimigo> inimigos = new ArrayList<Inimigo>();

        inimigos.add(new Inimigo("Black Knight", 100));
        inimigos.add(new Inimigo("Silver Knight", 200));

        Gson gson = new GsonBuilder().
            setPrettyPrinting().create();

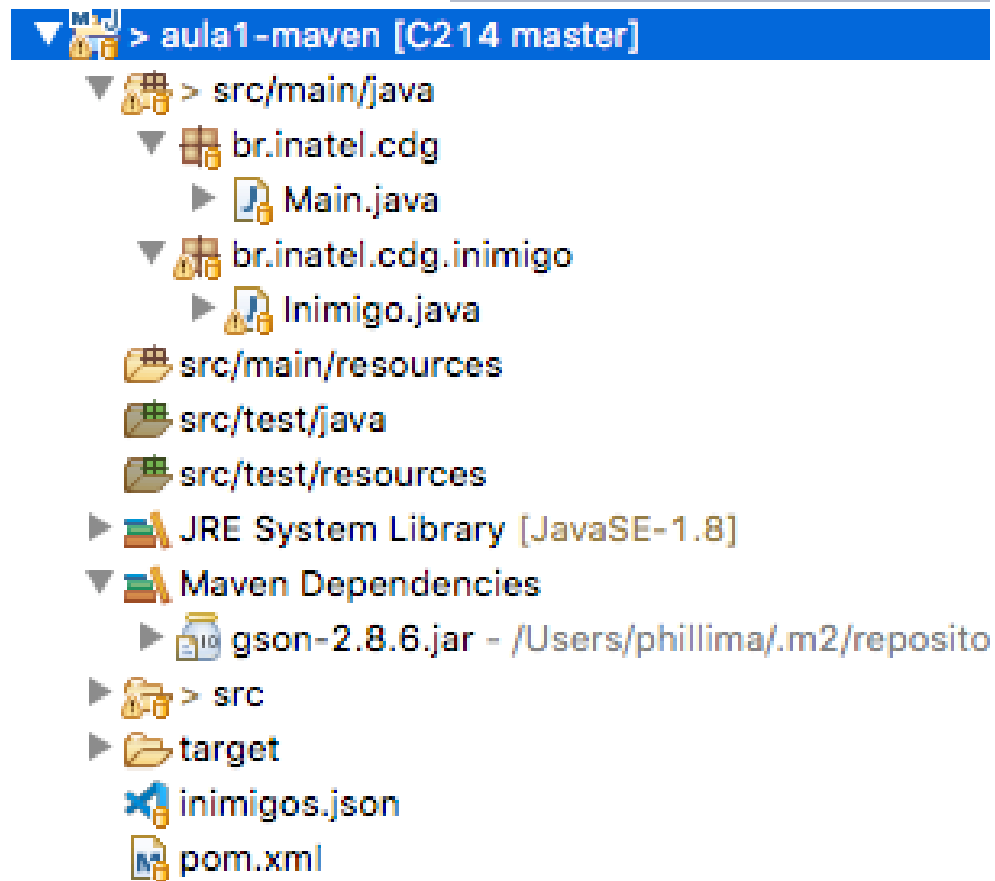
        String json = gson.toJson(inimigos);

        FileWriter writer;
        try {
            writer = new FileWriter("inimigos.json");
            writer.write(json);
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

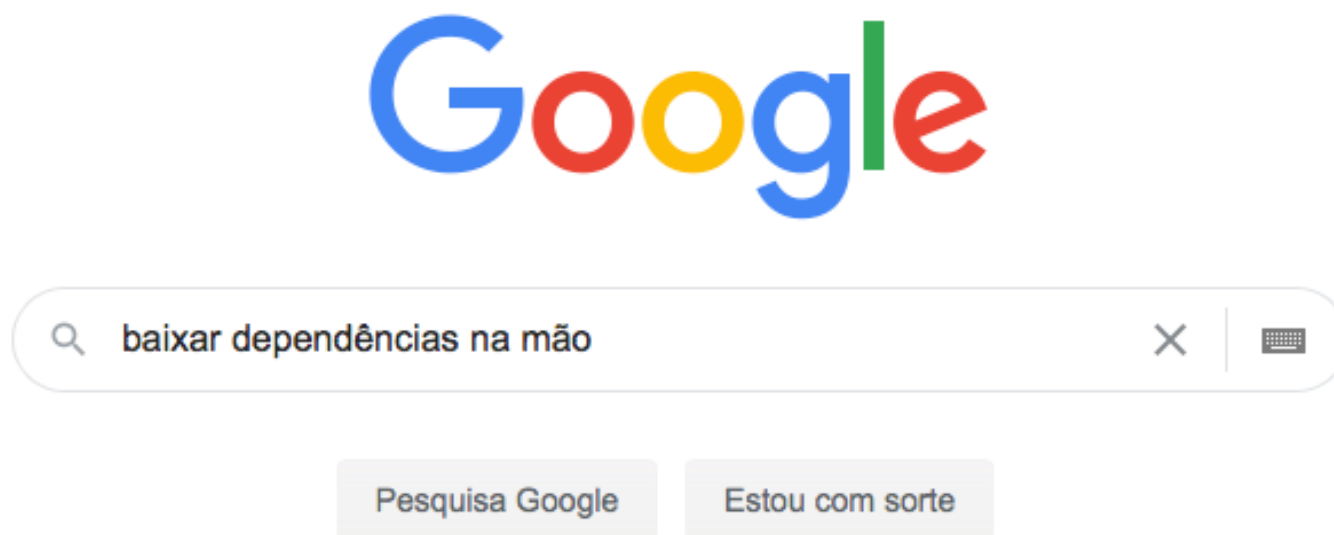
# Estrutura Final do Projeto!





 JSON gerado!

```
1  [  
2    {  
3      "nome": "Black Knight",  
4      "vida": 100.0  
5    },  
6    {  
7      "nome": "Silver Knight",  
8      "vida": 200.0  
9    }  
10 ]
```



# Implementações

 <https://github.com/phillima-inatel/C125>



# Material Complementar

 Instalando no Mac OS

 brew install maven

 Instalando no Linux (Ubuntu/Debian)

 apt-get install maven

# Material Complementar

## Instalando Maven no Windows:

 Tutorial: <https://mkyong.com/maven/how-to-install-maven-in-windows/>

 Necessário configurar JAVA\_HOME e MAVEN\_HOME

## Tutorial para C214

 PT1: <https://youtu.be/58RC3cgOPR0>

 PT2: <https://youtu.be/eD7OKxI39M8>



C125/C206 – Programação Orientada a Objetos  
com Java

# Automatização da Build com Maven

Prof. Phyllipe Lima  
phyllipe@inatel.br