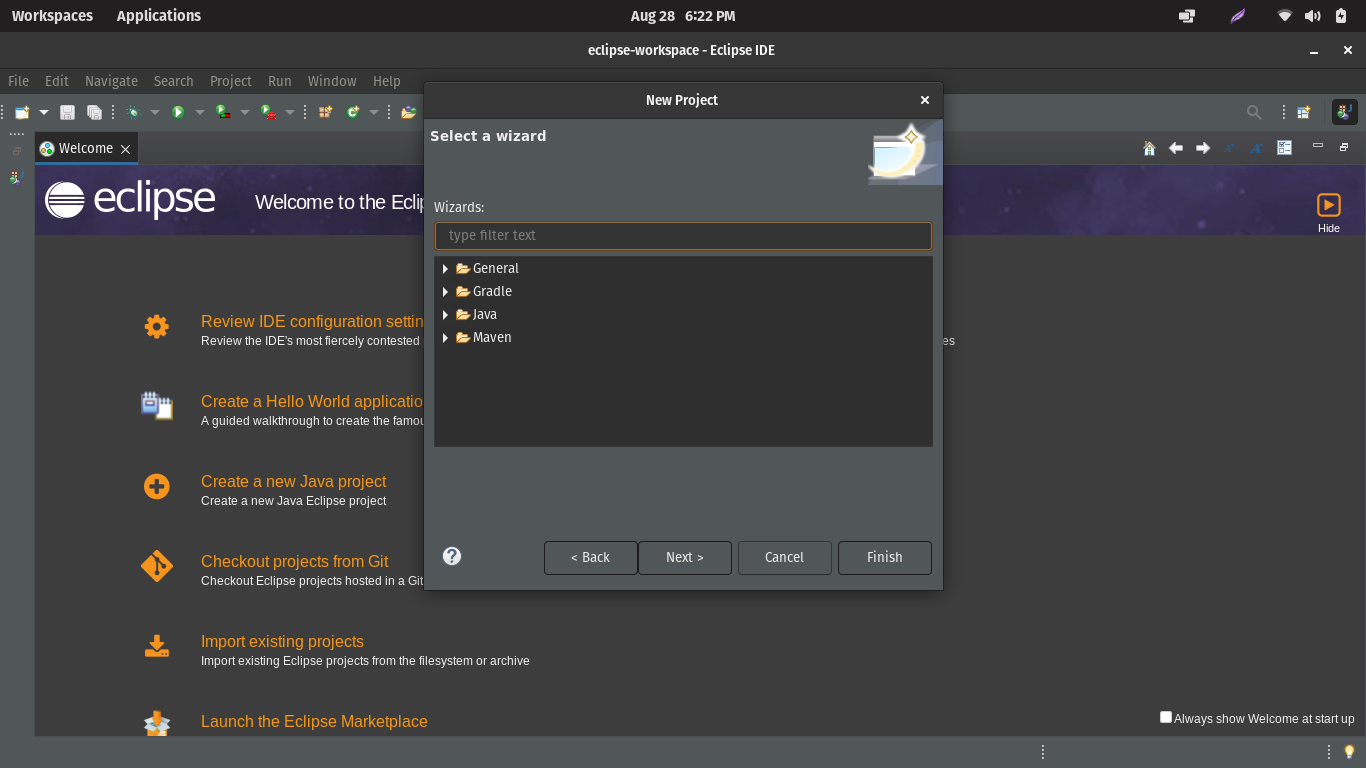
**Maven**

O Maven é, basicamente, uma ferramenta que permite a construção e gerenciamento de projetos Java e demais linguagens. Através do Maven é definida uma forma padronizada para estruturar, construir e gerenciar projetos. O maven pode também receber plugins que estendem suas funcionalidades e, além disso, conta com uma facilidade imensa na questão de gerenciamento de dependências. Enquanto sem o Maven é necessário fazer o trabalho manual relativo ao gerenciamento de diversos arquivos .jar, com o Maven é possível apenas escrever algumas linhas no arquivo pom.xml, e assim será feito o download de tudo aquilo que é preciso para a dependência.

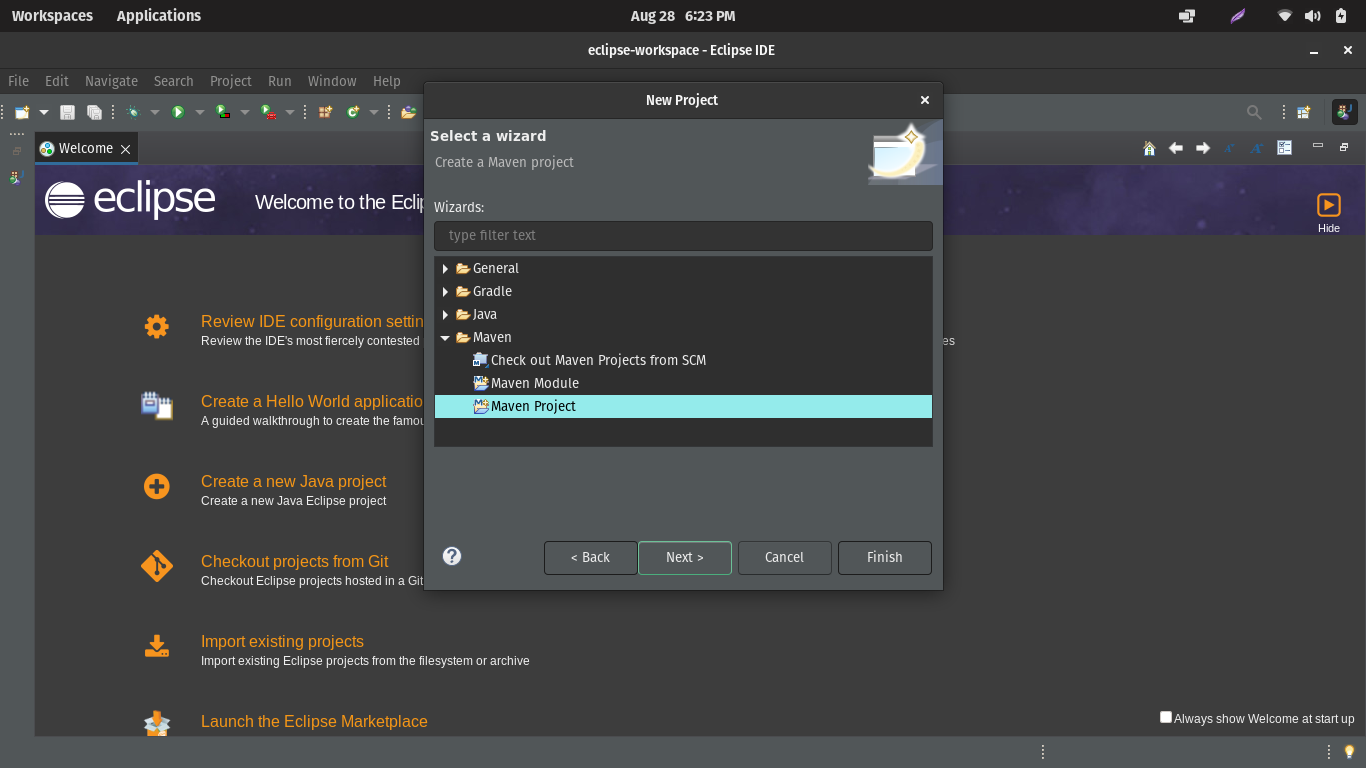
O pom.xml é o arquivo utilizado para fazer o controle do projeto Maven, conta com seus metadados, versão do Java, as Maven coordinates (group id, artifact id e version), das quais o Maven faz uso para identificar o projeto, plugins, dependências, entre outros. Basicamente todas as configurações necessárias para o devido funcionamento do projeto.

Dentro do Maven há diversos arquétipos (templates), que podem ser utilizados na criação de projetos, cada um para um tipo de aplicação diferente. Vamos exemplificar isso criando um projeto maven no Eclipse com o arquétipo geral no Maven através dos seguintes passos:

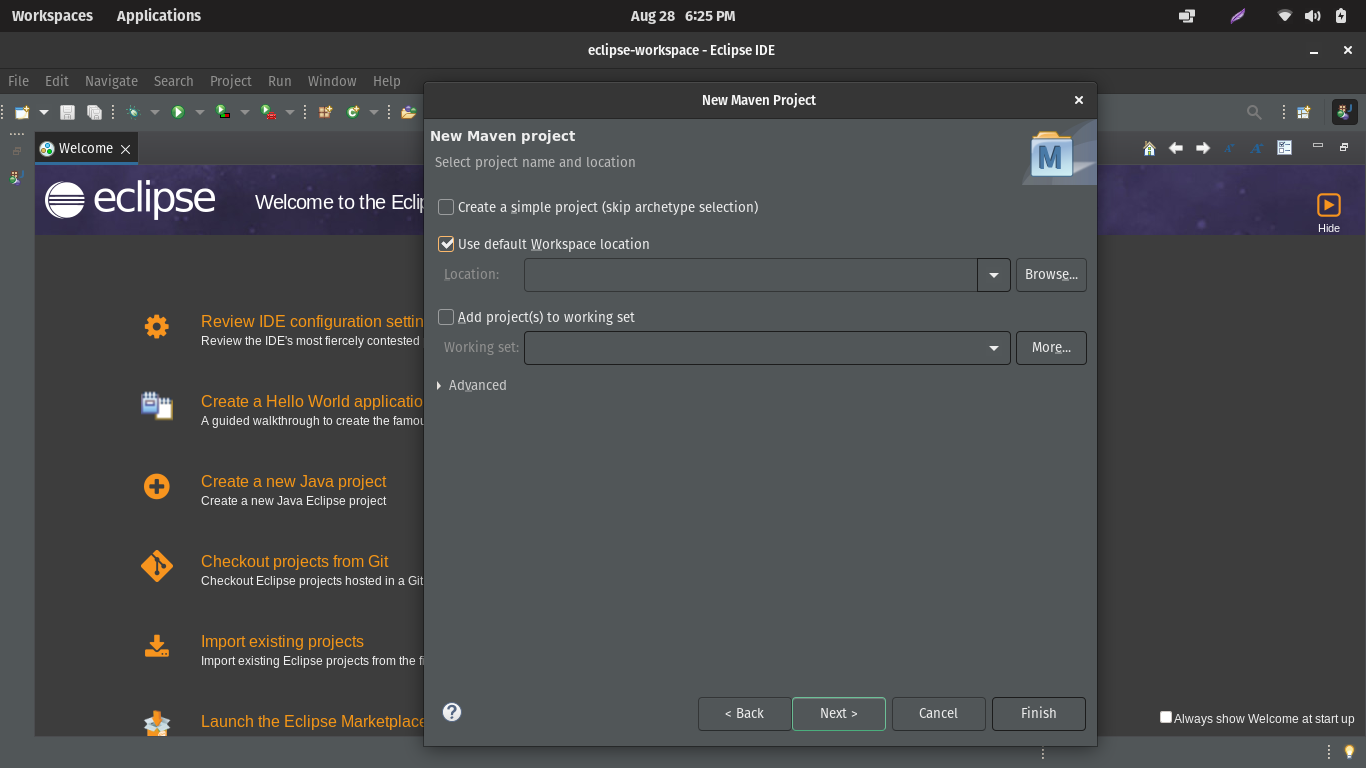
1. File -> New -> Project: você cairá nessa tela:



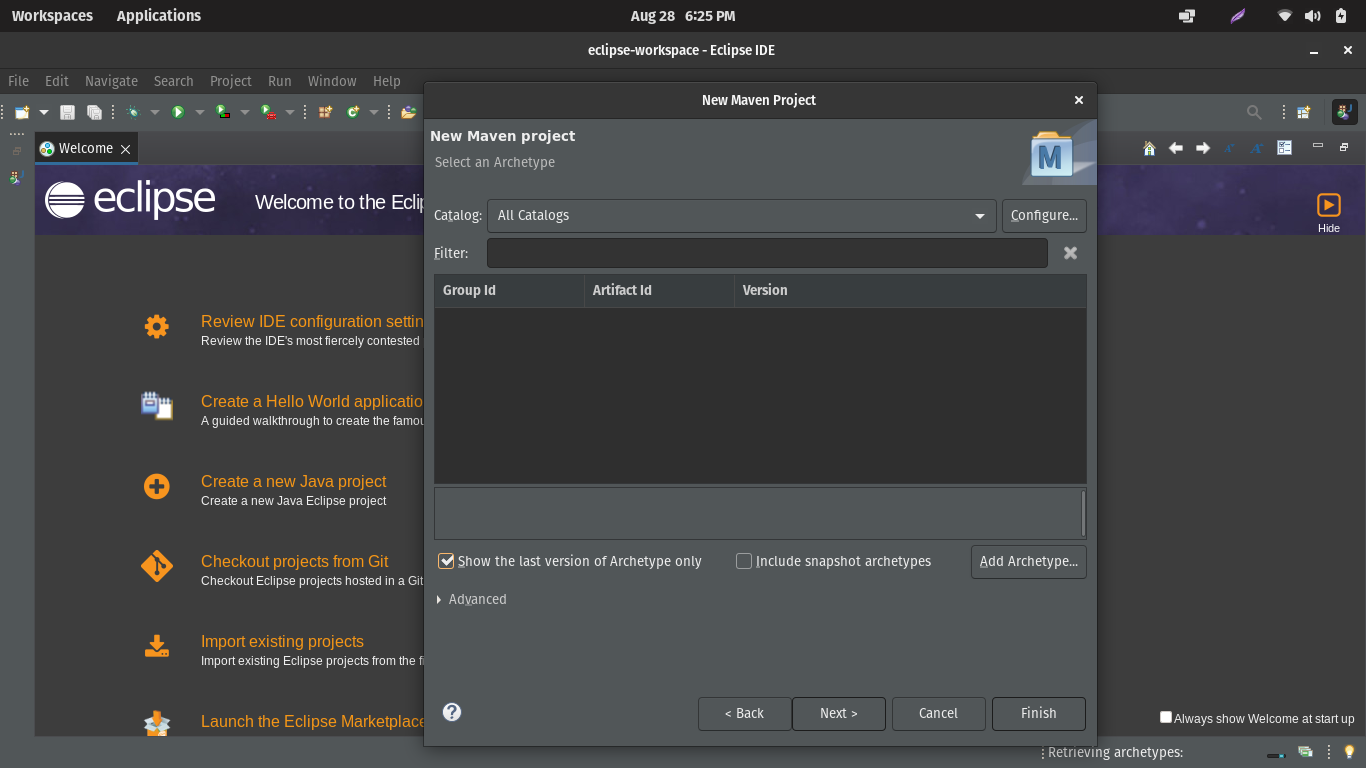
1. Maven -> Maven Project -> Next:



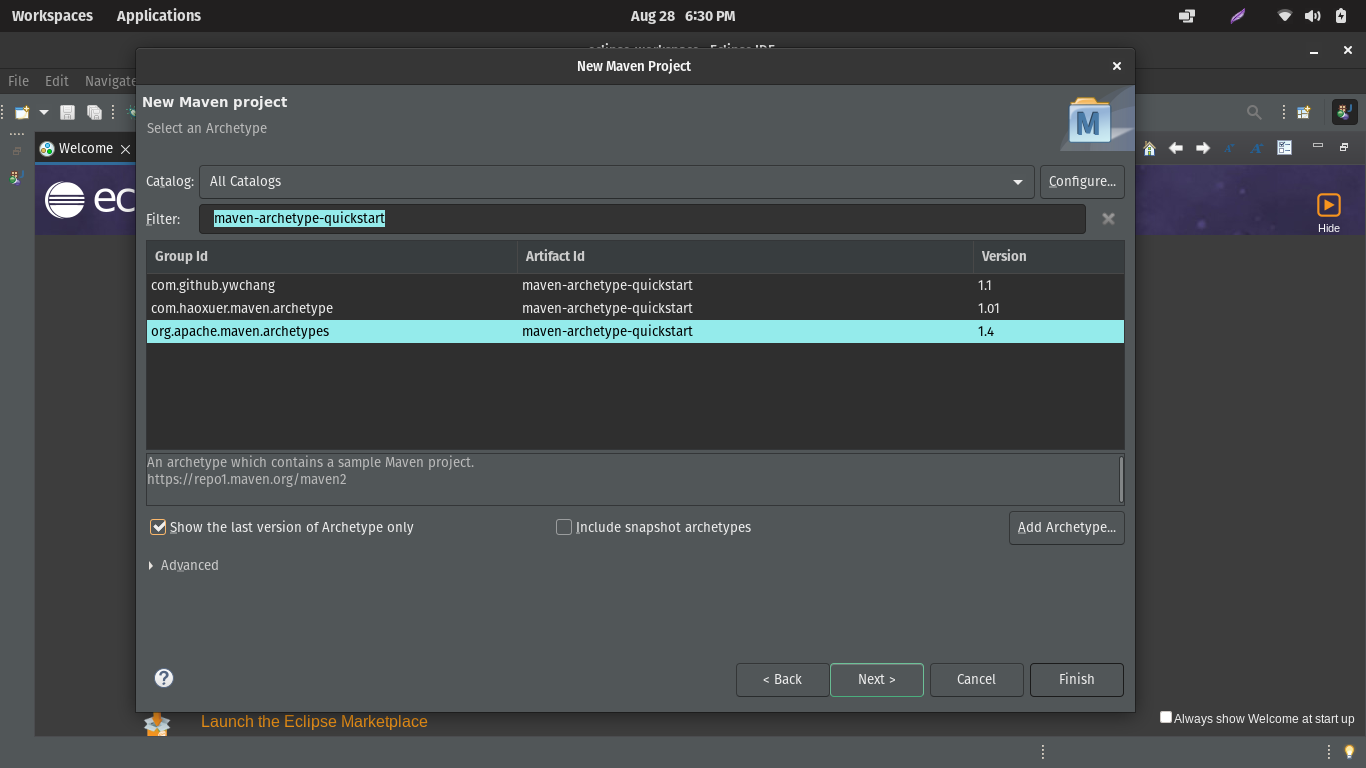
Você chegará nesta tela:



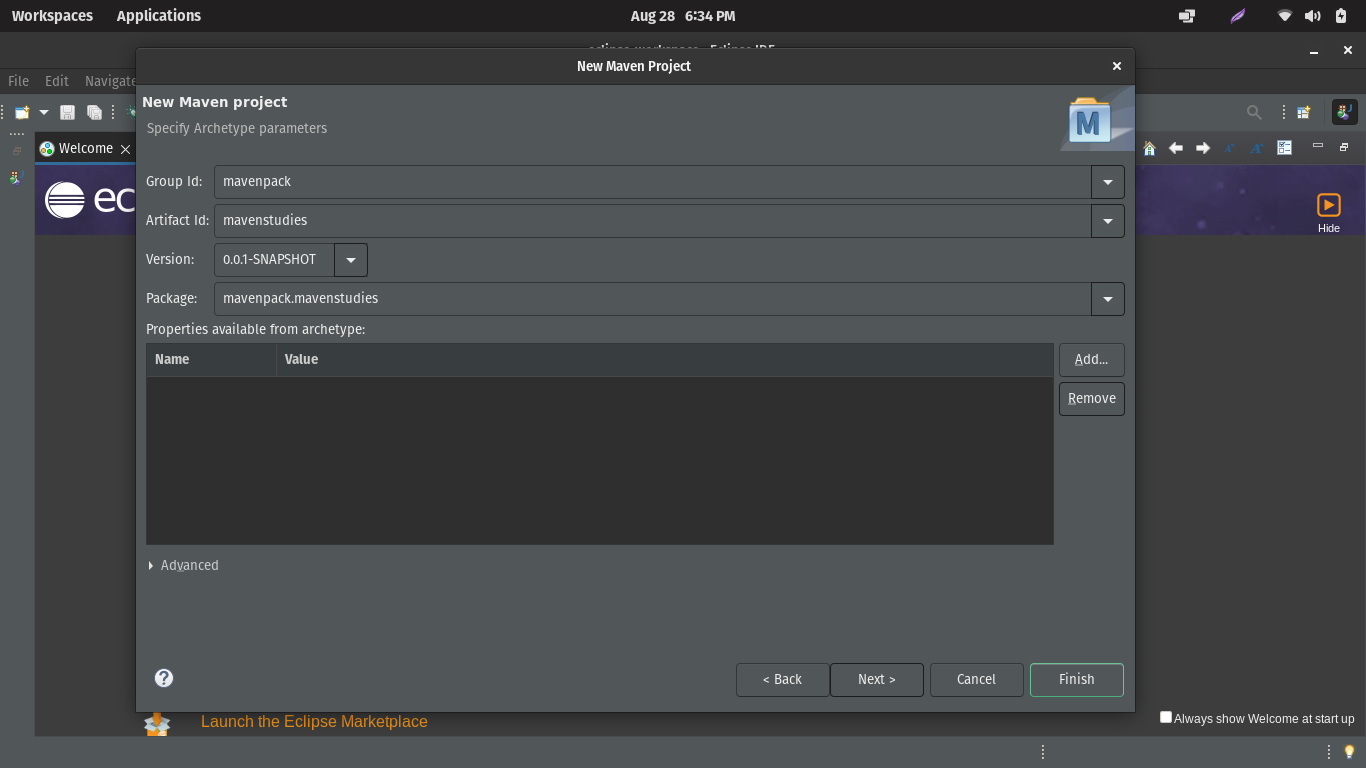
1. Clique em Next. Você chegará nesta tela:



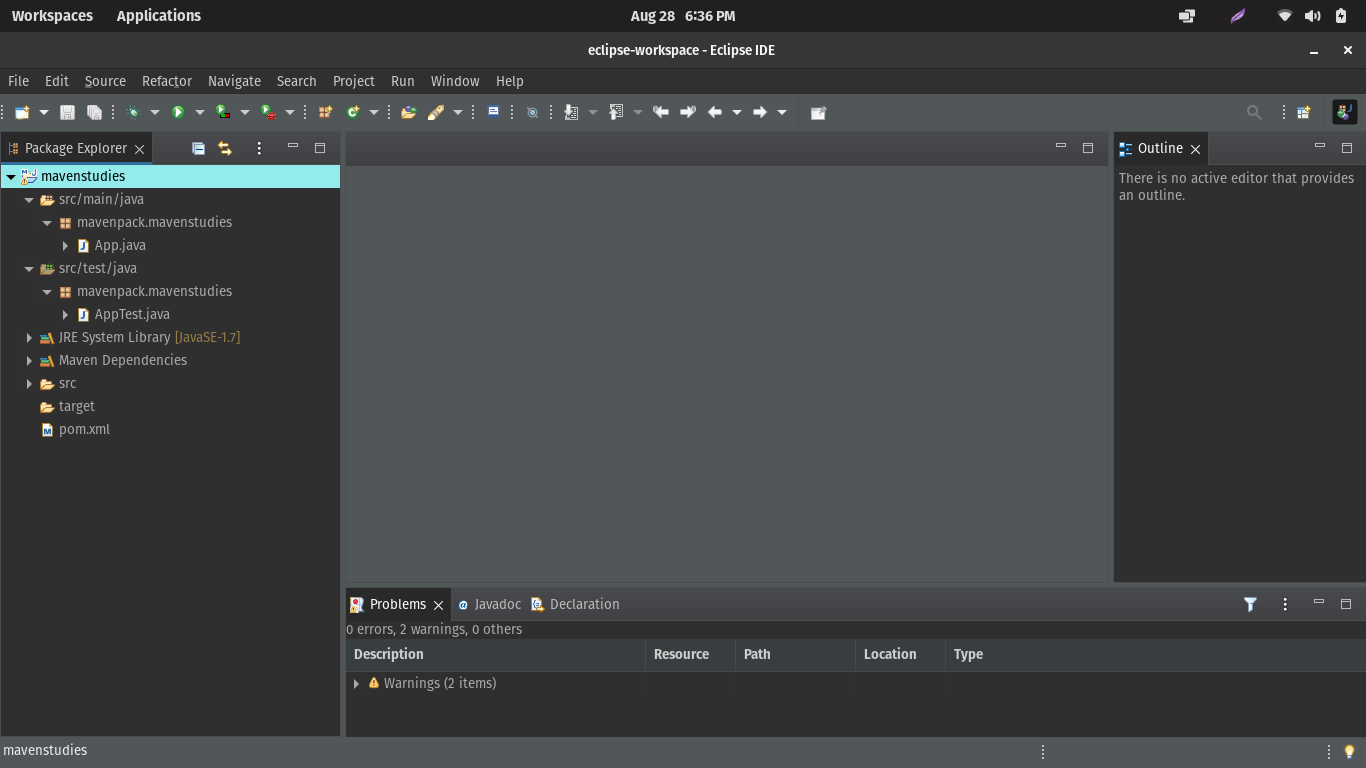
1. Aguarde o carregamento, busque e selecione o arquétipo “maven-archetype-quickstart”, versão 1.4. Clique em Next:



1. Aqui você deverá preencher alguns dados importantes, as Maven Coordinates que, como mencionado anteriormente, identificarão o projeto:
   1. Group Id: junto do artifact Id, será usado como nome do package;
   2. Artifact Id: será usado como nome do projeto;
   3. Version: versão com a qual o projeto inicia, pode ser importante para o controle do desenvolvimento à longo prazo. Por padrão o maven utiliza 0.0.1-SNAPSHOT.



1. Clique em Finish, o projeto será gerado:



Percebe-se que foi criada a seguinte estrutura:

1. Pasta main - na qual fica o código fonte;
2. O pacote “mavenpack.mavenstudies”, dentro da pasta main;
3. O diretório test, no qual se mantêm as classes com os testes unitários;
4. O diretório target, que é onde ficam localizados os resultados das compilações de código e demais processos do Maven;
5. O arquivo pom.xml. utilizado para fazer o controle do projeto Maven.

**Ciclos de vida e utilização de suas funções:**

Com nosso projeto já criado, antes de começarmos a usar a ferramenta é necessário entender os ciclos de vida, um dos pilares para o funcionamento do Maven. Em suma, os procedimentos de construção são definidos em etapas, ou seja, cada função do maven é uma etapa e uma ou mais dessas etapas forma um ciclo de vida. A ferramenta conta com 3 ciclos de vida primordiais, o  *default,* responsável pela implementação do projeto, o  *clean,* cuja função é de limpeza e o *site*, encarregado da documentação automática do projeto.

Esses 3 ciclos são compostos por diversos estágios. Além disso, algo importante a se mencionar é o fato de que os estágios são cumulativos e seguem uma ordem de execução, ou seja, quando é acionado o estágio 3, também são antes acionados, nesta ordem, os estágios 1 e 2.

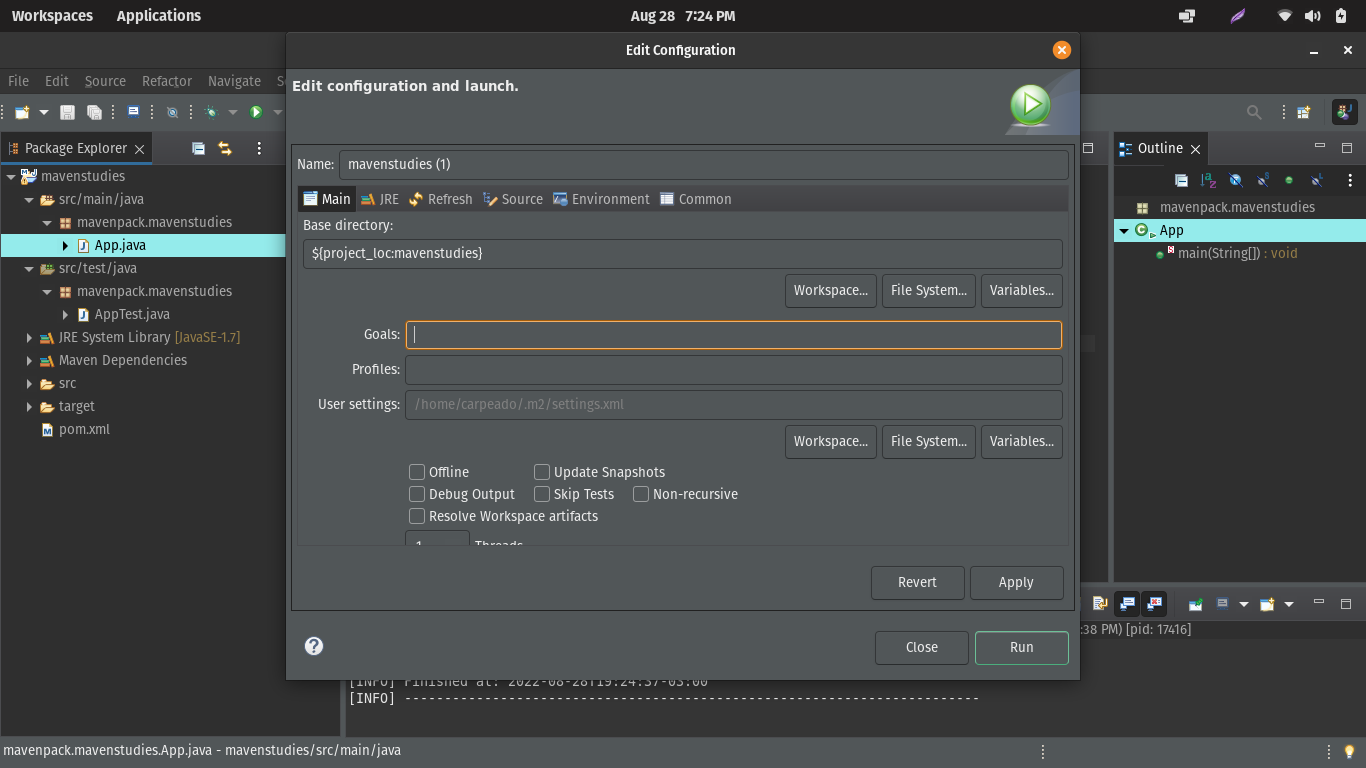
Para exemplificar a ideia de ciclos de construção, vamos iniciar detalhando o default, que conta com mais de 20 estágios, Dentre os principais estão:

1. validate: valida se as informações do projeto estão devidamente preenchidas e válidas;
2. compile: compila o código fonte;
3. test: executa os testes unitários;
4. package: empacota o código fonte e gera o executável no formato de arquivo definido para o projeto, por exemplo, .jar;
5. install: instala o pacote no repositório local;
6. deploy: instala o pacote em um repositório externo.

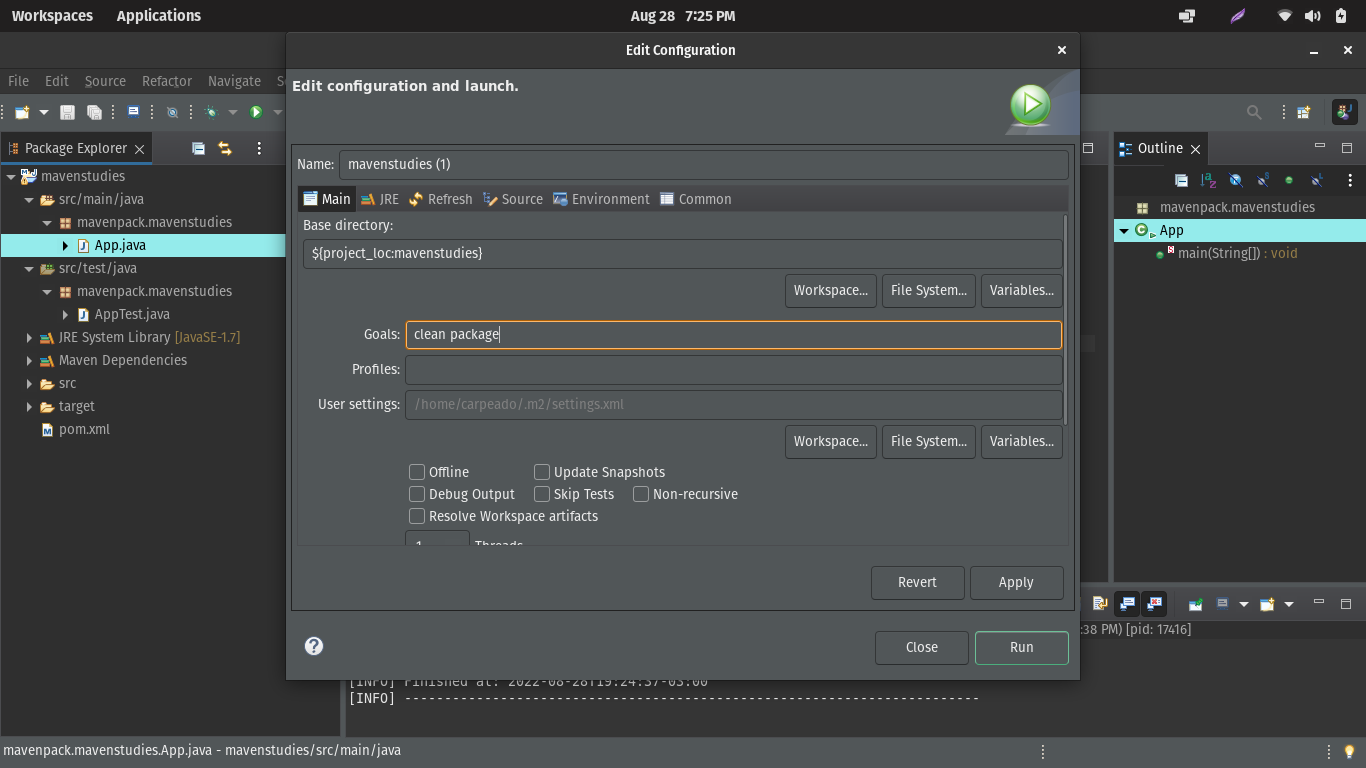
Dentre os listados acima está o package, o qual gera o executável do projeto, contudo, por conta da característica cumulativa dos estágios de um ciclo de vida, há alguns que vem antes dele, como o test e o compile (compila o código).

Para visualizar a execução dessa função podemos fazer uso do *clean package,* o clean serve para limpar as informações já existentes no target, onde serão colocadas informações novas e o arquivo .jar do projeto. O uso dessa e de demais etapas específicas pode ser feito, no Eclipse, da seguinte maneira:

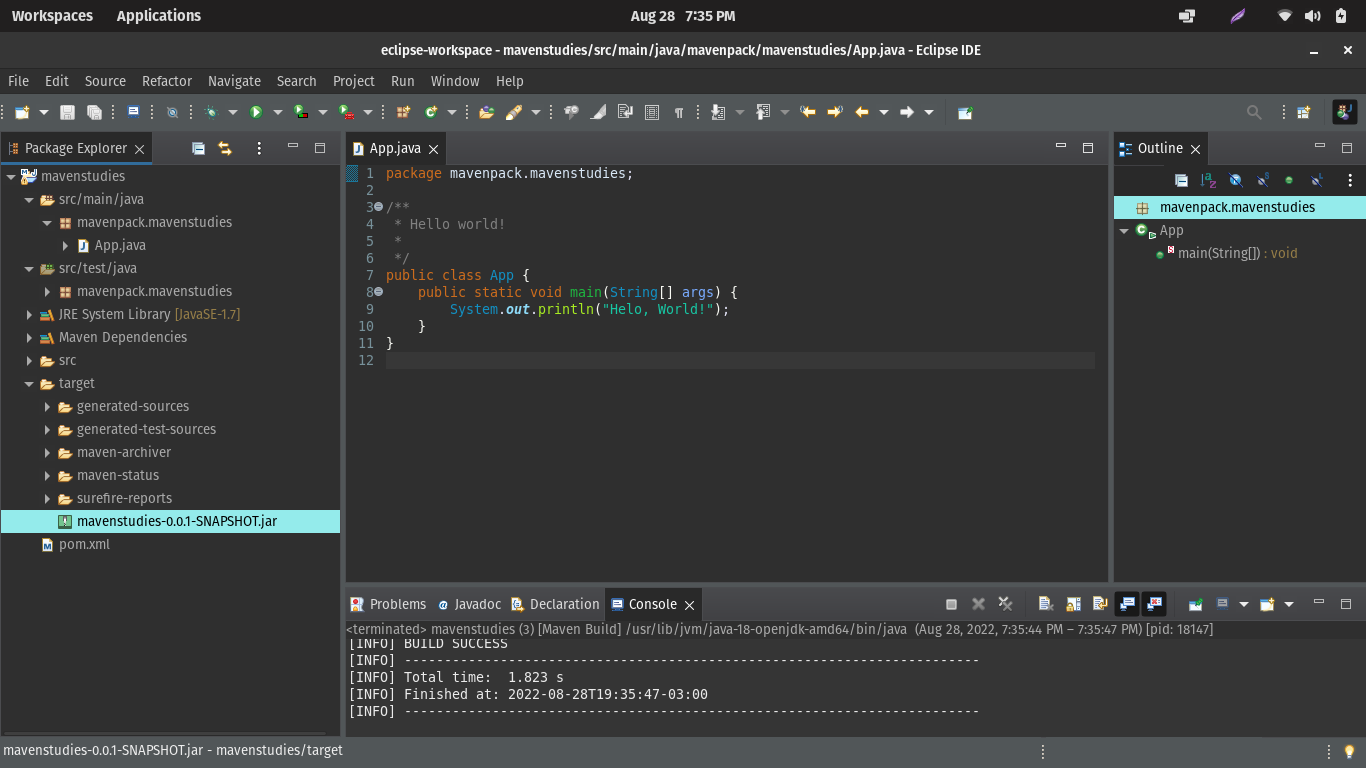
1. Clicar em Run -> Run As -> Maven build. Essa tela aparecerá:



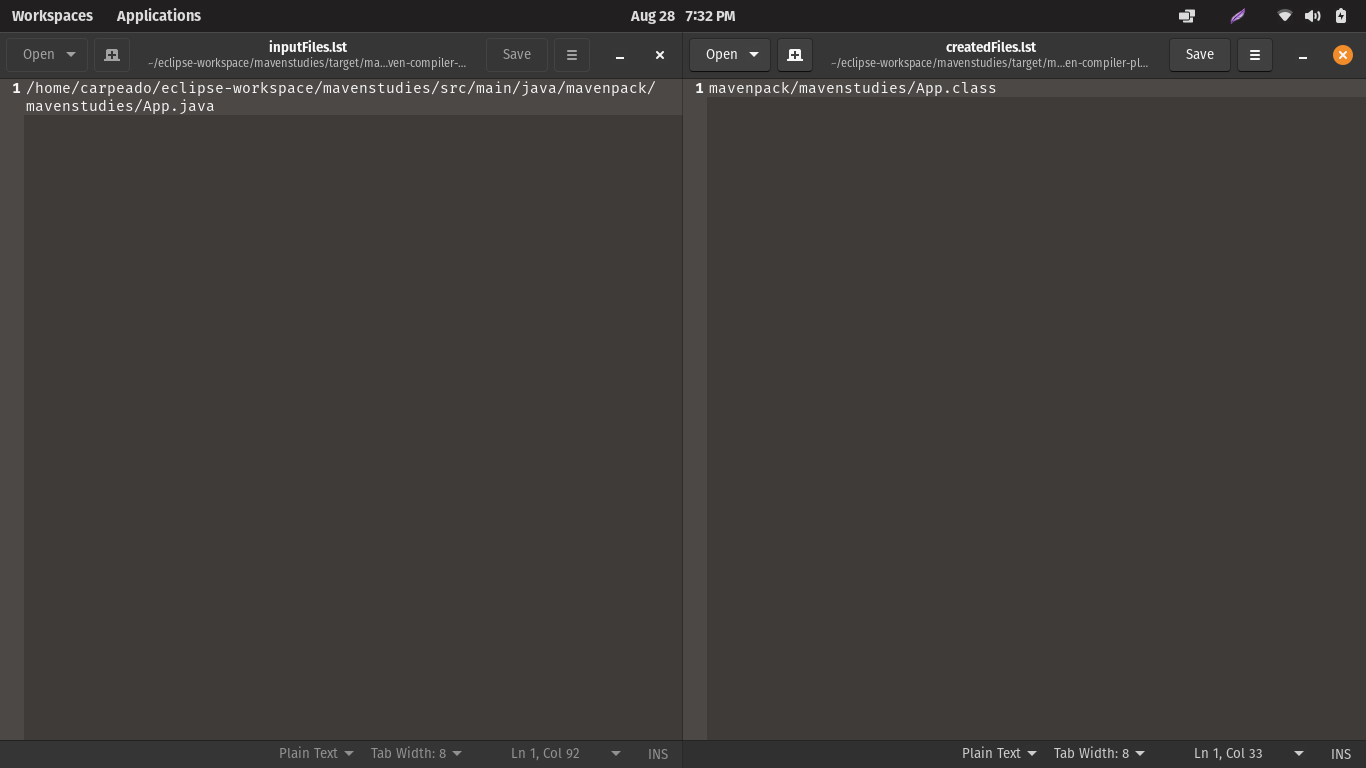
1. Em Goals você deverá especificar o que deseja fazer, em nosso caso: “clean package”:



1. Clique em Run. A função será executada, agora, se observamos o diretório target, iremos nos deparar com o recém gerado arquivo .jar:



Surgem também outros diretórios e arquivos, a maioria com finalidade de organização e compilação dos códigos. Temos, por exemplo, o diretório “maven-status”, o qual guarda arquivos com as informações das classes que foram compiladas, como esses:



**Dependências:**

Como mencionado anteriormente, com o Maven é possível gerenciar as bibliotecas externas do projeto de forma prática através do arquivo pom.xml, dentro do qual há uma lista com as dependências, o Maven tenta encontrá-las em duas localizações: primeiro é realizada uma busca no repositório local *“/home/User/.m2/repository”*, caso não seja encontrado determinado item, a procura segue no repositório público oficial “[*http://repo.maven.apache.org/maven2/*](http://repo.maven.apache.org/maven2/)*”,* se for encontrada a dependência desejada, será feita uma cópia para o repositório local e ocorrerá a disponibilidade dela para o projeto, deste modo, o Maven não precisará mais consultar o repositório público quando a mesma dependência for requisitada outra vez. Além disso, também é possível fazer uso de dependências de outros repositórios, para tal basta que seja feito o devido registro no arquivo pom.xml. Outro detalhe valioso é o fato de que, no caso de haver dependências de uma dependência, o Maven irá identificá-las e realizar o mesmo processo para cada uma automaticamente e disponibilizá-las para uso.

Como já mencionado, para adicionar uma biblioteca externa ao projeto Maven, basta definir isso no arquivo pom.xml, vamos realizar um exemplo utilizando o .jar necessário para conexão com o postgresql:

<dependency>

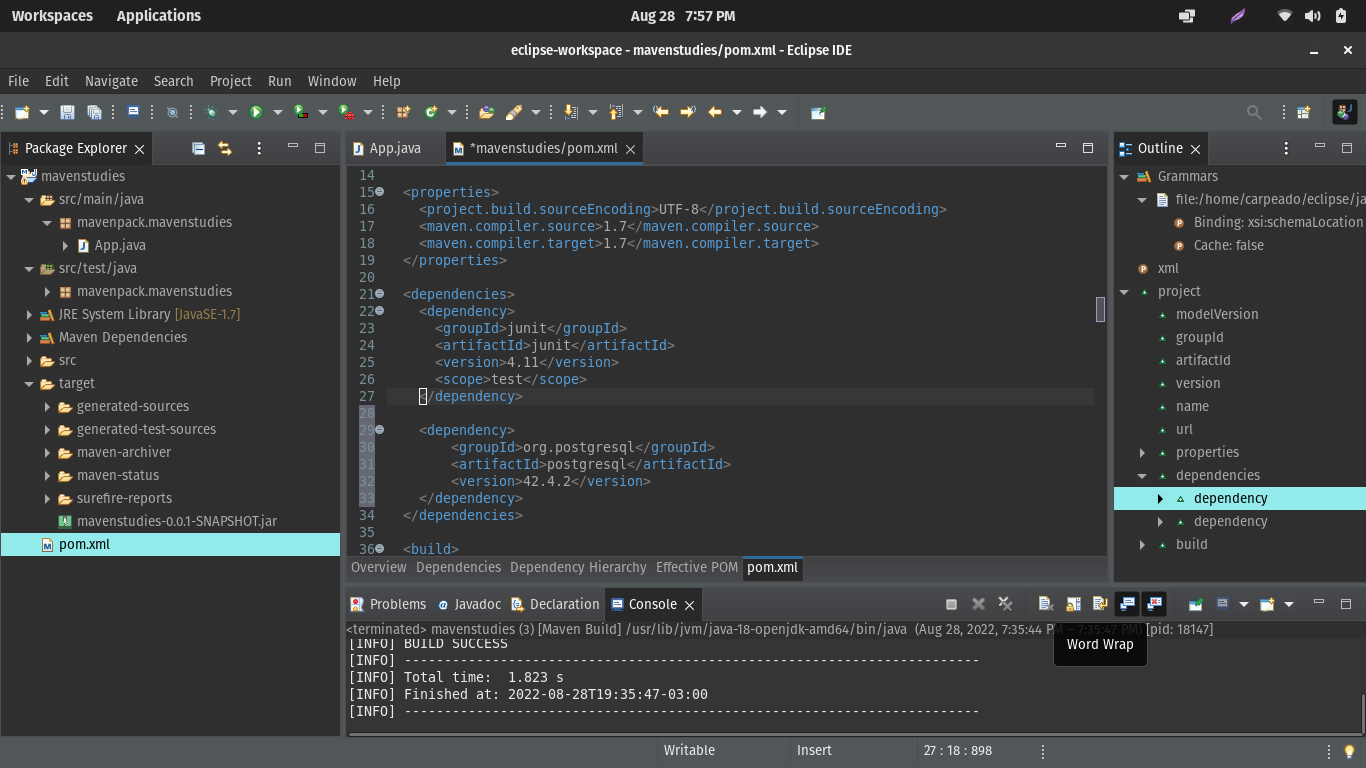
<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.4.2</version>

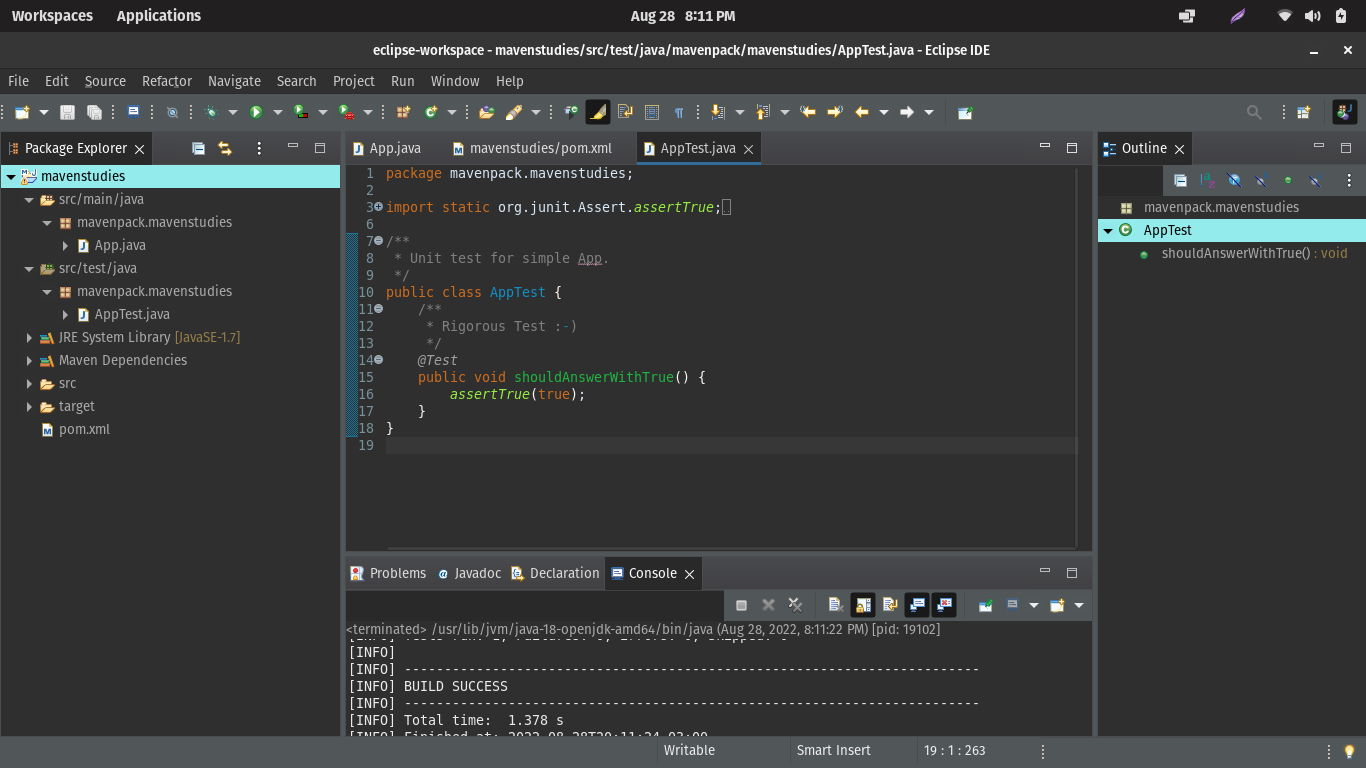
</dependency>

É preciso que o código acima seja colocado no pom.xml entre as tags <dependencies>, da seguinte forma:



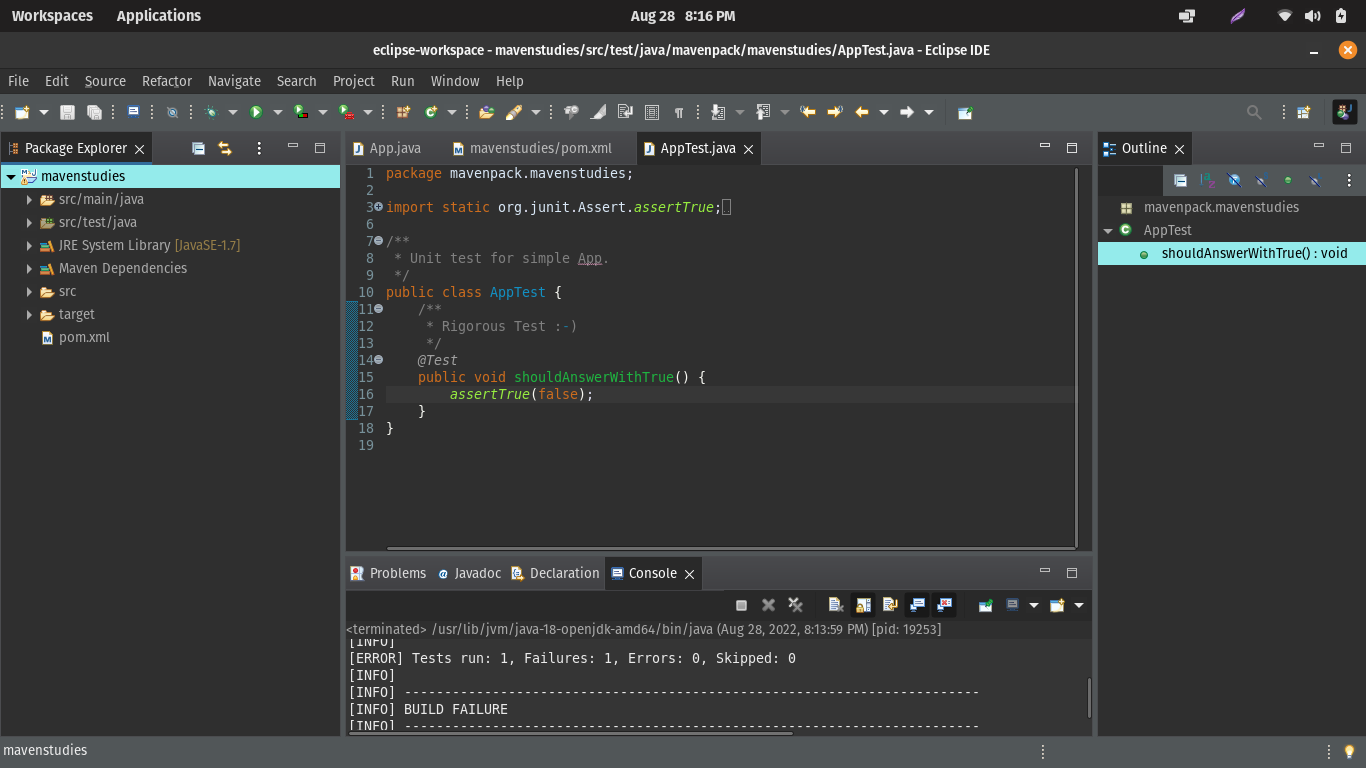
**Sobre os testes do Maven:**

Não pretendemos aprofundar nessa área, mas aqui vai uma introdução sobre. As classes de teste do maven funcionam através do Junit, uma biblioteca externa com a qual o projeto é automaticamente gerado pelo Eclipse. Essa biblioteca fornece alguns métodos que permitem testar coisas específicas do nosso código, por exemplo:



No caso desse teste, o retorno deve ser de não haver nenhuma falha ao executar o Maven test, como é possível averiguar no console.

Se utilizarmos, no entanto, esse código:



Será possível reparar no console, após a execução do maven test, que os testes não correram como deveriam e também será mostrada a quantidade de falhas.

**Exercícios:**

1. Crie um projeto Maven com o arquétipo maven-archetype-quickstart.
2. Utilize este projeto para colocar os códigos de conexão com banco de dados feitos nos últimos encontros.
3. Adicione a dependência do postgresql ao seu projeto e tente rodar os códigos, se houver algum problema, tente encontrar a solução.
4. Utilize o *clean compile* do Maven, dê um refresh no projeto e veja o resultado.
5. Utilize o *clean test* do Maven, dê um refresh no projeto e veja o resultado.
6. Utilize o *clean package* do Maven, dê um refresh no projeto e veja o resultado.
7. Utilize o *clean site* do Maven, dê um refresh no projeto e veja o resultado.
8. Utilize apenas o *clean* do Maven, dê um refresh no projeto e veja o resultado.