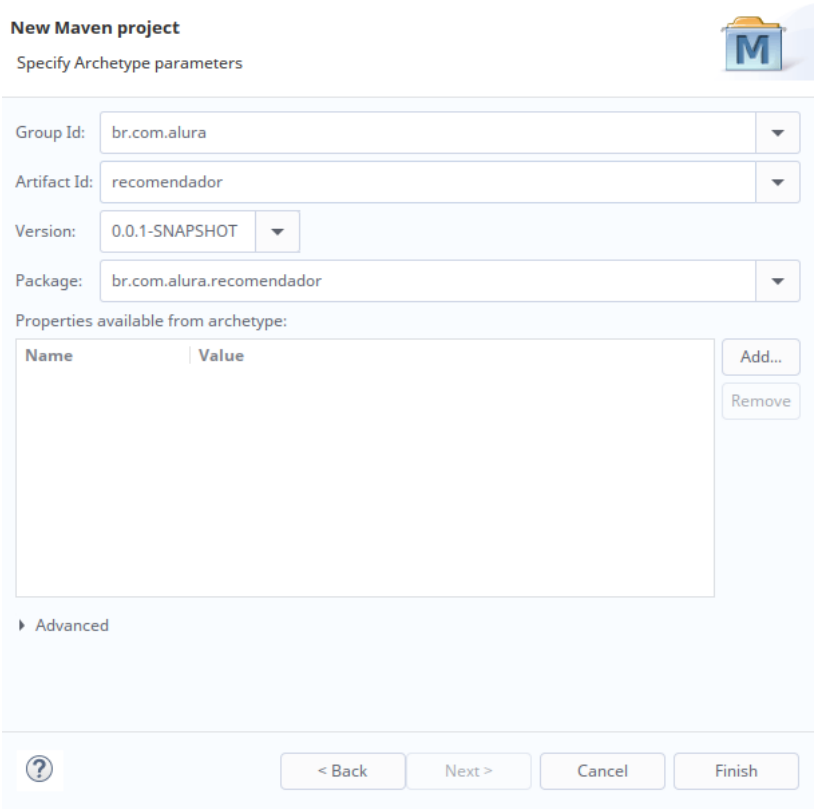
02

Pondo em prática

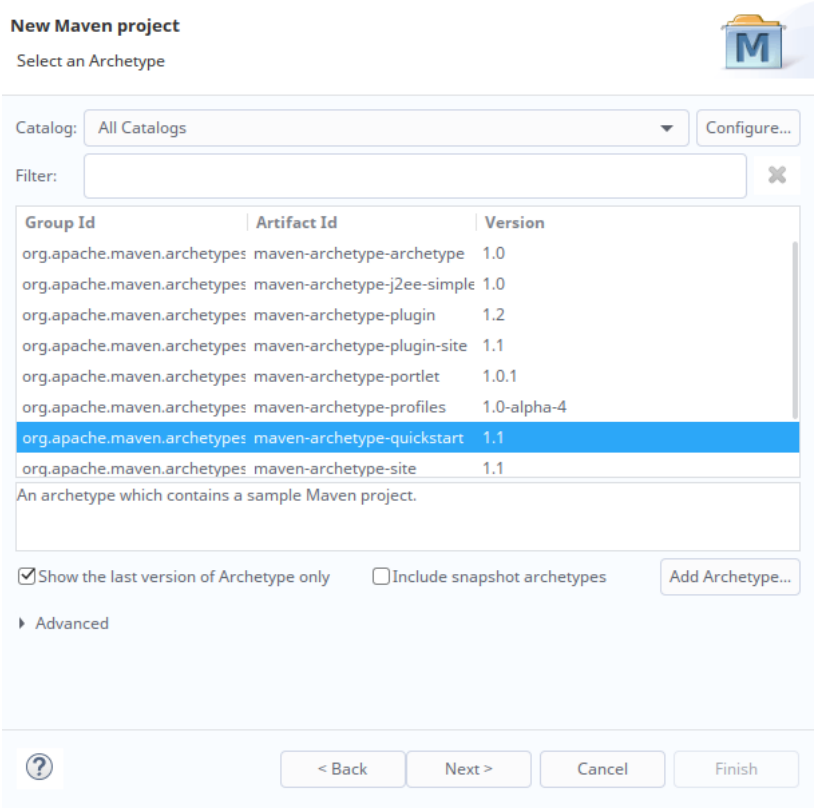
Transcrição

Vamos colocar o código que simulamos em prática? Primeiro baixamos o Eclipse (http://www.eclipse.org/downloads), a versão para Java EE Developers. Após o download descompactamos em um diretório de nossa preferência e depois o abrimos .

O próximo passo é criar um novo projeto **Maven** para que tenhamos as dependências baixadas a medida que precisamos e esse projeto será do tipo **Quick Start** (opção já selecionada).



Depois disso precisamos informar que o *GroupId* e o *Artifact* do projeto serão *br.com.alura* e *recomendador*, respectivamente.



Pode acontecer um erro informando que o Maven não conseguiu resolver o *artifact* do projeto, para resolver isso precisamos baixar o Maven e executá-lo manualmente pelo menos esta primeira vez. Para isso baixamos o Maven no site oficial ou clicando aqui (https://maven.apache.org/download.cgi). Vamos descompatar o mesmo e depois executar o seguinte comando onde queremos criar o projeto.

```
mvn archetype:generate -DgroupId=br.com.alura -DartifactId=recomendador -DarchetypeArtifactId=ma
```

Antes disso precisamos garantir que o Maven esteja no PATH do sistema operacional, se não o comando não funcionará nos sistemas UNIX. E isso é bem simples de se fazer, basta executar o seguinte:

```
PATH=$PATH:CAMINHO_PARA_BIN_DO_MAVEN
```

O que neste caso seria:

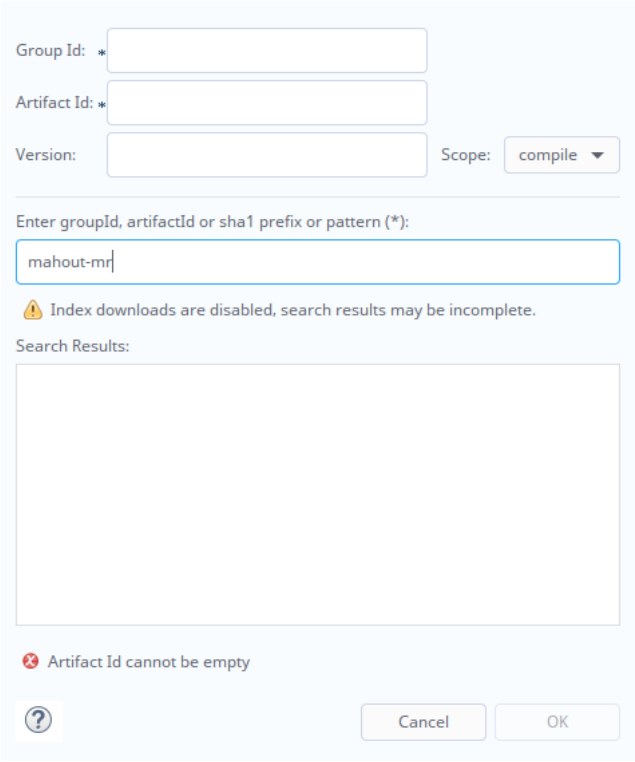
```
PATH=$PATH:/Users/alura/Downloads/apache-maven-3.3.9/bin
```

Utilizando o comando `mvn --version` devemos visualizar uma mensagem que indica a versão do Maven e dessa forma garantimos que tudo funcionou. Agora sim podemos criar um projeto com o comando anterior e depois disso importá-lo como projeto existente no Eclipse.

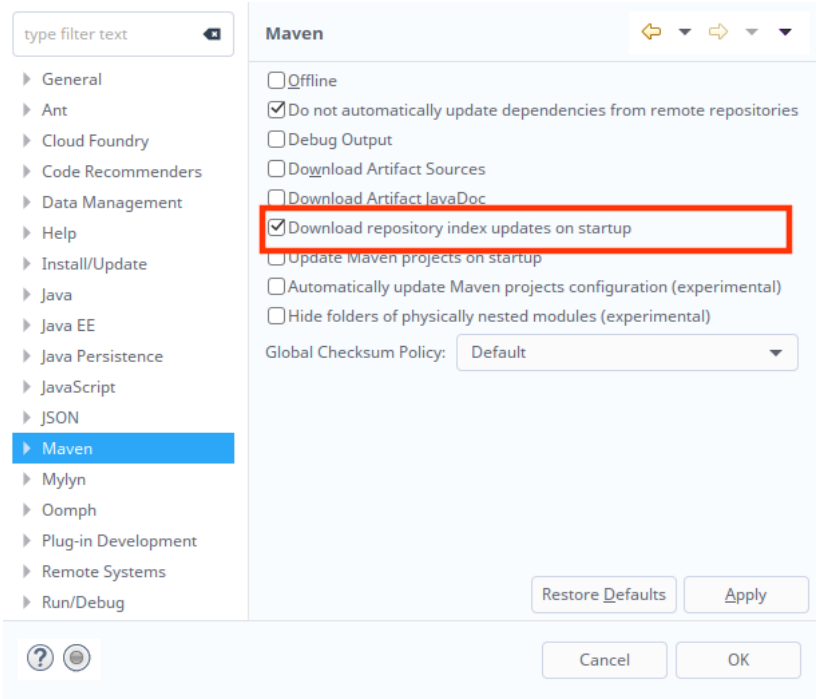
> Caso esse processo de criar o projeto com o Maven tenha soado complicado, veja nosso curso de Maven (https://cursos.alura.com.br/course/maven-build-do-zero-a-web).

Para escrever devidamente algum código ainda precisamos adicionar o Mahout em nosso projeto como uma dependência. Dessa forma o Maven se preocupa em baixá-lo e adicioná-lo em nosso projeto. Fazer esse processo manualmente é bem trabalhoso.

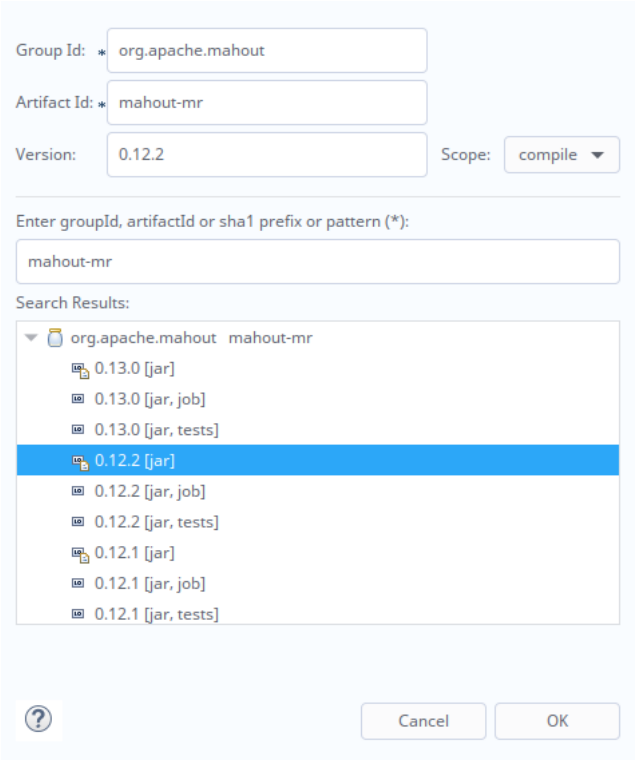
O que precisamos fazer é clicar com o botão direito do mouse sobre o projeto e selecionar as opções `Maven -> Add depedency` para adicionar uma dependência e na caixa de pesquisa digitar `mahout-mr` e teclar *enter*.



Note que a pesquisa não está listando os resultados e isso acontece justamente por que o padrão do plugin do Maven para o Eclipse não está configurado para isso. Vamos alterar essa configuração indo até as *Preferências do Eclipse* e na seção *Maven* marcar a opção que indica que o Maven deve baixar atualizações de bibliotecas ao inicializar.



Após isso, precisamos reiniciar o Eclipse e aguardar que abra novamente para que baixe as atualizações dos repositórios Maven. Após isso podemos adicionar a dependência do *mahour-mr*, conforme já vimos, e dessa vez escolheremos o **jar**, a versão **0.12.2**.



É possível adicionar o Mahout diretamente colando a dependência dentro do `pom.xml` do projeto. Porém, adicionando o *mahout-mr* como fizemos estamos adicionando várias outras dependências dele mesmo.

Outra dependência que precisamos adicionar é a do Hadoop Client, que será utilizada pelo Mahout. Neste caso, iremos digitar a dependência diretamente, só para vermos uma segunda forma de adicionarmos dependências em nosso projeto Maven. No arquivo `pom.xml` dentro da tag `dependecies` adicionaremos:

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <version>2.7.2</version>
</dependency>
```

Pronto! Já temos as dependências configuradas. Vamos criar nosso arquivo de dados onde teremos os dados que devem ser analisados pelo recomendador. O arquivo será um *CSV* como comentamos antes e já deixamos separados aqui uma cola para ser utilizada.

Baixar CSV de dados (https://raw.githubusercontent.com/alura-cursos/machine-learning-introducao-aos-sistemas-de-recomendacoes/master/src/main/resources/dados.csv)

Com tudo isso pronto, podemos criar a classe que fará uso de tudo que configuramos até então para recomendar itens baseado nesses dados. Já vimos esse código antes, então, vamos escrevê-lo. Primeiro criamos a classe `RecomendaProdutos` no pacote `br.com.alura` e nela criamos também o método `main` .

> Exclua outras classes que já tenham sido criadas juntamente com o projeto. Provável que uma classe App esteja no projeto. Pode apagá-la.

```
public class RecomendaProdutos {
  public static void main(String[] args) {
```

```
  }
}
```

Lembra-se quais eram os passos necessários até poder usar de fato o recomendador? Vamos fazê-los agora! Primeiro precisamos ler o arquivo `dados.csv` e criar o modelo.

```
public class RecomendaProdutos {
  public static void main(String[] args) throws IOException {
    File file = new File("dados.csv");
    FileDataModel model = new FileDataModel(file);
  }
}
```

Agora podemos criar as funções auxiliares! A primeira delas é baseada no usuário e a segunda na proximidade, depois disso criaremos o recomendador:

```
public class RecomendaProdutos {
  public static void main(String[] args) throws IOException, TasteException {
    File file = new File("dados.csv");
    FileDataModel model = new FileDataModel(file);

    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity, model);
    UserBasedRecommender recommender = new GenericUserBasedRecommender(model, neighborhood, sim
  }
}
```

O uso de `UserNeighborhood` ao invés de `ThresholdUserNeighborhood` e demais abreviações foi feito apenas para se basear na interface da classe ao invés de utilizar um objeto da classe direto.

Agora com o recomendador pronto, podemos solicitar 3 recomentações para o usuário 2 e em um laço, imprimir que recomendações foram feitas.

```
List<RecommendedItem> recommendations = recommender.recommend(2, 3);
for (RecommendedItem recommendation : recommendations) {
  System.out.println(recommendation);
}
```

Executando o código como temos agora o resultado será impresso como apresentado abaixo:

```
RecommendedItem[item:12, value:4.8328104]
RecommendedItem[item:13, value:4.6656213]
RecommendedItem[item:14, value:4.331242]
```

Então, caso queiramos criar um recomendador em nossos sistemas, tudo que precisamos está feito nestes passos: criar um aquivo *csv* com os dados de identificador de usuário, identificador de item e a nota de avaliação.

Depois disso, no código, lemos este arquivo e dele criamos o modelo, a função de similaridade e de proximidade. Logo em seguida, podemos sair pedindo recomendações.

Note que caso ocorra alteração do número de recomendações para mais, o recomendador não fará mais que três recomendações pois o usuário 2 de 9 itens já avaliou 6, e se o número de recomendações for menor que 3, os de maior nota serão recomendados.

Já somos capazes de fazer recomendações e entender o que se passa por trás da lógica de recomendações. Nosso próximo passo antes de utilizar dados reais será avaliar um pouco mais algumas variações desse mesmo código.