Este documento possui como objetivo descrever passo-a-passo como gerar um projeto para fazer refactorings utilizando MoDisco e ATL.

Ferramentas utilizadas:

* Eclipse 2019-09 (4.13.0)
* MoDisco (1.2.0.201805182005)
  + Para instalar a versão correta do MoDisco, na versão 4.13.0 do Eclipse, é necessário fazer as seguintes instalações manualmente:
    - <http://download.eclipse.org/facet/updates/release/1.3.0/>
    - <http://download.eclipse.org/modeling/mdt/modisco/updates/release/1.2.0/>
* ATL

1. Gerar modelo Java utilizando MoDisco

Botão direito sobre o projeto Java 🡪 Discoveres 🡪 Discover Java Project... 🡪 Sete a flag **SERIALIZE\_TARGET** para **true**

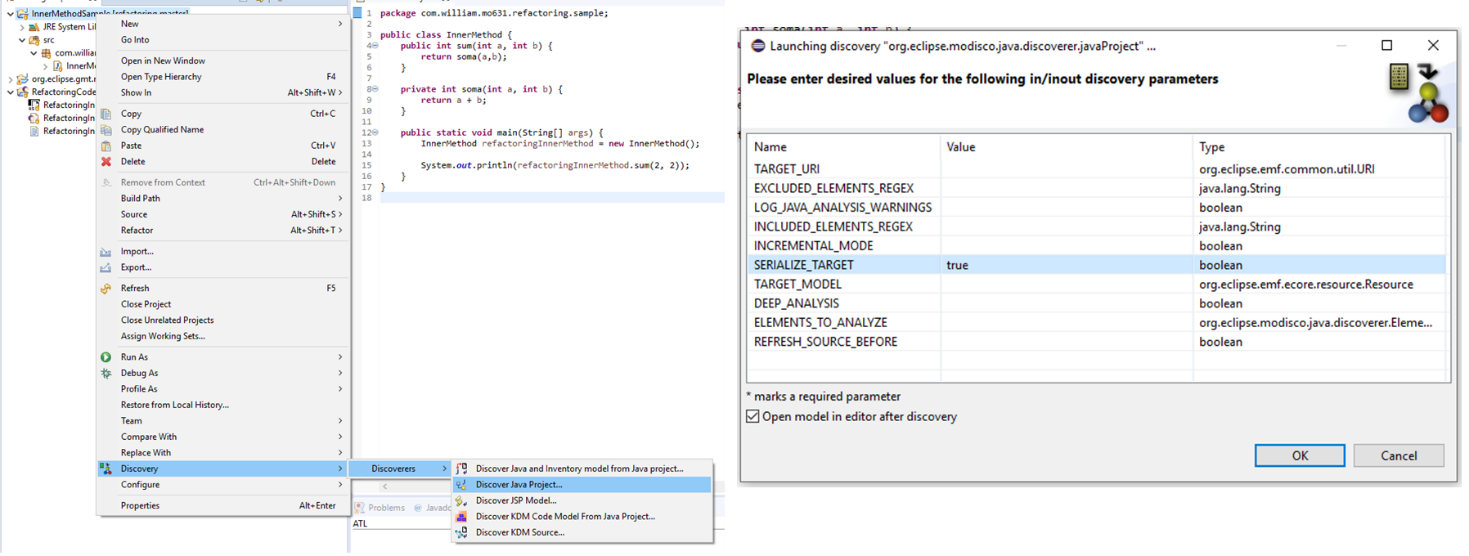


Figura - Como gerar modelo Java e configuração do MoDisco

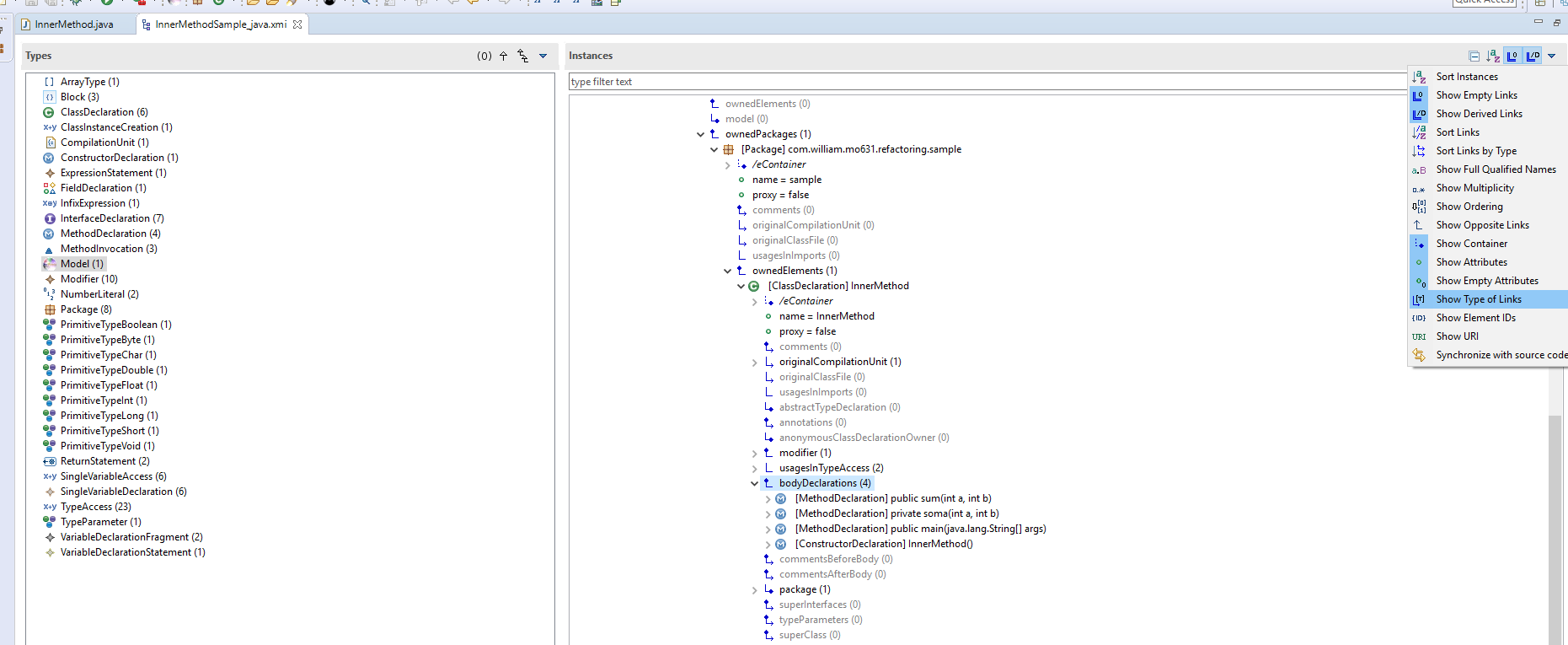


Figura - Visualização do modelo gerado e personalização para exibir o tipo dos elementos (facilita para gerar as queries OCL)

1. Gerar projeto ATL

File 🡪 New 🡪 Project 🡪 ATL 🡪 ATL Project

Dê um nome para seu projeto e clique em **Finish**

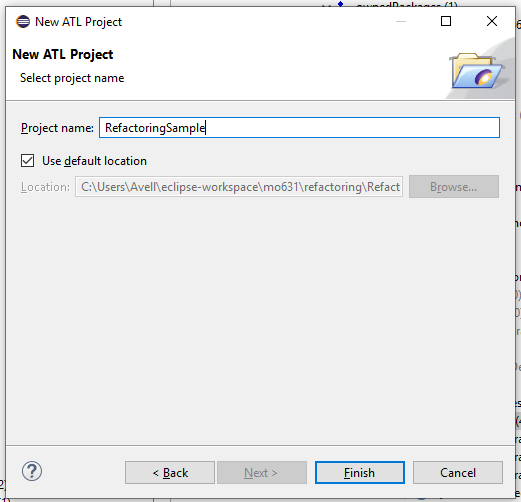


Figura 3 - Criando projeto ATL

Adicione um novo arquivo ATL ao projeto criado

Botão direito sobre o projeto 🡪 New 🡪 Other 🡪 ATL 🡪 ATL file

Coloque um nome ao seu arquivo e clique em **Finish**

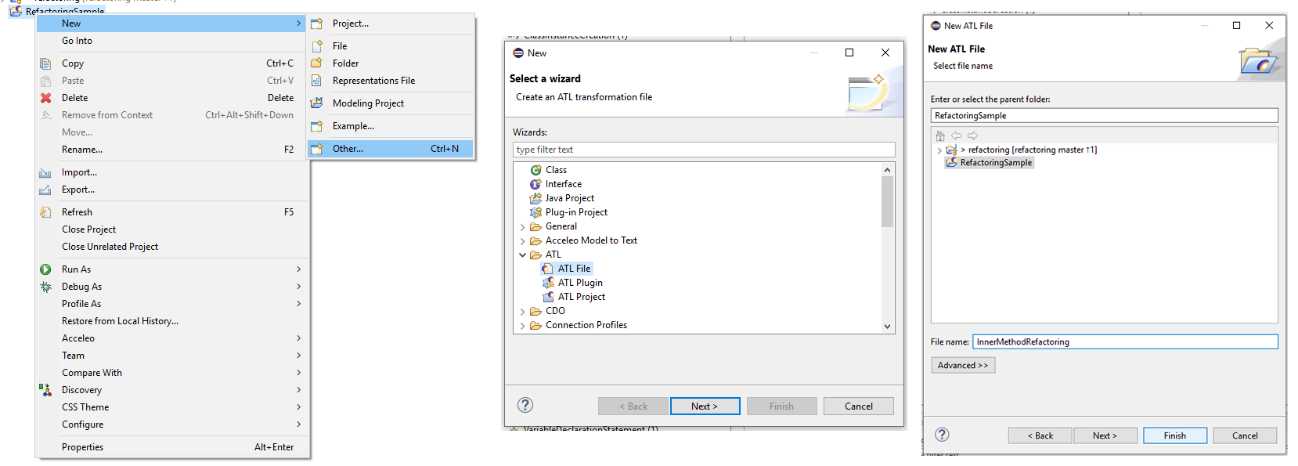


Figura 4 - Adicionando arquivo ATL ao projeto

O projeto está com um erro. Para corrigir é necessário fazer algumas alterações no arquivo ATL indicando qual a transformação é desejada. Para fazer isso, primeiro é necessário adicionar a referência ao metamodelo do Java utilizado pelo MoDisco.

Botão direito sobre o projeto 🡪 Import 🡪 Plug-in Development 🡪 Plug-ins and Fragments 🡪 Clique no botão **Next** 🡪 Clique novamente em **Next** 🡪Filtre por *modisco.java* 🡪 selecione o plugin: **org.eclipse.gmt.modisco.java.browser.customization** 🡪 Clique em **Add** 🡪 Clique em **Finish**

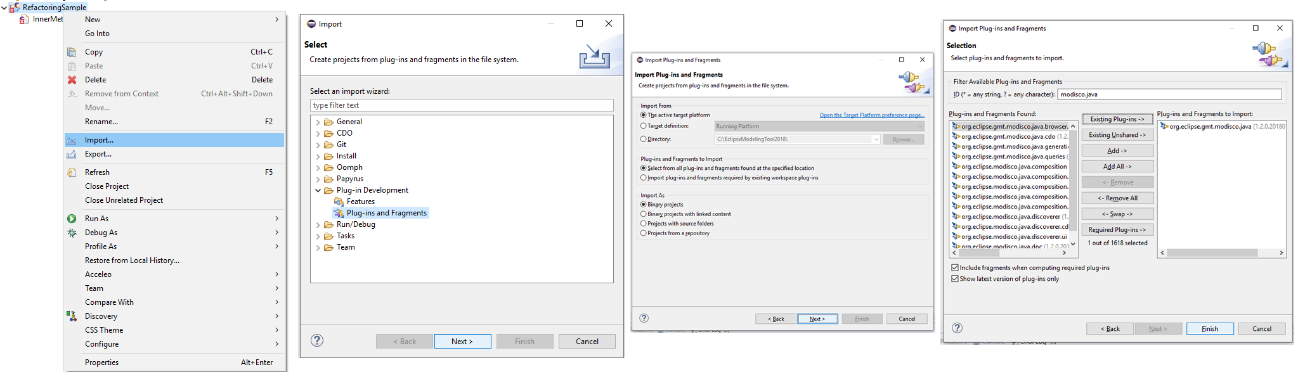


Figura 5 - Adicionando referência ao plug-in do MoDisco que possui o metamodelo Java

O projeto ainda estará com erro, agora é necessário configurar o arquivo ATL com as transformações desejadas.

Abra o arquivo ATL adicionado e insira as seguintes informações:

**-- @atl2010** 🡪 indica o compilador ATL a ser utilizado

**-- @path Java=/org.eclipse.gmt.modisco.java/model/java.ecore** 🡪 Indica o caminho do metamodelo a ser utilizado nas transformações (*java.ecore*) e atribui um apelido ao mesmo (*Java*).

**create OUT: Java refining IN: Java;** 🡪 indica que a transformação ocorrerá do modelo de entrada específico (IN: Java) para o modelo de saída específico (OUT: Java). Como o projeto visa criar refactorings, utilizamos a modo de refinamento (refining), dessa forma somente serão alterados os elementos que sofrerem alguma alteração durante o processo de transformação M2M.

**Obs.:** Somente é possível usar o modo de refinamento se o modelo de entrada e o modelo de saída são os mesmos.

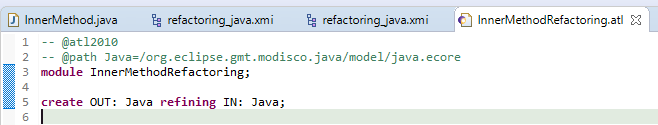


Figura 6 - Arquivo ATL atualizado

Após esses passos o projeto já não apresentará erros, porém ainda não faz nenhuma transformação, pois não há nenhuma tarefa no arquivo ATL para isso.

Afim de terminar a preparação do ambiente, agora é necessário configurar a execução do projeto, para isto siga os seguintes passos:

Botão direito sobre o projeto ATL 🡪 Run As 🡪 Run Configurations... 🡪 Duplo clique sobre ATL Transformation 🡪 Renomeie a configuração 🡪 Clique em workspace e selecione seu projeto 🡪 Clique em Ok

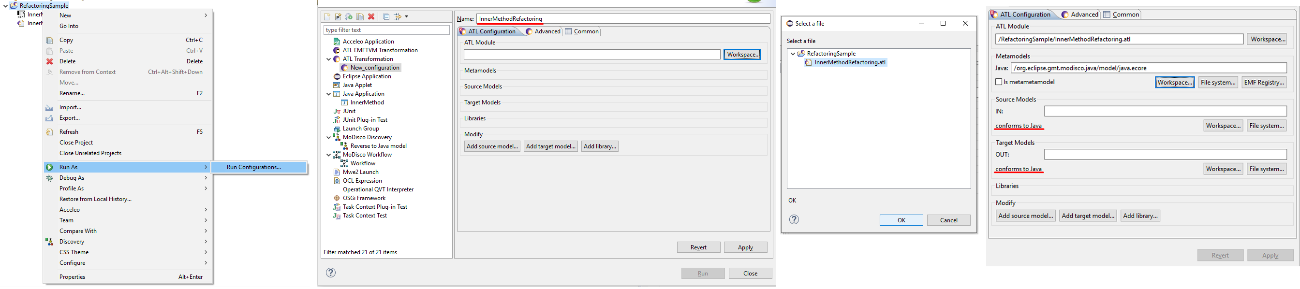


Figura 7 - Configurando execução do projeto ATL (1)

Tanto em *Source Models* quanto em *Target Models*, será apresentado o modelo o qual a transformação deve ser conforme, no caso desse projeto Java.

Para finalizar a configuração, é necessário inserir o XMI a ser transformado e também o XMI a ser gerado. Para tal siga os passos a seguir.

Source Models 🡪 IN: 🡪 Selecione o arquivo gerado a partir do MoDisco

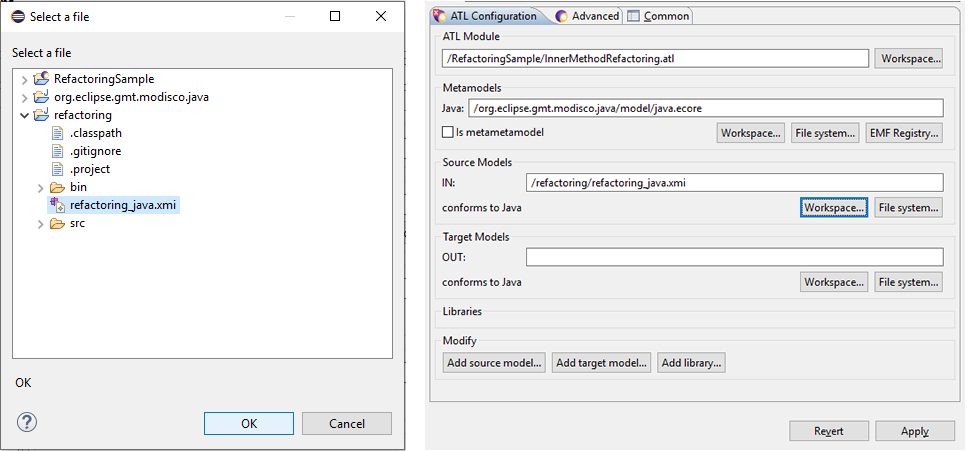


Figura 8 - Configurando execução do projeto ATL (2)

Target Models 🡪 OUT: 🡪 Selecione o local onde será gerado o arquivo resultante da transformação 🡪 Dê um nome para o arquivo resultante, sendo que o final necessariamente deve ser \_java.xmi (sufixo utilizado pelo MoDisco para reconhecer e gerar o código resultante da transformação).

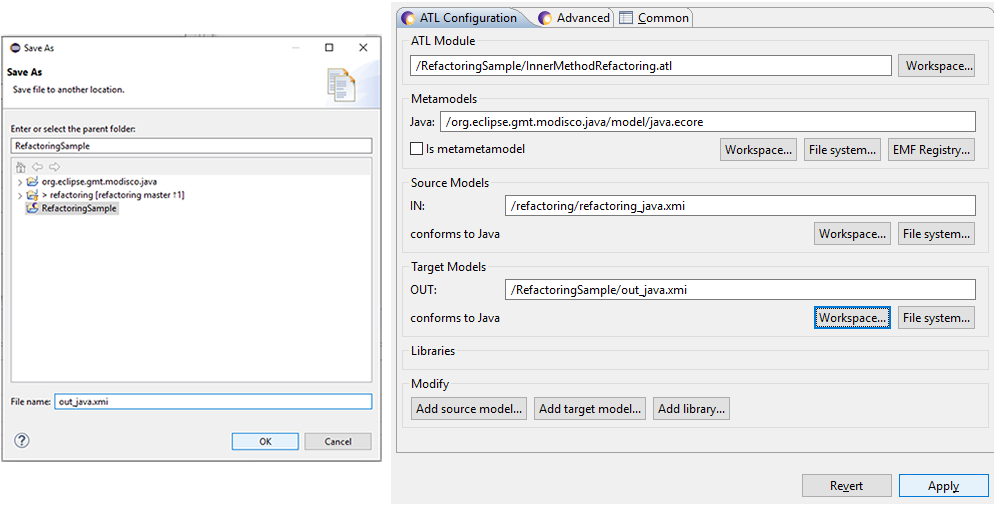


Figura 9 - Configurando execução do projeto ATL (3)

Finalizada esta etapa nos resta adicionar as funções para realizar de fato o refactoring do código.

1. Funções ATL para realizar o refactoring
2. Geração do código Java após a aplicação do refactoring

Para gerar o código Java novamente, será criado um novo projeto utilizando Gradle para gerenciar as dependências.

File 🡪 New 🡪 Other 🡪 Gradle Project 🡪 Next🡪 Next 🡪 Dê um nome ao seu projeto 🡪 Finish

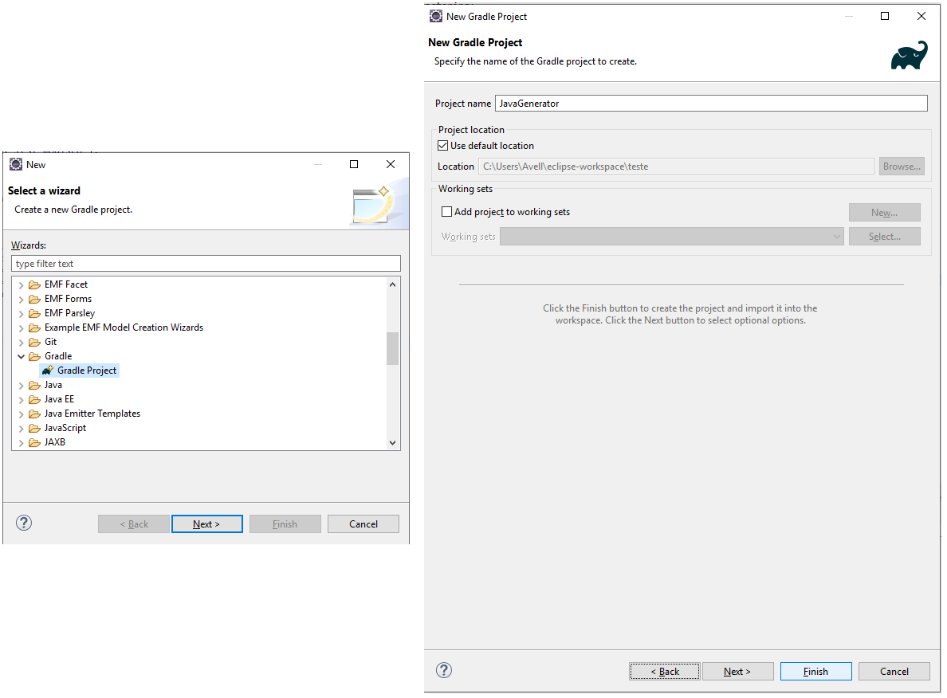


Figura - Criando projeto Gradle

O projeto Gradle já será criado com a dependência do projeto Guava fornecido pelo Google, porém caso não haja tal dependência explícita em seu projeto, apenas adicione-a. Seu arquivo de dependências deverá ser parecido com o demostrado na Figura 11.

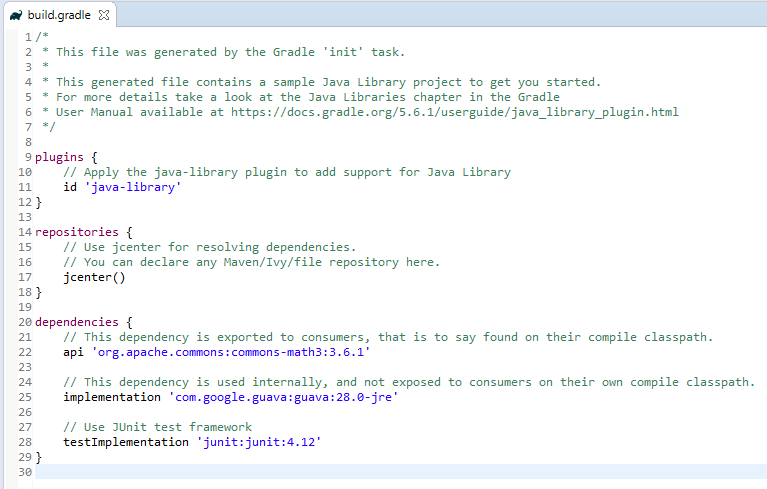


Figura - Arquivo de dependências build.gradle

Assim como foi feito no projeto ATL, é necessário adicionar os plugins do MoDisco ao projeto que estamos criando agora para recriar o código Java. Os plugins a serem adicionados são:

* org.eclipse.gmt.modisco.java.generation
* org.eclipse.modisco.util.emf.core

OBS.: O passo-a-passo sobre como importar plugins a um projeto no eclipse pode ser visto na Figura 5.

Após adicionar todas as dependências, vamos implementar a classe Java responsável por gerar o código após o refactoring ser aplicado. A classe deve seguir o modelo apresentado a seguir.

|  |
| --- |
| Classe Generation.java |
| package com.william.mo631.generator;  import java.io.File;  import java.io.IOException;  import java.util.ArrayList;  import org.eclipse.emf.common.util.URI;  import org.eclipse.emf.ecore.EPackage;  import org.eclipse.emf.ecore.resource.Resource;  import org.eclipse.emf.ecore.xmi.XMIResource;  import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;  import org.eclipse.emf.ecore.xmi.impl.XMIResourceImpl;  import org.eclipse.gmt.modisco.java.emf.JavaPackage;  import org.eclipse.gmt.modisco.java.generation.files.GenerateJavaExtended;  public class Generation {  public static void main(String[] args) throws IOException {  EPackage.Registry.INSTANCE.put(JavaPackage.eNS\_URI, JavaPackage.eINSTANCE);  XMIResourceFactoryImpl xmiResourceFactoryImpl = new XMIResourceFactoryImpl() {  public Resource createResource(URI uri) {  XMIResource xmiResource = new XMIResourceImpl();  return xmiResource;  }  };  Resource.Factory.Registry.INSTANCE.getExtensionToFactoryMap().put("xmi", xmiResourceFactoryImpl);  GenerateJavaExtended javaGenerator = new GenerateJavaExtended(  URI.createFileURI("src/main/resources/refactoringTest\_java.xmi"),  new File("src/main/generated/"),  new ArrayList<Object>());  javaGenerator.doGenerate(null);  }  } |