# Compile Kermeta Project

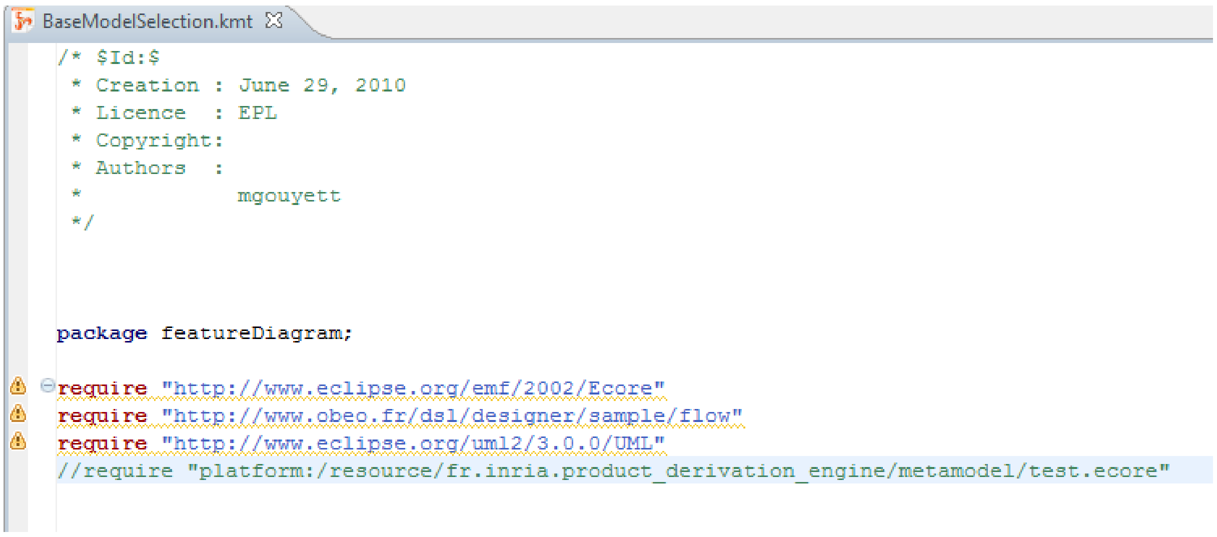
This document will present how to compile a Kermeta Project.

You need to update the following jars :

- kermeta.compilo.scala-0.0.2-SNAPSHOT.jar
- language.framework.scala-0.10.0-SNAPSHOTS.jar

## 1. Require metamodel URI

If you want to derive a model conforms to a given metamodel you need to require it on a Kermeta file. You can either require a metamodel URI or the ecore file of the metamodel. You need to deploy metamodel plugin before.



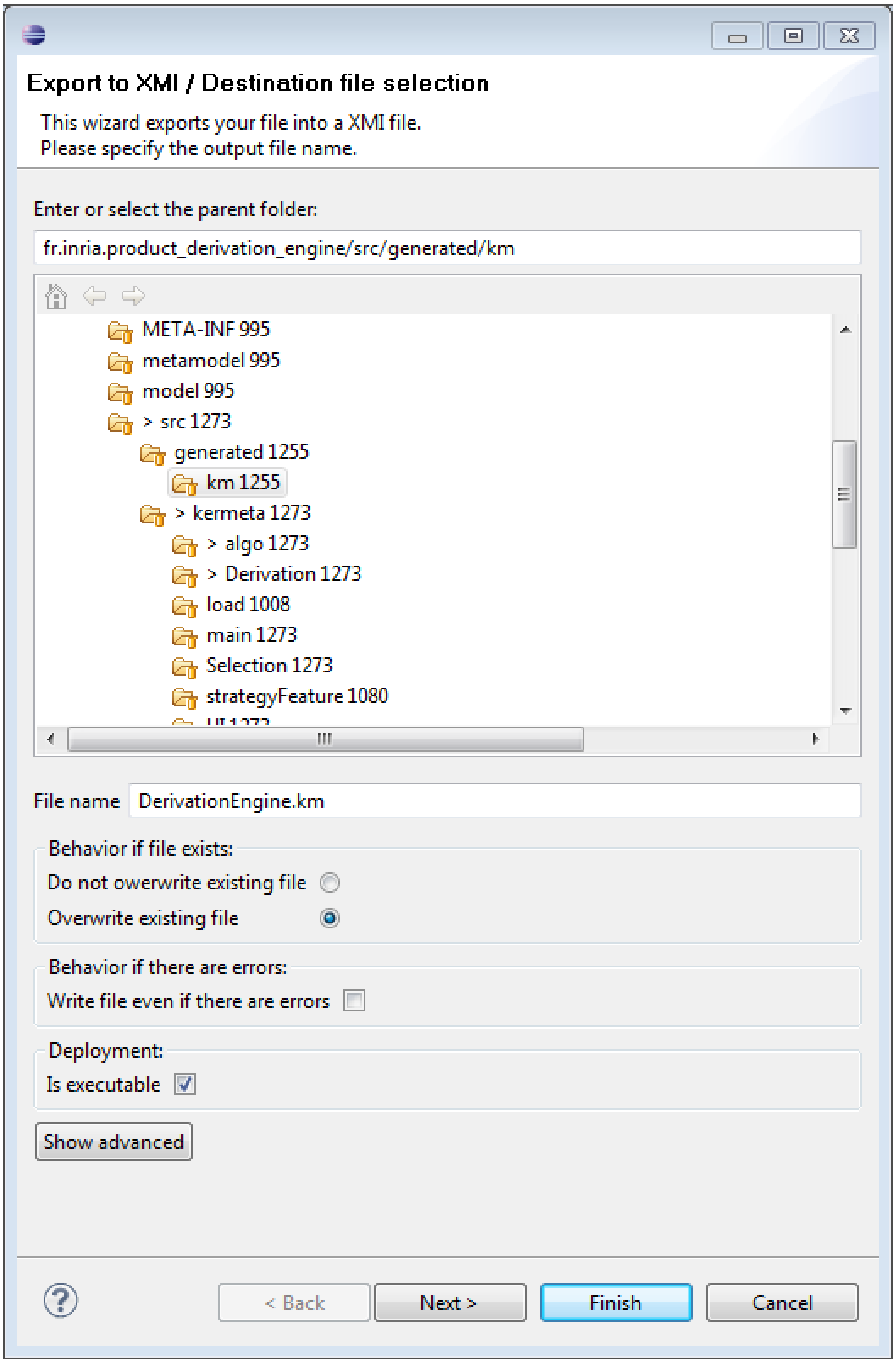
BaseModelSelection.kmt

```
/* $Id:$
 * Creation : June 29, 2010
 * Licence  : EPL
 * Copyright:
 * Authors  :
 *           mgouyett
 */


package featureDiagram;

require "http://www.eclipse.org/emf/2002/Ecore"
require "http://www.obeo.fr/dsl/designer/sample/flow"
require "http://www.eclipse.org/uml2/3.0.0/UML"
//require "platform:/resource/fr.inria.product_derivation_engine/metamodel/test.ecore"
```

## 2. Create Kermeta model file (.km)

We supposed you have a Kermeta project without any errors and a Kermeta main class (it contains mainClass and mainOperation tag). A first step to compile your Kermeta project as a jar is to generate a new executable Kermeta Model file (.km).

So, right click on the Kermeta main class on the Eclipse Package Explorer View -> Kermeta-> Generate km (Kermeta model).


Export to XMI / Destination file selection

This wizard exports your file into a XMI file.
Please specify the output file name.

Enter or select the parent folder:

fr.inria.product_derivation_engine/src/generated/km

```
META-INF 995
metamodel 995
model 995
> src 1273
  generated 1255
    km 1255
  > kermeta 1273
    > algo 1273
    > Derivation 1273
    load 1008
    main 1273
    Selection 1273
    strategyFeature 1080
```

File name: DerivationEngine.km

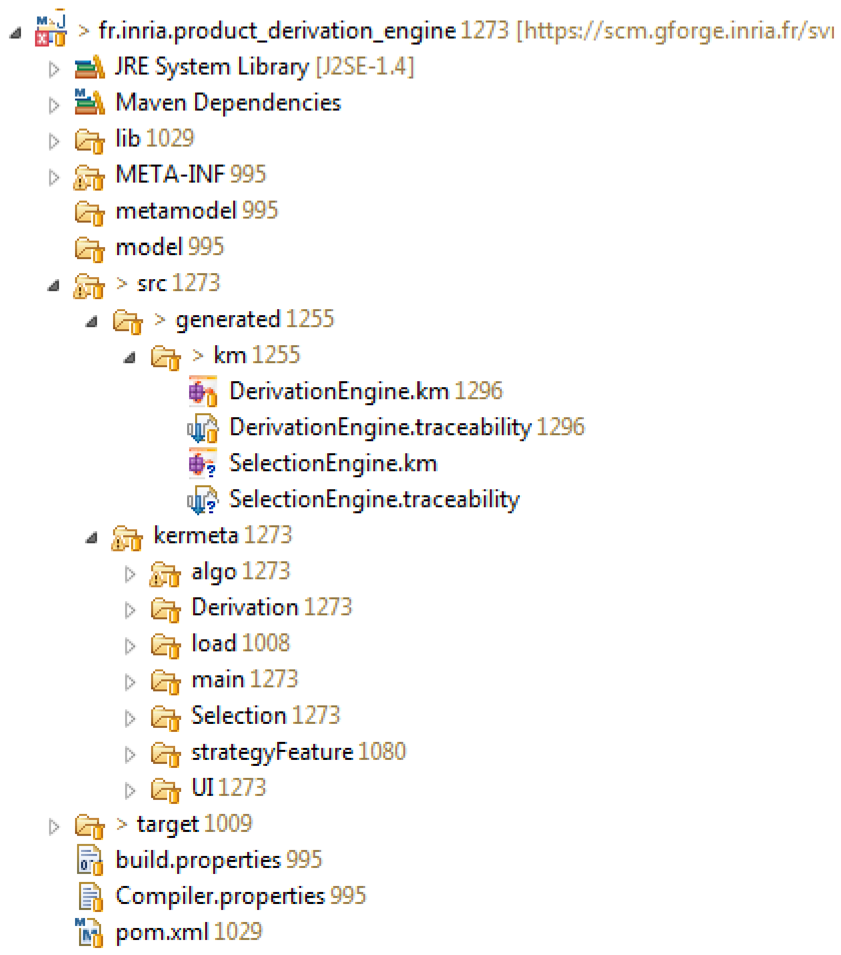
Behavior if file exists:
Do not owerwrite existing file ○
Overwrite existing file ●

Behavior if there are errors:
Write file even if there are errors ☐

Deployment:
Is executable ☑

Store it in on the folder src/generated/km (if it does not exists create it) and **do not forget to check executable box**. A new km file that represent your Kermeta code appears.

Our project should look like this :

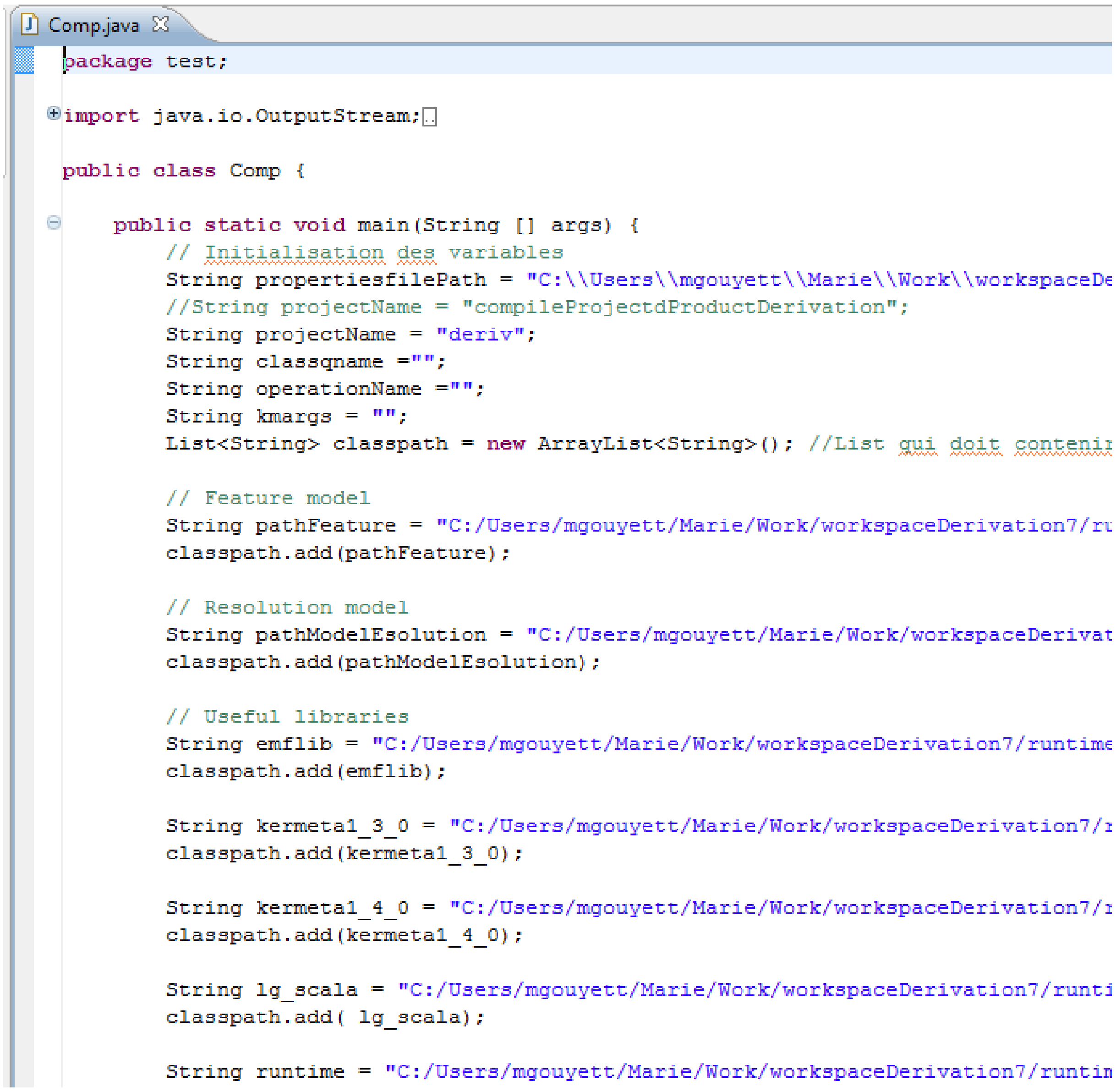


```
> fr.inria.product_derivation_engine 1273 [https://scm.gforge.inria.fr/svı
    JRE System Library [J2SE-1.4]
    Maven Dependencies
    lib 1029
    META-INF 995
    metamodel 995
    model 995
    > src 1273
        > generated 1255
            > km 1255
                DerivationEngine.km 1296
                DerivationEngine.traceability 1296
                SelectionEngine.km
                SelectionEngine.traceability
        kermeta 1273
            algo 1273
            Derivation 1273
            load 1008
            main 1273
            Selection 1273
            strategyFeature 1080
            UI 1273
    > target 1009
    build.properties 995
    Compiler.properties 995
    pom.xml 1029
```

## 3. Use Java Project compileProjects

Go to the Java project compileProject and :

1. Change all path in the java file Comp.java with your own absolute path to your current workspace.

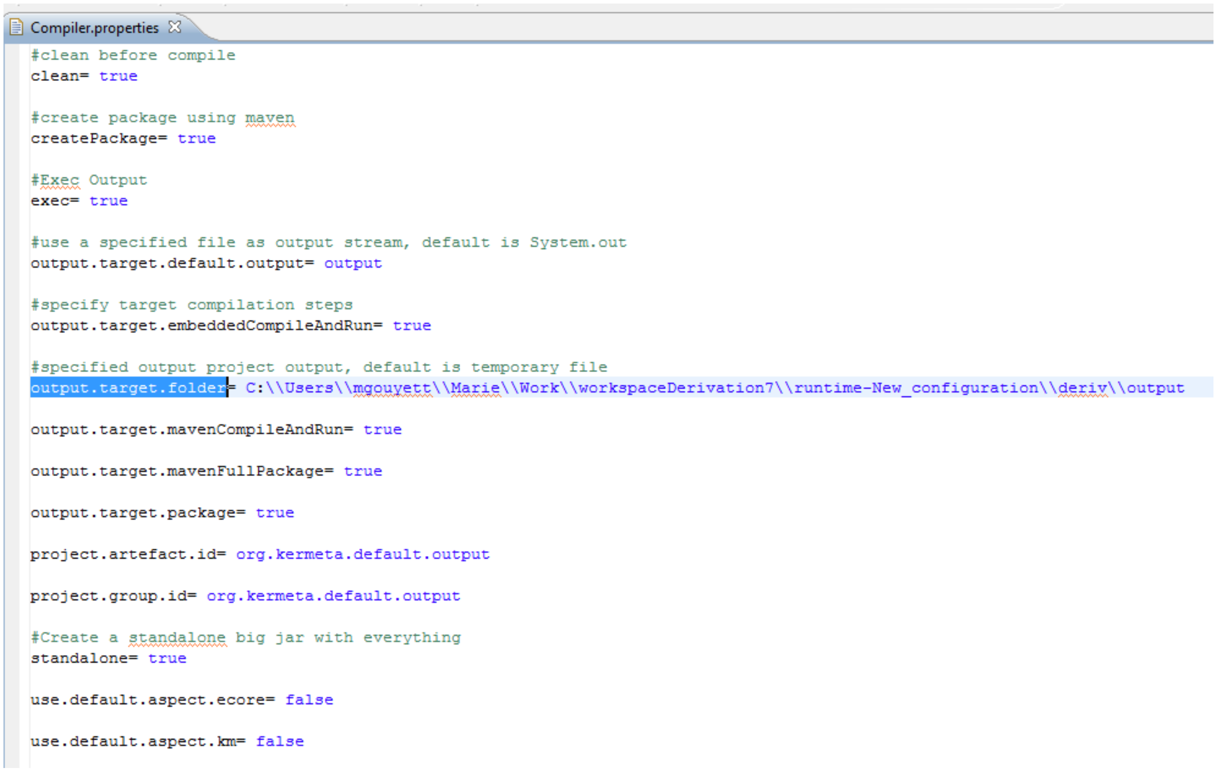


```
package test;

import java.io.OutputStream;

public class Comp {

    public static void main(String [] args) {
        // Initialisation des variables
        String propertiesfilePath = "C:\\Users\\mgouyett\\Marie\\Work\\workspaceDe
        //String projectName = "compileProjectdProductDerivation";
        String projectName = "deriv";
        String classqname ="";
        String operationName ="";
        String kmargs = "";
        List<String> classpath = new ArrayList<String>(); //List qui doit contenir

        // Feature model
        String pathFeature = "C:/Users/mgouyett/Marie/Work/workspaceDerivation7/ru
        classpath.add(pathFeature);

        // Resolution model
        String pathModelEsolution = "C:/Users/mgouyett/Marie/Work/workspaceDerivat
        classpath.add(pathModelEsolution);

        // Useful libraries
        String emflib = "C:/Users/mgouyett/Marie/Work/workspaceDerivation7/runtime
        classpath.add(emflib);

        String kermeta1_3_0 = "C:/Users/mgouyett/Marie/Work/workspaceDerivation7/r
        classpath.add(kermeta1_3_0);

        String kermeta1_4_0 = "C:/Users/mgouyett/Marie/Work/workspaceDerivation7/r
        classpath.add(kermeta1_4_0);

        String lg_scala = "C:/Users/mgouyett/Marie/Work/workspaceDerivation7/runti
        classpath.add( lg_scala);

        String runtime = "C:/Users/mgouyett/Marie/Work/workspaceDerivation7/runtin
```

2. Change compile propertiesFilePath (first variable of the program) to match with your current workspace

3. Select a name for your output project, here “deriv”.

4. On the file Compiler.properties on the folder properties change the output.target.folder your own path to our project target folder (here deriv/output)

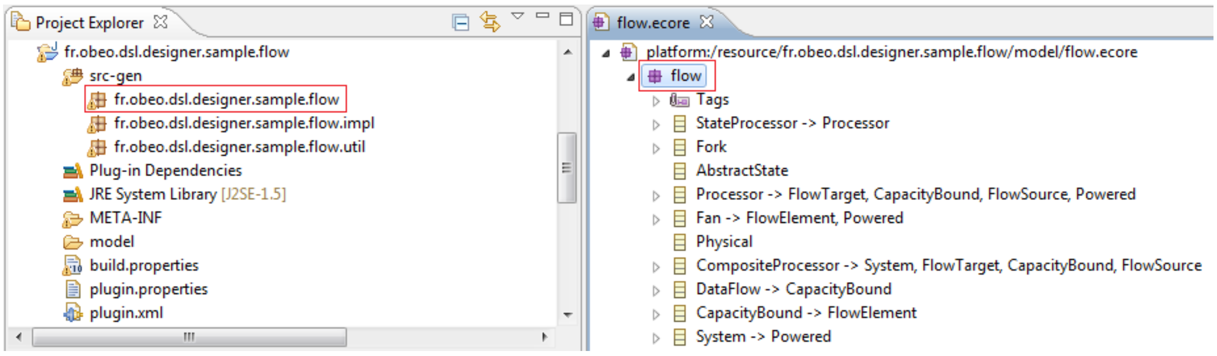

Compiler.properties

```
#clean before compile
clean= true

#create package using maven
createPackage= true

#Exec Output
exec= true

#use a specified file as output stream, default is System.out
output.target.default.output= output

#specify target compilation steps
output.target.embeddedCompileAndRun= true

#specified output project output, default is temporary file
output.target.folder= C:\\Users\\mgouyett\\Marie\\Work\\workspaceDerivation7\\runtime-New_configuration\\deriv\\output

output.target.mavenCompileAndRun= true

output.target.mavenFullPackage= true

output.target.package= true

project.artefact.id= org.kermeta.default.output

project.group.id= org.kermeta.default.output

#Create a standalone big jar with everything
standalone= true

use.default.aspect.ecore= false

use.default.aspect.km= false
```

5. Add our metamodel jar project on the folder lib of the compileProject project and add it on the java file Comp.java as in the figure above.

6. If your metamodel package name does not have the same name as the name of the package in the EMF java project (like in the figure above), you need to change the package name by

```
// Change package name to adjust ecore model package name and
corresponding EMF Java code package
		kermeta.utils.TypeEquivalence.packageEquivelence().put("flow","
fr.obeo.dsl.designer.sample.flow");
```


Project Explorer

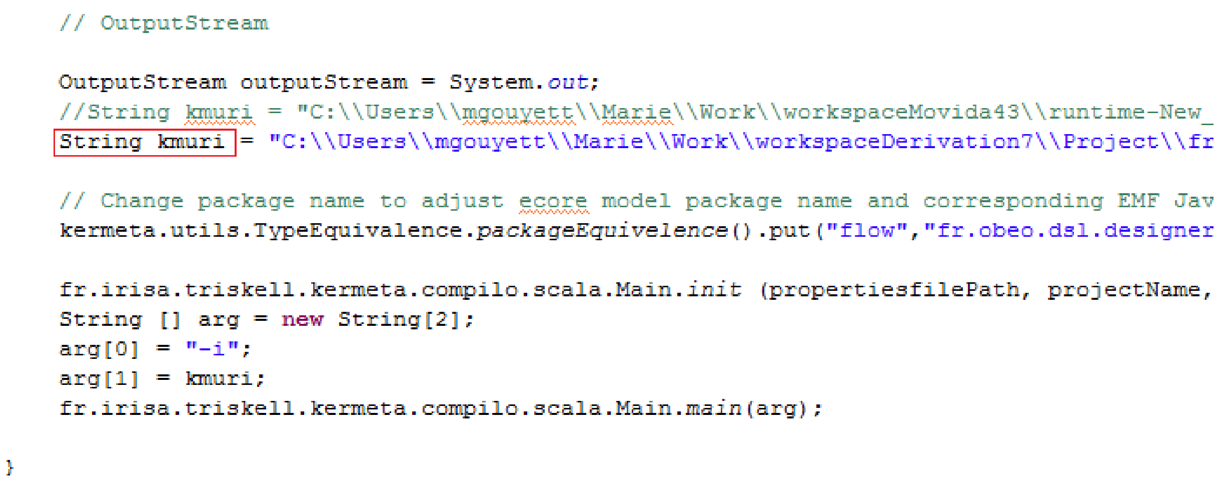
```
fr.obeo.dsl.designer.sample.flow
  src-gen
    fr.obeo.dsl.designer.sample.flow
    fr.obeo.dsl.designer.sample.flow.impl
    fr.obeo.dsl.designer.sample.flow.util
  Plug-in Dependencies
  JRE System Library [J2SE-1.5]
  META-INF
  model
  build.properties
  plugin.properties
  plugin.xml
```

flow.ecore

```
platform:/resource/fr.obeo.dsl.designer.sample.flow/model/flow.ecore
  flow
    Tags
    StateProcessor -> Processor
    Fork
    AbstractState
    Processor -> FlowTarget, CapacityBound, FlowSource, Powered
    Fan -> FlowElement, Powered
    Physical
    CompositeProcessor -> System, FlowTarget, CapacityBound, FlowSource
    DataFlow -> CapacityBound
    CapacityBound -> FlowElement
    System -> Powered
```

The figure above shows that flow meta-model does not have the same package name as in the EMF generated Java code.

7. Change the String kmuri value to the path to your own kermeta model created in the last section.



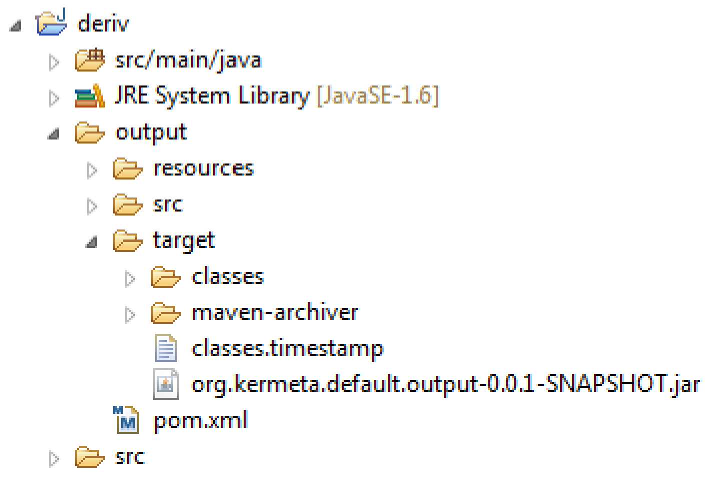
```
    // OutputStream

    OutputStream outputStream = System.out;
    //String kmuri = "C:\\Users\\mgouyett\\Marie\\Work\\workspaceMovida43\\runtime-New_
    String kmuri = "C:\\Users\\mgouyett\\Marie\\Work\\workspaceDerivation7\\Project\\fr

    // Change package name to adjust ecore model package name and corresponding EMF Jav
    kermeta.utils.TypeEquivalence.packageEquivelence().put("flow","fr.obeo.dsl.designer

    fr.irisa.triskell.kermeta.compilo.scala.Main.init (propertiesfilePath, projectName,
    String [] arg = new String[2];
    arg[0] = "-i";
    arg[1] = kmuri;
    fr.irisa.triskell.kermeta.compilo.scala.Main.main(arg);

}
```

Now, launch Comp.java file. It compiles Kermeta project into a Scala project on deriv/output.

## 4. Create application jar

To create the application jar click on the deriv/output folder -> Maven-> Package.

Your deriv project should look like this :

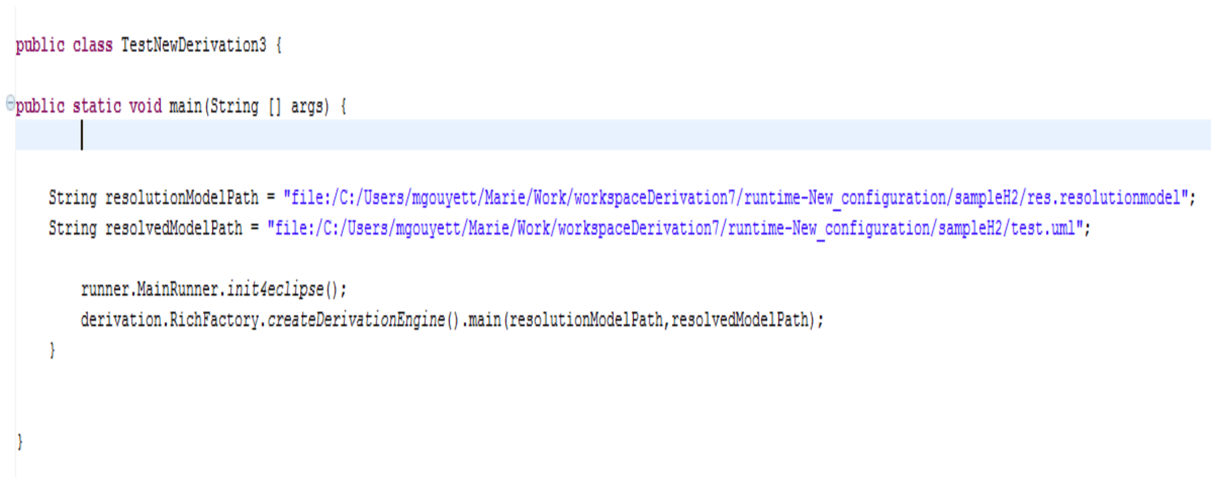


```
deriv
  src/main/java
  JRE System Library [JavaSE-1.6]
  output
    resources
    src
    target
      classes
      maven-archiver
      classes.timestamp
      org.kermeta.default.output-0.0.1-SNAPSHOT.jar
    pom.xml
  src
```

## 5. Use your compiled program on a Java project

Create a new Java project and add on its classpath all libraries used in the CompileProject and the org.kermeta.default.output-0.0.1-SNAPSHOT.jar.

To execute deriv, use the following main :



```
public class TestNewDerivation3 {

public static void main(String [] args) {


    String resolutionModelPath = "file:/C:/Users/mgouyett/Marie/Work/workspaceDerivation7/runtime-New_configuration/sampleH2/res.resolutionmodel";
    String resolvedModelPath = "file:/C:/Users/mgouyett/Marie/Work/workspaceDerivation7/runtime-New_configuration/sampleH2/test.uml";

        runner.MainRunner.init4eclipse();
        derivation.RichFactory.createDerivationEngine().main(resolutionModelPath,resolvedModelPath);
    }



}
```

```
runner.MainRunner.init4eclipse();

        // Change this line according to your compiled project
        derivation.RichFactory.createDerivationEngine().main(resolutionModelP
    ath,resolvedModelPath);
```