



# Formations OpenEmbeDD

## *Kermeta 1.2*

Premier niveau

# Plan

- Installer *Kermeta*
- Environnement *Kermeta* dans Eclipse
- *Kermeta* : le langage
- Ingénierie Dirigée par les Modèles
- Modelisation orientée Aspects
- Autres fonctionnalités

Plus d'information : <http://kermeta.org/documents/>

# I - Installer *Kermeta*

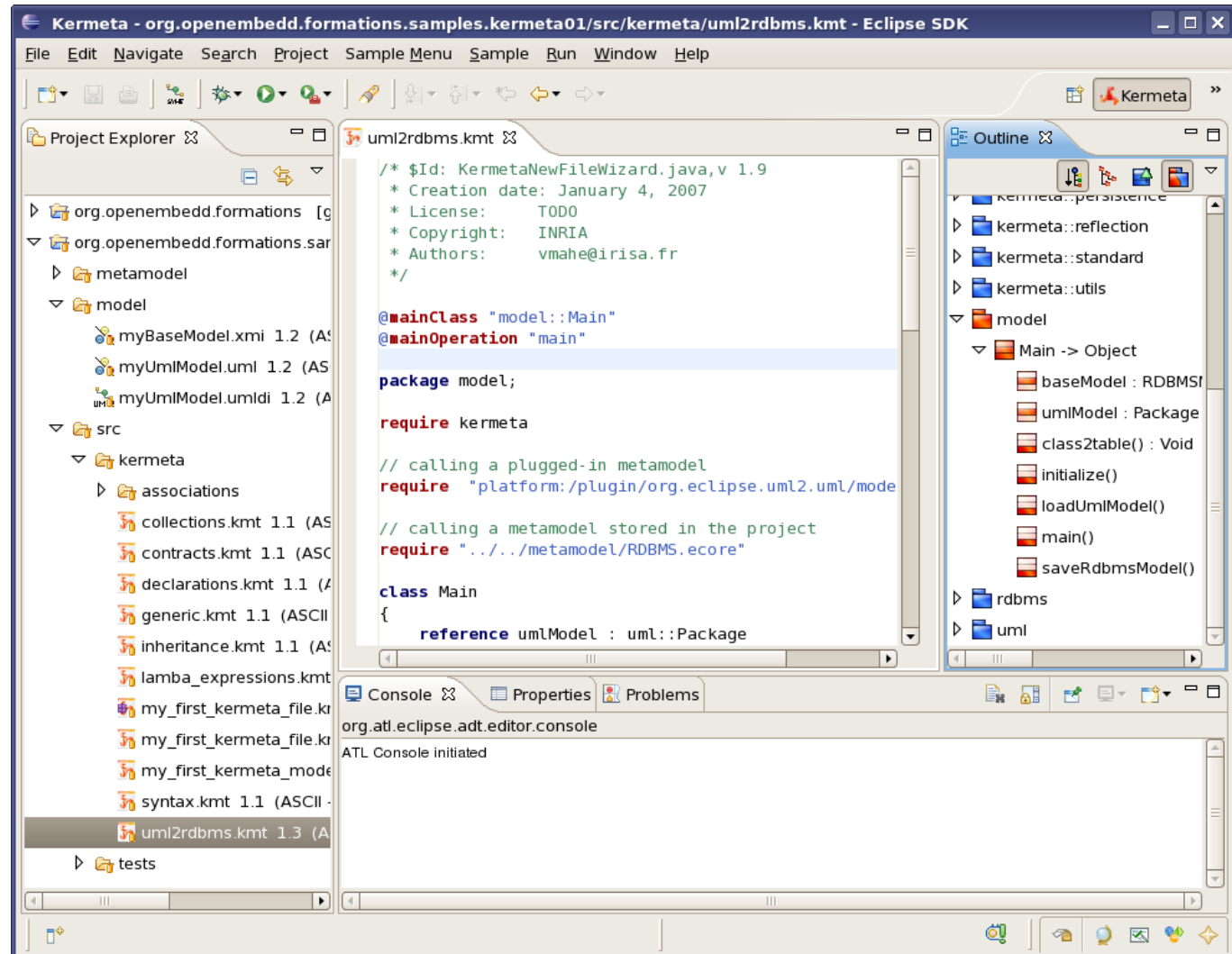
- Télécharger un Eclipse SDK 3.4.1
  - ... avec l'environnement Java
  - Le déballer
  - Lancer Eclipse(un Eclipse 3.4.1 déjà fonctionnel peut faire l'affaire)
- Paramétrer le site update d'OpenEmbeDD
  - *Help -> Software Updates -> Available Software*
  - Add a New Remote Site
    - « OpenEmbeDD experimental »
    - <http://openembedd.org/experimental/update>  
(pour avoir accès à la plus récente version)
- Installer ce dont vous avez besoin
  - Sélectionner « *Generic Modelling Tools* » + « *Samples* »
  - Cliquer sur le bouton « Install »
  - Finir l'installation puis redémarrer Eclipse

# Kermeta

## Un environnement

## II - Environnement : Eclipse et *Kermeta*

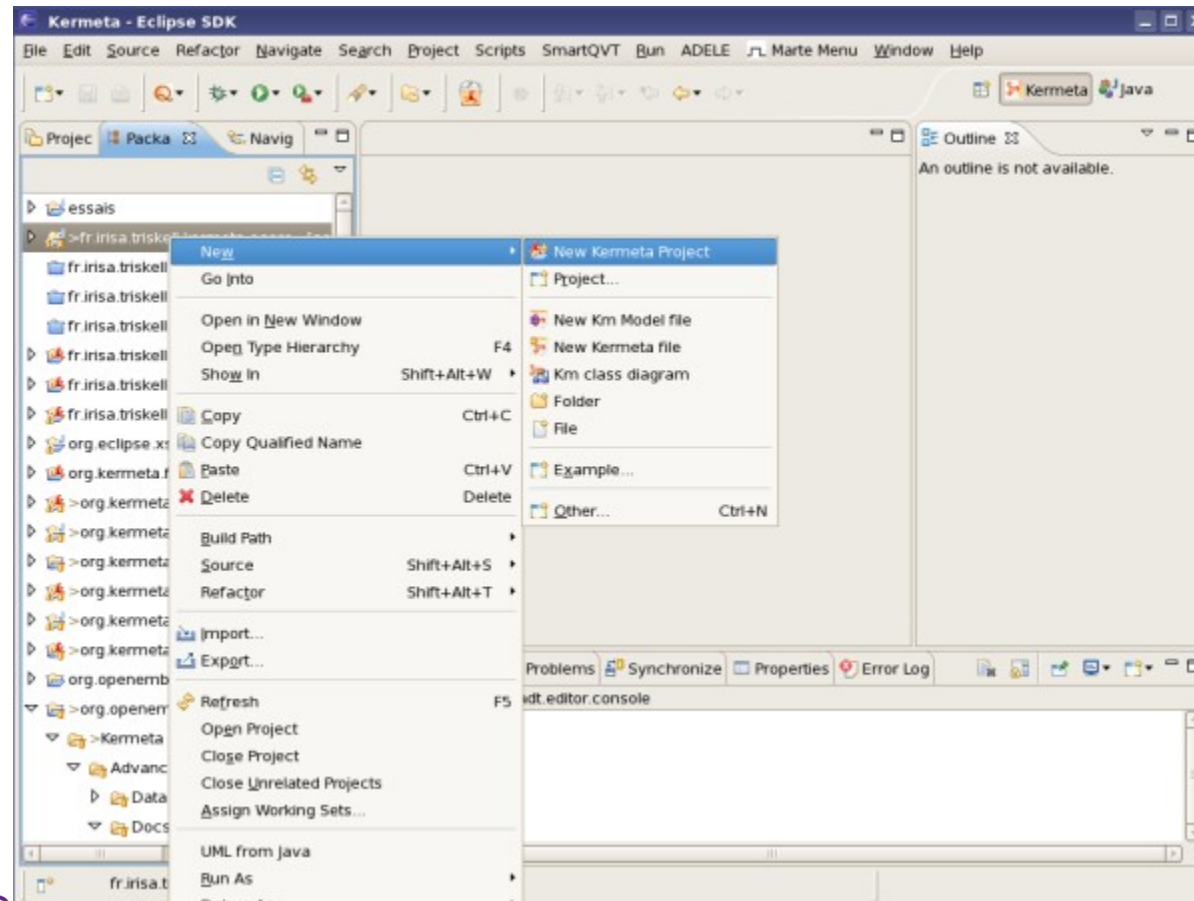
- Kermeta* est totalement intégré à Eclipse :



## II – Environnement : la perspective

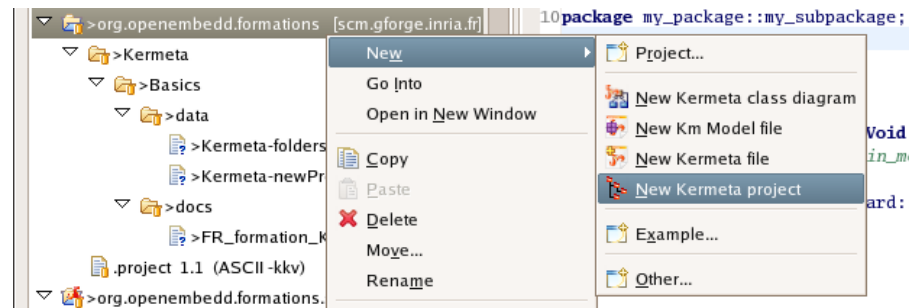
### Kermeta dispose de sa propre perspective :

- Raccourcis contextuels adhoc
- Assistants « Nouveau projet » & « Nouveau fichier »

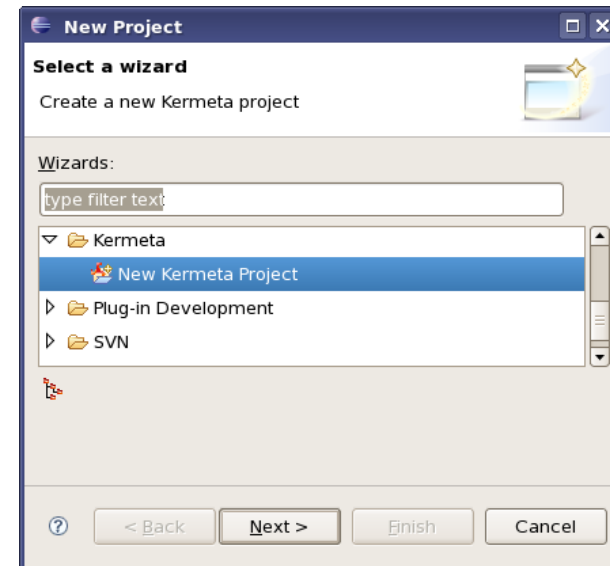


## II – Environnement : nouveau projet

- Un projet kermeta se crée par appel à l'assistant via le menu contextuel :

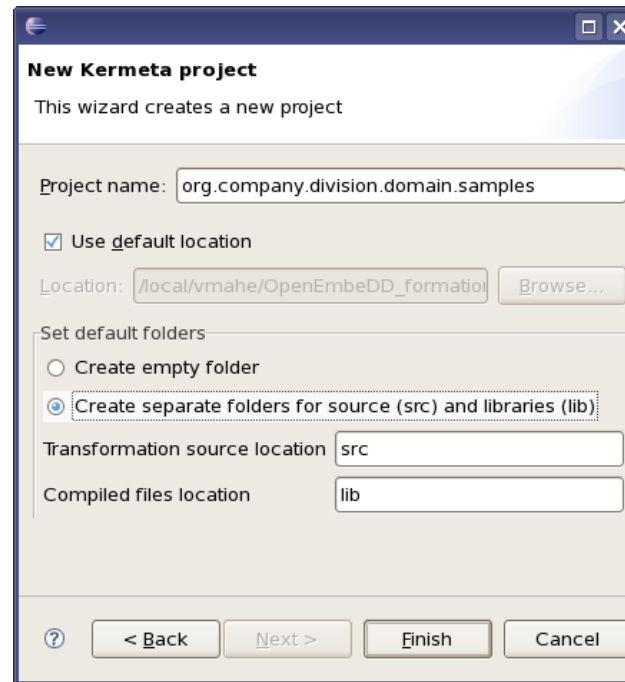


- ou bien par le menu général -> projet :

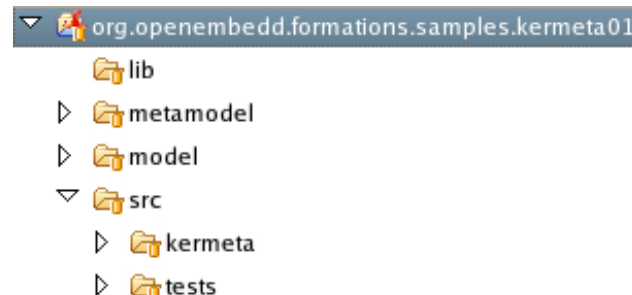


## II – Environnement : projet Kermeta

- L'assistant réalise une structuration des projets Kermeta telle que préconisée :



Ce qui donne :

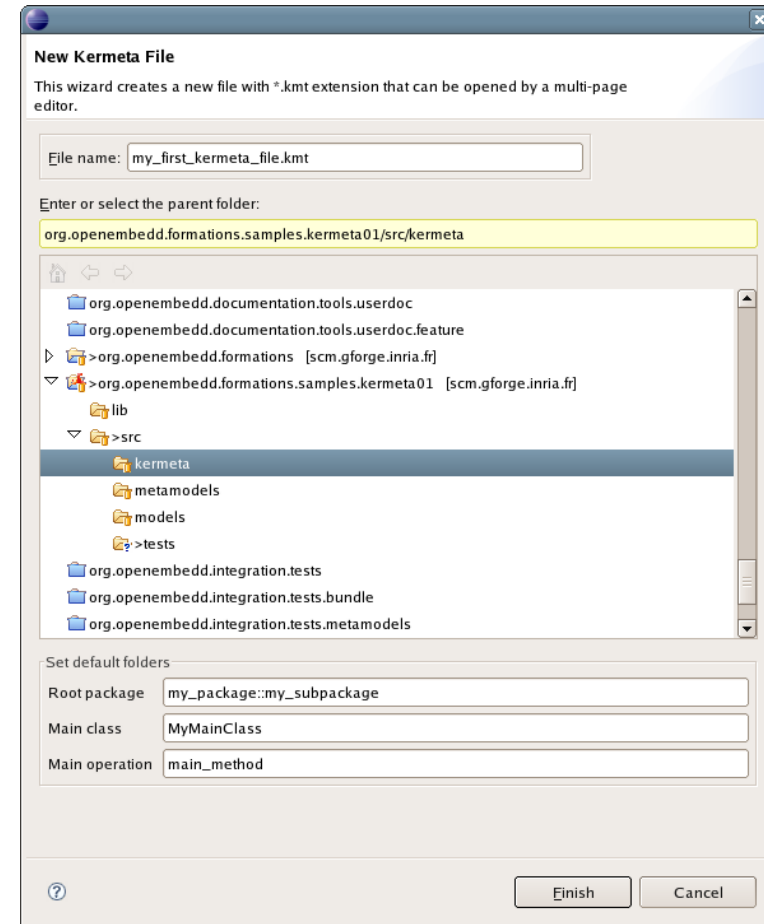




## II – Environnement : nouveau fichier

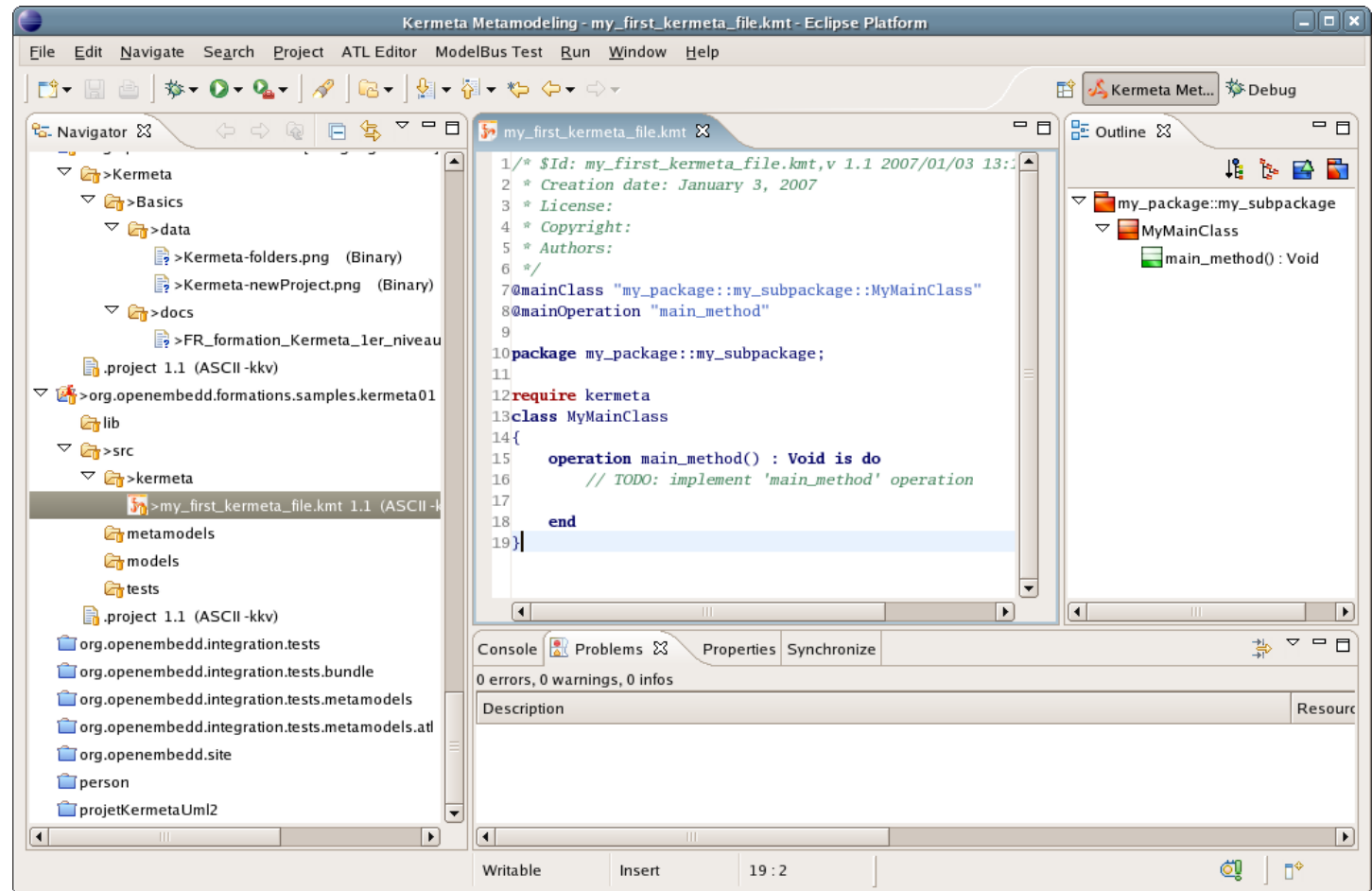
### Un assistant dédié :

- Nom
- Emplacement dans le projet
- Paquetage de référence
- Classe principale
- Méthode de lancement



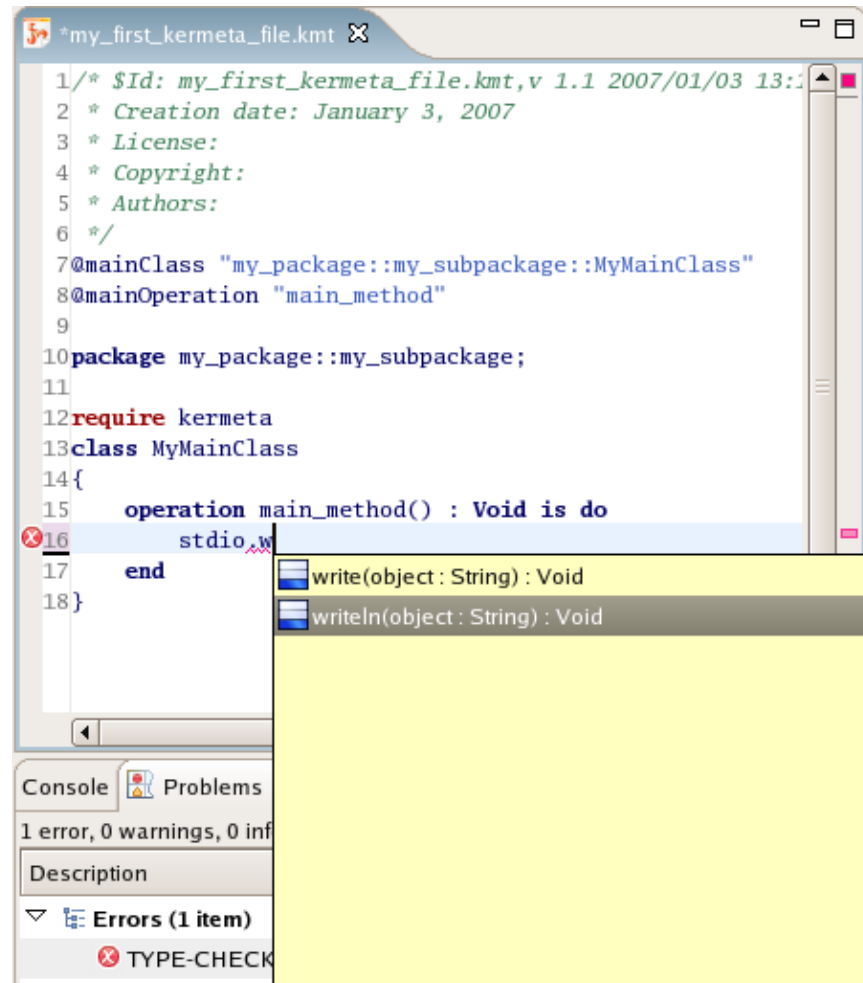
## II – Environnement : édition d'un fichier

- L'assistant génère le nouveau fichier et l'ouvre dans l'éditeur de texte :



## II – Environnement : édition d'un fichier

- Ajoutons un peu de code au fichier :



```

1/* $Id: my_first_kermeta_file.kmt,v 1.1 2007/01/03 13:11:11 $
2 * Creation date: January 3, 2007
3 * License:
4 * Copyright:
5 * Authors:
6 */
7@mainClass "my_package::my_subpackage::MyMainClass"
8@mainOperation "main_method"
9
10package my_package::my_subpackage;
11
12require kermeta
13class MyMainClass
14{
15    operation main_method() : Void is do
16        studio.w
17    end
18}
    
```

write(object : String) : Void  
 writeln(object : String) : Void

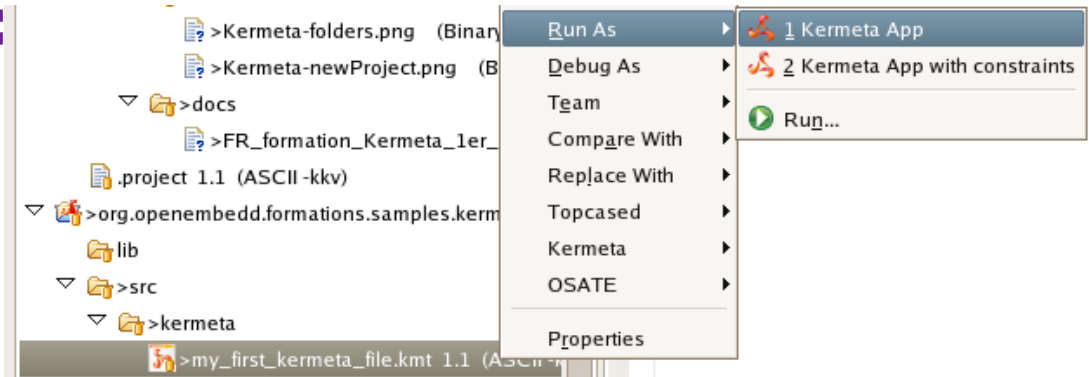
Console Problems  
 1 error, 0 warnings, 0 info  
 Description  
 Errors (1 item)  
 TYPE-CHECK

## II – Environnement : exécuter Kermeta

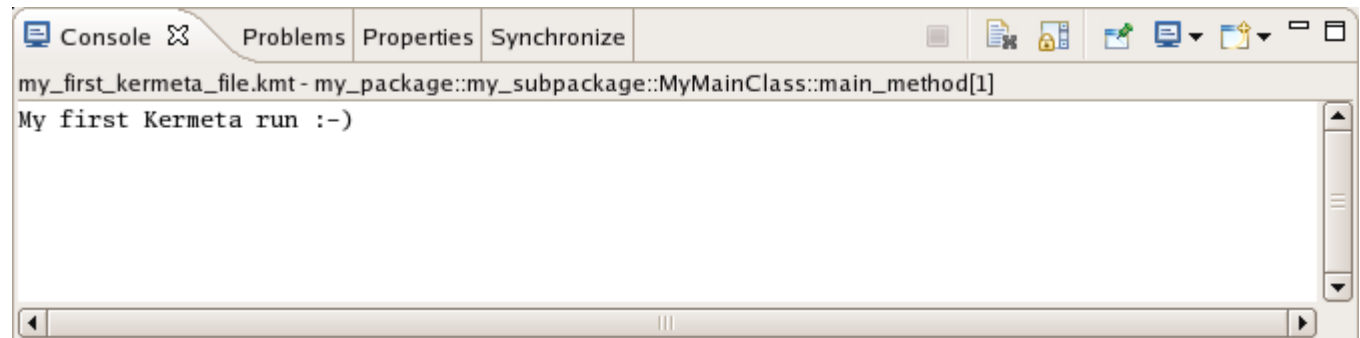
- Complétons :

```
13 class MyMainClass
14 {
15     operation main_method() : Void is do
16         stdio.writeln("My first Kermeta run :-)")
17     end
18 }
```

- Puis lançons :



- La console vient au premier plan et affiche :



## II – Environnement : *KUnit*

### Tests unitaires avec KUnit

```
tests_suite01.kmt
/* $Id: tests_suite01.kmt,v 1.1 2007/01/09 08:17:59 vmahe Exp $
 * Creation date: January 3, 2007
 * License:
 * Copyright:
 * Authors:
 */
@mainClass "my_package::subpackage::MyTestSuite"
@mainOperation "runTests"

package my_package::subpackage;

require kermeta
require "platform:/resource/org.openembedd.formations.samples.kermeta01/src/kermeta/my_first_kermeta_file.kmt"

class MyTestSuite inherits kermeta::kunit::TestRunner
{
    operation runTests() : Void is do
        // Here, we run our first test case
        run(FirstTestCase)
        printTestResult
    end
}

class FirstTestCase inherits kermeta::kunit::TestCase
{
    reference a : kermeta::standard::Integer
    reference b : kermeta::standard::Integer

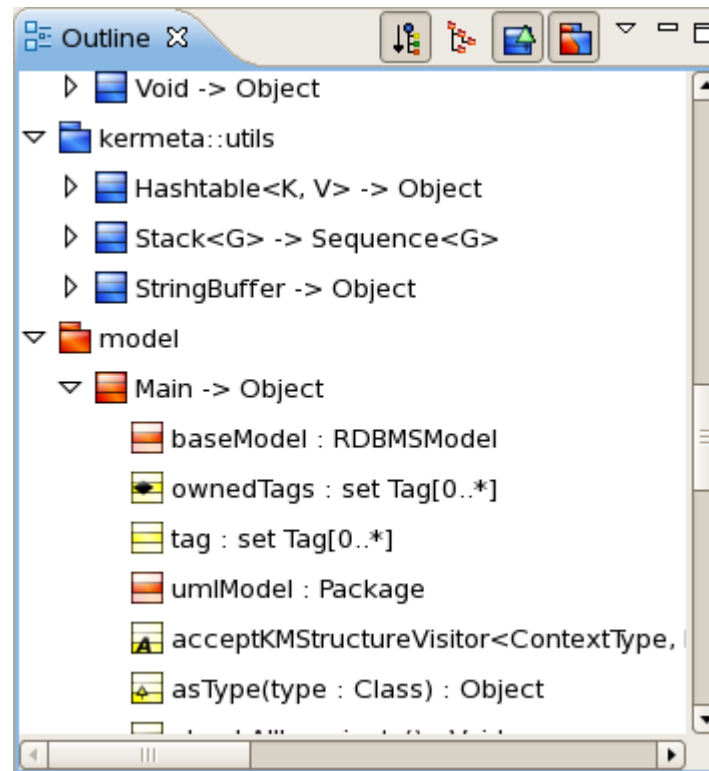
    method setUp() is do
        a := 0
        b := 1
    end

    method tearDown() is do // TODO
    end

    // test methods name must begin with "test" to be processed
    operation testSuccessDemo() is do
        assertTrueWithMsg(b > a, "The testSuccessDemo should demonstrate a test success")
    end
    operation testFailureDemo() is do
        assertTrueWithMsg(a > b, "The testFailureDemo should demonstrate a test failure")
    end
    operation testErrorDemo() is do
        var i : kermeta::standard::Integer init 1/0
        assertTrueWithMsg(a > (b/a), "The testErrorDemo should demonstrate a error interception")
    end
}
```

## II – Environnement : divers

### • « Outline » : affichage des librairies



# Kermeta

## Le langage

# III – Langage Kermeta : généralités

- C'est un langage Objet

```
class MyMainClass
{
    operation main_method() : Void is do
        stdio.writeln("My first Kermeta run :-)")
    end
}
```

- Sa syntaxe est impérative

- Le code bénéficie d'un typage fort, vérifié à la volée

- Kermeta offre la généricité :

```
class Queue<G>
{
    reference elements : oset G[*]
    operation enqueue(e : G) : Void is do
        elements.add(e)
    end
    operation dequeue() : G is do
        result := elements.first
        elements.removeAt(0)
    end
}
```



# III – Langage Kermeta : généralités

## • Héritage multiple :

```

abstract class AText
{
    operation addOp(textToAdd : kermeta::standard::String) is abstract
}
class LeftHand inherits AText
{
    reference text : kermeta::standard::String
    method addOp(textToAdd : kermeta::standard::String) is
    do
        if text != void then
            text.append(textToAdd)
        else
            text := textToAdd
        end
    end
}
class RightHand inherits AText
{
    reference text : kermeta::standard::String
    method addOp(textToAdd : kermeta::standard::String) is
    do
        if text != void then
            textToAdd.append(text)
            text := textToAdd
        else
            text := textToAdd
        end
    end
}
class CapitalText inherits LeftHand, RightHand
{
    method addOp(textToAdd : kermeta::standard::String) from LeftHand is
    do
        super(textToAdd)
    end
}
    
```

# III – Langage Kermeta : généralités

## • Éléments de syntaxe :

```
package my_package::subpackage;

require kermeta

class SyntaxClass
{
  // composition attributes
  attribute myAtt : X
  // pointer-like attributes
  reference myObj : X
  // affectation to an "attribute" deletes former
  // container attribute
  operation main() : Void is do
    // temporary variable declaration
    // + initialization
    var v1 : SyntaxClass init SyntaxClass.new
    var v2 : SyntaxClass init SyntaxClass.new
    var anObj : X // declaration without
                  // initialization
    anObj := X.new // affectation with a new X object

    v1.myAtt := anObj
    // v1 has an attribute
    stdio.writeln(v1.myAtt.toString)

    v2.myAtt := v1.myAtt // transfert of "anObj"
                        // from v1 to v2
    // v1 has loose its attribute (print <void>)
    stdio.writeln(v1.myAtt.toString)
  end
}
class X
{
  method toString() : kermeta::standard::String is do
    result := "I'm an X object"
  end
}
```

```
class Rectangle
{
  attribute length : kermeta::standard::Integer
  attribute width : kermeta::standard::Integer

  // read-only property derived from length/width
  property surface : kermeta::standard::Integer
  getter is do
    result := length * width
  end
}
class Cube
{
  attribute width : kermeta::standard::Integer
  attribute surface : kermeta::standard::Integer
  attribute volume : kermeta::standard::Integer

  // read-write property
  property edge : kermeta::standard::Integer
  getter is do
    result := width
  end
  setter is do
    width := value
    surface := value * value * 6
    volume := value * value * value
  end
}
```

# III – Langage Kermeta : généralités

**Bloc de code :**

```
do
    // my code : locally declared variables are not visibles outside the block
end
```

**Conditions :**

```
var boolCond : kermeta::language::structure::Boolean init true
// conditional block
if boolCond then
    // block for true value of the condition
else
    // block for false value of the condition
end
// conditional expression => affectation
var s : kermeta::standard::String
s := if boolCond then "its true !" else "its a joke ;-)" end
```

**Boucle :**

```
from
    var i : kermeta::standard::Integer init 0
until
    i == 10
loop
    /* code to be done 10 times
       .... */
    i := i + 1 // don't forget to increment the counter :-)
end
```

**Exceptions :**

```
operation raiseException() is do
    raise kermeta::exceptions::Exception.new
end

operation handleException() is
do    // some code which raise an exception
    self.raiseException
rescue (e : kermeta::exceptions::Exception)
    // do something if exception of Exception type has been raised in block
end
```

# III – Langage Kermeta : généralités

## • Commentaires

- Fin de ligne
- Plusieurs lignes

```
// a "line" comment
```

```
/* a multi line  
comment */
```

- Annotation nommée


```
@descr      "a named annotation"  
operation myAnnotatedMethod() is abstract
```

- Annotation anonyme

```
/** anonymous multi line annotation */  
reference anAnnotatedObject :  
kermeta::language::structure::Object
```

affichage en bulle d'aide

```
operation main() is do  
  myAnnotatedMethod  
  anAnn  
end
```

```
class ForComments{  
  ...  
   operation myAnnotatedMethod() is abstract  
  ...  
}  
@descr "a named annotation"
```

## • Sucre syntaxique

```
package root_package;  
require kermeta  
using kermeta::language::structure
```

```
class X  
{  
  /* avoid writing kermeta::language::structure::Object */  
  reference anAnnotatedObject : Object  
}
```

# III – Langage Kermeta : généralités

## • Variables

- Syntaxe : a..z, A..Z, 0..9, « ~ », « \_ »
- Mots réservés : utilisables précédés de « ~ »

## • Énumérations

- Déclaration
- Usage

```
enumeration Seasons { spring; summer; automn; winter; }
```

```
operation x ( val : Seasons) is do  
if val == Seasons.spring then stdio.writeln("It's Spring") end  
end
```

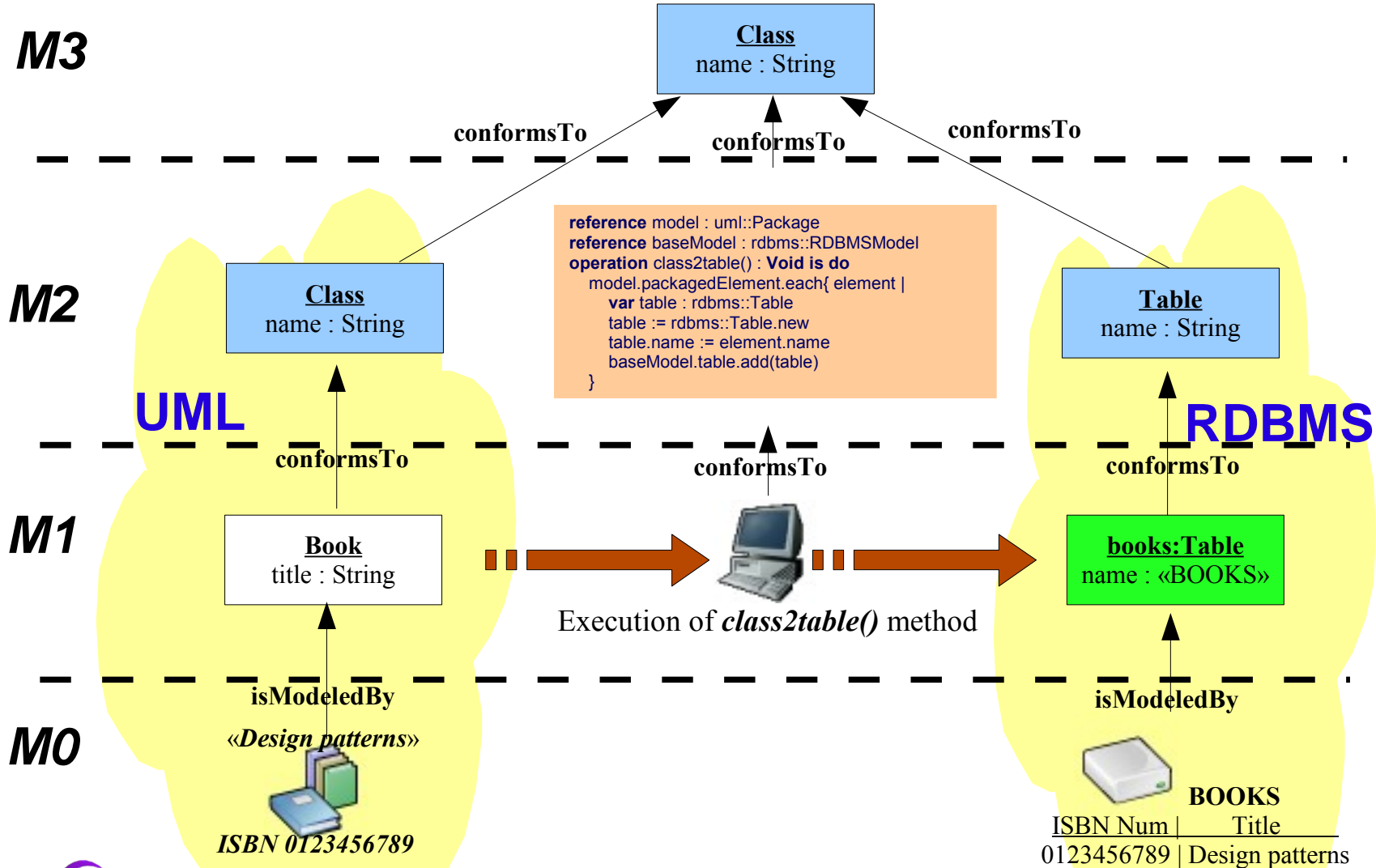
## • Types primitifs

- Integer <=> Java Integer
- String <=> Java String mais peu de méthodes (append, ...)
- Boolean <=> Java Boolean
- Character [incomplet]
- Real [incomplet]

# IDM

**modèles,  
méta-modèles,  
méta-méta-modèles**

# IV – Modèles : méta et méta-méta



# VI – Modèles : chargement

## • Déclaration du méta-modèle

```
// calling a metamodel stored in the project (bad)
require "../metamodels/RDBMS.ecore"

// calling a plugged-in metamodel (better)
require "/plugin/org.eclipse.uml2.uml/model/UML.ecore"

// calling a plugged-in metamodel (the best)
require "http://www.eclipse.org/uml2/2.1.0/UML"
```

## • Chargement d'un modèle

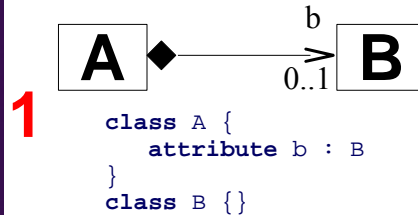
```
operation loadUmlModel() is do
    var inputRep : kermeta::persistence::EMFRepository init kermeta::persistence::EMFRepository.new
    var inputRes : kermeta::persistence::EMFResource
    inputRes := inputRep.createResource("../models/myUmlModel.uml",
                                        "platform:/plugin/org.eclipse.uml2.uml/model/UML.ecore")
    inputRes.load() // if use getResource(aModel), no need load()
    var pack : uml::Package
    pack := inputRes.one
end
```

## • Sérialisation d'un modèle

```
operation saveRdbmsModel() is do
    var outputRepository : kermeta::persistence::EMFRepository
    init kermeta::persistence::EMFRepository.new
    var outputResource : kermeta::persistence::EMFResource
    outputResource := outputRepository.createResource("../models/myBaseModel.xmi",
                                                       "../metamodels/RDBMS.ecore")
    outputResource.add(baseModel)
    outputResource.save()
end
```

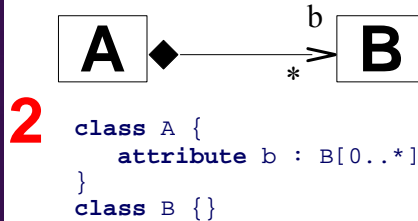


# IV – Modèles : associations

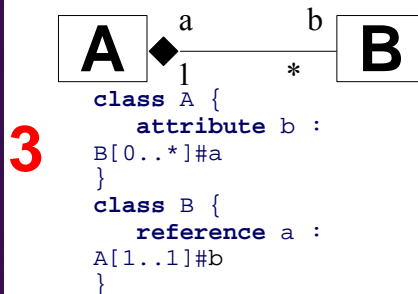


usage

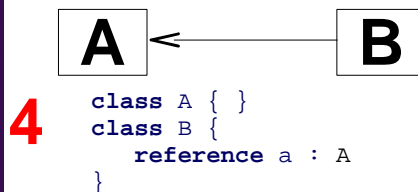
```
var a1 : A init A.new
var b1 : B init B.new
a1.b := b1
var b2 : B
b2 := a1.b
```



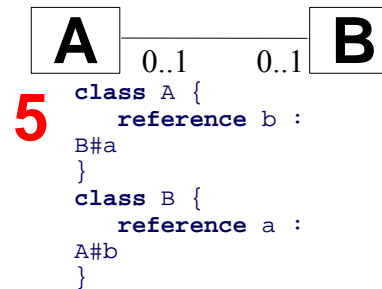
```
var a1 : A init A.new
var b1 : B init B.new
a1.b.add(b1)
var bees : OrderedSet<B>
init OrderedSet<B>.new
bees.addAll(a1.b)
```



```
var a1 : A init A.new
var b1 : B init B.new
a1.b.add(b1)
var a2 : A
a2 := b1.a
```

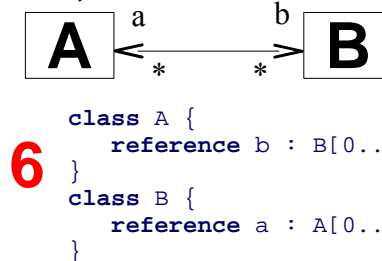


```
var a1 : A init A.new
var b1 : B init B.new
b1.a := a1
var a2 : A
a2 := b1.a
```

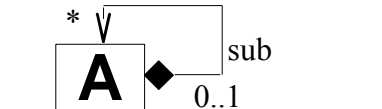


usage

```
var a1 : A init A.new
var b1 : B init B.new
a1.b := b1
var b2 : B
b2 := b2.a.b
var a2 : A
a2 := b1.a
```



```
var a1 : A init A.new
var b1 : B init B.new
a1.b.add(b1)
var bees : OrderedSet<B>
init OrderedSet<B>.new
bees.addAll(a1.b)
var aees : OrderedSet<A>
init OrderedSet<A>.new
aees.addAll(b1.a)
```



```
var a1 : A init A.new
var a2 : A init A.new
a1.sub.add(a2)
var a3 : A
a3 := a1.sub.first
```



```
var a1 : A init A.new
var a2 : A init A.new
a1.sub.add(a2)
var a3 : A
a3 := a2.up
```

## IV – Modèles : collections

### • Fonctions à la OCL déjà implémentées sur les collections :

- `aCollection.each { e | do`  
`/* traiter 'e' */`  
`end }`
- `aBoolean := aCollection.forAll { e | /* condition */ }`
- `aCollection2 := aCollection.select { e | /* condition */ }`
- `aCollection2 := aCollection.reject { e | /* condition */ }`
- `aCollection2 := aCollection.collect { e | /* valeur */ }`
- `anObject := aCollection.detect { e | /* condition */ }`
- `aBoolean := aCollection.exists { e | /* condition */ }`

### • Autres

- `10.times { i | do`  
`/* code à exécuter 10 fois */`  
`end }`

## IV – Modèles : collections

- 4 types de collections

	Not Ordered	Ordered
Unique	<b>Set</b>	<b>OrderedSet</b>
Not Unique	<b>Bag</b>	<b>Sequence</b>

- Usage :

```

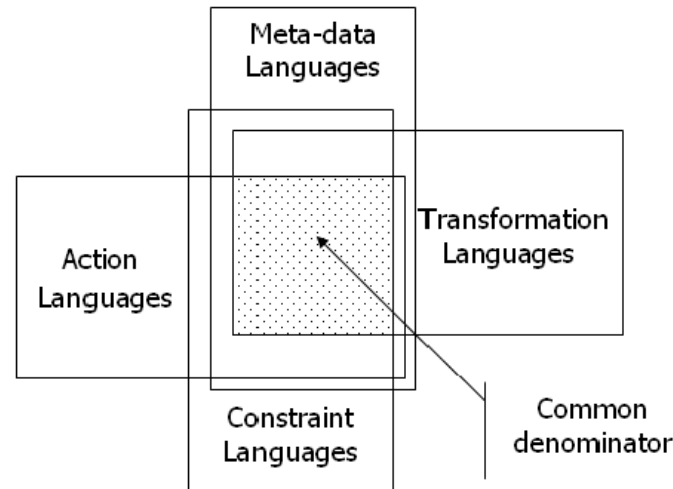
var myColl1 : set Integer[0..*]
var myColl2 : oset String[0..*]
var myColl3 : bag Boolean[0..*]
var myColl4 : seq Package[0..*]

// Fill in myColl1
myColl1.add(10)
myColl1.add(50)

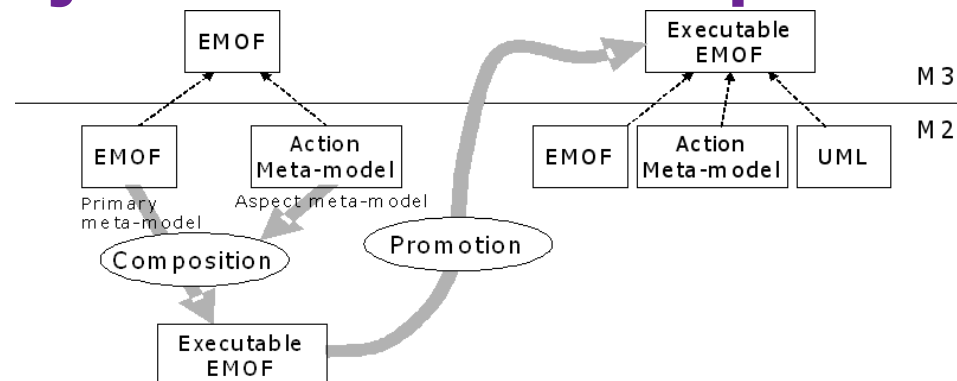
init kermeta::standard::Set<Integer>.new
init kermeta::standard::OrderedSet<String>.new
init kermeta::standard::Bag<Boolean>.new
init kermeta::standard::Sequence<Package>.new
    
```

## IV – Modèles : un langage d'action

- À la croisée des chemins, Kermeta :

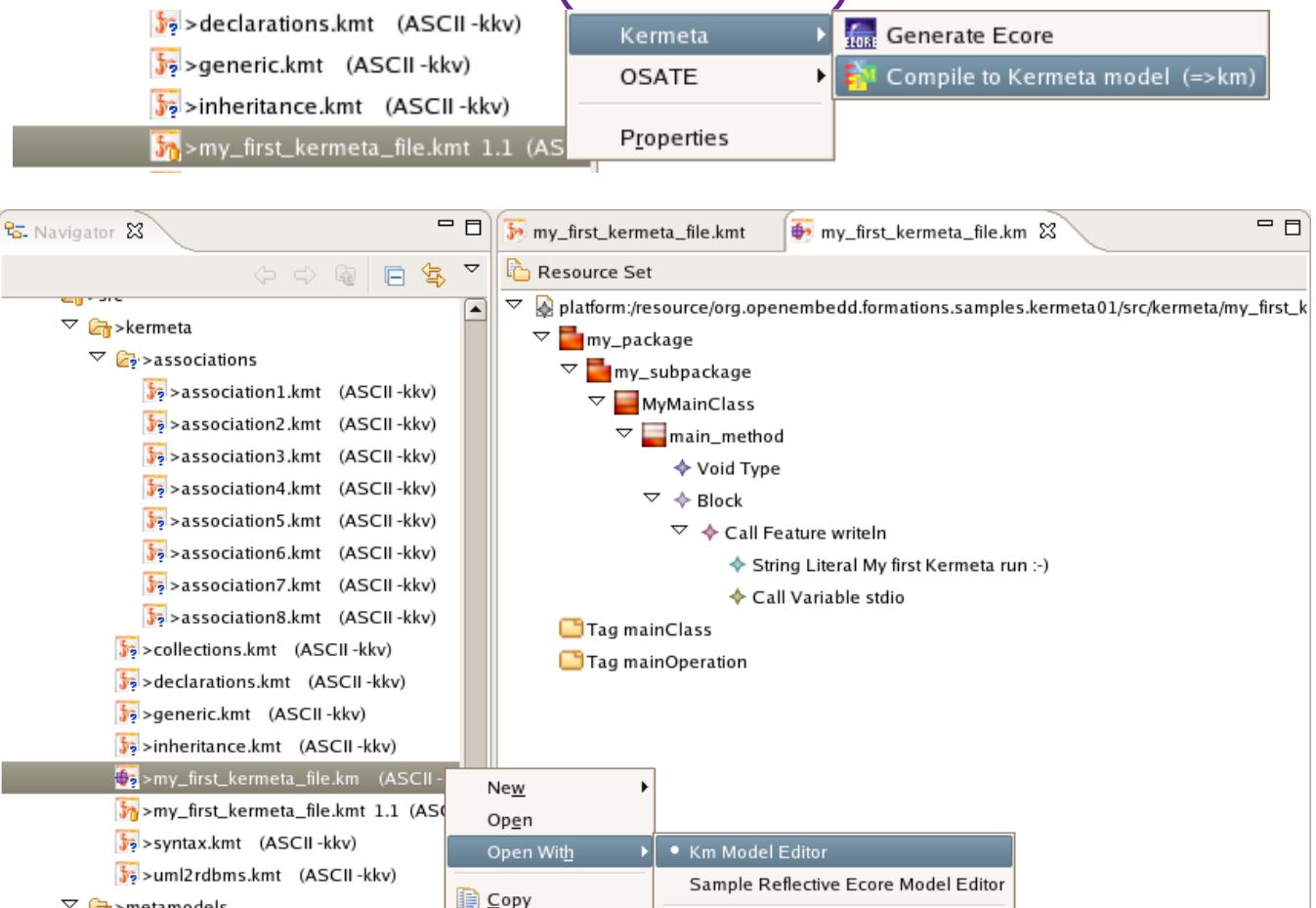


- Kermeta ajoute une sémantique à EMOF :



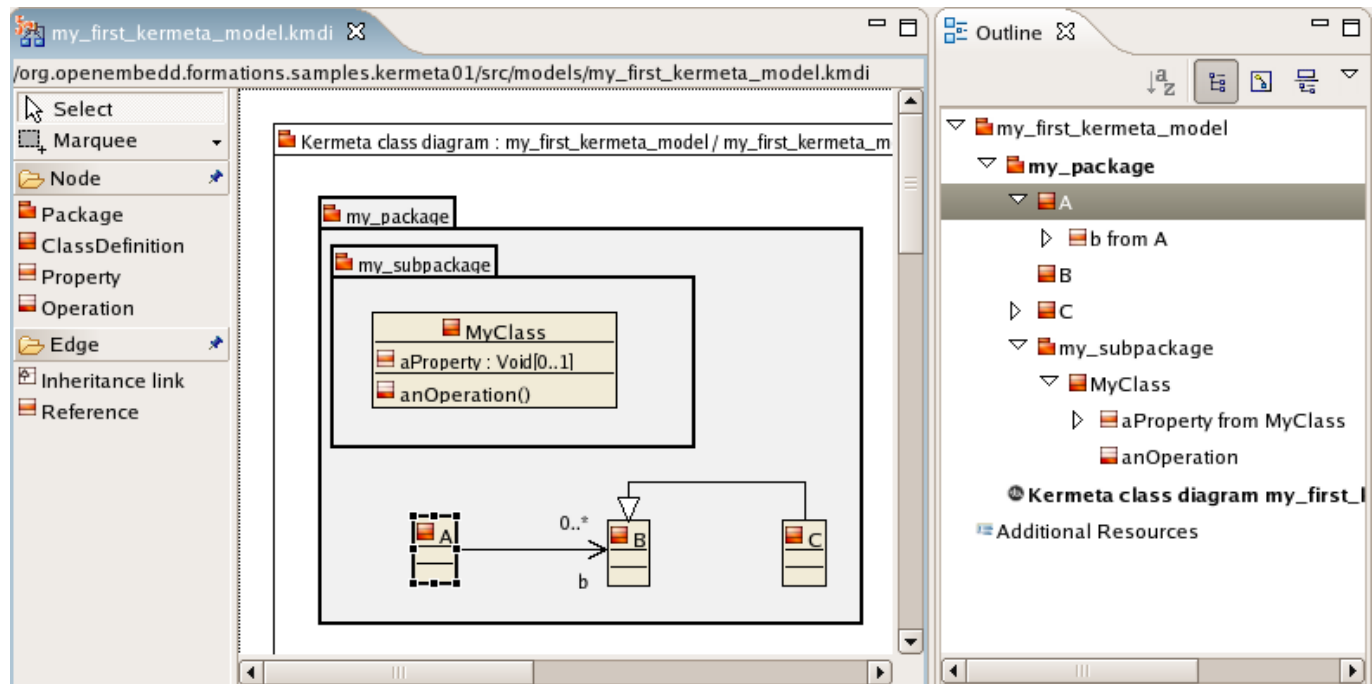
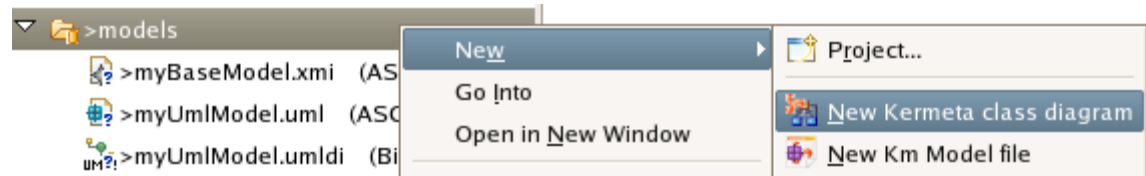
## IV – Modèles : un langage d'action

- Kermeta est aussi un méta-modèle
  - Transformation en KM (ou Ecore) :



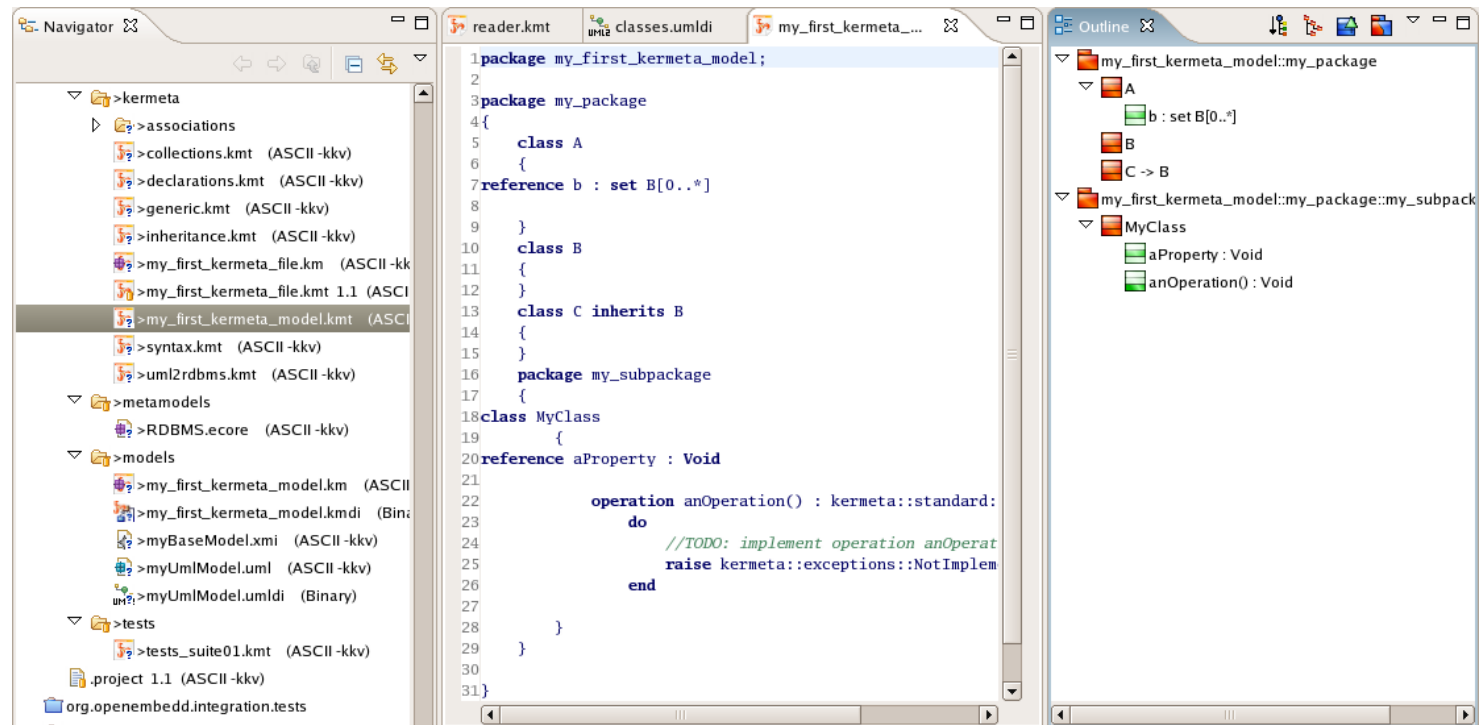
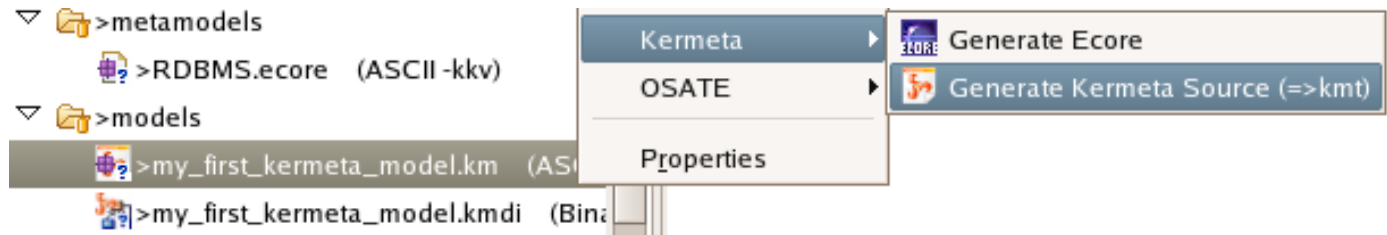
## IV – Modèles : éditeur graphique

- Kermeta possède son propre éditeur graphique :



# IV – Modèles : éditeur graphique

## • Passer du graphique au code :



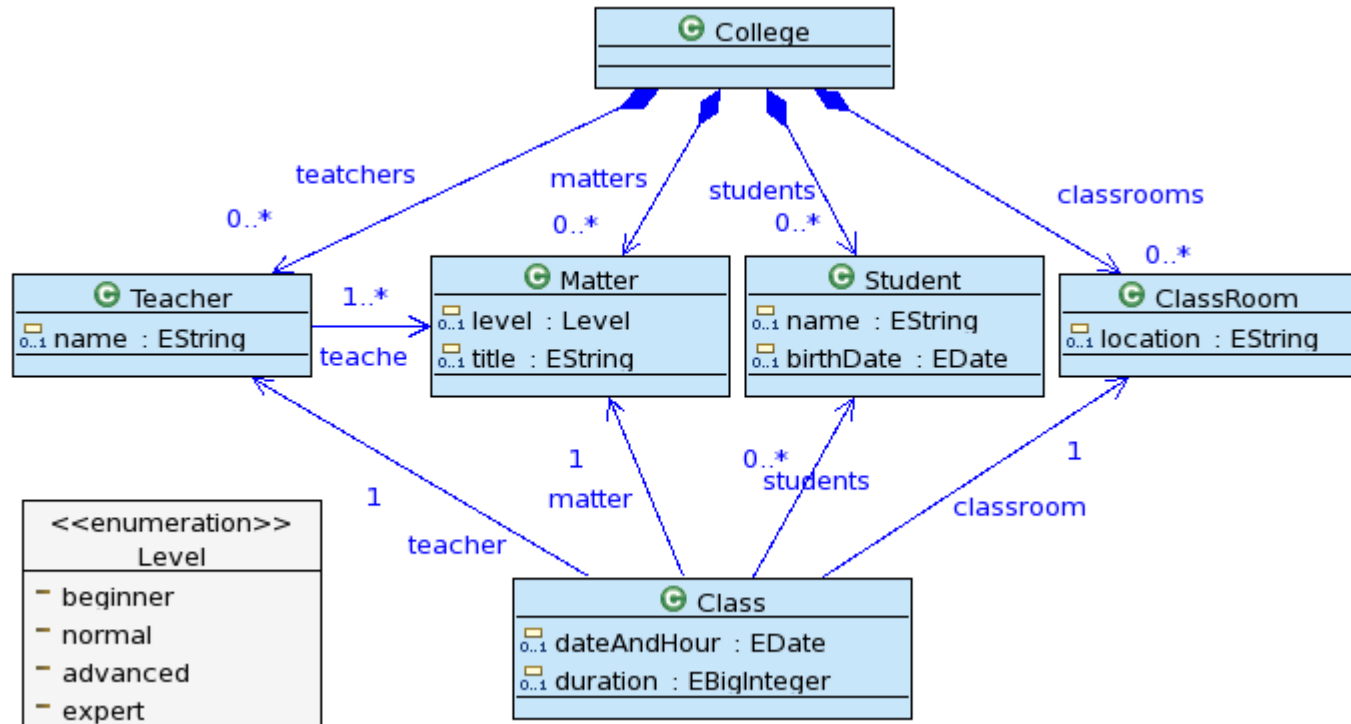
# Aspects

## Enrichissez vos méta-modèles



# V – Aspects : enrichissez vos méta-modèles

- Imaginez un méta-modèle d'école(s)



# V – Aspects : enrichissez vos méta-modèles

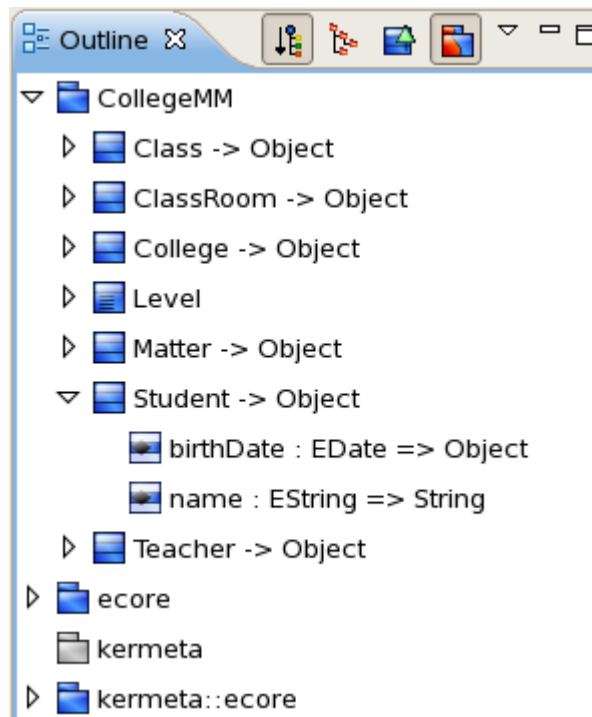
- Créez un fichier kermeta qui le référence

```
package CollegeMM;
```

```
require kermeta
```

```
require "platform:/resource/org.openembedd. formations.samples.kermeta01/metamodel/CollegeMM.ecore"
```

- Vous obtenez l'outline correspondant



# V – Aspects : enrichissez vos méta-modèles

## • Un aspect permet d'ajouter une classe

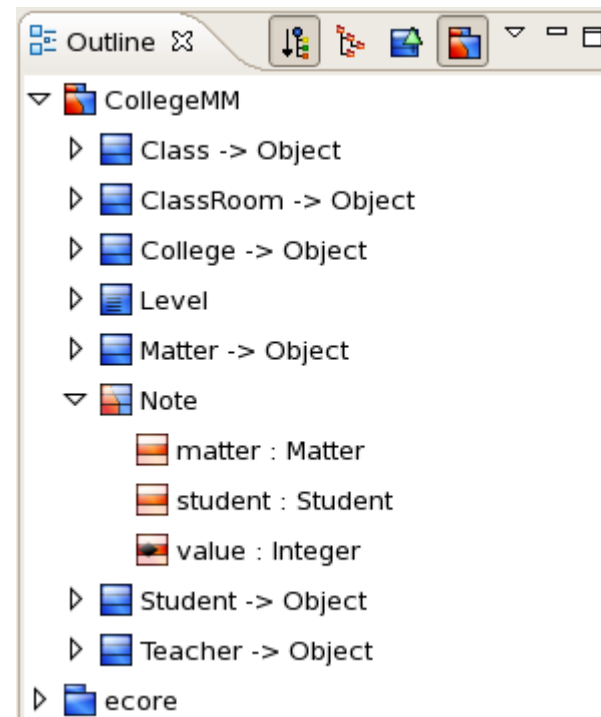
```
package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd. formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
    attribute ~value : kermeta::standard::Integer

    reference student : Student
    reference matter : Matter
}
```

## • L'outline affiche l'aspectisation



# V – Aspects : enrichissez vos méta-modèles

## • Ajout des opposites + nouvelle opération

```
package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd. formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
    attribute ~value : kermeta::standard::Real
    // add the opposite for managing notes from students/matters
    reference student : Student#notes
    reference matter : Matter#notes
}

aspect class Student {
    reference notes : Note[0..*]#student
    property average : kermeta::standard::Real
    getter is do
        var total : kermeta::standard::Real
        notes.each{ n | total := total + n.~value }
        result := total / notes.size.toReal
    end
}

aspect class Matter {
    reference notes : Note[0..*]#matter
    property average : kermeta::standard::Real
    getter is do
        var total : kermeta::standard::Real
        notes.each{ n | total := total + n.~value }
        result := total / notes.size.toReal
    end
}
```

# V – Aspects : enrichissez vos méta-modèles

## • Factoriser le traitement via l'héritage

```
package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd.formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
    attribute ~value : kermeta::standard::Real
    // add the opposite for managing notes from students/matters
    reference student : Student#notes
    reference matter : Matter#notes
}

aspect class Notable {
    operation average(notes : Note[0..*]) : kermeta::standard::Real is do
        var total : kermeta::standard::Real
        notes.each{ n | total := total + n.~value }
        result := total / notes.size.toReal
    end
}

aspect class Student inherits Notable {
    reference notes : Note[0..*]#student
    property averageNote : kermeta::standard::Real
    getter is do
        result := average(notes)
    end
}

aspect class Matter inherits Notable {
    reference notes : Note[0..*]#matter
    property average : kermeta::standard::Real
    getter is do
        result := average(notes)
    end
}
```

# Kermeta

## Autres fonctionnalités

# VI – Autres : programmation par contrats

## • Syntaxe :

```
class StringTool
{
  reference stringTable : Collection<String>

  // an invariant constraint
  inv noVoidTable is
    do stringTable != void end

  // an operation with contracts
  operation concatenate(first : String,
                        second : String) : String
    pre noVoidInput is
      do first != void and second != void end

    post noVoidOutput is
      do result != void end

  // operation body
  is do
    result := first
    result.append(second)
  end
}
```

# VI – Autres : programmation par contrats

## • Programme de test :

```
class MyClass
{
  operation main() : Void is do
    // new tool : its stringTable must be initialized
    var st1 : StringTool init StringTool.new
    st1.stringTable := Set<String>.new
    var s1 : String
    var s2 : String

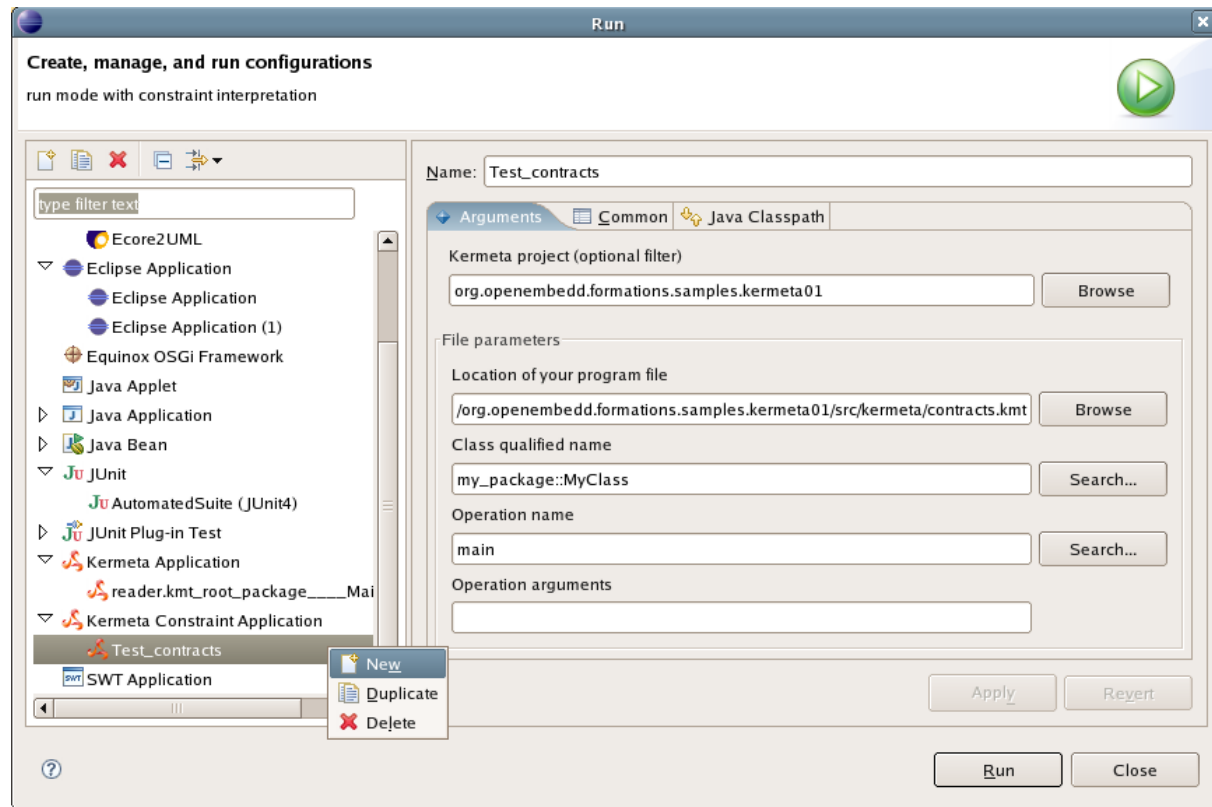
    do
      // void strings should raise exception
      st1.concatenate(s1, s2)
    rescue (err : ConstraintViolatedPre)
      stdio.writeln("expected err " + err.toString)
    end

    do
      // new tool without table
      var st2 : StringTool init StringTool.new
      st2.checkInvariants
    rescue (err : ConstraintViolatedInv)
      stdio.writeln("expected err " + err.toString)
    end
  end
}
```

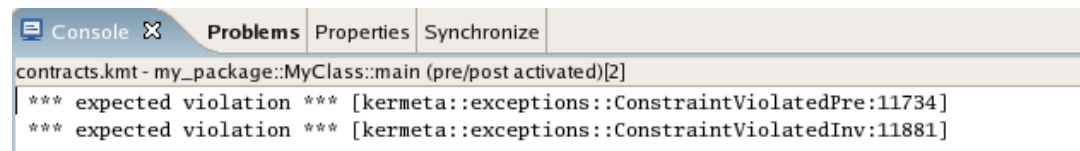


# VI – Autres : programmation par contrats

## Lancement en mode « contrats » :



## Résultat :



# VI - Autres : passerelle Java / Kermeta

## • Passerelle Java :

possibilité d'appeler des types et fonctions Java

```
/** An implementation of a StdIO class in Kermeta using existing Java standard input/output */
class StdIO {
  /** write the object to standard output */
  operation write(object : Object) : Void is do
    result ?= extern fr::irisa::triskell::kermeta::runtime::basetypes::StdIO.write(object)
  end
  /** read an object from standard input */
  operation read(prompt : String) : String is do
    result ?= extern fr::irisa::triskell::kermeta::runtime::basetypes::StdIO.read(prompt)
  end
}
```

```
/** Java Implementation of wrapper called from kermeta */
public class StdIO{
  // ..... //
  // Implementation of method read(prompt : String)
  public static RuntimeObject read(RuntimeObject prompt) {
    java.lang.String input = null;
  }
}
```

## VI – Autres fonctionnalités

- **Expressions dynamiques**  
interprétation à la volée de code passé en variable
- **$\lambda$  expressions**  
créer ses propres fonctions
- **ModelType**  
rapprocher des méta-modèles
- **Fonctionnalités en cours de développement :**
  - Clone élaboré
  - ...