Memento for the Kermeta Language

It is an Object language

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

Kermeta offers genericity…

```
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
}
```

.. and multiple inheritance

```
class CapitalText inherits LeftHand, RightHand {
    method addOp(textToAdd : String)
            from LeftHand is
        do
            super(textToAdd)
        end
}
```

Block of code

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

Conditions

```
var boolCond : 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
```

Loop

```
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
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 an Exception has been raised
  end
```

Kermeta language : bases

Syntaxe elements

```
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 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
}
```

Kermeta language : bases

Comments

End of line

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

Multiple lines

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

Named annotation

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

Anonymous annotation

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

Syntatic sucre

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

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

Enumerations

Declaration

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

Use

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

Variables

Syntax : `a..z`, `A..Z`, `0..9`, « ~ », « _ »

Key words : usable if prefixed by « ~ »

Primitifs types

`Integer` <=> Java Integer

`String` <=> Java String

`Boolean` <=> Java Boolean

`Character` [partial]

`Real` [partial]

4 kind of collections

	Not Ordered	Ordered
Unique	**Set**	**OrderedSet**
Not Unique	**Bag**	**Sequence**

Use

```
var myCol1 : (set) Integer[0..*] init
    kermeta::standard::Set<Integer>.new
// Fill in myCol1
myCol1.add(10)
myCol1.add(50)
```

Kermeta language : models

Declaration of the needed metamodel

```
// 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"
```

Loading a model

```
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
```

Serializing a model

```
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
```

Functions on collections

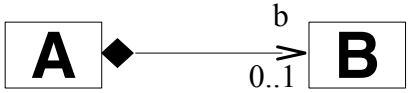
```
aCollection.each { e | do
   /* process 'e' */
end }
aBoolean := aCollection.forAll { e |
   /* condition */ }
aCollection2 := aCollection.select { e |
   /* condition */ }
aCollection2 := aCollection.reject { e |
   /* condition */ }
aCollection2 := aCollection.collect { e |
   /* value */ }
anObject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

Other functions

```
10.times { i | do
  /* code to execute 10 times */
end }
```

Kermeta language : associations

1




```
class A {
   attribute b : B
}
class B {}
```

use

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

2


```
class A {
   attribute b :
B[0..*]
}
class B {}
```

use

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

3


```
class A {
   attribute b :
B[0..*]#a
}
class B {
   reference a :
A[1..1]#b
}
```

use

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

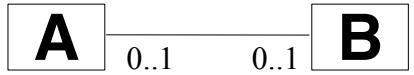
4


```
class A { }
class B {
   reference a : A
}
```

use

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

5




```
class A {
   reference b : B#a
}
class B {
   reference a : A#b
}
```

use

```
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
```

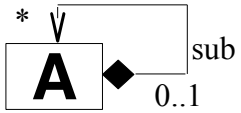
6


```
class A {
   reference b : B[0..*]
}
class B {
   reference a : A[0..*]
}
```

use

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

7

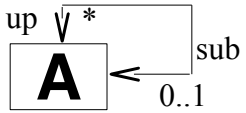



```
class A {
   attribute sub : A[0..*]
}
```

use

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

8



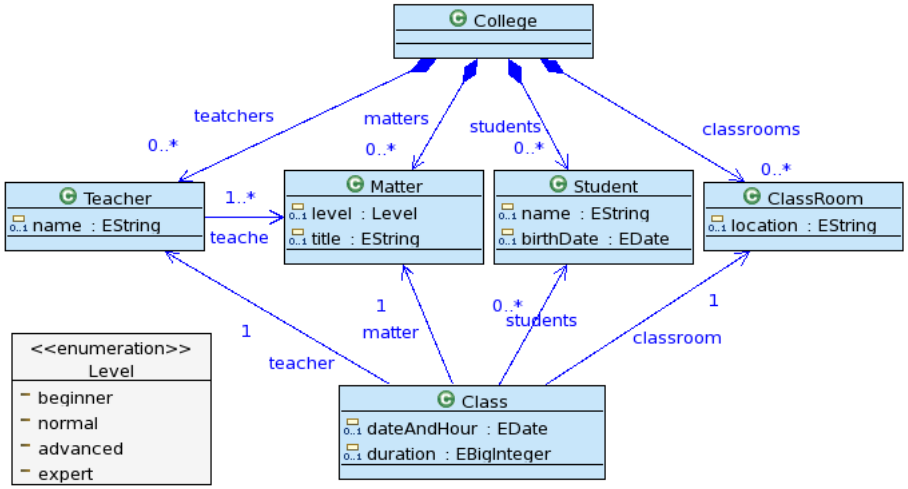

```
class A {
   reference sub : A[0..*]#up
   reference up : A#sub
}
```

use

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

Aspects : enrichissez vos méta-modèles

Imagine a metamodel about schools



Reference it

```
package CollegeMM;

require kermeta
require "../metamodel/CollegeMM.ecore"
```

Add a class with aspect

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

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

Add opposites + new operation

```
package CollegeMM;

require kermeta
require "../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
}
```

Autres : programmation par contrats

Syntax

```
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
}
```

Program which verify contracts

```
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
}
```

Other functionalities

- **Dynamic expressions**

  Code passed as parameter is interpreted on the fly

- **Gateway to Java**

  Call Java types and functions

```
/** 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;
}
```

- Lambda expressions

Create your own functions

- Functionalities under development
  - “Model” type
  - Require OCL rules file

…