EMF course - PACT

Etienne Borde
www.etienneborde.fr

Objectives

- Collective software development requires to focus on integration.
- John develops functionality A;
  Mike develops functionality B
- How to ensure A will work with B when starting the project (T_0)?
- How to ensure A will work with B latter in the project (T_0+dT)?

Anticipating Integration: Example

- At T0, John and Mike agree on a common data structure they will both rely on; they start their developments.
- At T0+dT, John and Mike have changed the data structure because of their own needs.
- What if John and Mike talk about these changes 1 week before delivery?

Anticipating Integration

- *Share* a common data structure and the *evolutions* of this data structure

- *Discuss and motivate* the modifications of this data structure

What is EMF and How is that useful to do that?

EMF Introduction

- **EMF stands for “Eclipse Modeling Framework”**
  - E : EMF is integrated to the Eclipse IDE, one of the most used IDE today. EMF is used in many Eclipse projects.
  - M : EMF defines a format (ecore) to “model” data structures instead of “coding” data structures
  - F : EMF is a “framework” offering different services, in particular Java code generation.

How is EMF helpful for integration

- **EMF helps to**
  - Define a common architecture
  - Communicate on this architecture
  - Produce the code of this architecture

Here, architecture means “assembly of software components”

Visual and compact representation of classes and relationships among them

Organization of the presentation

1. **Presentation of EMF**
   - an extension to the Java Course
   - how to use it in PACT

2. **Tutorial about "how to install, initialize, and use EMF" in Eclipse.**

Key concepts of Modeling

- Modeling is an activity of software engineering that aims at representing the architecture (assembly of components) of a software application in order to:
  - Ease discussions between experts (this feature will be useful for PACT reviews)
  - Anticipate integration by centralizing the definition of the software architecture
  - Generate the concrete implementation (code) corresponding to the modelled architecture (this will be the main practical feature used in the scope of PACT)
  - Analyse the model to ensure the application meets a given set of requirements (this is out of the scope of PACT)

When to use EMF in PACT

- **For the Analysis review:**
  - To represent the decomposition of the system into subsystems and components
  - To represent interactions between components
    - Existing (reused) components
    - Components to be produced during the project

When to use EMF in PACT

- **For the specification review**
  - Represent the implementation of components with classes, methods, attributes, etc….
  - Ensure traceability with the results of the analysis review

Class Diagram: the model part of EMF

- **Graphical representation of**
  - Packages (EPackage)
  - EMF data types
  - Classes (EClass), with their attributes (EAttribute) and methods (EOperation)
  - Inheritence between classes
  - References (EReference) between classes
  - Enums (EEnum)
  - Personalized data types (EDataType)

Used for analysis review

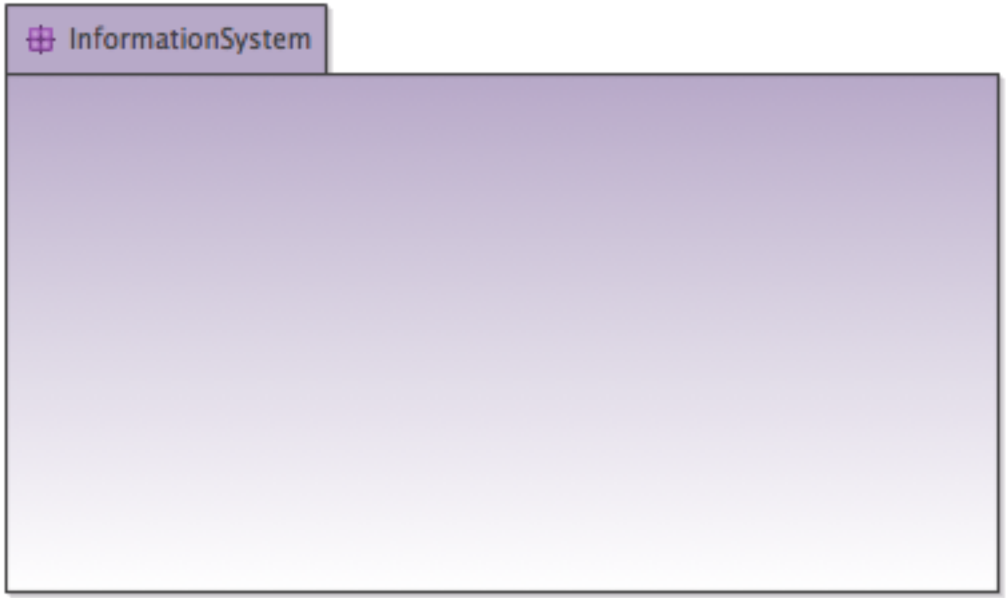
Packages in Java

- A package is a folder that groups other packages and classes

- A package defines a namespace in order to allow several classes to have the same name without ambiguities (for instance when referencing other classes with the import instruction)

Packages in EMF

- **Packages (Epackage)**



Primitive Data types in EMF

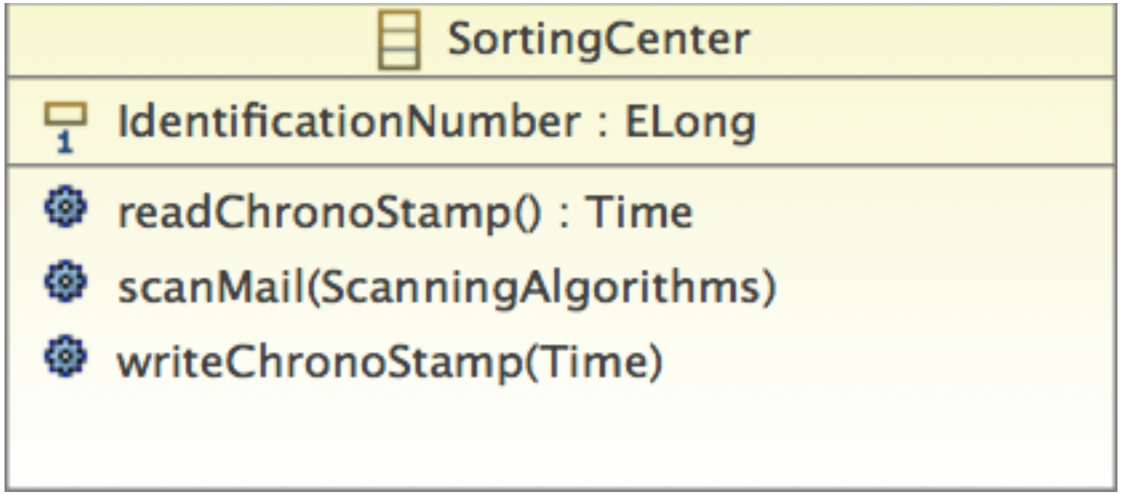
- **EMF defines its own data types, simply wrapping Java data types**
  - EInt → Integer
  - ELong → Long
  - EDouble → Double
  - …
- **Generally, the mapping Java data → EMF data type is trivial.**
- **Besides, the Java data type is indicated next to an EMF data type.**

Classes in Java

- In a Java program, objects are represented by classes that describe operations and data contained in this object.
- Data is represented by attributes, which type can be a primitive data type (int, long, etc…) or another class
- Operations are represented by methods with parameters and return type

Classes in EMF

- Classes (EClass), with their attributes (EAttribute) and methods (EOperation)



SortingCenter

IdentificationNumber : ELong

readChronoStamp() : Time

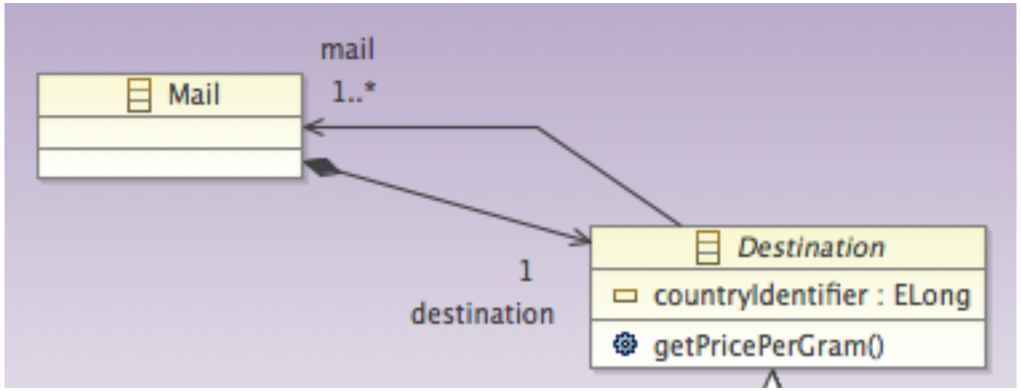
scanMail(ScanningAlgorithms)

writeChronoStamp(Time)

- Note that attributes are represented this way only when their type is a simple type (not a class typically)

Attributes of type “another class”

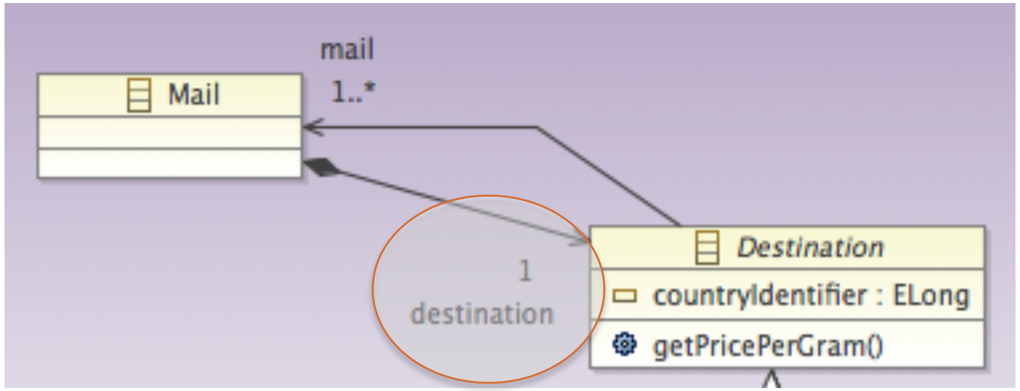
- References (EReference) between classes



- A reference represents an attribute of a class (or interface) which type is another class or interface.

About references

- **A reference represents an attribute of a class (or interface) which type is another class or interface**
- **A reference has**
  - A direction: the source of the reference holds an attribute of type the target of the reference.
  - A name: the name of the attribute in the source class
  - A numeration: 1, N..* (N>0), or *. It mainly tells if the attribute is a list of reference or a simple reference
  - Optional properties: EContainment, EOpposite.



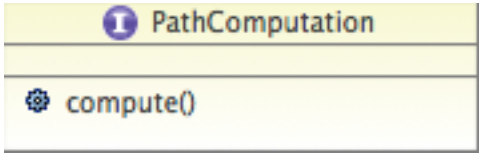
Note the default position of the name and numeration:
next to the destination of the reference

Interfaces in Java

- **An interface defines a contract for the implementation of software components:**
  - Classes that implement an interface I have to implement all the methods defined in I
  - Classes that use an interface I can use all the methods defined in I

Interfaces in EMF

- Interfaces, with their attributes (EAttribute) and methods (EOperation)



PathComputation

compute()

- Note that the “implements” relationship between interfaces and classes does not explicitly exist in EMF. To represent this in EMF, a class can inherit an interface even though IT IS NOT ALLOWED IN JAVA.

Abstract Classes in Java

- **An abstract class is almost like a class, except that it cannot be instantiated.**
  - It represent an incomplete class, with some default attributes and methods
  - It must be specialized thanks to inheritance and the concrete versions that inherit an abstract class can be instanciated

Inheritance in Java

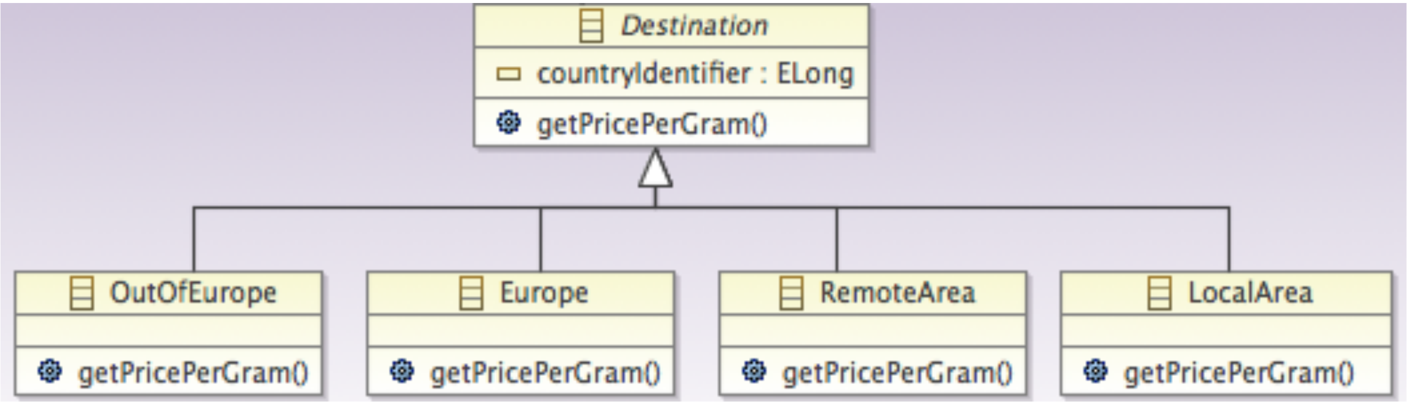
- **A class can inherit another class,**
  - to extend the definition of the super-class
  - To override the definition of methods from the super-class
- **An interface can also inherit another interface**
- **Three rules must be respected in Java:**
  - A class can inherit from one and only one class
  - An interface can inherit from several interfaces
  - A class can implement several interfaces
- **These rules are not true for EMF, but the Java code generated by EMF respects those three rules**

Inheritance in EMF

- Inheritance in EMF is more permissive that in Java
- However, the Java code generated from an EMF model will respect the rules of Java

Inheritance in EMF

- Inheritance between classes

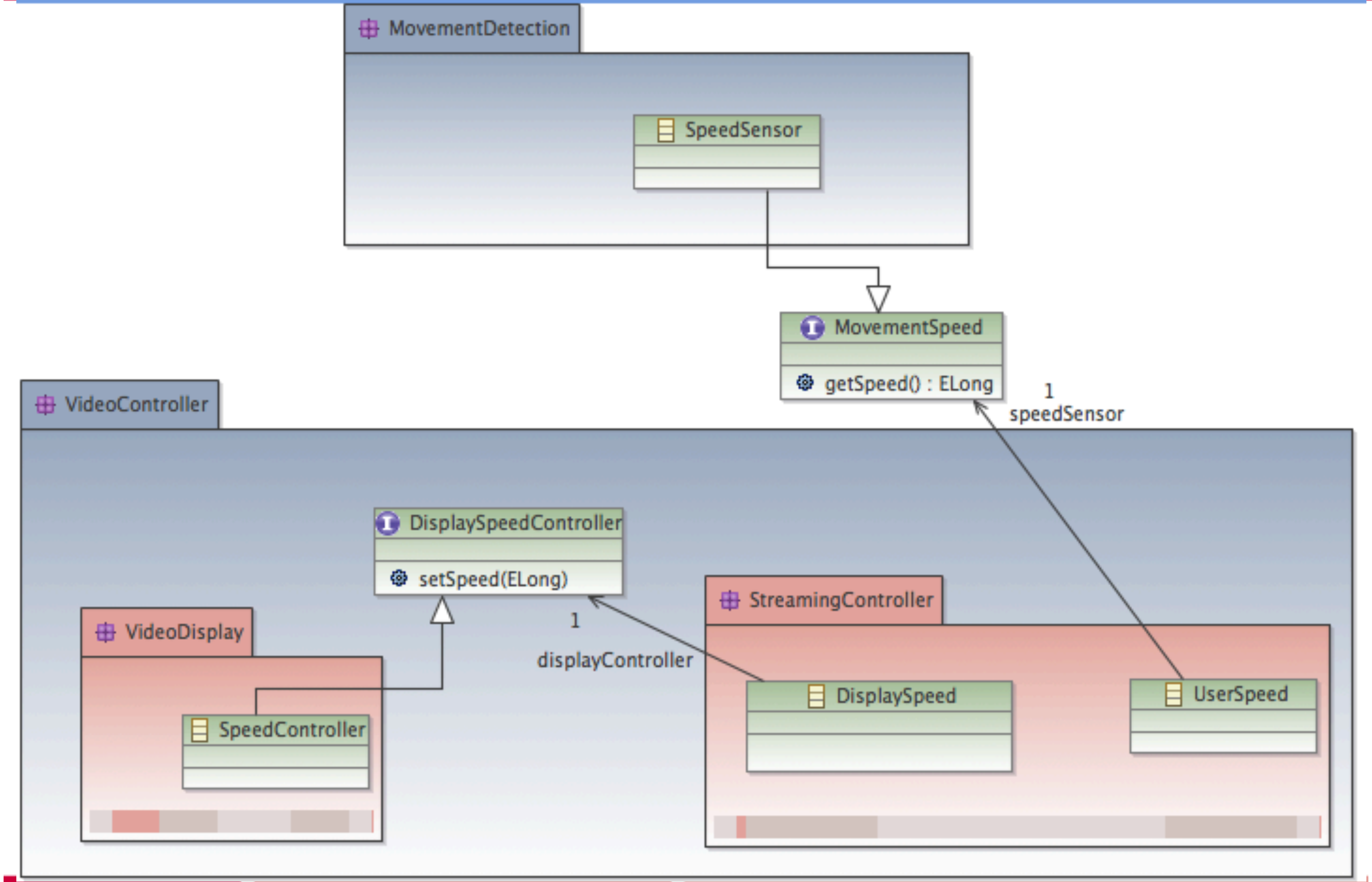


- In this example, OutOfEurope, Europe, RemoteArea, and LocalArea inherit *Destination*
- Note that Destination appears in Italic because it is an *abstract* class
- Note that as opposed to Java, Classes can inherit interfaces … Which (almost) means that the class implements this interface

How to use EMF for analysis review

- **For the analysis review:**
  - Represent subsystems with BLUE packages
  - Represent components with RED packages
  - Represent interactions with GREEN interfaces and classes responsible for components interactions

Example


MovementDetection
SpeedSensor
MovementSpeed
getSpeed() : ELong
1
speedSensor
VideoController
DisplaySpeedController
setSpeed(ELong)
1
displayController
VideoDisplay
SpeedController
StreamingController
DisplaySpeed
UserSpeed

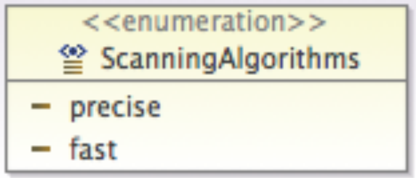
Class Diagram: the model part of EMF

- **Graphical representation of**
  - Packages (EPackage)
  - EMF data types
  - Classes (EClass), with their attributes (EAttribute) and methods (EOperation)
  - Inheritence between classes
  - References (EReference) between classes
  - Enums (EEnum)
  - Personalized data types (EDataType)

Used for specification review

Enumerated data type (enum)

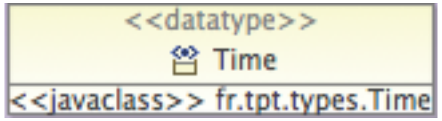
- Enums (EEnum) : data type that can take a value among a predefined list of possible values



- Here, a variable of type “ScanningAlgorithms” can take 2 possible values:
  - precise
  - fast

Existing data types in EMF

Personalized data types (EDataType)

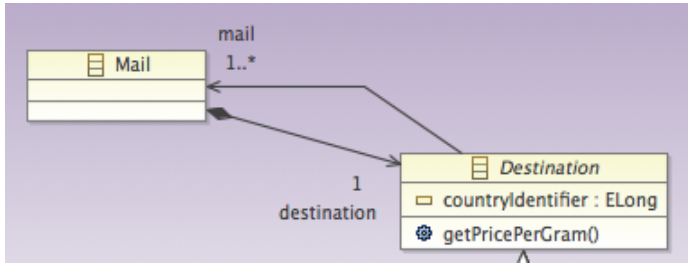


<<datatype>>
Time
<<javaclass>> fr.tpt.types.Time

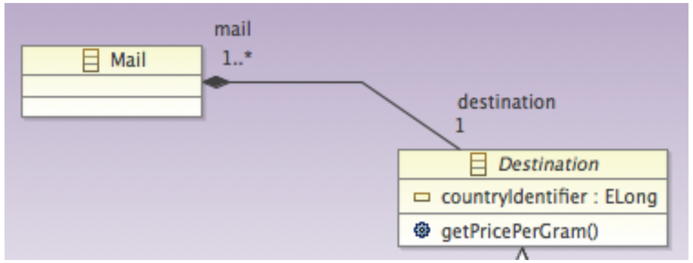
- Personalized data types aim at representing classes defined in existing Java code
- For instance, types defined in a library “math” or in a library “kinect” will be referenced in a EMF diagram using the EDataType
- The existing type is identified in the EDataType by its “qualified name”: the name of the class prefixed by the name of the package in which it is defined

EOpposite reference, example

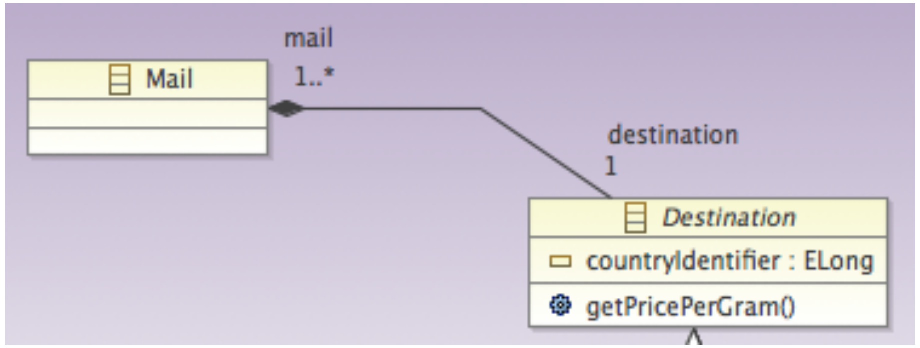
- When a class *Mail* references a class *Destination* that also references *Mail*:



- Reference mail and destination can be set as EOpposite:



EOpposite reference, good consequence



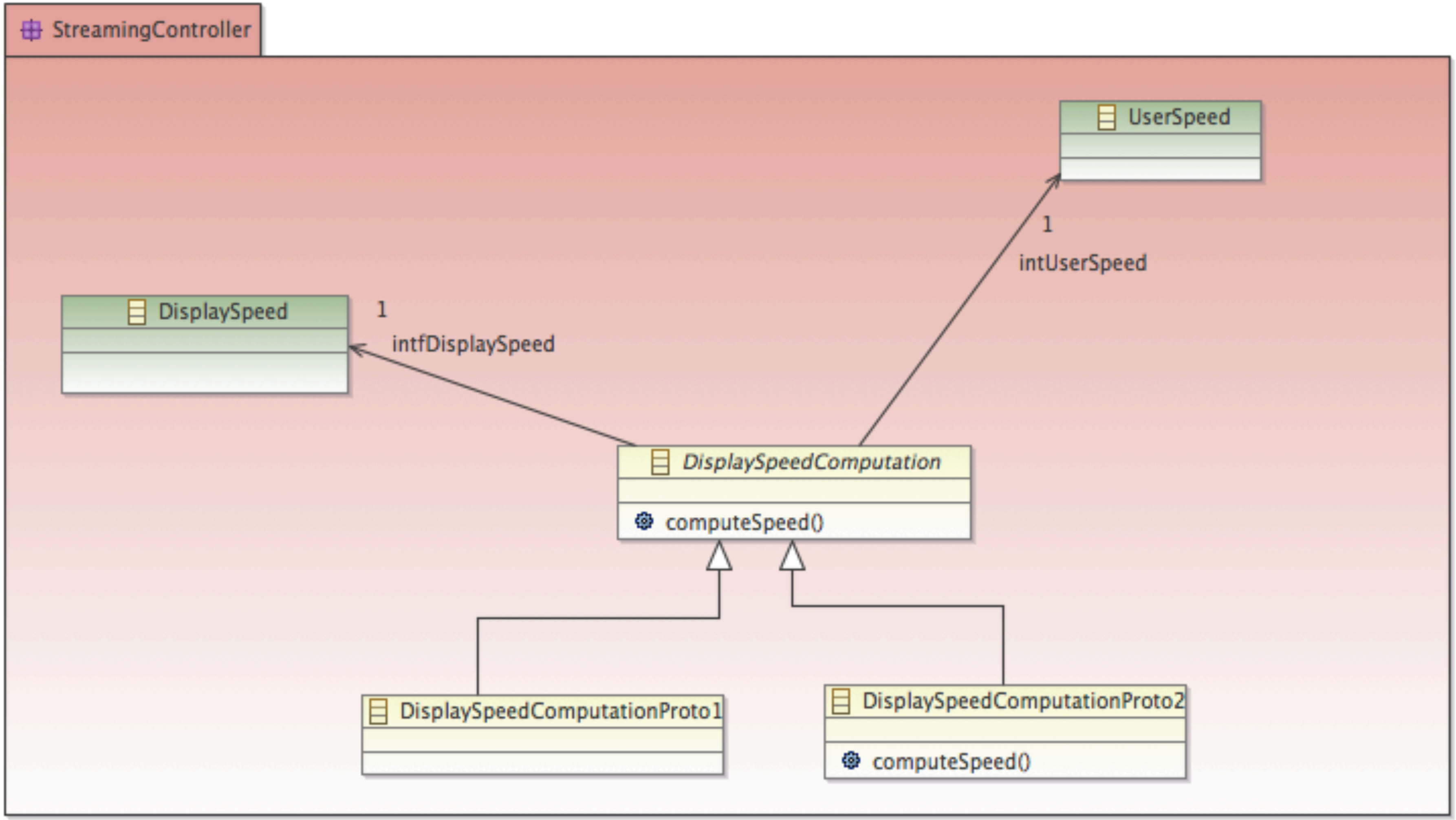
- When an object *m* of class *Mail* set an object *d* as its *destination*, then *m* is automatically set as the *mail* of *d*.

How to use EMF for specification review

- **In the analysis review:**
  - Components were represented by RED packages
  - Interactions between components were represented with GREEN interfaces and classes

- **For the specification review:**
  - Represent the internals of components (RED packages) by giving the class diagram of their implementations. Use default (yellow) color for this.

Example


StreamingController
UserSpeed
1
intUserSpeed
DisplaySpeed
1
intfDisplaySpeed
DisplaySpeedComputation
computeSpeed()
DisplaySpeedComputationProto1
DisplaySpeedComputationProto2
computeSpeed()

Organization of the presentation

1. **Presentation of EMF**

2. **Tutorial about "how to install, initialize, and use EMF" in Eclipse.**

   - Install
   - Create and edit a diagram
   - Generate Java code (for advanced Java programmers)

Installation procedure

- **In Eclipse,**
  - Go to Help --> Install New Software
  - Click on "Add »
  - Fill in the form

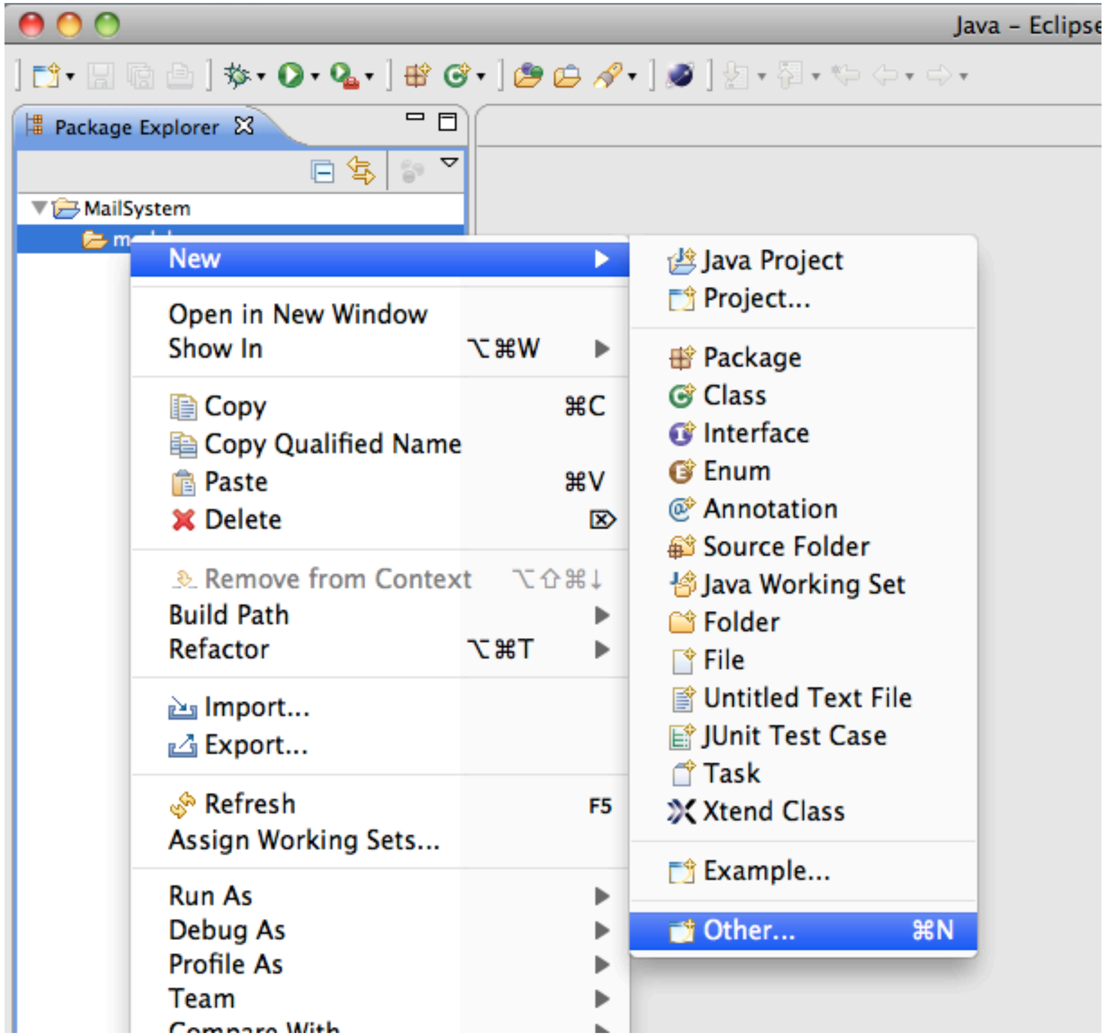
    Name: Juno

    Location: http://download.eclipse.org/releases/juno
  - Then select "Juno" for "work with"
  - Extend the menu "Modeling" and select

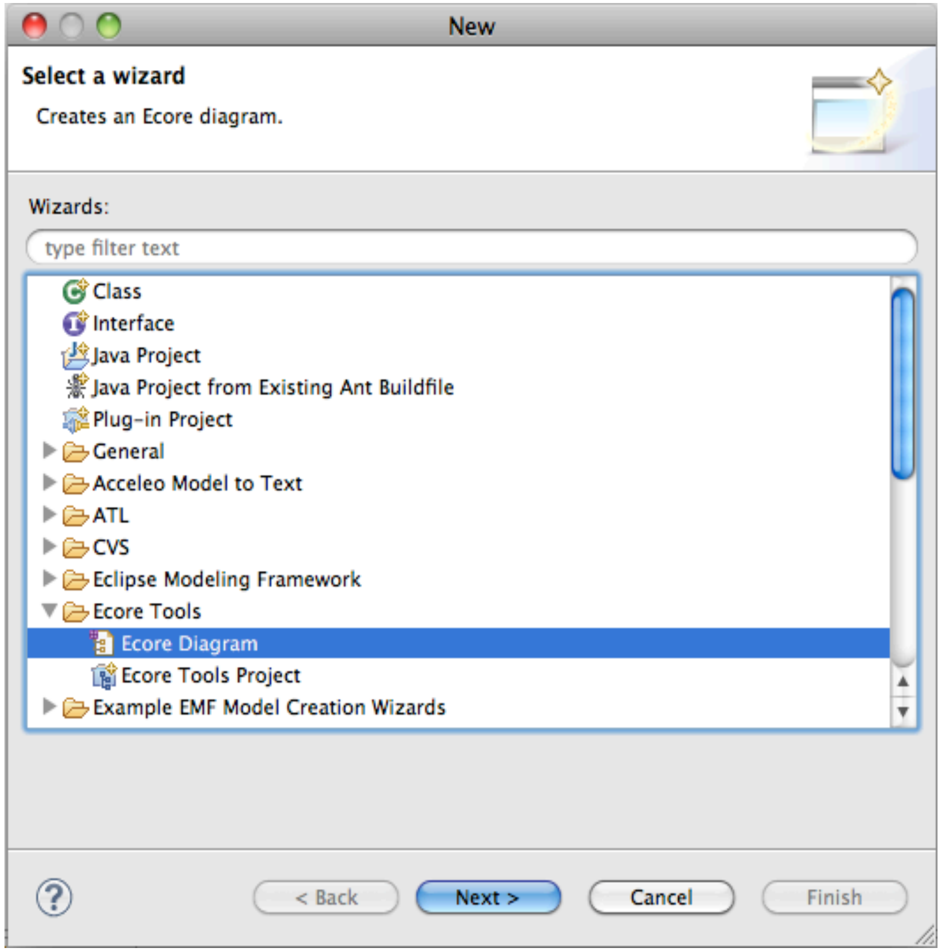
    "Ecore Tools SDK" and

    "EMF - Eclipse Modeling Framework SDK".
  - Click on "Next" then "Finish ».

Initialization



Initialization



New

Select a wizard

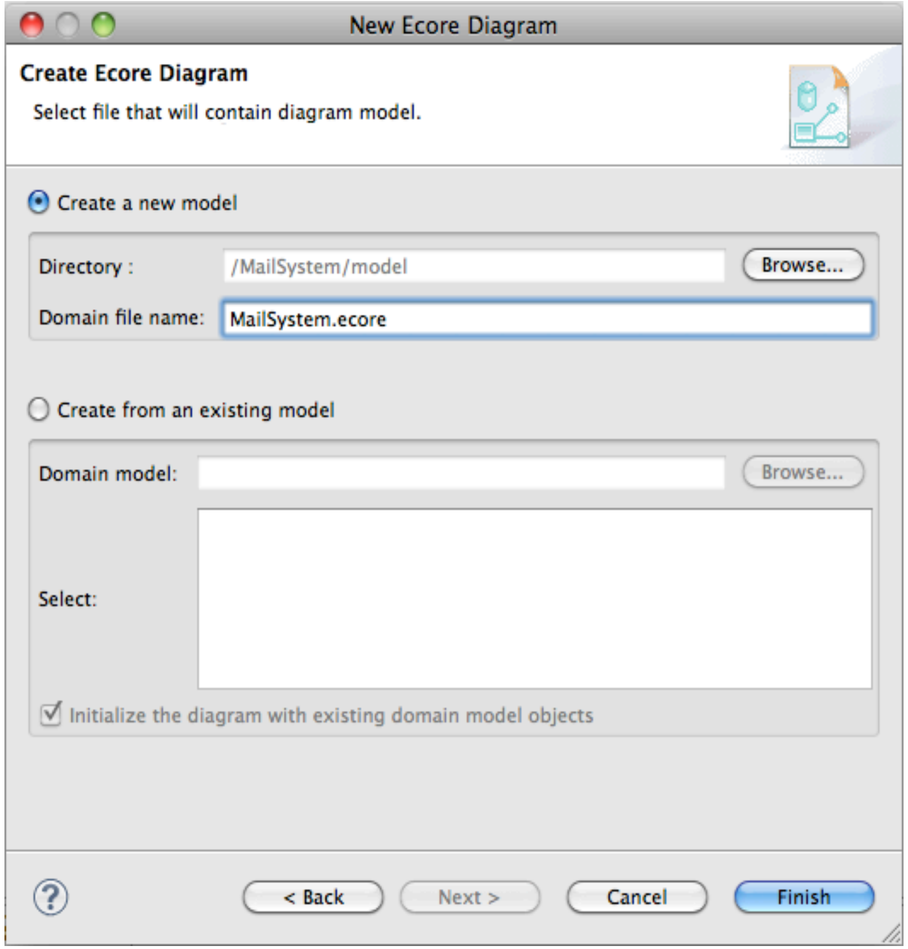
Creates an Ecore diagram.

Wizards:

type filter text

- Class
- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- Acceleo Model to Text
- ATL
- CVS
- Eclipse Modeling Framework
- Ecore Tools
  - Ecore Diagram
  - Ecore Tools Project
- Example EMF Model Creation Wizards

Initialization



New Ecore Diagram

Create Ecore Diagram

Select file that will contain diagram model.

● Create a new model

Directory : /MailSystem/model

Domain file name: MailSystem.ecore

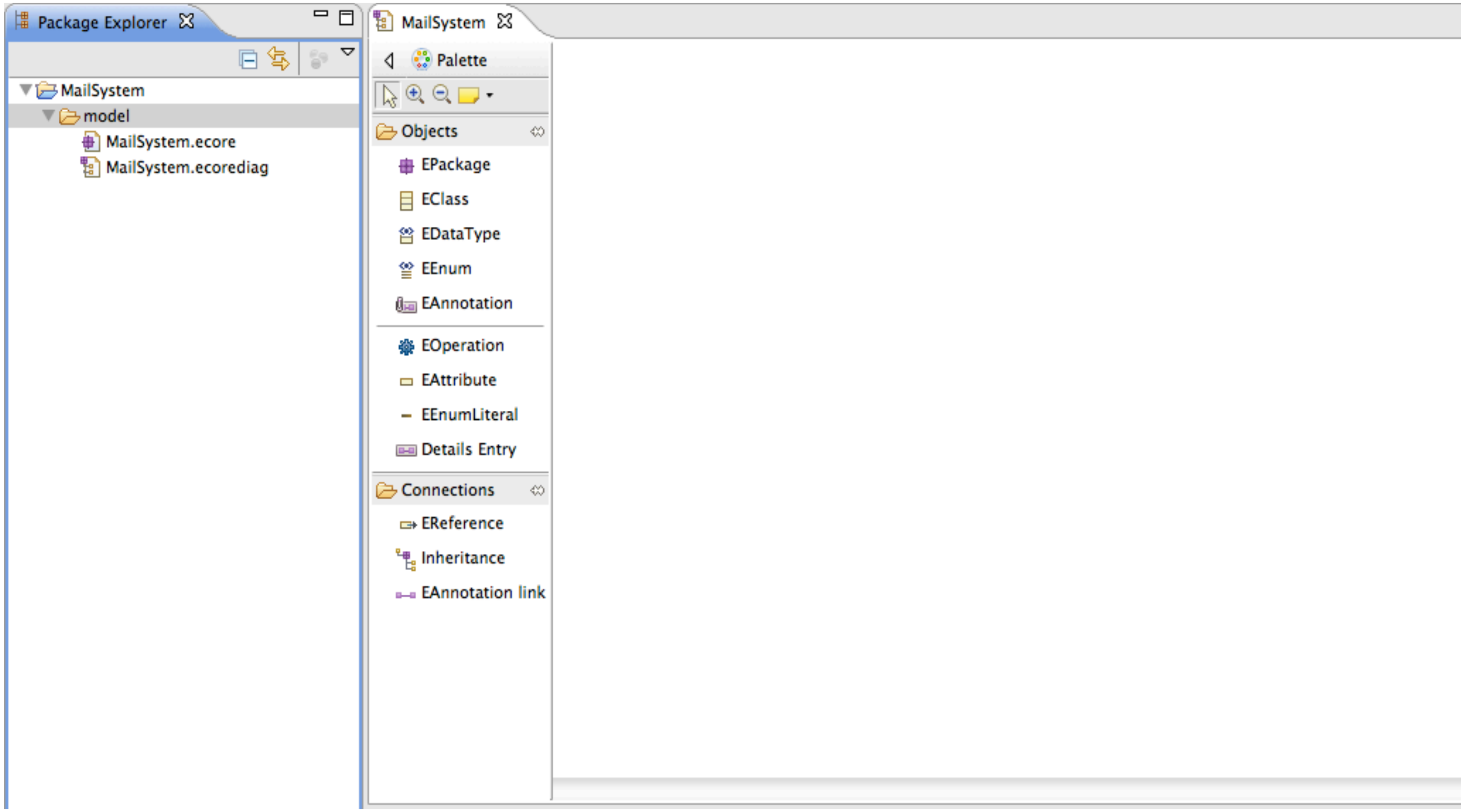
○ Create from an existing model

Domain model:

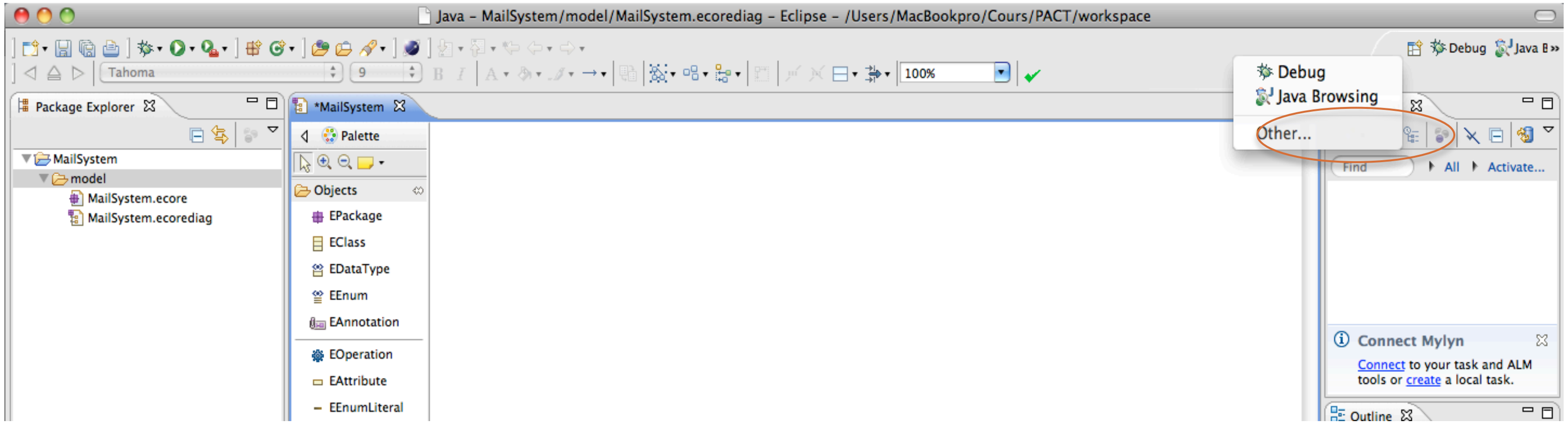
Select:

☑ Initialize the diagram with existing domain model objects

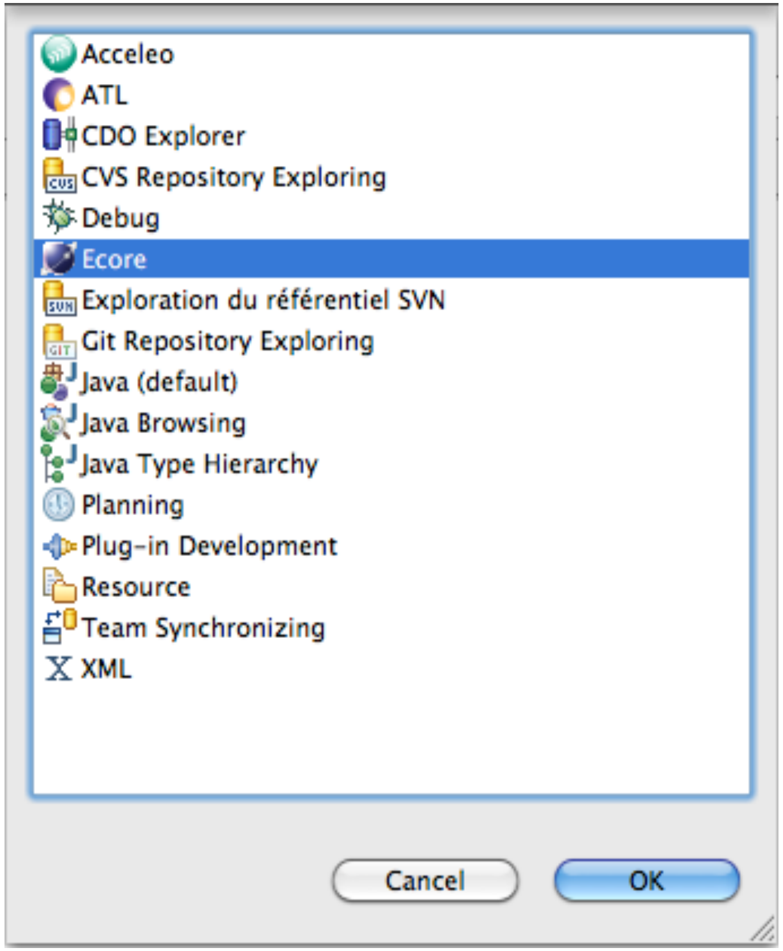
Initialization



Initialization

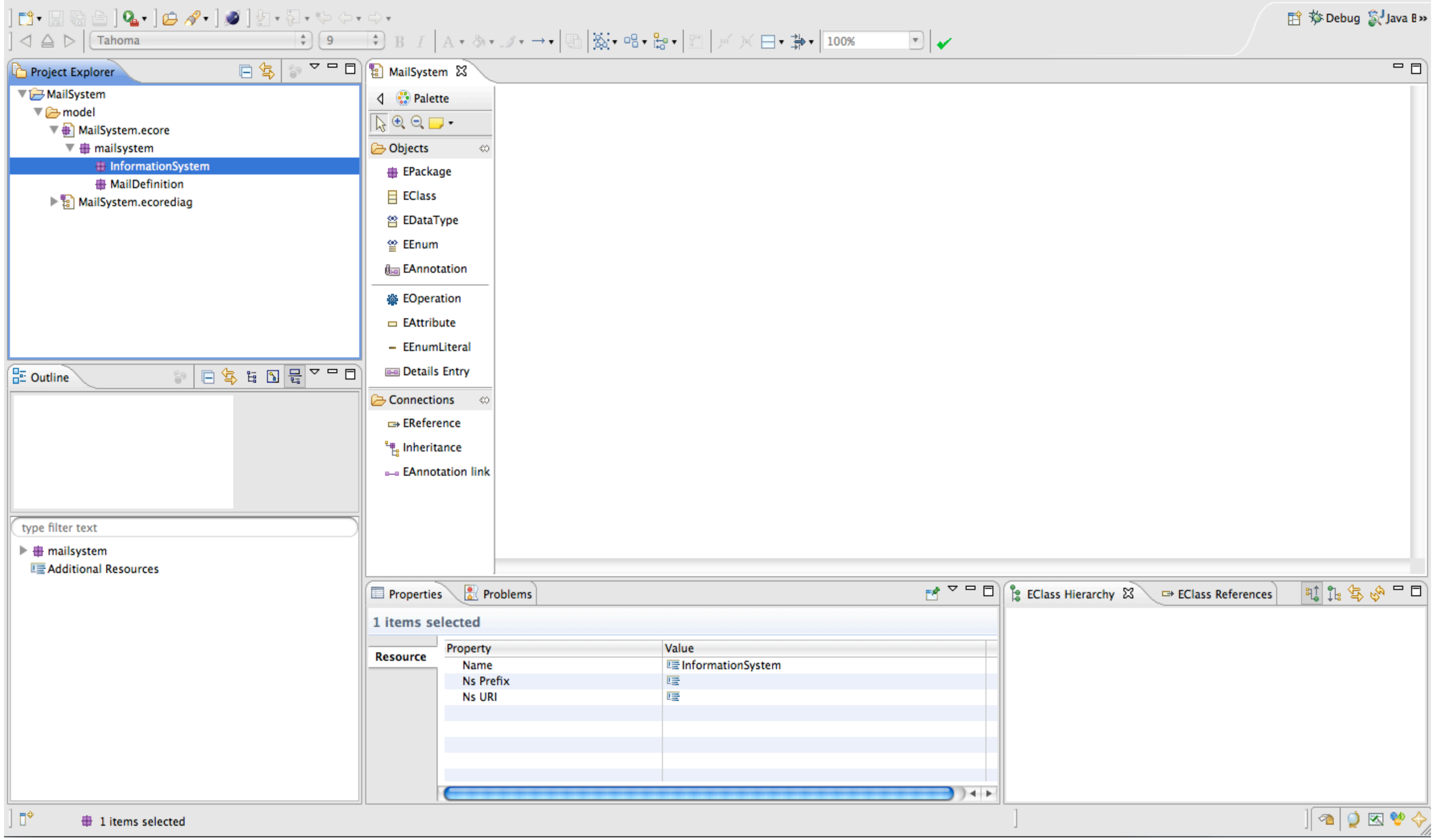


Initialization



- Acceleo
- ATL
- CDO Explorer
- CVS Repository Exploring
- Debug
- Ecore
- Exploration du référentiel SVN
- Git Repository Exploring
- Java (default)
- Java Browsing
- Java Type Hierarchy
- Planning
- Plug-in Development
- Resource
- Team Synchronizing
- XML

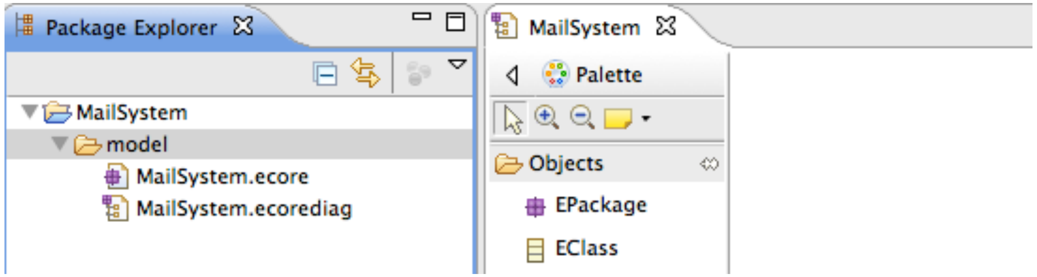
Initialization



1 items selected

Property	Value
Name	InformationSystem
Ns Prefix	
Ns URI	

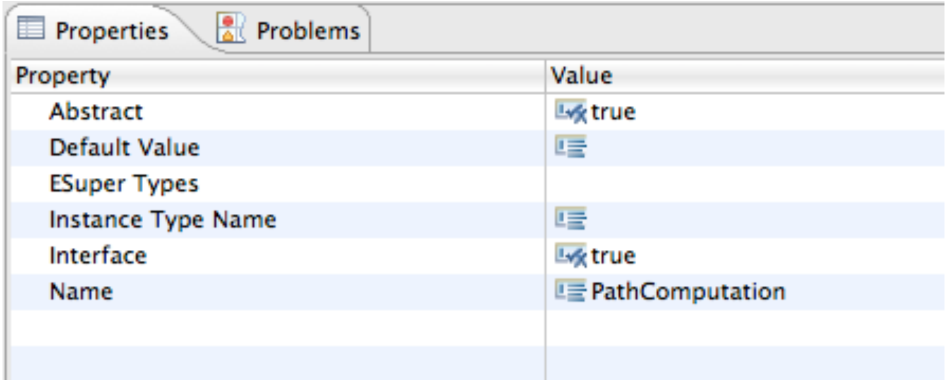
EMF Edition



- **Two types of editable files are created:**
  - Ecore model (.ecore)
  - Ecore diagram (.ecorediag)
- **They are automatically synchronized by the EMF Framework: a modification in the .ecore will be visible in the .ecorediag and vice-versa.**
- **The diagram (. ecorediag) is just a graphical representation of the content of the model (.ecore).**
- **The diagram does not necessarily represent all the content of the model**
- **There can exist several diagrams for one model, called "views" of the model. To do this, copy paste the ecorediagram file and change its name.**

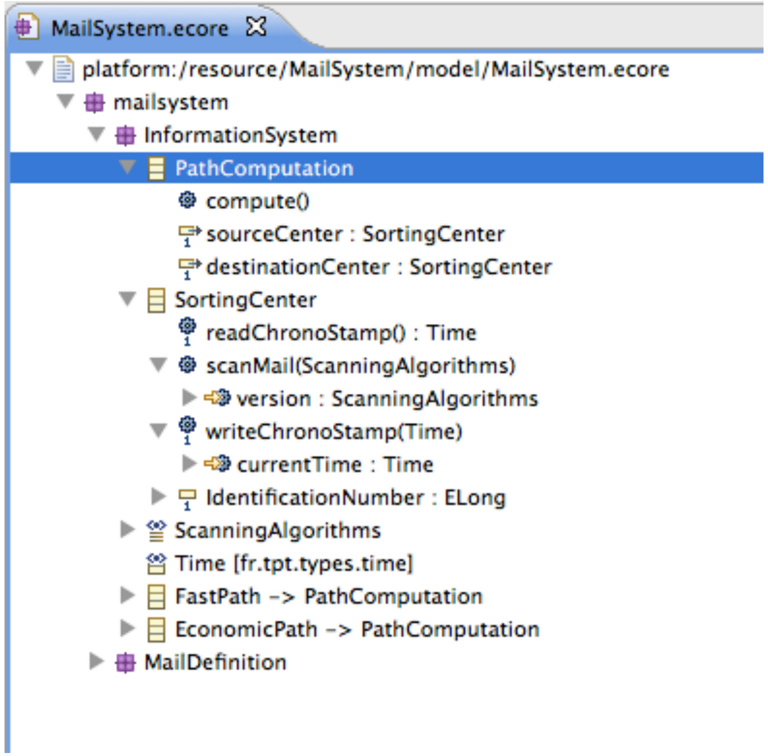
Edition of the Ecore model

- For each type of element (class, attribute, method, etc…) most of its configuration will be done in the "Property view"



Properties | Problems

Property	Value
Abstract	true
Default Value	
ESuper Types	
Instance Type Name	
Interface	true
Name	PathComputation

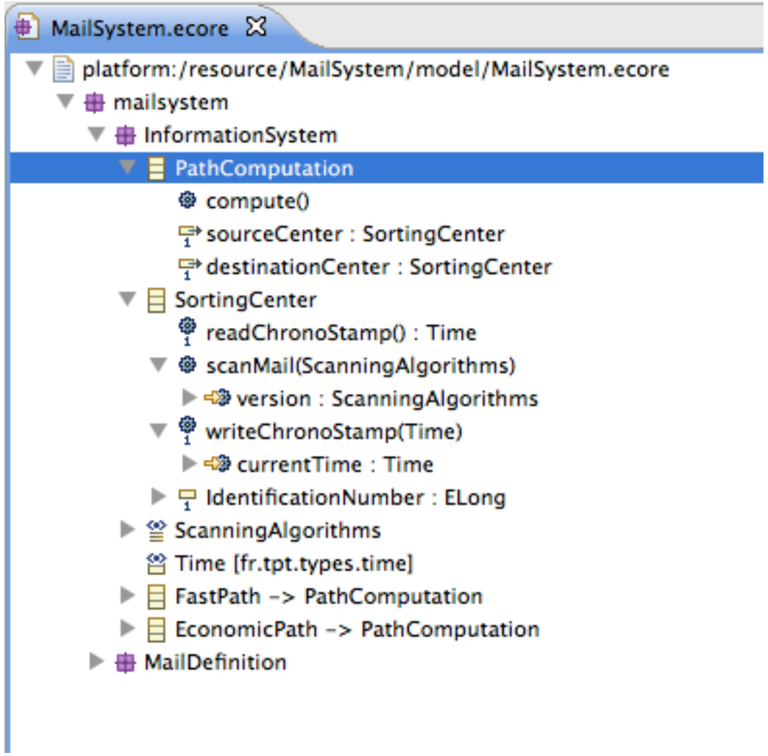


MailSystem.ecore

```
platform:/resource/MailSystem/model/MailSystem.ecore
  mailsystem
    InformationSystem
      PathComputation
          compute()
          sourceCenter : SortingCenter
          destinationCenter : SortingCenter
      SortingCenter
          readChronoStamp() : Time
        scanMail(ScanningAlgorithms)
          version : ScanningAlgorithms
        writeChronoStamp(Time)
          currentTime : Time
        IdentificationNumber : ELong
      ScanningAlgorithms
      Time [fr.tpt.types.time]
      FastPath -> PathComputation
      EconomicPath -> PathComputation
    MailDefinition
```

Edition of the Ecore model

- **When the ECore file is open, its content appears as a tree of packages, classes, methods, attributes, etc…**
- **To create a new element, write click on an existing element (by default there is at least a root package)**
  - Go to “New Child” or “New Sibling”
  - New Child => elements to be created as a sub-element of the selected element
  - New Sibling => elements to be created as a sub-element of the parent of the selected element
- **To delete an element, right click on the element → Delete.**



MailSystem.ecore

```
platform:/resource/MailSystem/model/MailSystem.ecore
  mailsystem
    InformationSystem
      PathComputation
        compute()
        sourceCenter : SortingCenter
        destinationCenter : SortingCenter
      SortingCenter
        readChronoStamp() : Time
        scanMail(ScanningAlgorithms)
          version : ScanningAlgorithms
        writeChronoStamp(Time)
          currentTime : Time
        IdentificationNumber : ELong
      ScanningAlgorithms
      Time [fr.tpt.types.time]
      FastPath -> PathComputation
      EconomicPath -> PathComputation
    MailDefinition
```

One file per subsystem or component

- **Behind the graphical editor, EMF models are stored in files with an ugly format…**
- **If you have conflicting modifications in one of these file, merging modifications will be tricky…**

- **Practical advice:**
  - Create one EMF model for the whole system, to be defined all together
  - Create one EMF model per subsystem, or component, so that only 2 students are working together on a model

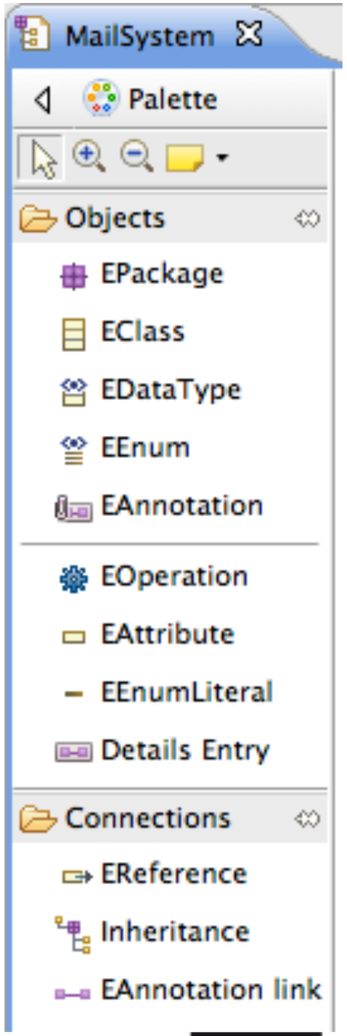
  - Referencing a model from another model is easy, a drag and drop from the package explorer to the content of the diagram should do the thing.

Edition of the Diagram

- To add an element, select the element type in the palette, then click on the part of the diagram where you want to add it (empty space, package, attributes section of a class, methods section of a class, etc…).
- To configure an element, select it in the diagram and edit the properties in the "Property view".
- To delete an element, right click on it

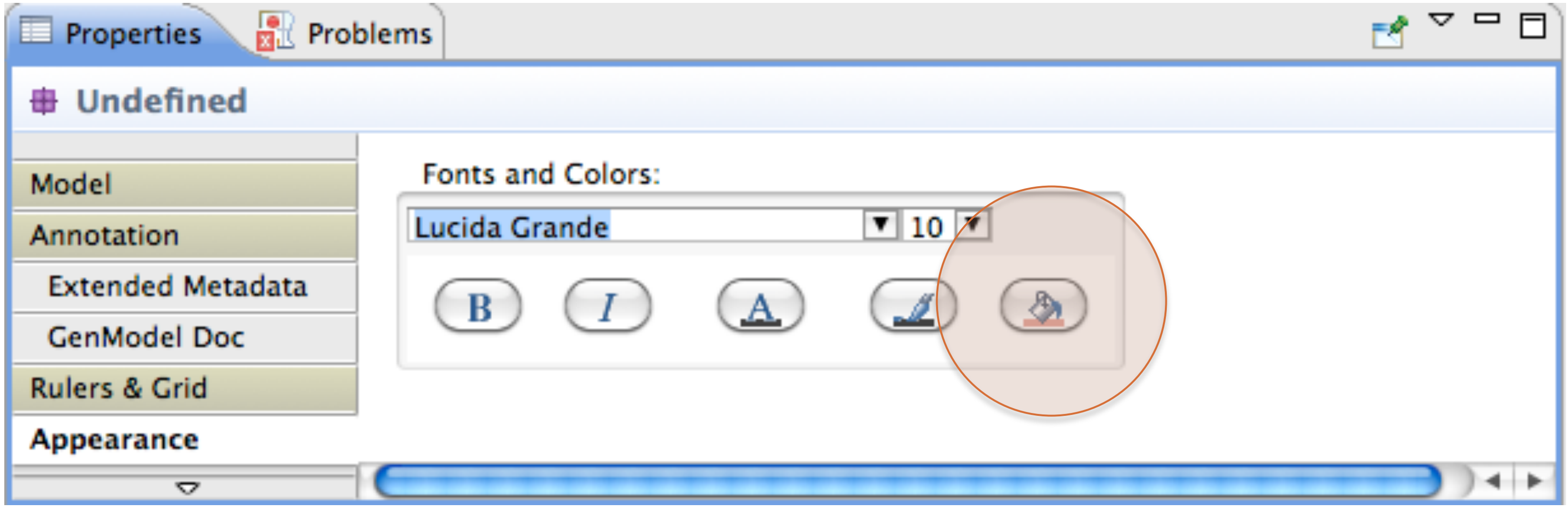
→delete from model will delete it from the diagram (.ecorediag) and the model (.ecore).

→ delete from diagram will delete it from the diagram (.ecorediag) but the element will remain in the model (.ecore).



Change the color of an element

Select the element you want to change the color, and go to the property view. Then click on the painting sign and select the color

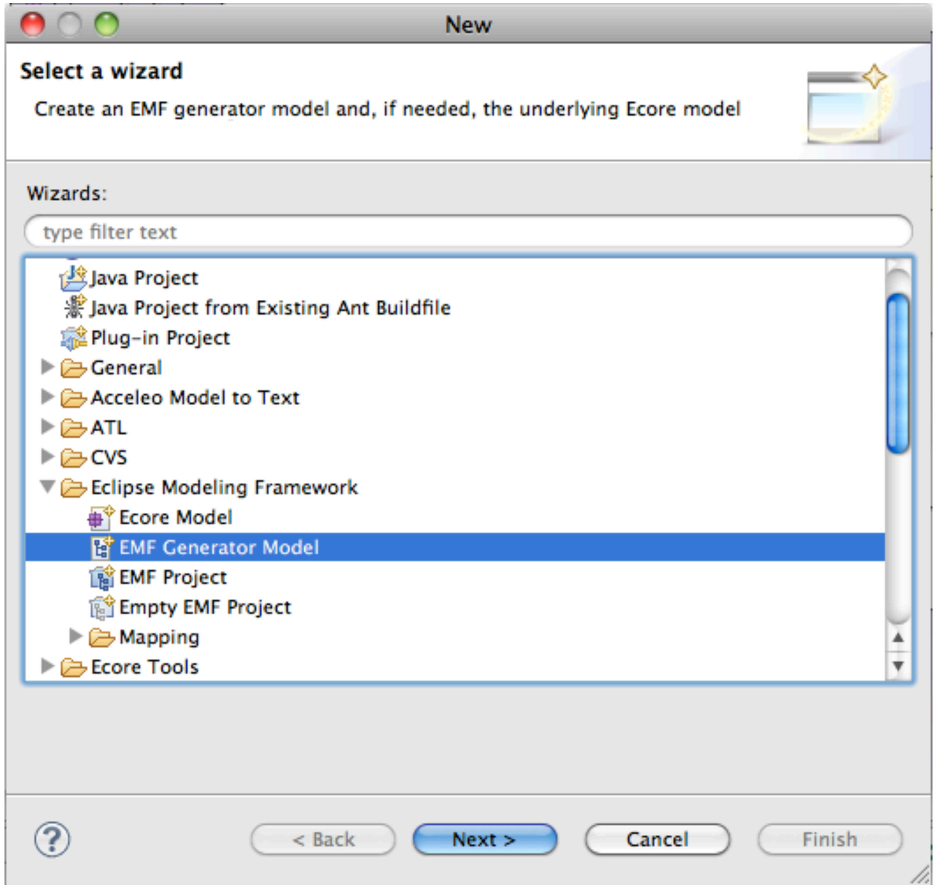


Code Generation (advanced programmers)

- **ONE WAY code generation: no synchronization of modifications of the Java classes in the EMF model**
- **EMF will generate the Java code corresponding to your model:**
  - Interfaces
  - Classes
  - Attributes
  - Inheritance
  - References
  - Etc…

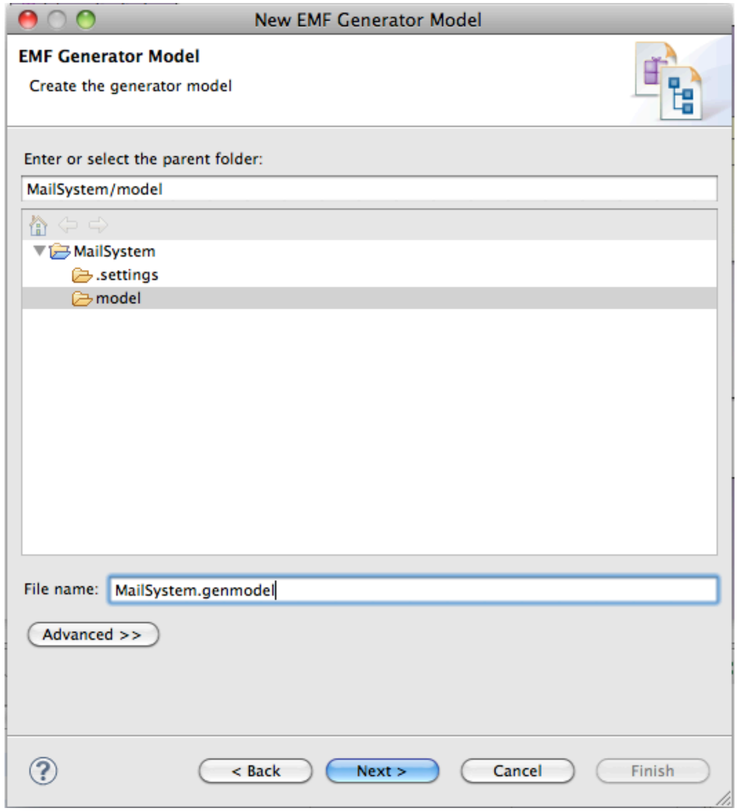
Configure the Code Generation

- **The code generation is driven by the content of a .genmodel file**
- **We explain hereafter how to create this file:**
  - Right click on the foldre where you want to ad the file
  - Click on New → Other
  - Select EMF Generator Model (see picture)



Create .genmodel file

- Fill in the name of the file, then select ecore as type of importation



New EMF Generator Model

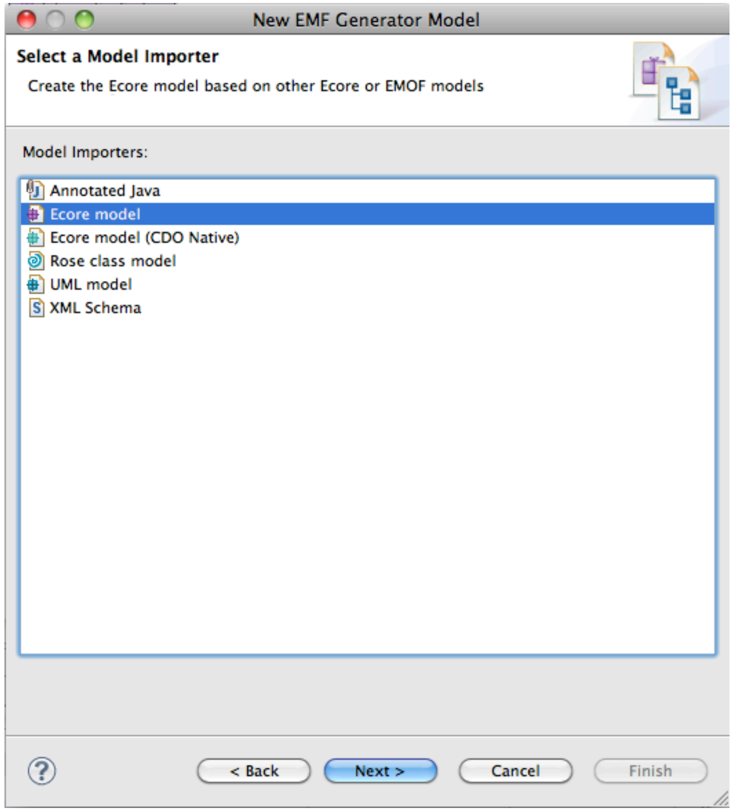
EMF Generator Model

Create the generator model

Enter or select the parent folder: MailSystem/model

- MailSystem
  - .settings
  - model

File name: MailSystem.genmodel



New EMF Generator Model

Select a Model Importer

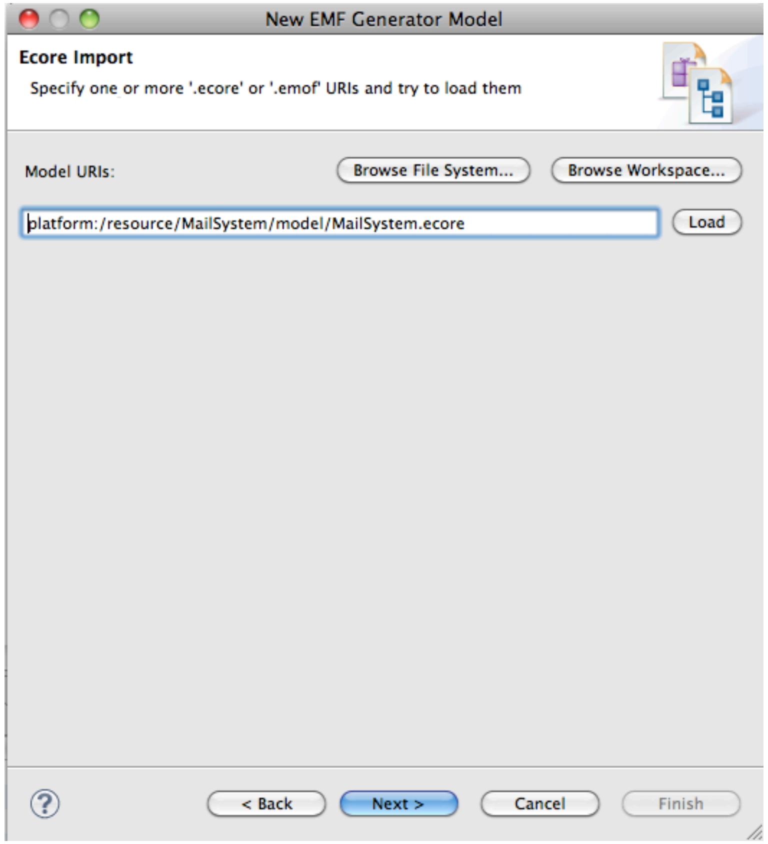
Create the Ecore model based on other Ecore or EMOF models

Model Importers:

- Annotated Java
- Ecore model
- Ecore model (CDO Native)
- Rose class model
- UML model
- XML Schema

Create .genmodel file

- Select ecore model (via the Browse workspace button) and click on next, then click on finish

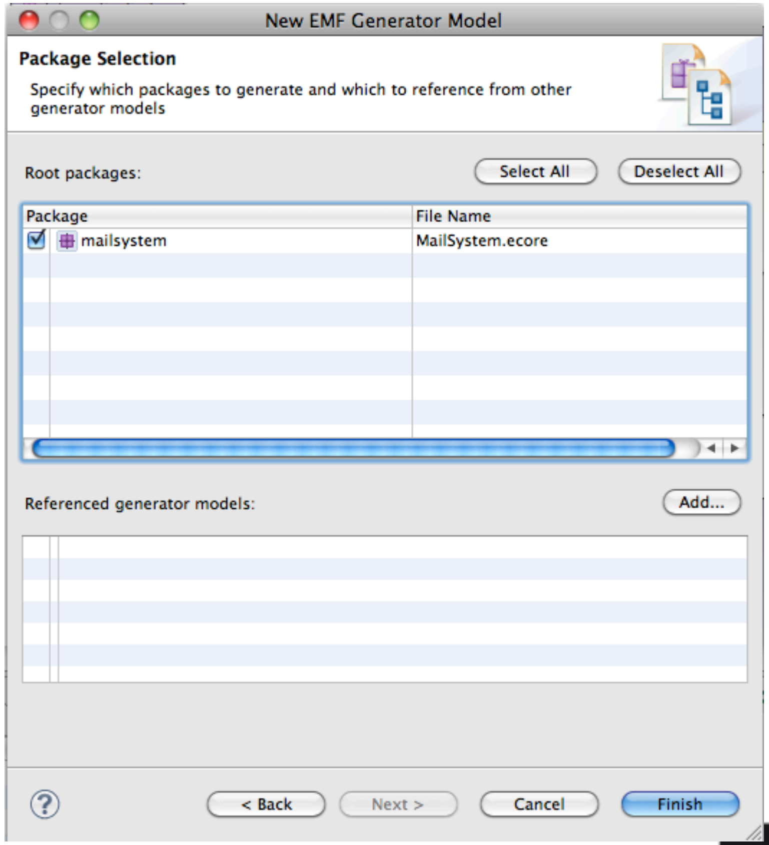


New EMF Generator Model

Ecore Import

Specify one or more '.ecore' or '.emof' URIs and try to load them

Model URIs: platform:/resource/MailSystem/model/MailSystem.ecore



New EMF Generator Model

Package Selection

Specify which packages to generate and which to reference from other generator models

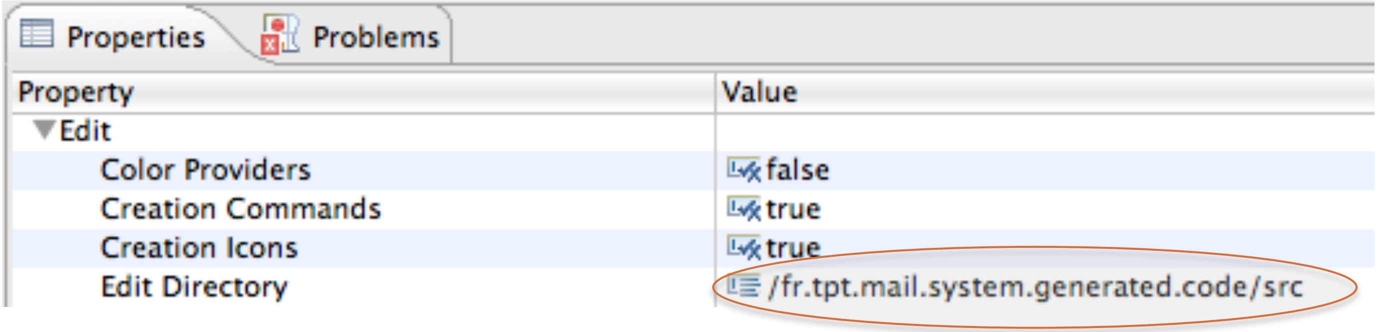
Root packages:

Package	File Name
☑ mailsystem	MailSystem.ecore

Referenced generator models:

Configure .genmodel file

- **“everything” is in the property view**
  - when you select the root of the genmodel content, fill in:



Property	Value
▼Edit	
Color Providers	false
Creation Commands	true
Creation Icons	true
Edit Directory	/fr.tpt.mail.system.generated.code/src

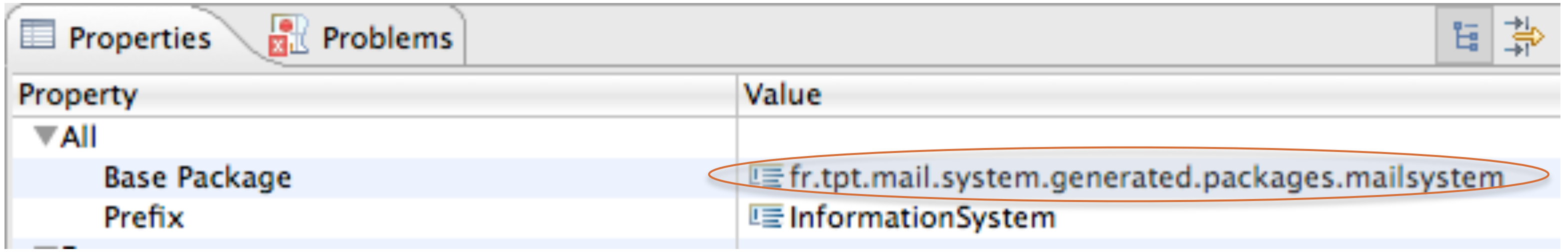
Edit Directory, Editor Directory, Model Directory, Test Directory.

/ identifies the root of the workspace
fr.tpt. … .generated.code identifies a Java project in the workspace
src is a source folder in this Java project

- The value of these properties will tell EMF where to generate code. Preferably select a folder in a Java Project.

Configure .genmodel file

- **"everything" is in the property view**
  - When you select a package of in the genmodel content, fill in:



Property	Value
▼All	
Base Package	fr.tpt.mail.system.generated.packages.mailsystem
Prefix	InformationSystem

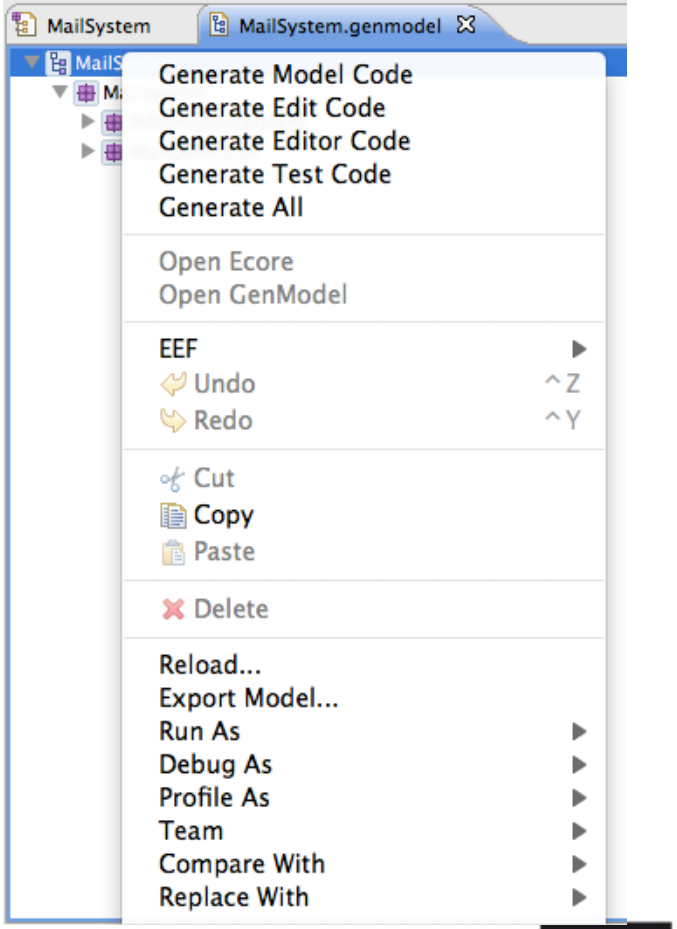
BasePackage.

Choose a name that respect you naming conventions (they should be documented)

  - The value of this property will tell EMF which prefix to use to generated Java Packages

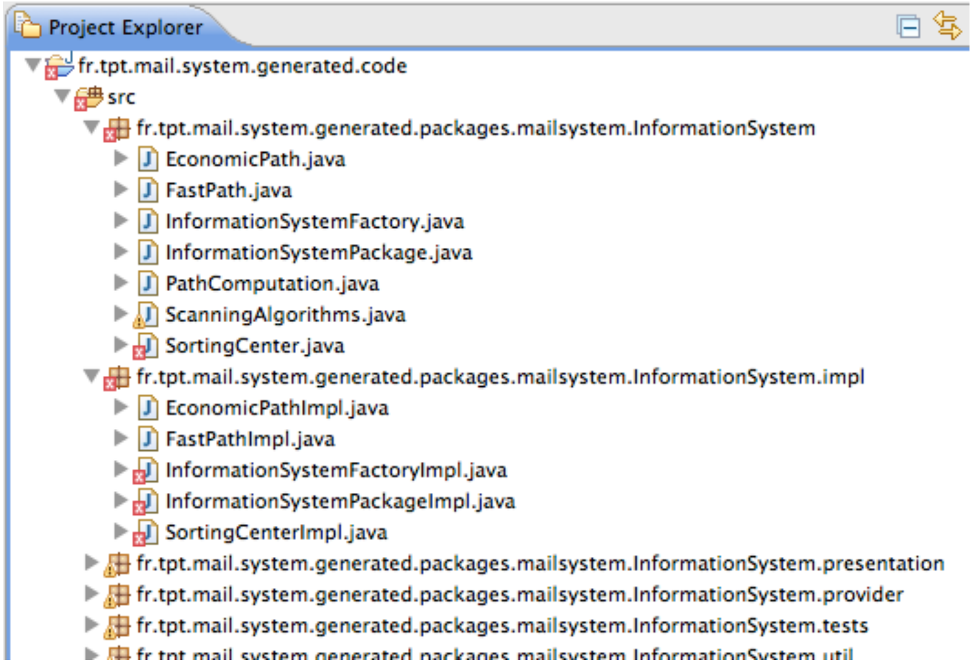
Code generation actions

- Right-click on the root of the genmodel tree, click on Generate All



Result of the code generation

- After generation, the generated code is generally compiled
- Compilation Errors are generally due to unresolved dependencies, Class “fr.tpt.types.Time” in our example (used in the personalized data type Time)
- This type of errors are fixed by updating the build path of the Java project



Project Explorer

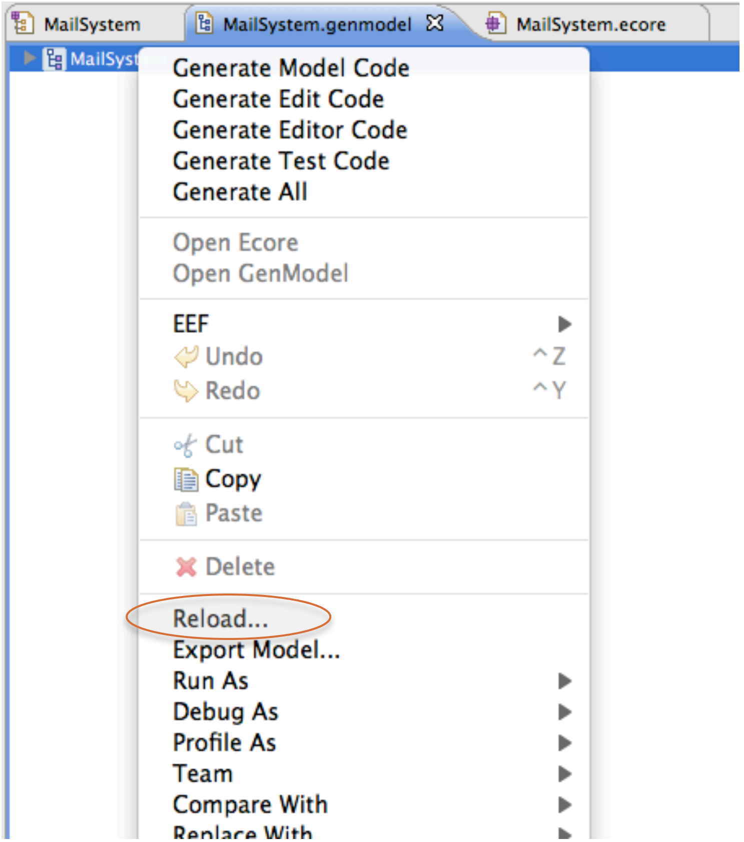
```
▼ fr.tpt.mail.system.generated.code
  ▼ src
    ▼ fr.tpt.mail.system.generated.packages.mailsystem.InformationSystem
      ▶ EconomicPath.java
      ▶ FastPath.java
      ▶ InformationSystemFactory.java
      ▶ InformationSystemPackage.java
      ▶ PathComputation.java
      ▶ ScanningAlgorithms.java
      ▶ SortingCenter.java
    ▼ fr.tpt.mail.system.generated.packages.mailsystem.InformationSystem.impl
      ▶ EconomicPathImpl.java
      ▶ FastPathImpl.java
      ▶ InformationSystemFactoryImpl.java
      ▶ InformationSystemPackageImpl.java
      ▶ SortingCenterImpl.java
    ▶ fr.tpt.mail.system.generated.packages.mailsystem.InformationSystem.presentation
    ▶ fr.tpt.mail.system.generated.packages.mailsystem.InformationSystem.provider
    ▶ fr.tpt.mail.system.generated.packages.mailsystem.InformationSystem.tests
    ▶ fr.tpt.mail.system.generated.packages.mailsystem.InformationSystem.util
```

Re-generate the code

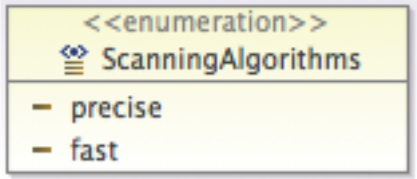
- **When you have modified the EMF model and you want to regenerate the code, reload the model (.ecore) into the generation model (.genmodel)**

  → Click on next, next, finish.

- **Then re-execute the code generation actions**



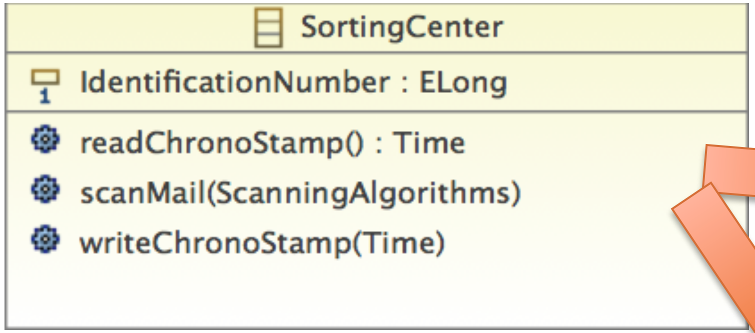
Generated code



<<enumeration>>
ScanningAlgorithms
- precise
- fast

```
public final class ScanningAlgorithms extends AbstractEnumerator {
  …
  public static final int FAST = 0;
  public static final int PRECISE = 1;
  …
}
```

Generated code


SortingCenter

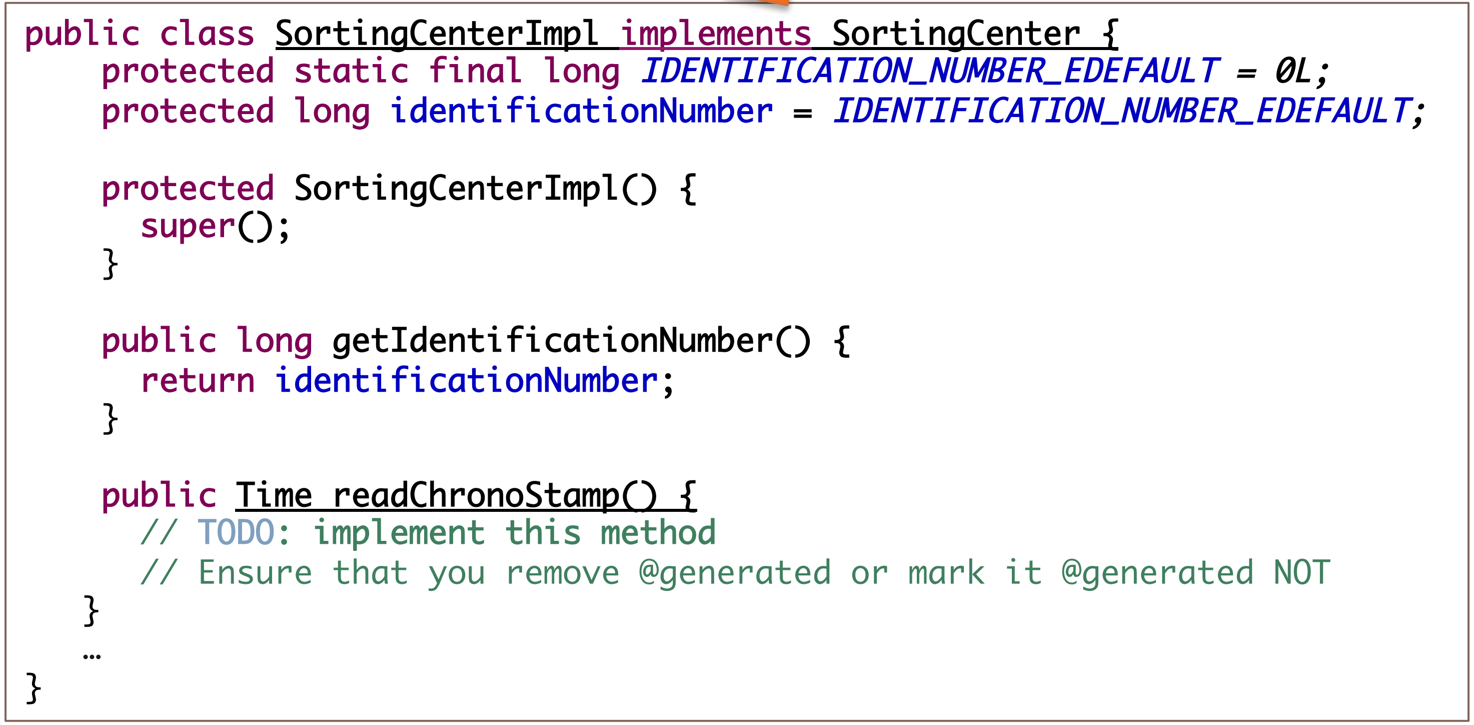
IdentificationNumber : ELong

readChronoStamp() : Time

scanMail(ScanningAlgorithms)

writeChronoStamp(Time)

```
public interface SortingCenter
{
    long getIdentificationNumber();
    void setIdentificationNumber(long value);
    Time readChronoStamp();
    …
}
```


```
public class SortingCenterImpl implements SortingCenter {
    protected static final long IDENTIFICATION_NUMBER_EDEFAULT = 0L;
    protected long identificationNumber = IDENTIFICATION_NUMBER_EDEFAULT;

    protected SortingCenterImpl() {
      super();
    }

    public long getIdentificationNumber() {
      return identificationNumber;
    }

    public Time readChronoStamp() {
      // TODO: implement this method
      // Ensure that you remove @generated or mark it @generated NOT
   }
   …
}
```

Incremental code generation

- There is no synchronisation between the Java code and the EMF model, and ONLY EMF TO JAVA is possible (modifying the java code will not change the EMF model).

- When executed the first time, EMF generates code with annotations of type:

  @generated

- When re-executed, code that is not preceded by this annotation is not modified. Another way (more explicit) to have this result is to complete the annotation by @generated NOT
- This is useful to modify the generated code, for instance to implement a skeleton of method generated by EMF

Generated code in PACT (for advanced programmers)

- **Once generated**
  - Should be put under version control (Git) in order to avoid systematic regeneration of code by all the users.
  - The EMF model should not be modified anymore…

- **In other words, EMF is not supposed to be used during the implementation phase.**

Sources of information

- Official Eclipse – EMF web page: http://www.eclipse.org/modeling/emf/
- Interesting (up-to-date) turorial: http://www.vogella.com/articles/EclipseEMF/article.html
- Interesting (a bit outdated) tutorials http://www.eclipse.org/articles/Article-Using%20EMF/using-emf.html http://www.openarchitectureware.org/pub/documentation/4.2/html/contents/emf_tutorial.html
- General course about modeling http://www.idt.mdh.se/kurser/dva411/index.php?pageId=lectures

Key concepts of Eclipse (key = used in this course)

- Project: any directory that contains a “*.project*” file created by Eclipse
- Workspace: any directory that contains a “*.metadata*” folder created by Eclipse. Note that the information contained in the “*.metadata*” folder point to the projects used in the workspace.
- Update site: a web interface to install plugins
- Resource: a file in Eclipse is called a resource