Model-Based Software Engineering

Lecture 03 – Metamodeling cont., OCL

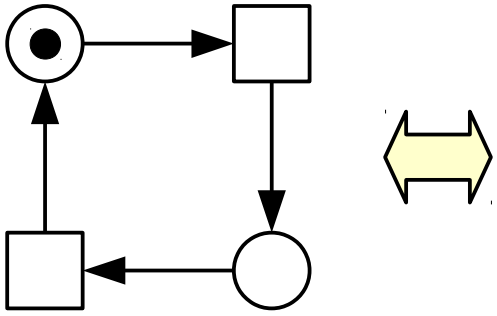
Prof. Dr. Joel Greenyer

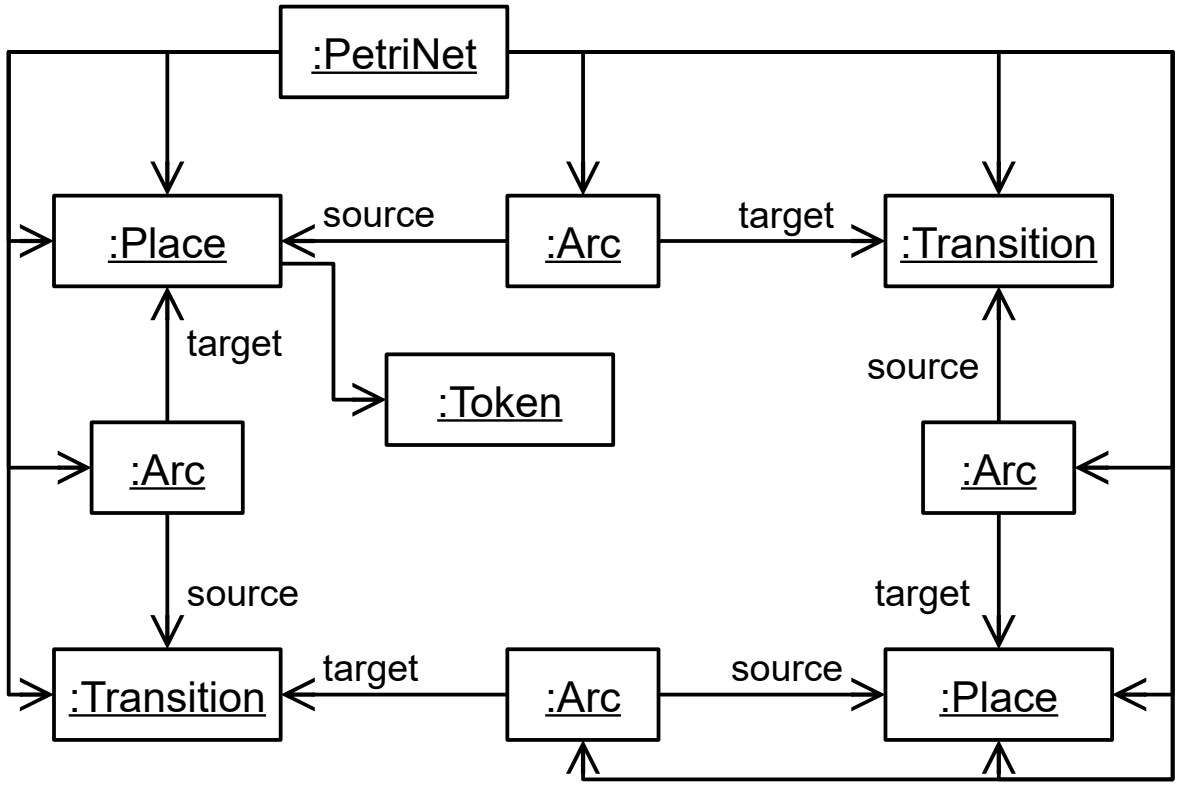
April 18, 2016

Object-Oriented Modeling Approach

in the last lecture...

- **Step 1**: Understand a model as a **structure of objects**
- For the example:





concrete syntax
(representation to the user)

abstract syntax
(internal structure, occurrences of language constructs and their relationships)

Model and Metamodel

in the last lecture...

metamodel

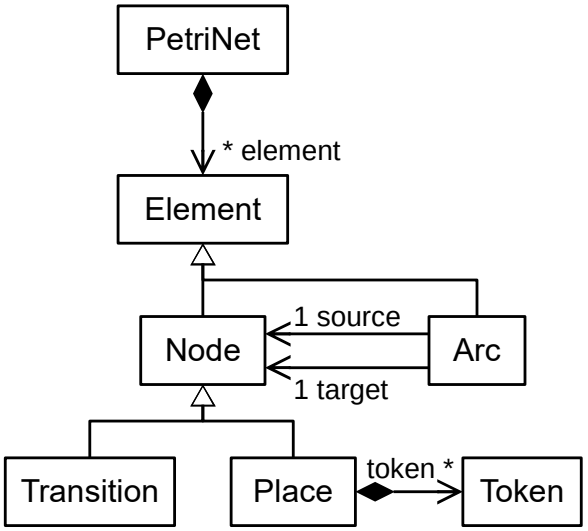
(language definition)

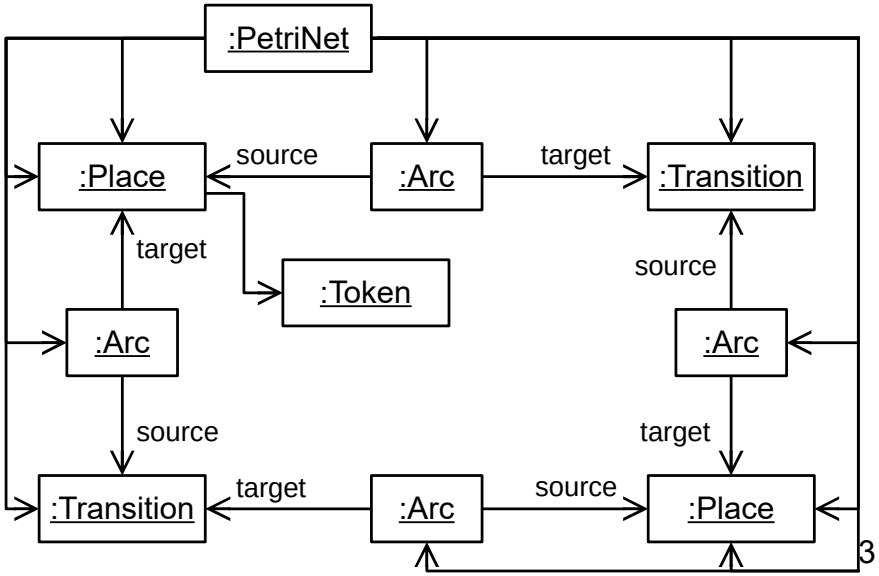
defines language of

instance of

model

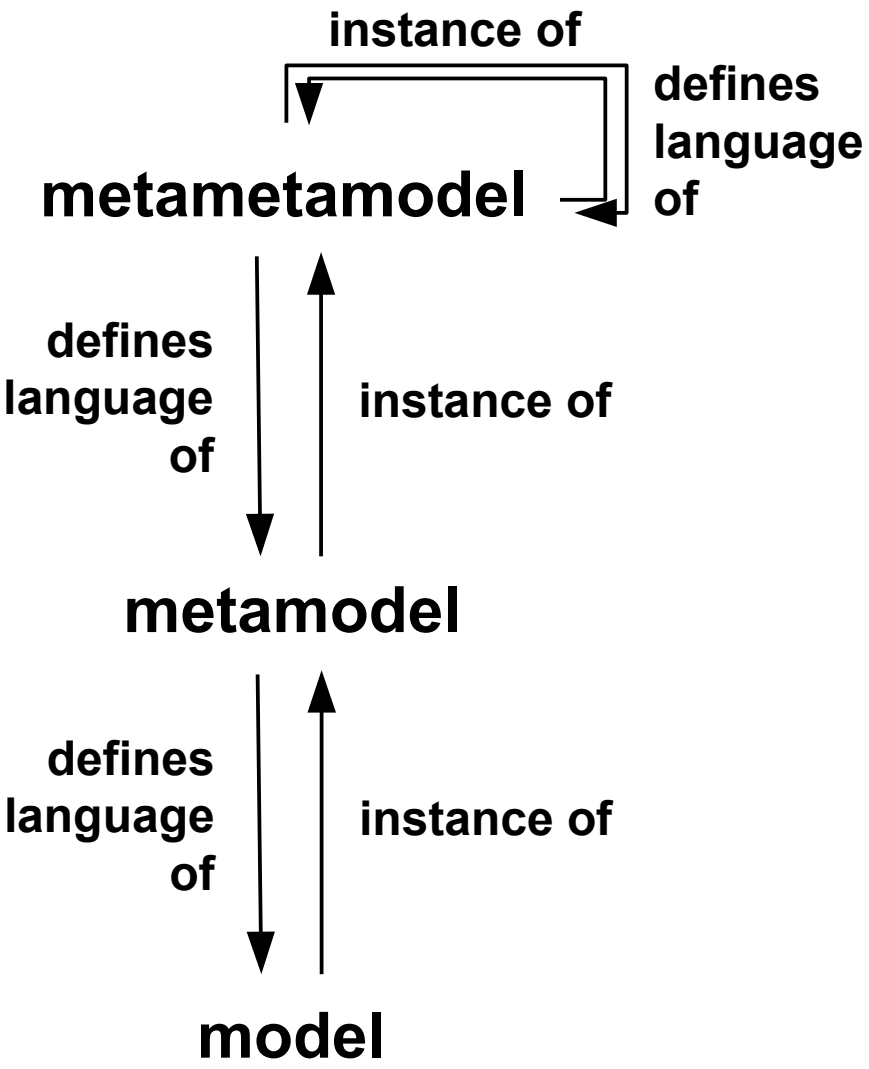
(model expressed in the language)

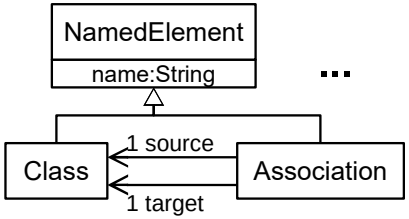


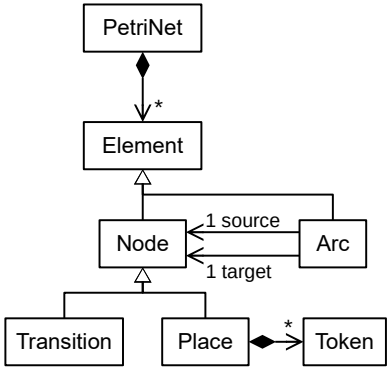


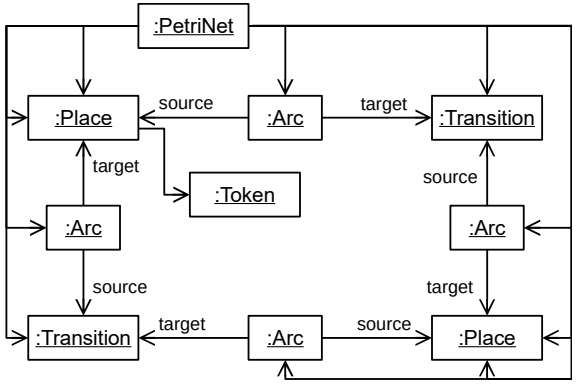
Meta-Levels

in the last lecture...









SOFTWARE ENGINEERING

in the last lecture...

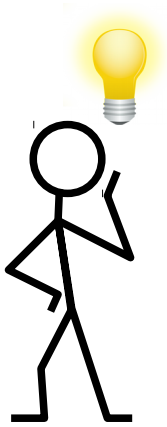
Typical Meta-Level Descriptions

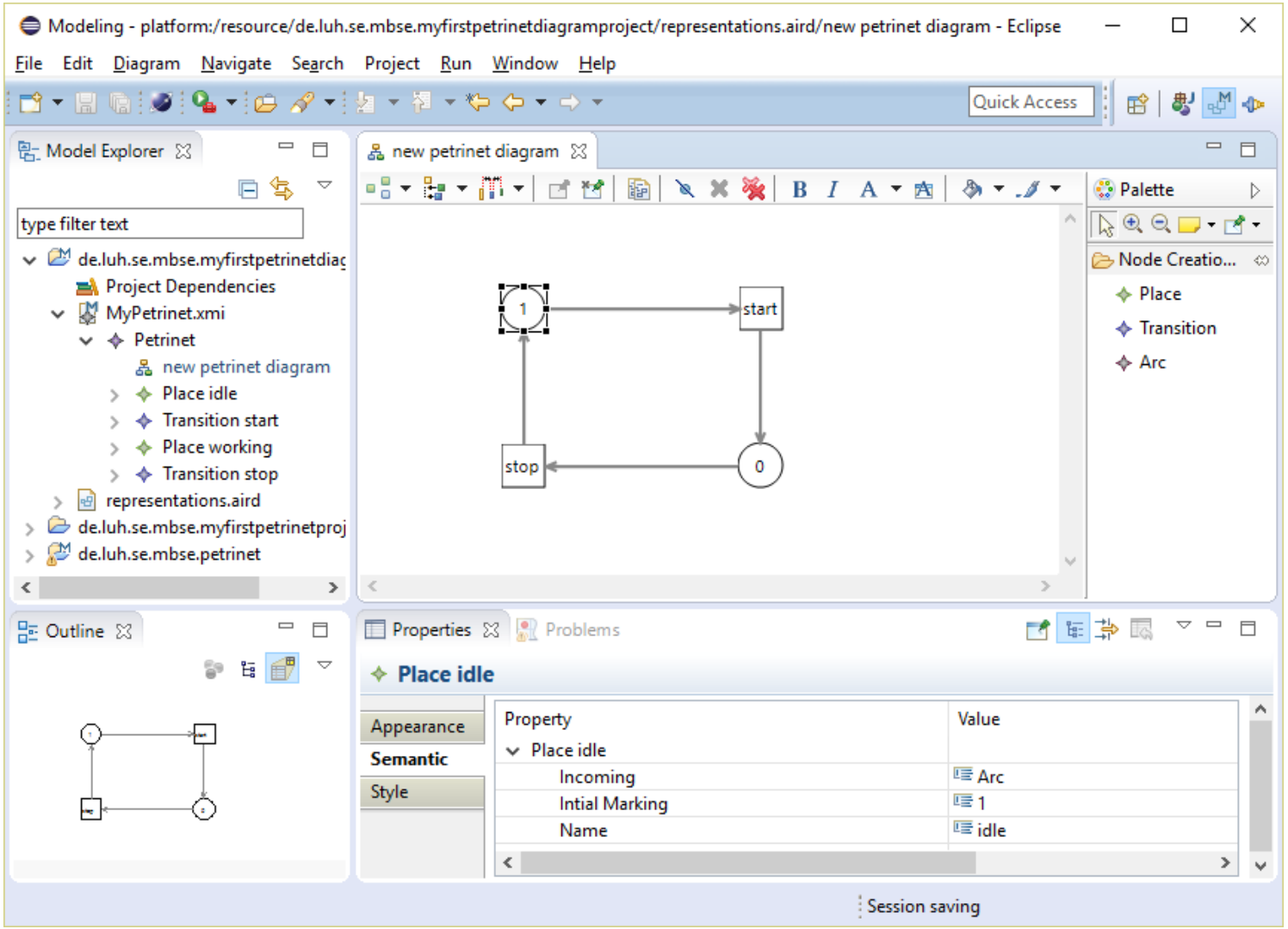
- Sometimes, we refer to the **four meta-levels** (M0-M3) originally defined by the MOF standard
  - MOF: Meta-Object Facility, standard by the OMG (see http://www.omg.org/mof/)

M3	**meta-metamodel** to define metamodels on M2, also describes itself
M2	**metamodels**, for defining a modeling language on M1
M1	**models** of data or processes
M0	**instance-model,** concrete data

Vision: Build a Petri Net Modeling Tool

in the last lecture...

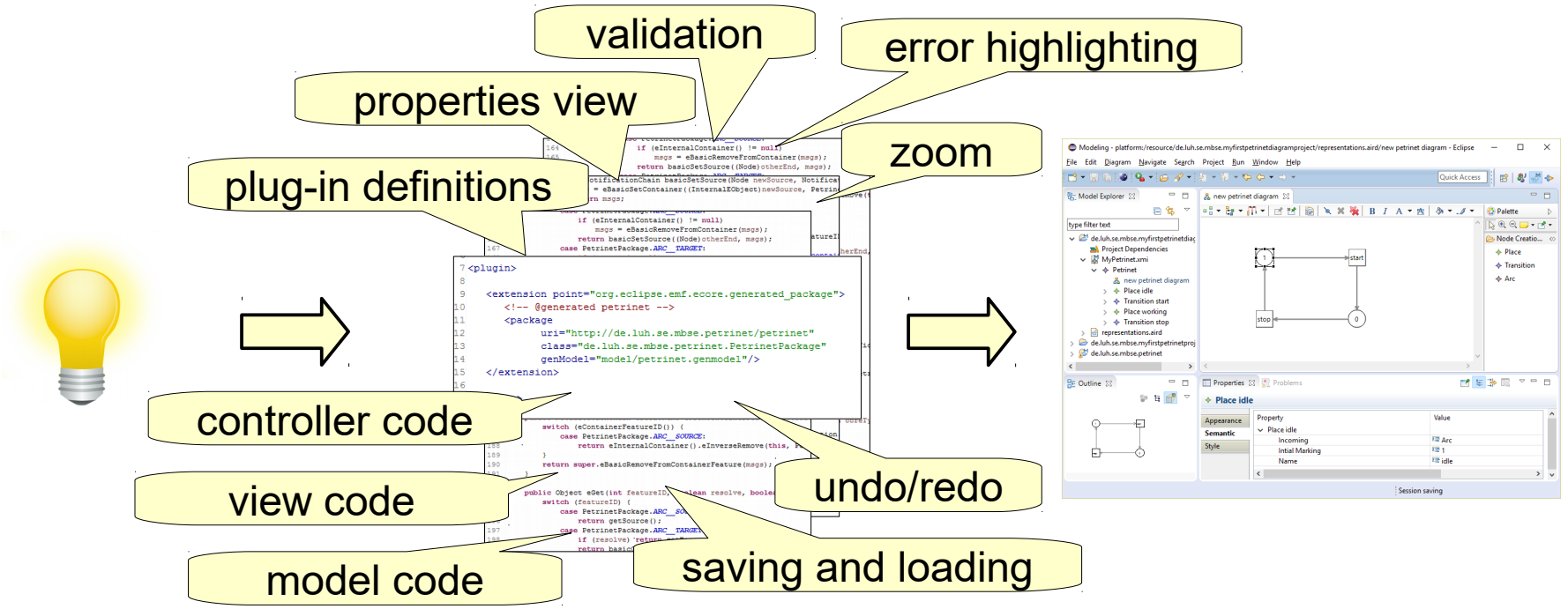


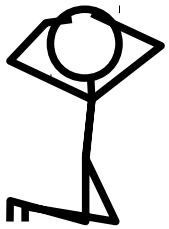


Build a Petri Net Modeling Tool

in the last lecture...

- Manual implementation: A lot of repetitive or generic code

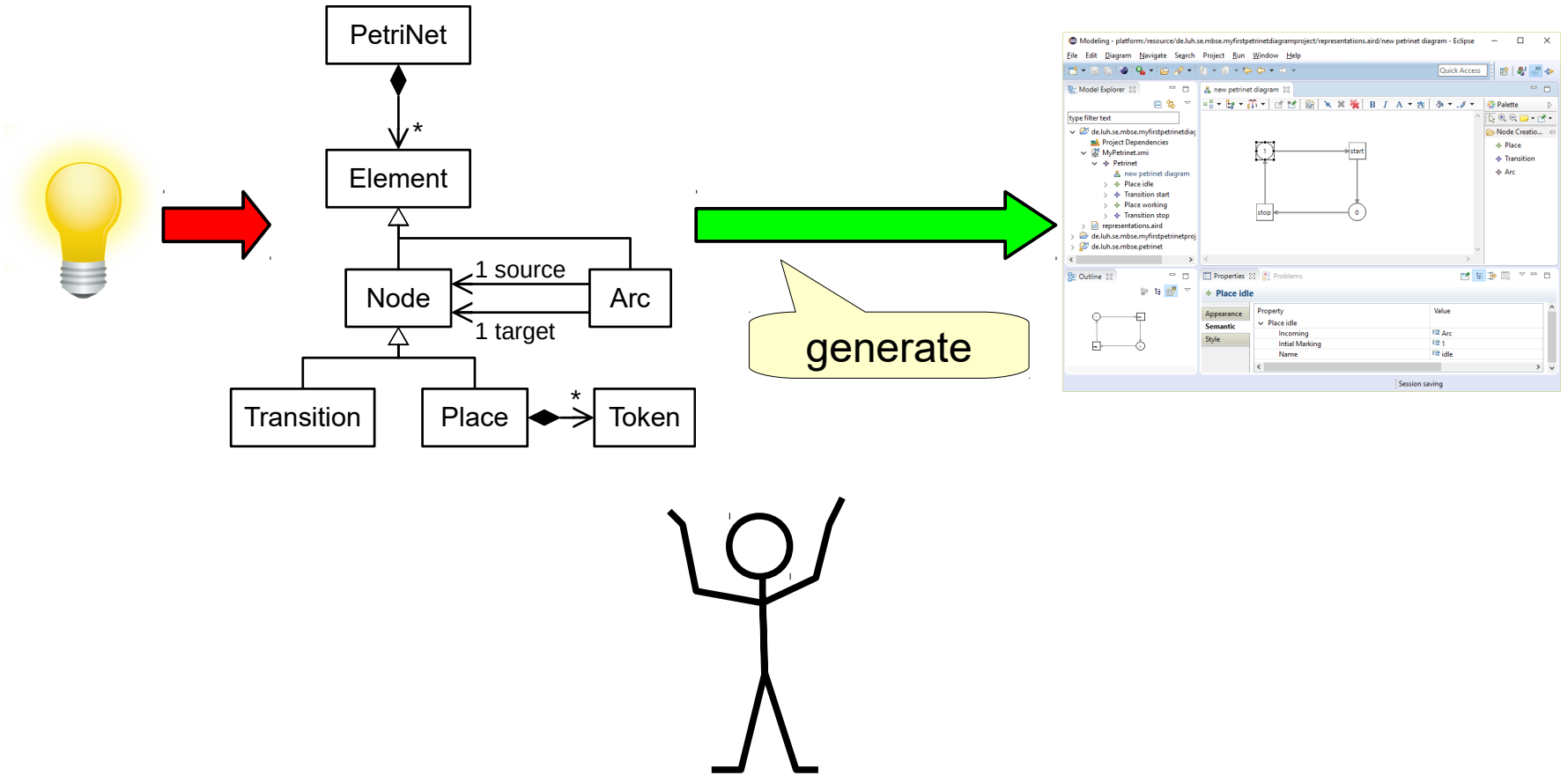




Build a Petri Net Modeling Tool

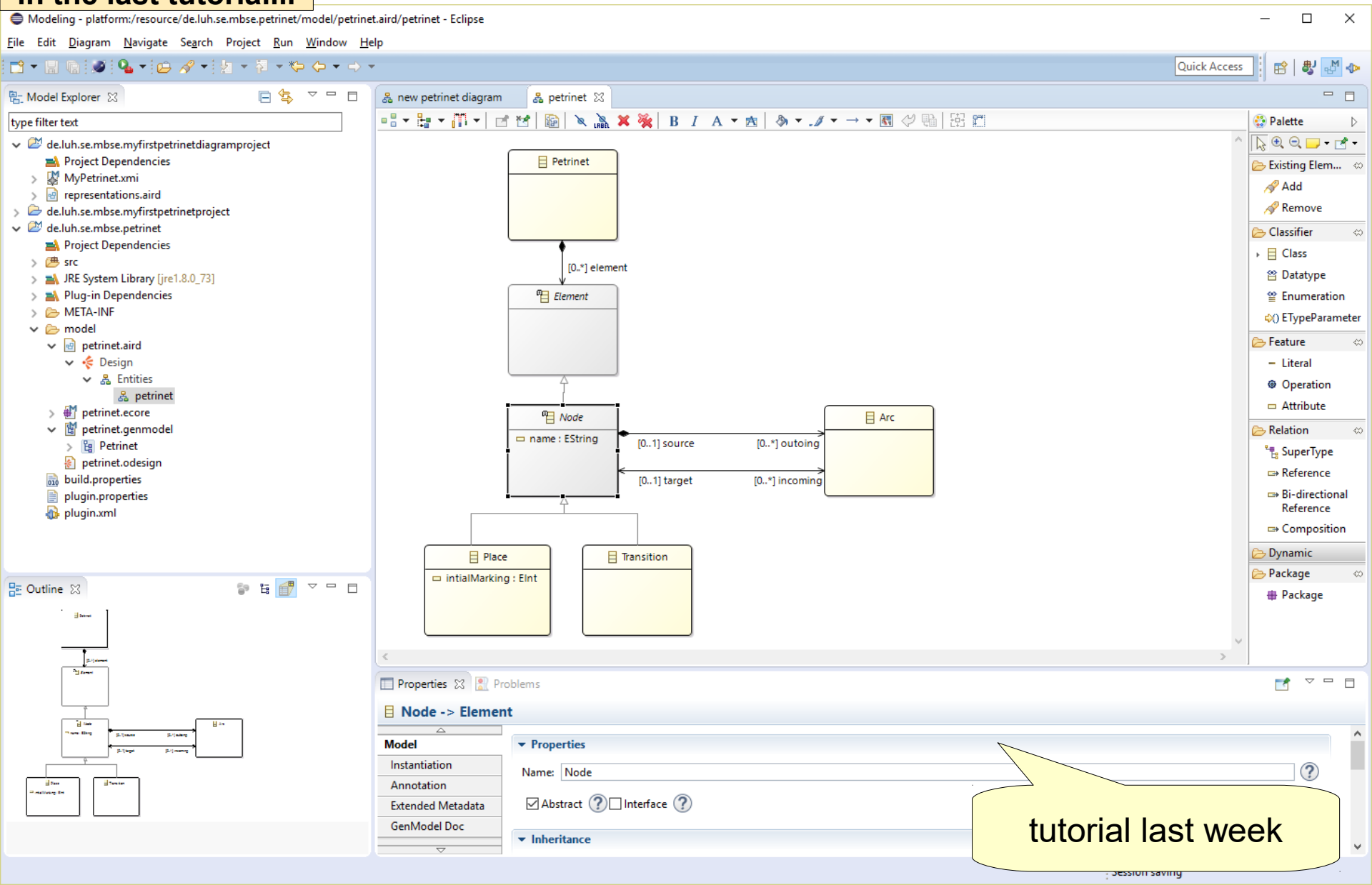
in the last lecture...

- Model-based approach for building modeling tools: Provide only a few conceptual models and generate tool automatically



Eclipse Modeling Framework (Modeling a Petri Net Metamodel)

in the last tutorial...



SOFTWARE ENGINEERING

2.5. Eclipse Modeling Framework (EMF) and Ecore

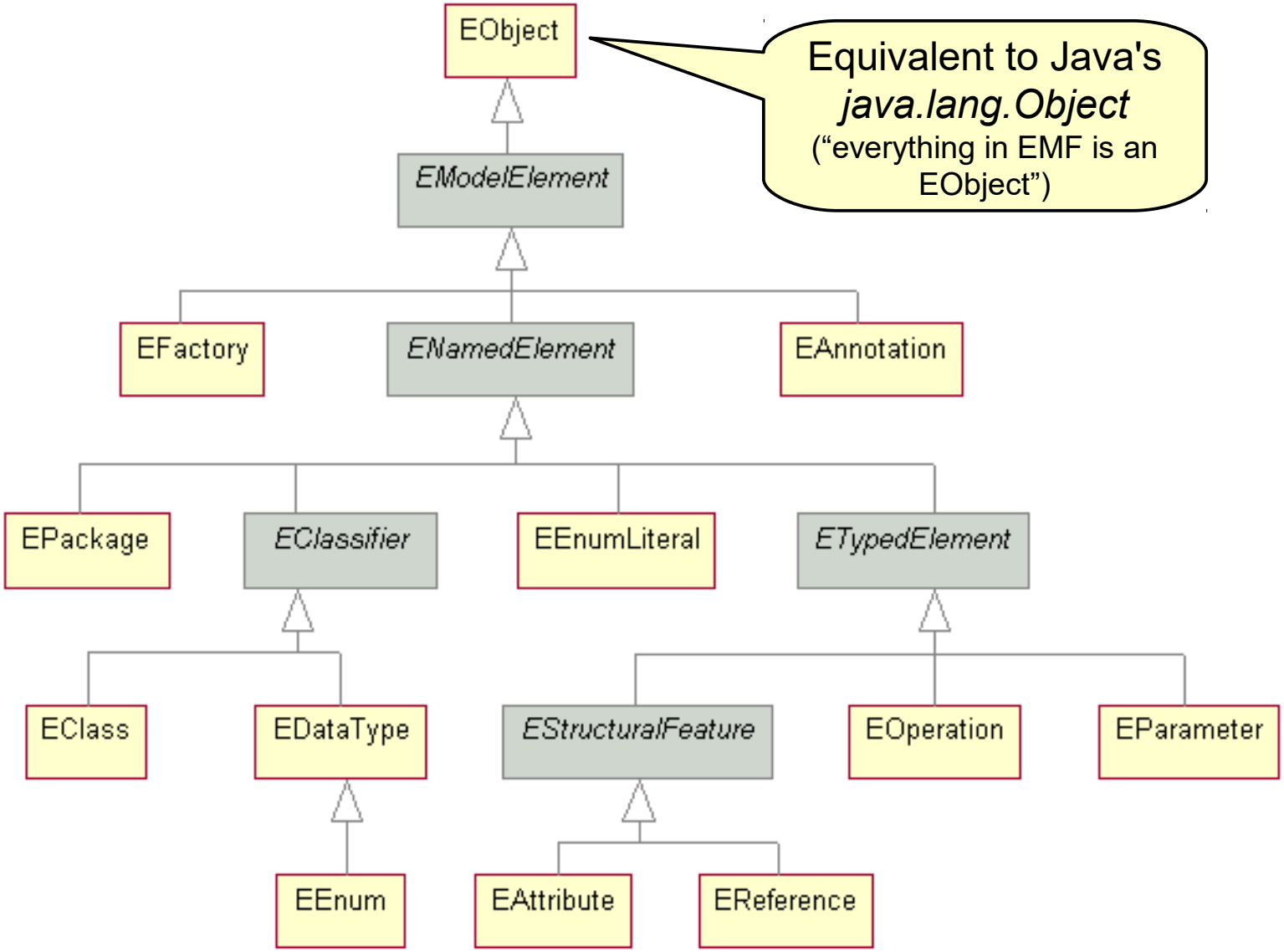
Leibniz Universität Hannover

Ecore vs EMOF

- MOF defines a meta-metamodel used to define metamodels in other OMG specifications (UML, IDL, CWM)
  - (see http://www.omg.org/mof/)
- MOF consists of two parts
  - "Essential MOF" (EMOF) is a core part of the MOF
  - the full MOF meta-metamodel is called "Complete MOF" (CMOF), which includes EMOF, i.e., is an extension of EMOF
  - EMOF and CMOF are *derived from the UML2 metamodel*
- Ecore is the meta-metamodel used by EMF
  - it is very similar to EMOF, some aspects are even simplified
  - it has some technology-specific specialties
  - we are going to look at Ecore mainly and later compare to MOF

A Close Look at the Ecore Meta-Metamodel

- ECore:

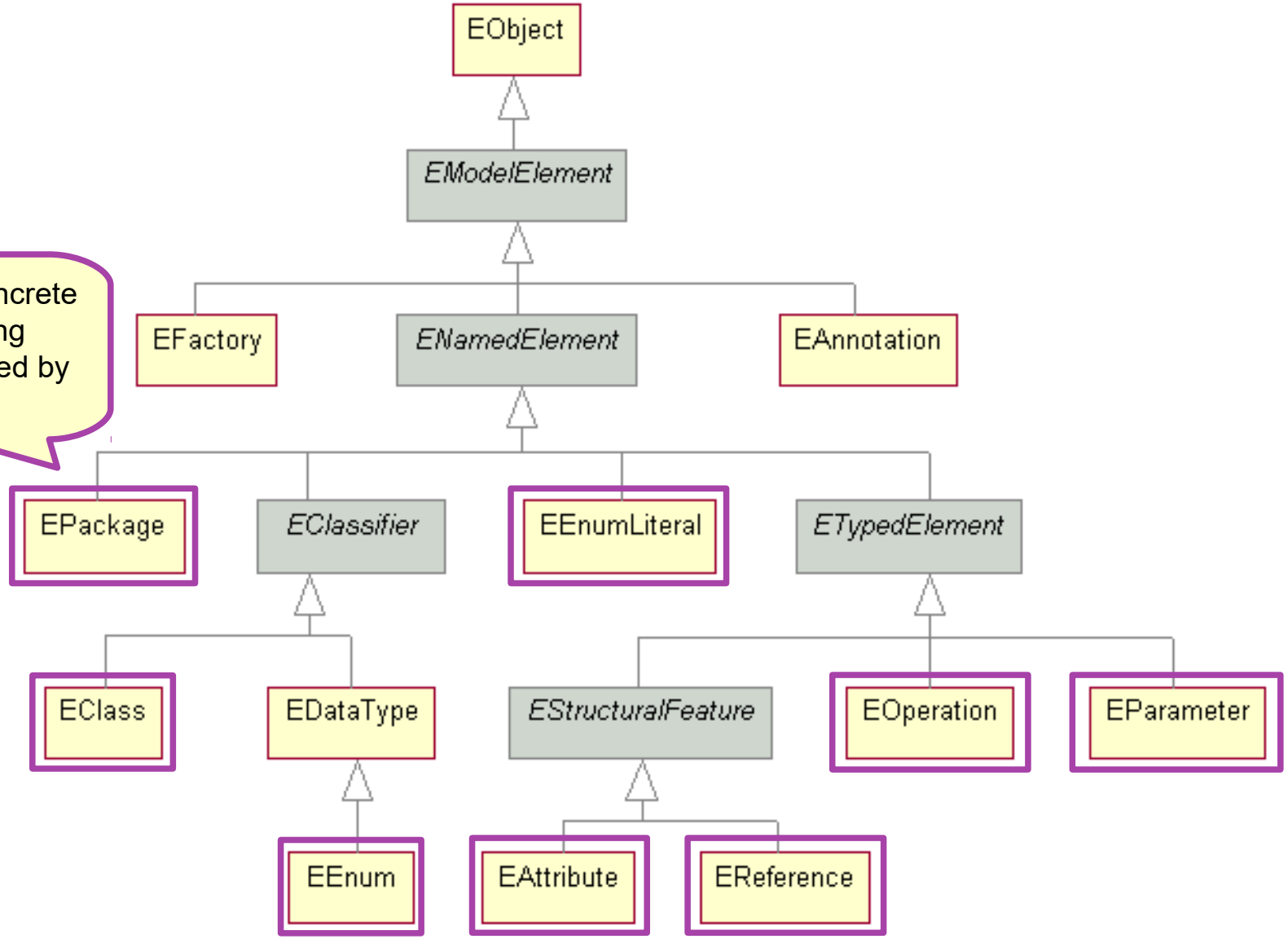


see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

A Close Look at the Ecore Meta-Metamodel

- ECore:

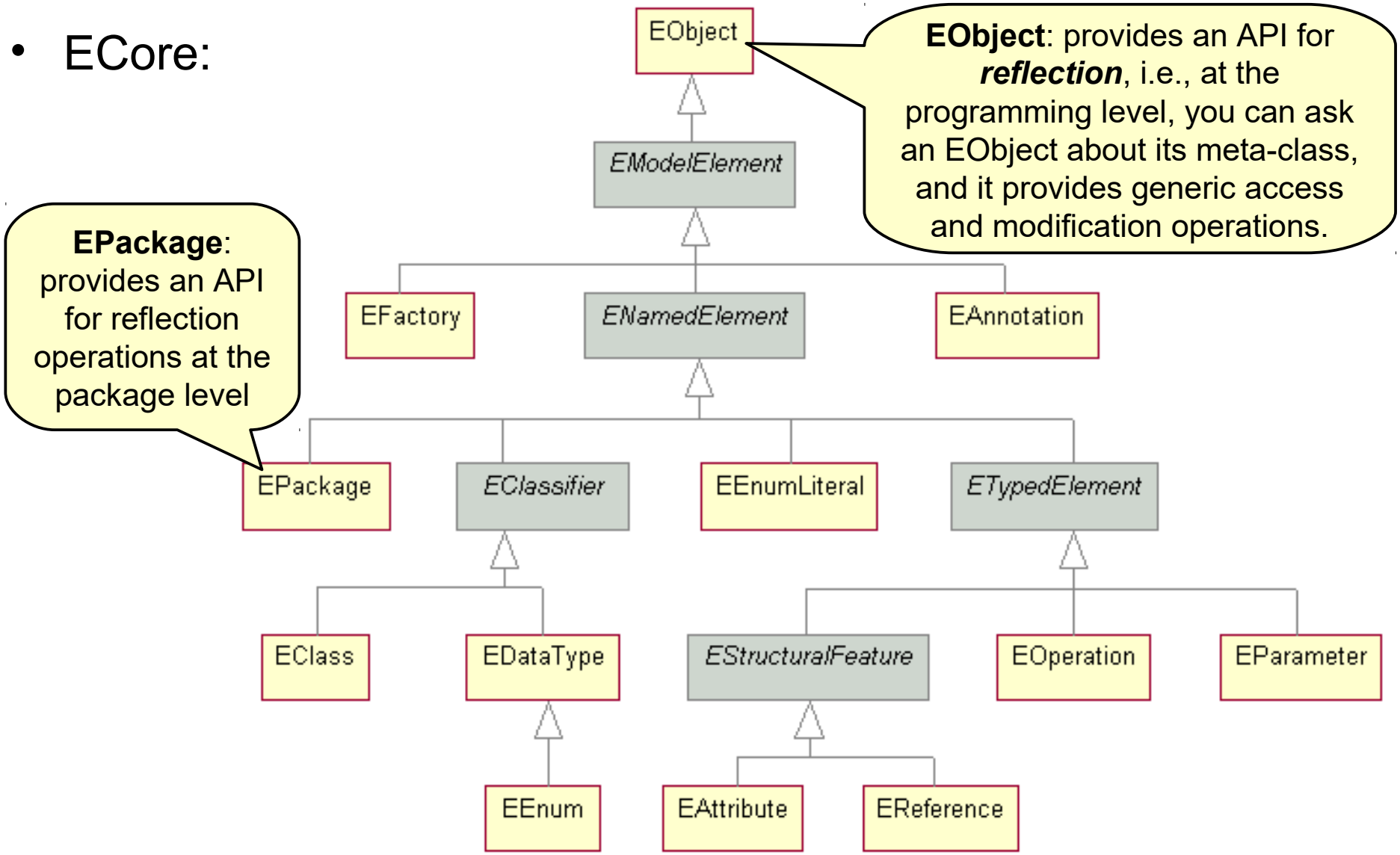
these are the concrete metamodeling concepts provided by ECore



see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

A Close Look at the Ecore Meta-Metamodel

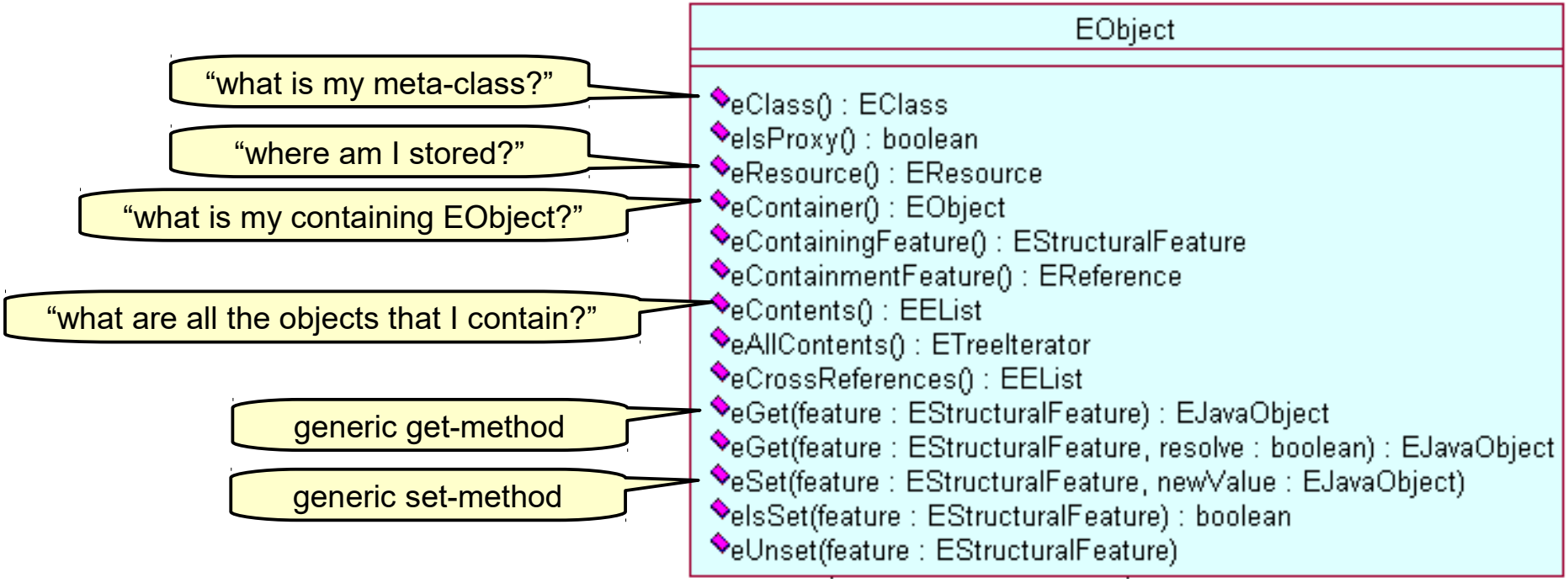
- ECore:



see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

A Close Look at the Ecore Meta-Metamodel

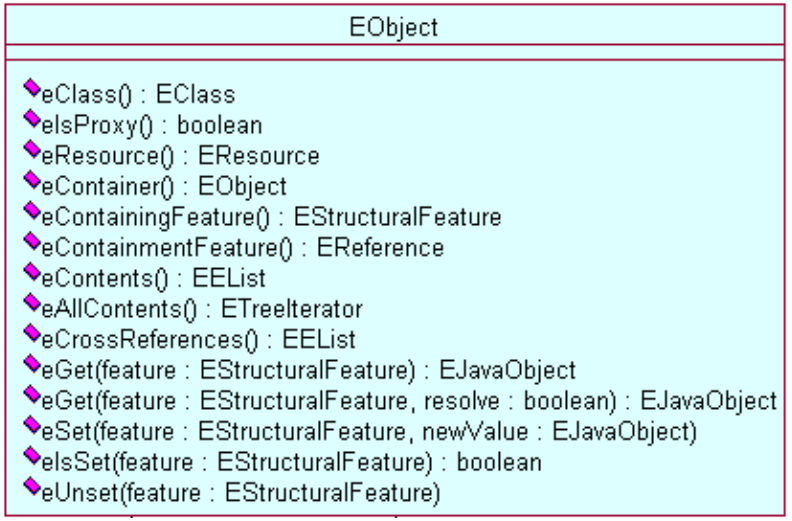
- The EObject API:



see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

A Close Look at the Ecore Meta-Metamodel

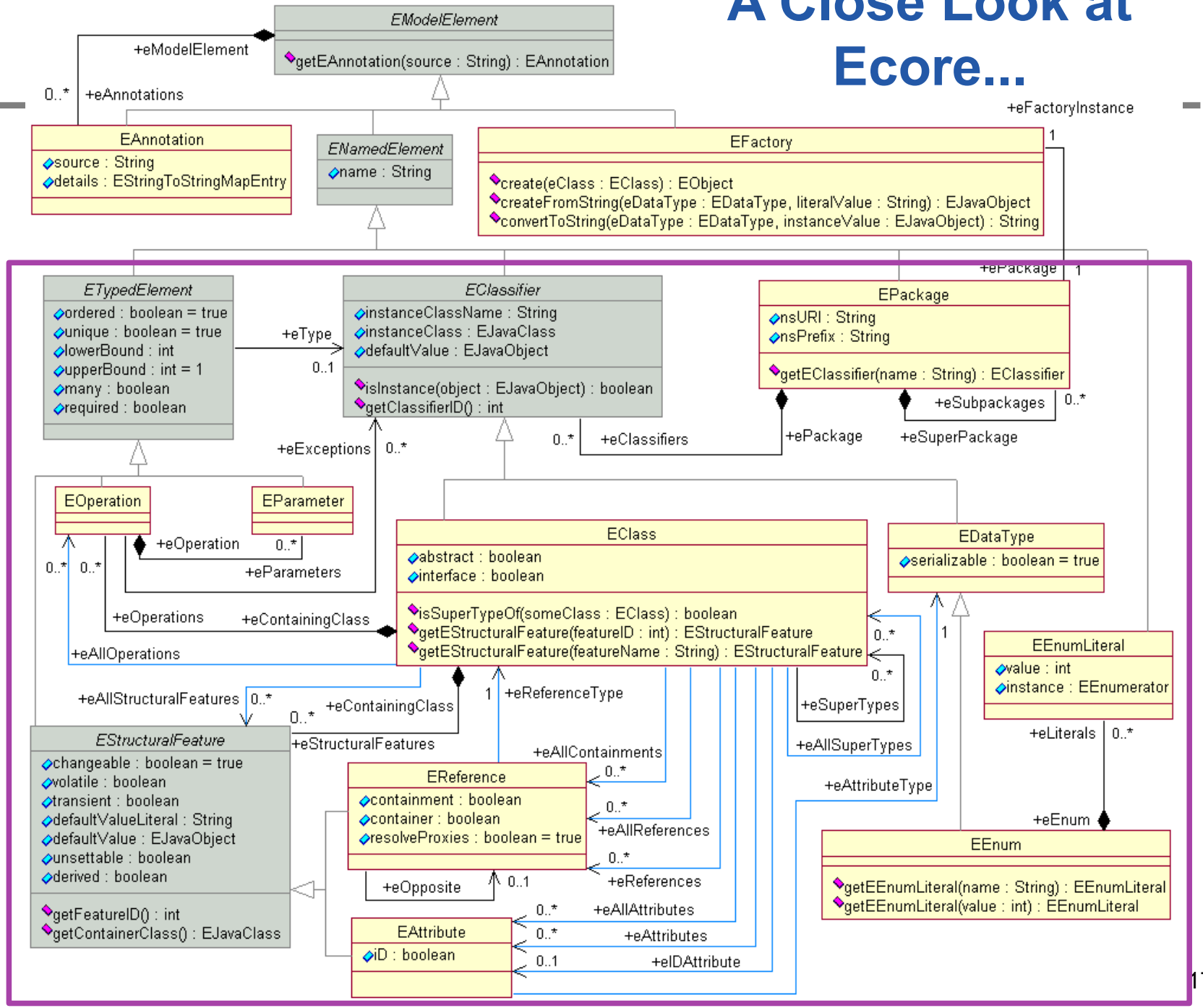
- The EObject API:



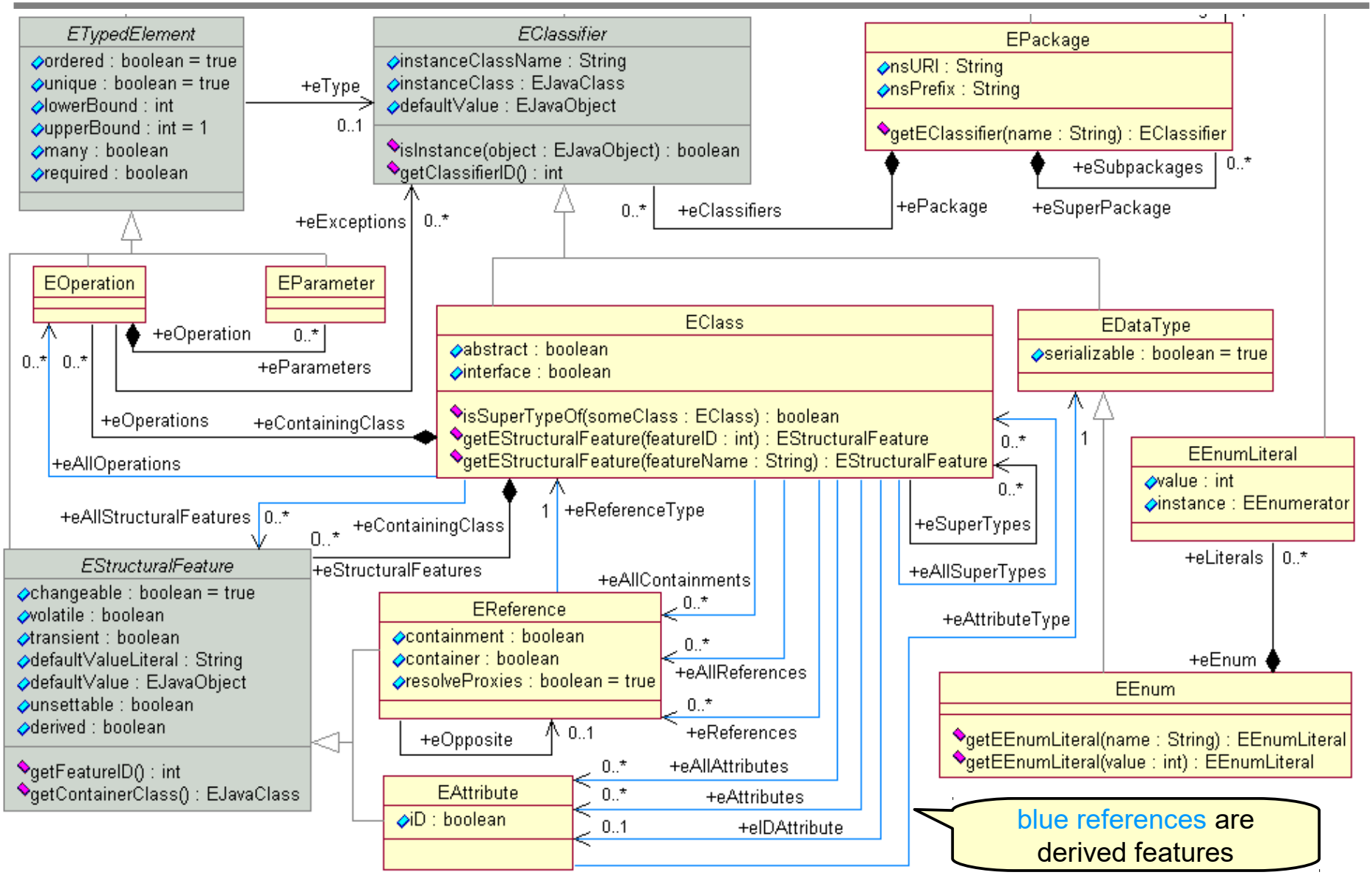
- **Example**:
  reflective `getName()` function:

```
public static String getName(EObject eObject){
        EClass eClass = eObject.eClass();
        EStructuralFeature nameFeature
                = eClass.getEStructuralFeature("name");
        if (nameFeature != null
                && "EString".equals(nameFeature.getEType().getName())){
                return (String) eObject.eGet(nameFeature);
        }else{
                return "";
        }
}
```

A Close Look at Ecore...

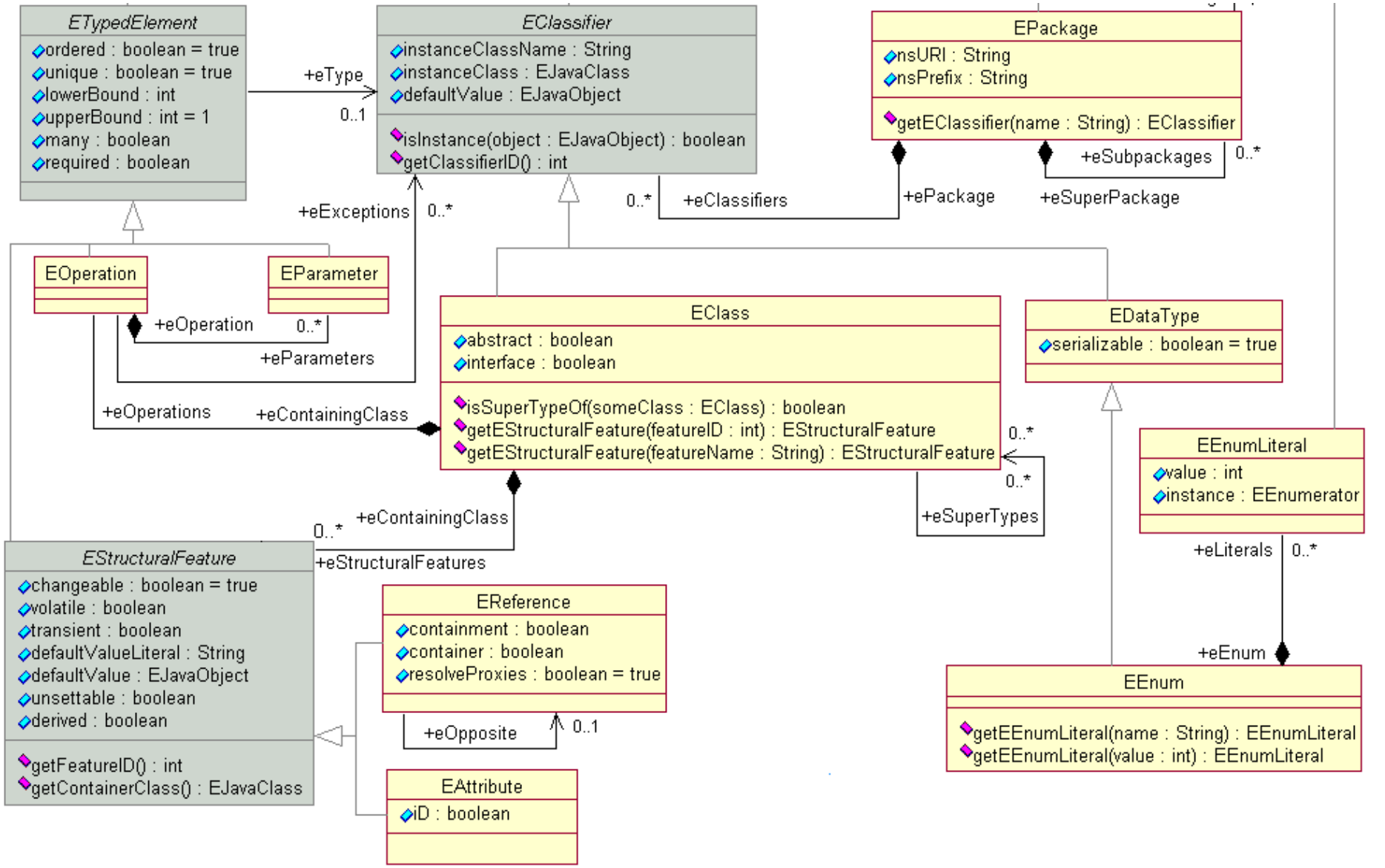


A Close Look at the Ecore Meta-Metamodel



see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

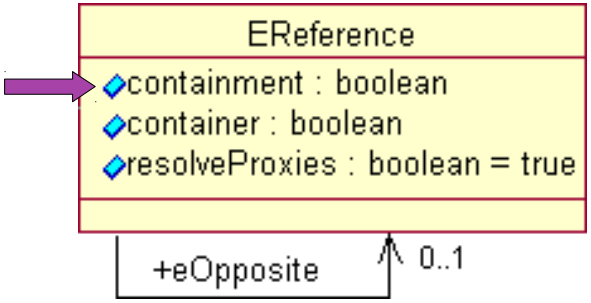
A Close Look at the Ecore Meta-Metamodel



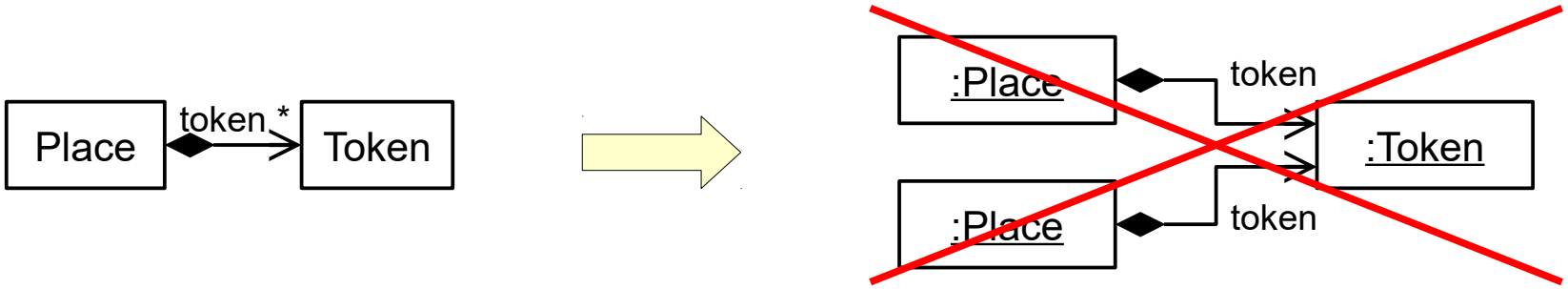
see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

Specialties of EReferences

- **Containment**:
  - An object can only be be contained in *at most one* other object at a time
    - it can be target of at most one containment link at a time

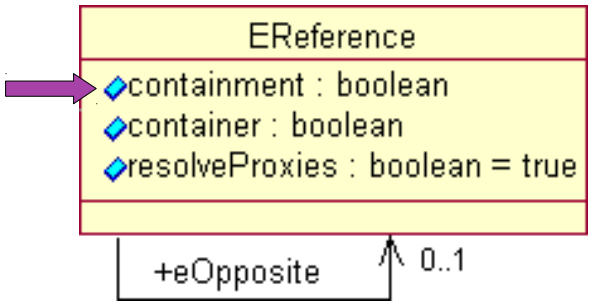


- Example:

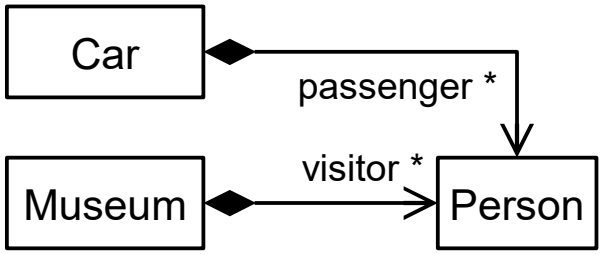


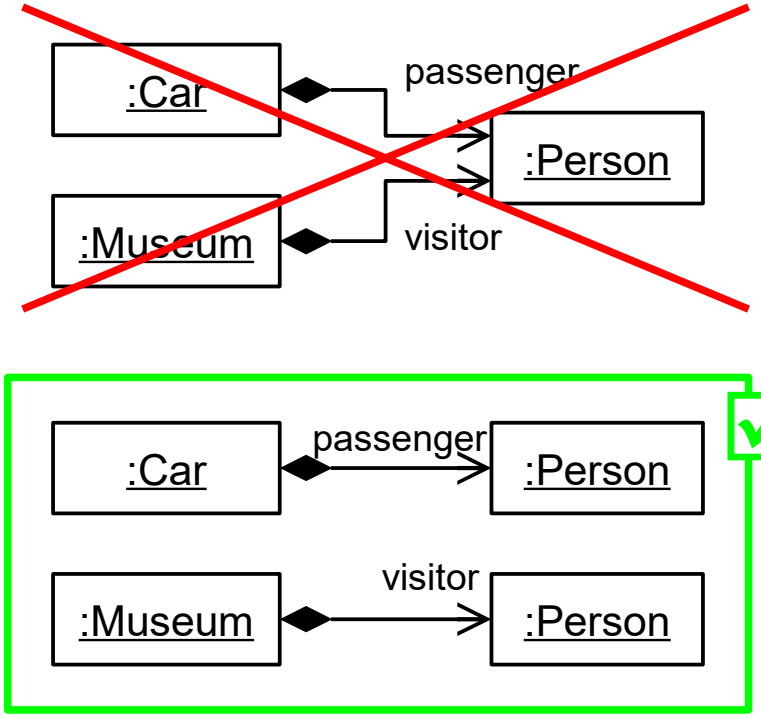
Specialties of EReferences

- **Containment**:
  - An object can only be be contained in *at most one* other object at a time
    - it can be target of at most one containment link at a time



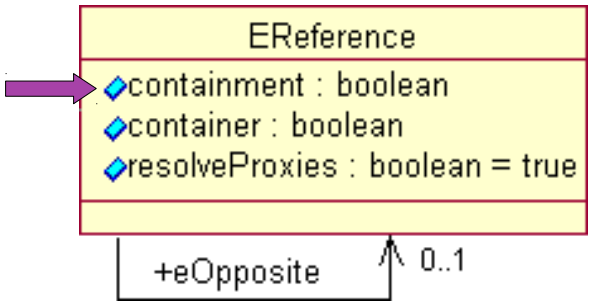
- Example:



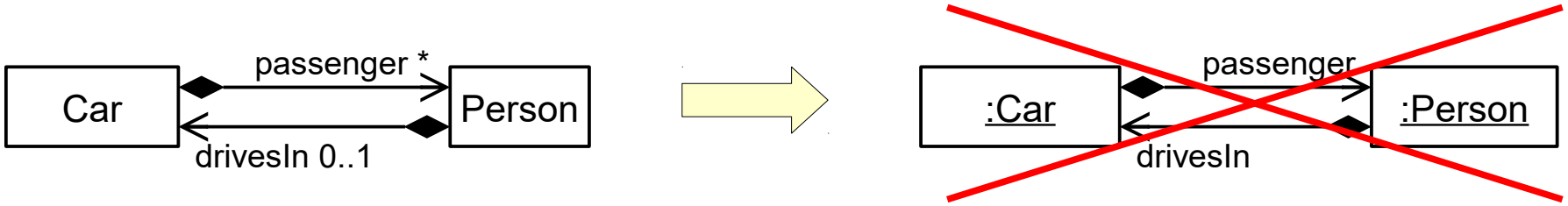


Specialties of EReferences

- **Containment**:
  - Containment links must not form a cycle
  - An object cannot contain any object that it is (transitively) contained in

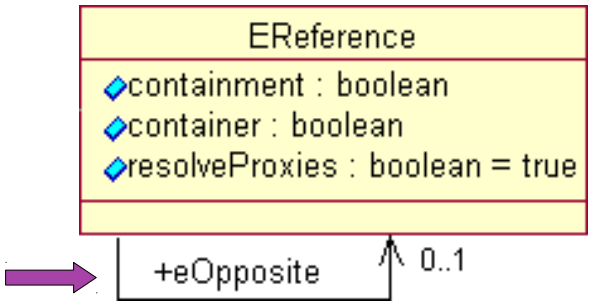


- Example:

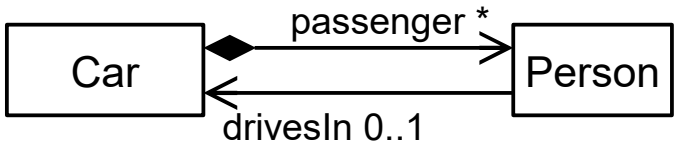


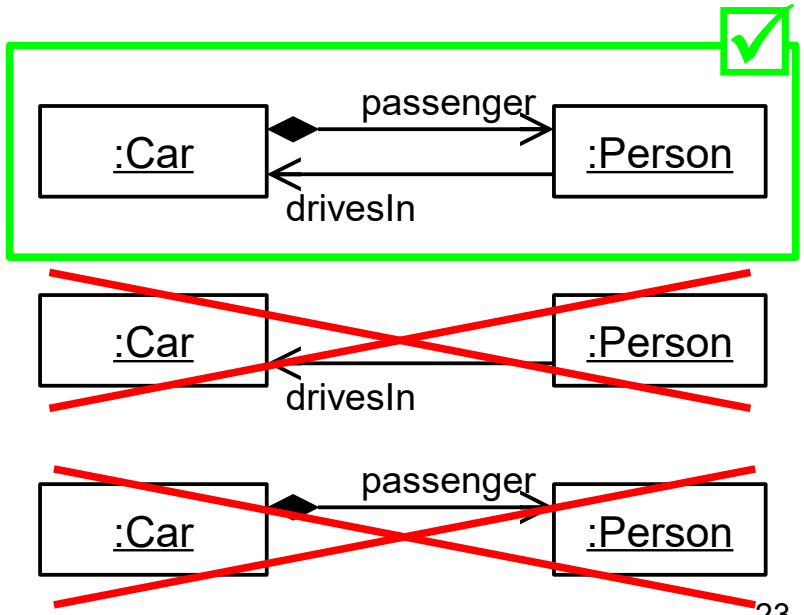
Specialties of EReferences

- **eOpposite**:
  - Two EReferences in opposite directions between two EClasses can be “opposites”
  - Thereby forming a bidirectional relationship
  - At the object level, there must be bidirectional links

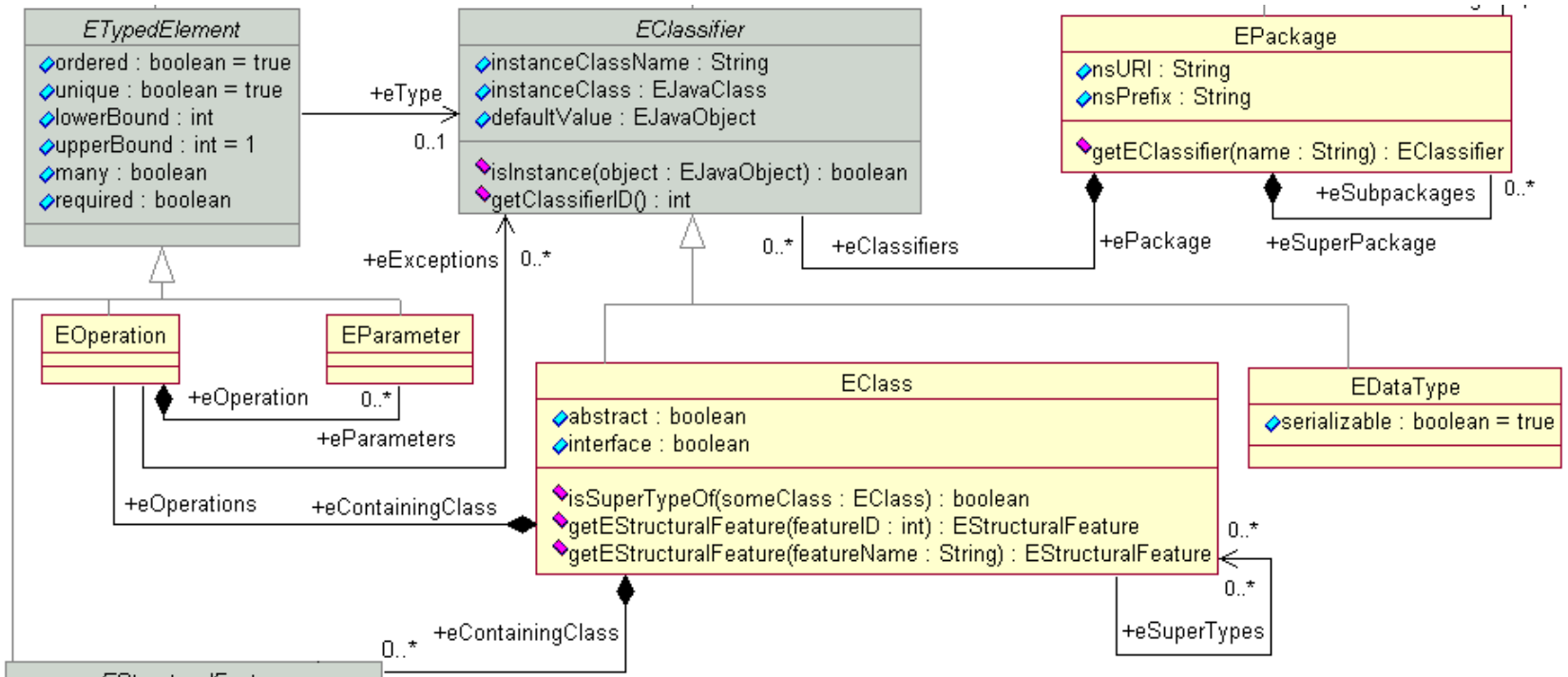


- Example:

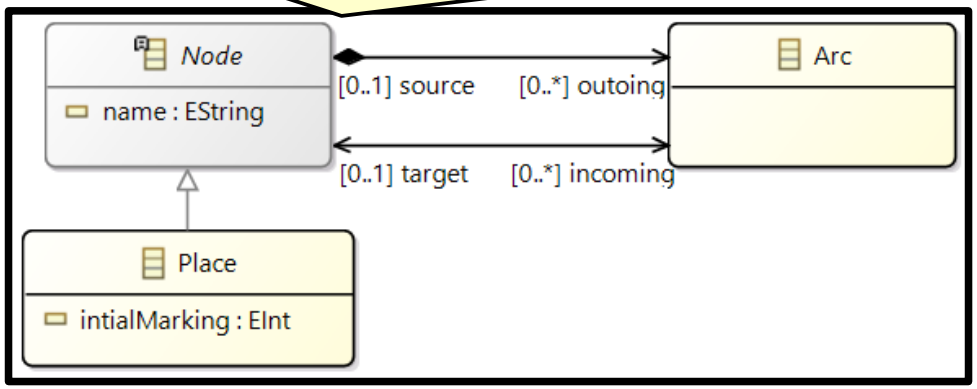




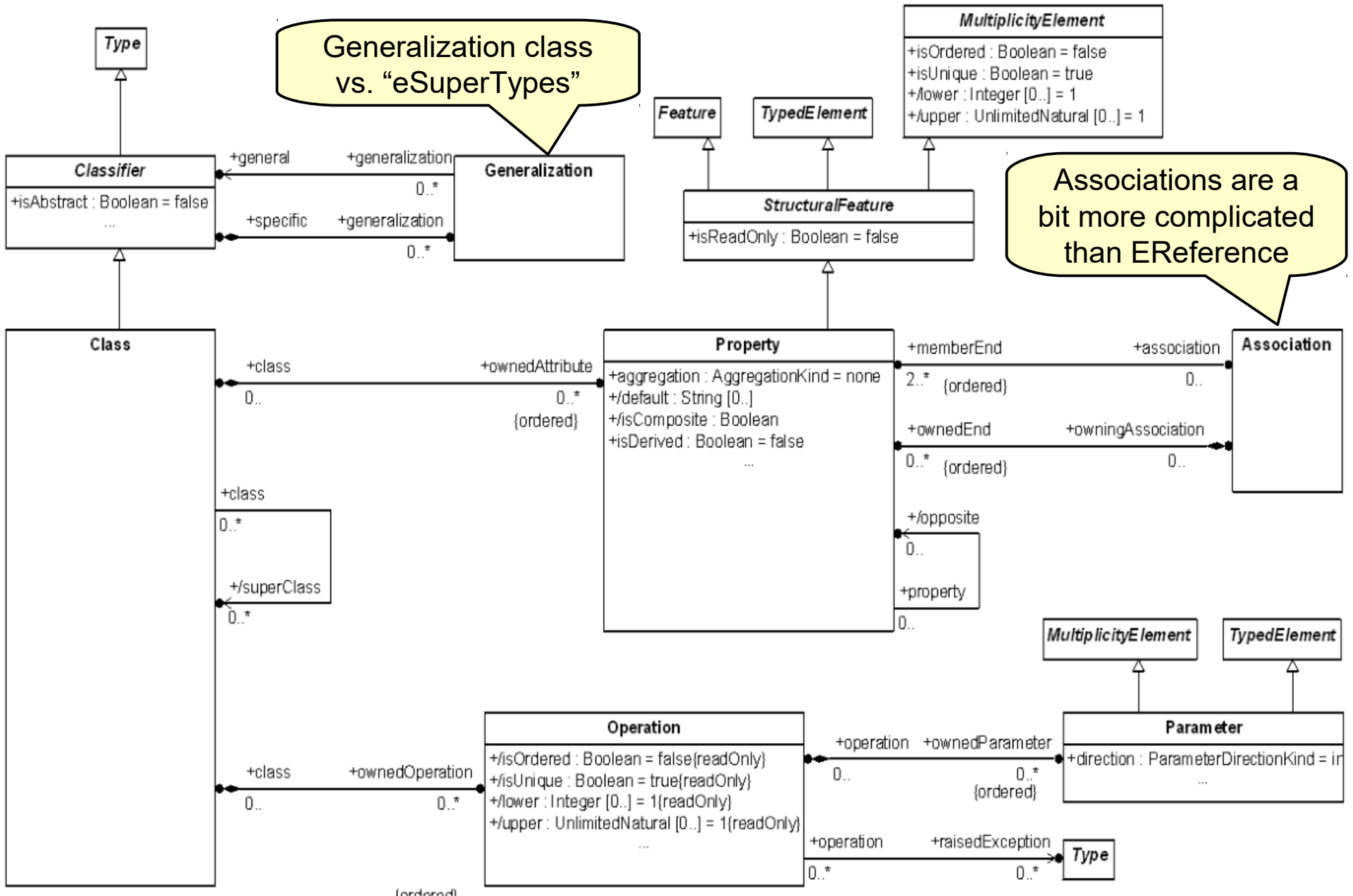
A Close Look at the Ecore Meta-Metamodel



Task: draw an object diagram for the abstract syntax of this Ecore diagram based on the given Ecore metamodel

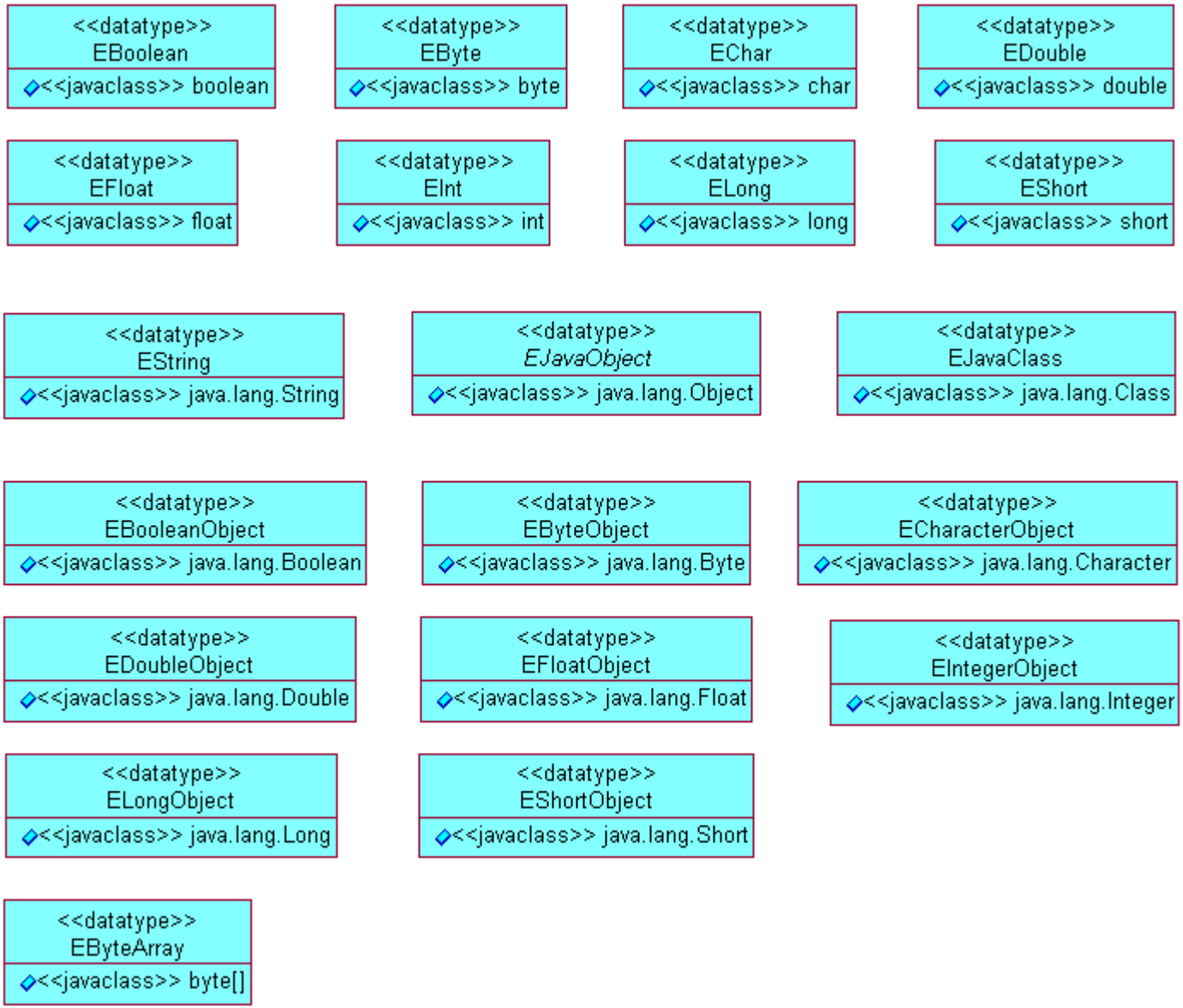


EMOF Classes – Difference to Ecore



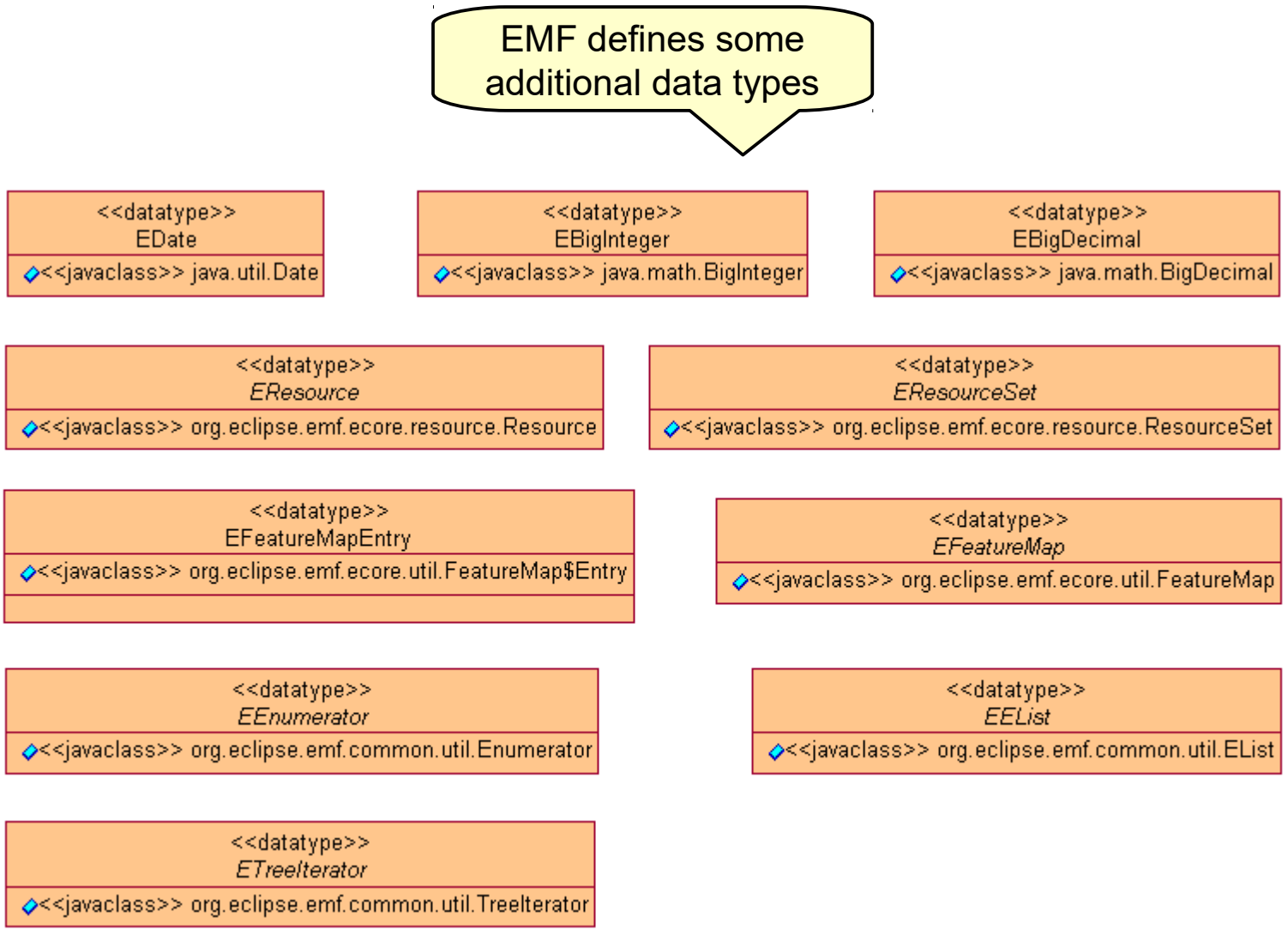
Data Types in EMF

EMF uses some special **data types** that are mapped to standard Java data types



see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

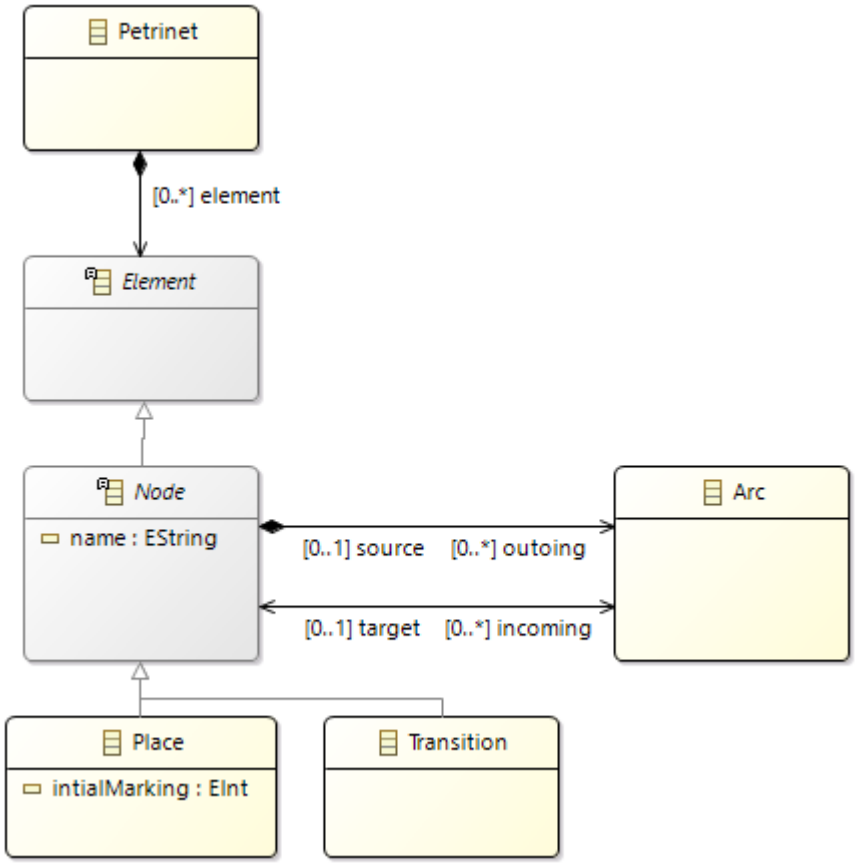
Data Types in EMF

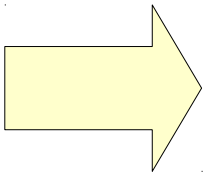


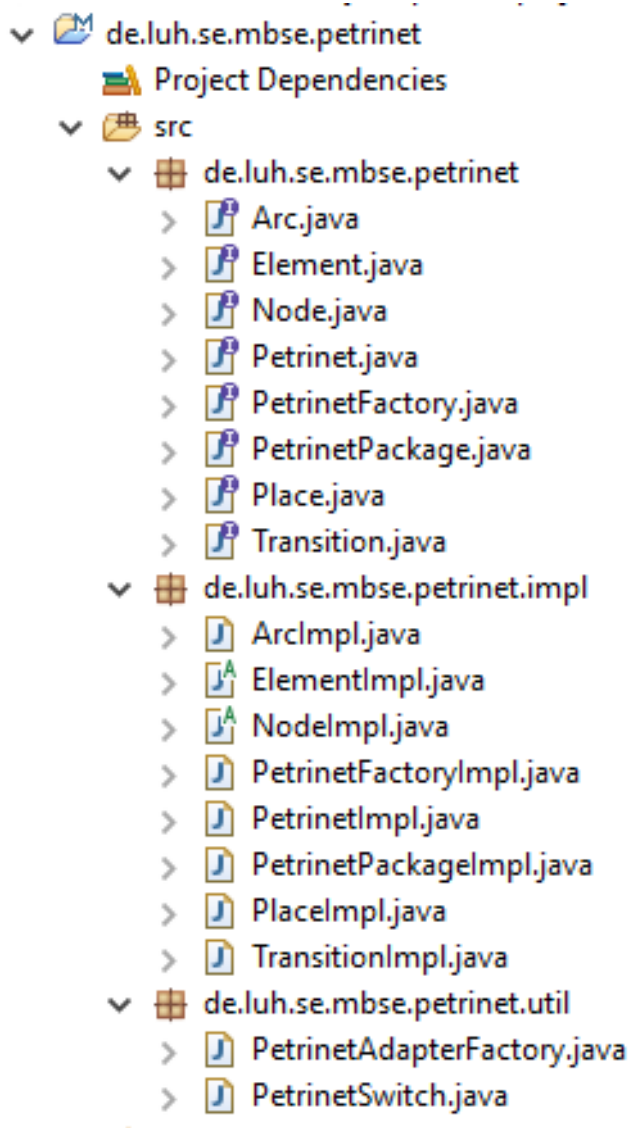
see http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

EMF Code Generation

- EMF supports Java code generation from Ecore models

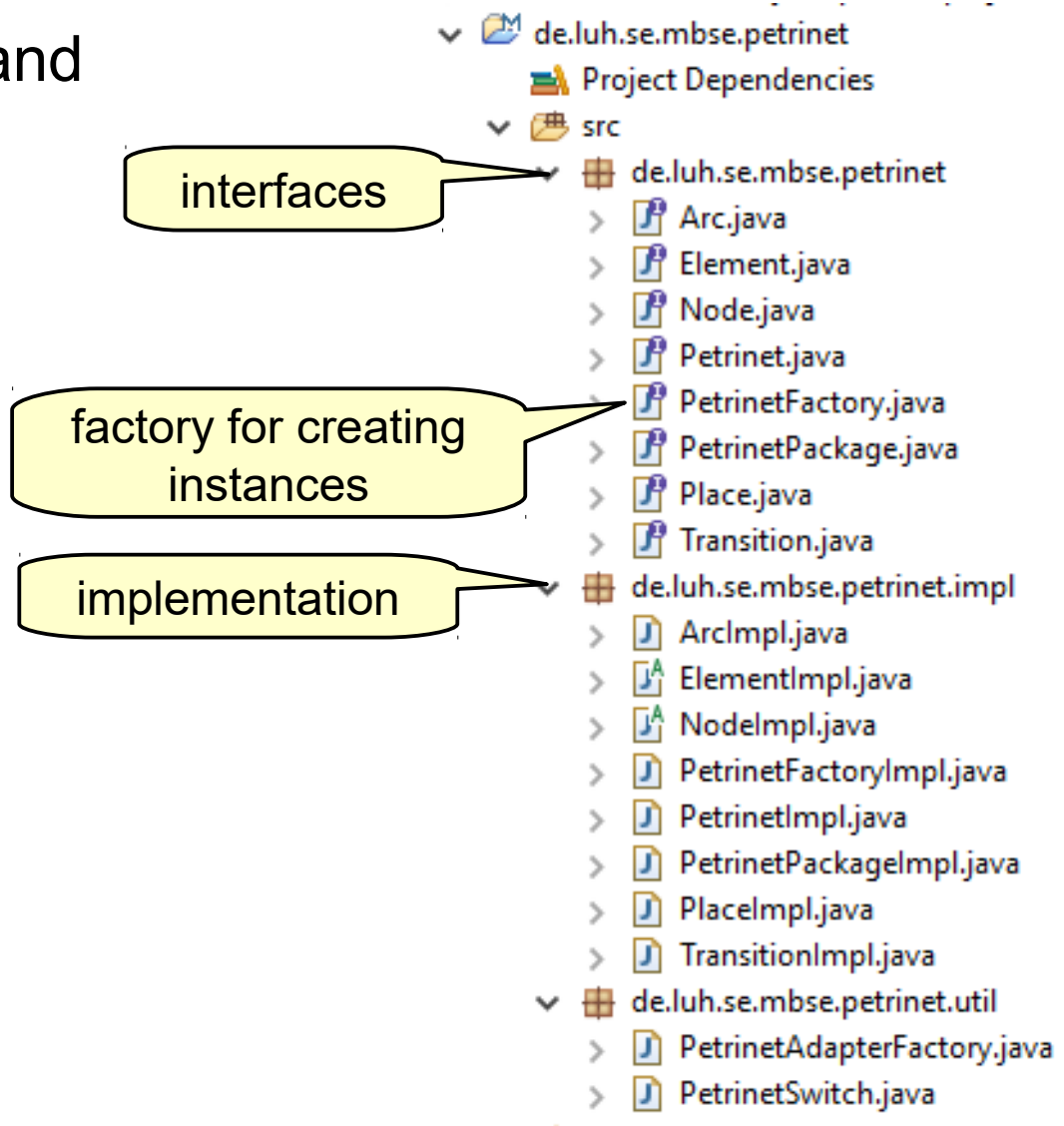






EMF Code Generation

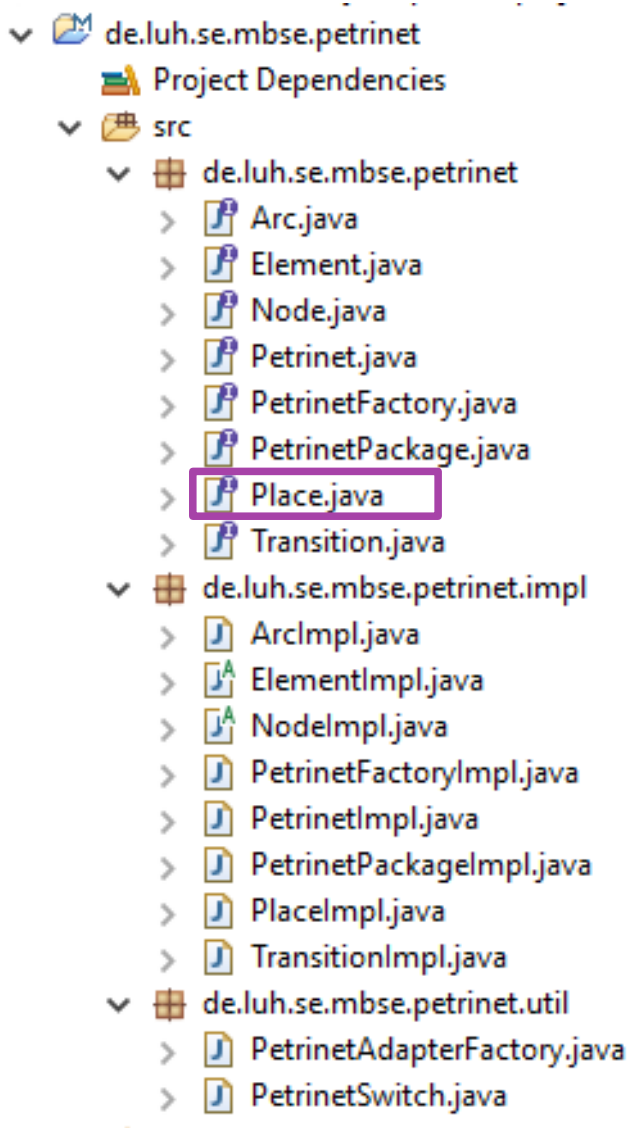
- Separation of Interfaces and Implementation



EMF Code Generation

- Separation of Interfaces and Implementation

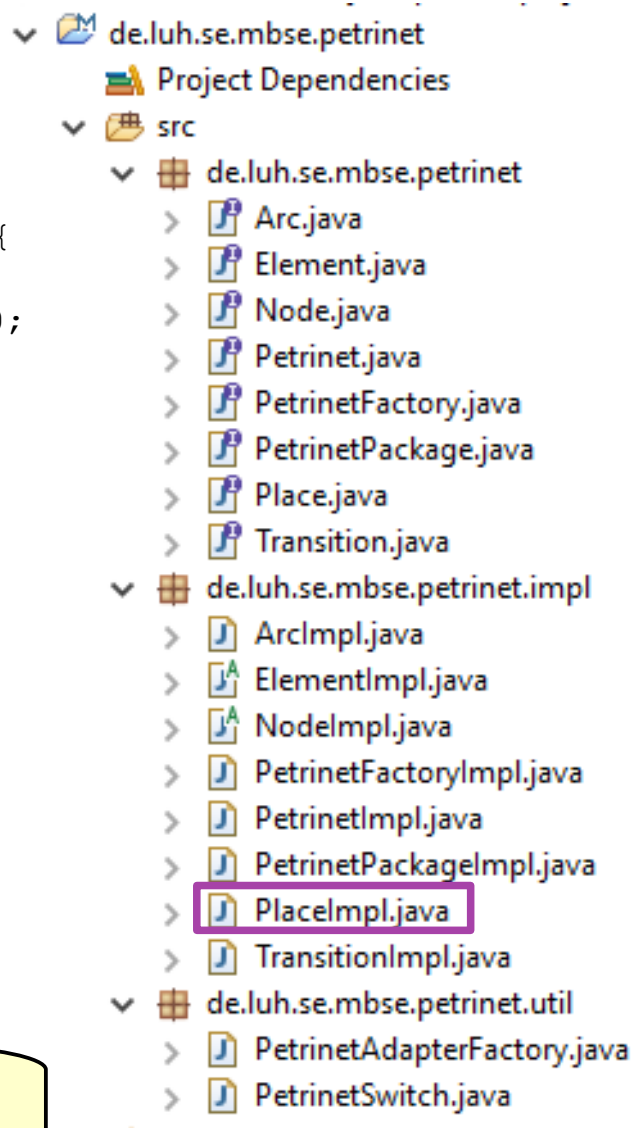
```
public interface Place extends Node {
    /**
     * Returns the value of the
     * '<em><b>Initial Marking</b></em>' attribute.
     * @generated
     */
    int getInitialMarking();

    /**
     * Sets the value of the
     * '<em>Initial Marking</em>' attribute.
     * @generated
     */
    void setInitialMarking(int value);

    ...

} // Place
```



EMF Code Generation

- Separation of Interfaces and Implementation

```
public class PlaceImpl extends NodeImpl implements Place {

    protected static final int INTIAL_MARKING_EDEFAULT = 0;
    protected int intialMarking = INTIAL_MARKING_EDEFAULT;

    public int getIntialMarking() {
          return intialMarking;
    }

    public void setIntialMarking(int newIntialMarking) {
          int oldIntialMarking = intialMarking;
          intialMarking = newIntialMarking;
          if (eNotificationRequired())
                eNotify(new ENotificationImpl(this,
                Notification.SET,
                PetrinetPackage.PLACE__INTIAL_MARKING,
                oldIntialMarking,
                intialMarking));
    }

    ...

} //PlaceImpl
```

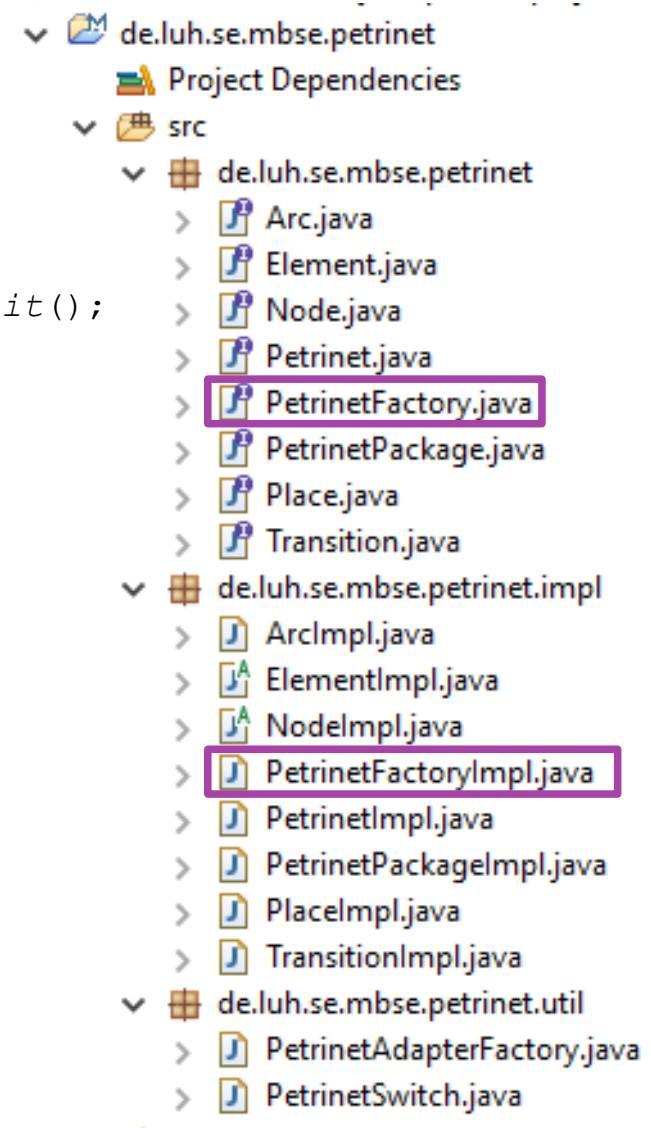


Notification mechanism (observer pattern) built in

EMF Code Generation

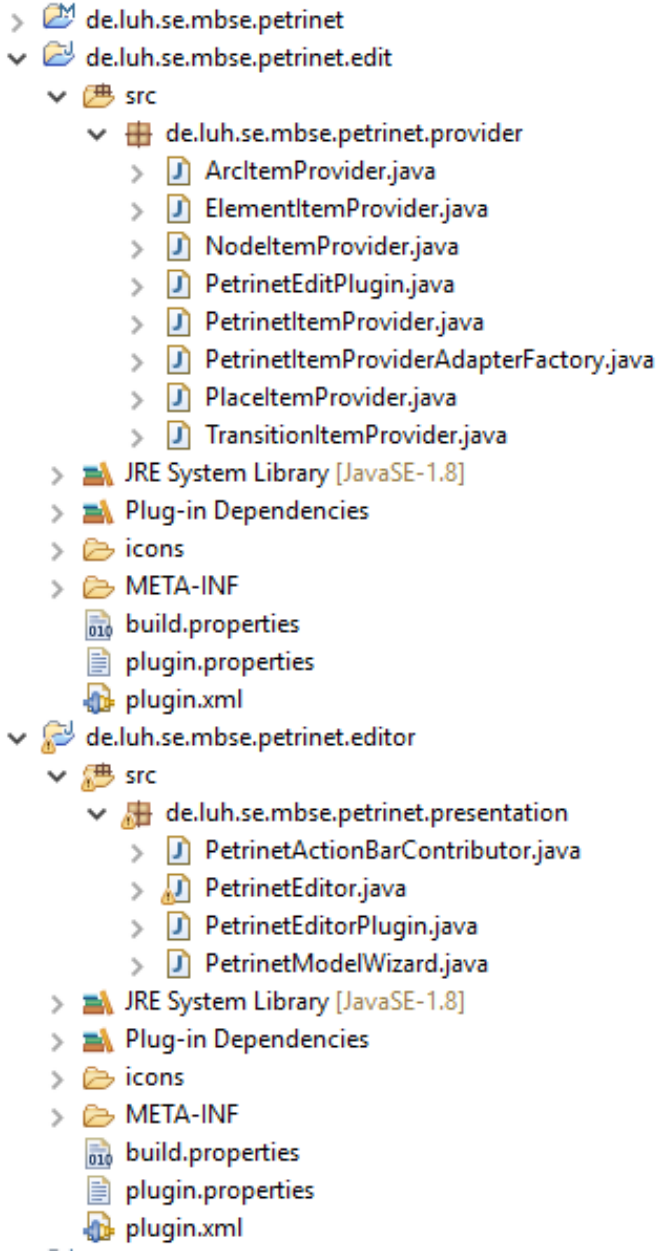
- Factory interface and implementation

```
public interface PetrinetFactory extends EFactory {

    PetrinetFactory eINSTANCE
    = de.luh.se.mbse.petrinet.impl.PetrinetFactoryImpl.init();

    Petrinet createPetrinet();
    ...
} //PetrinetFactory


public class PetrinetFactoryImpl extends EFactoryImpl
        implements PetrinetFactory {

    public static PetrinetFactory init() {
        return new PetrinetFactoryImpl();
    }

    public Petrinet createPetrinet() {
        PetrinetImpl petrinet = new PetrinetImpl();
        return petrinet;
    }
    ...

} //PetrinetFactoryImpl
```



Edit and Editor Code

- EMF also supports the generation of code for *building editors and views* within Eclipse:
  - *.edit* plug-in:
    - Content and label provider classes, for table and tree views
    - property source support for property pages
    - command framework for undo/redo
  - .editor plug-in:
    - tree editor
    - model creation wizards

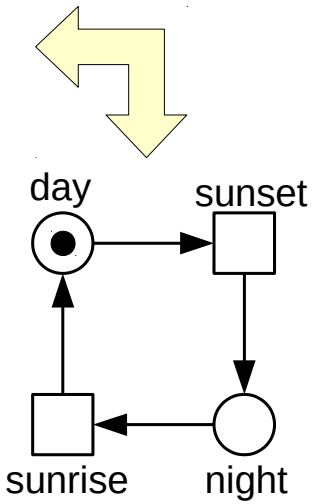


Model Persistence – XMI Serialization

- EMF models can be serialized in the **XML Metadata Interchange (XMI)** format
  - also an OMG standard: http://www.omg.org/spec/XMI/
  - XML format for exchanging models with metamodels conforming to MOF

Model Persistence – XMI Serialization (Example)

```
<?xml version="1.0" encoding="UTF-8"?>
<petrinet:Petrinet xmi:version="2.0"
        xmlns:xmi="http://www.omg.org/XMI"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:petrinet="http://www.example.org/petrinet"
        xsi:schemaLocation="http://www.example.org/petrinet
                ../de.luh.se.mbse.petrinet/model/petrinet.ecore">
  <element xsi:type="petrinet:Place" name="day"
        incoming="//@element.3/@outoing.0" initialMarking="1">
    <outoing target="//@element.1"/>
  </element>
  <element xsi:type="petrinet:Transition" name="sunset"
        incoming="//@element.0/@outoing.0">
    <outoing target="//@element.2"/>
  </element>
  <element xsi:type="petrinet:Place" name="night"
        incoming="//@element.1/@outoing.0">
    <outoing target="//@element.3"/>
  </element>
  <element xsi:type="petrinet:Transition" name="sunrise"
        incoming="//@element.2/@outoing.0">
    <outoing target="//@element.0"/>
  </element>
</petrinet:Petrinet>
```



Dynamic EMF

- The generated code allows us to create instances of our Ecore models
  - for example:

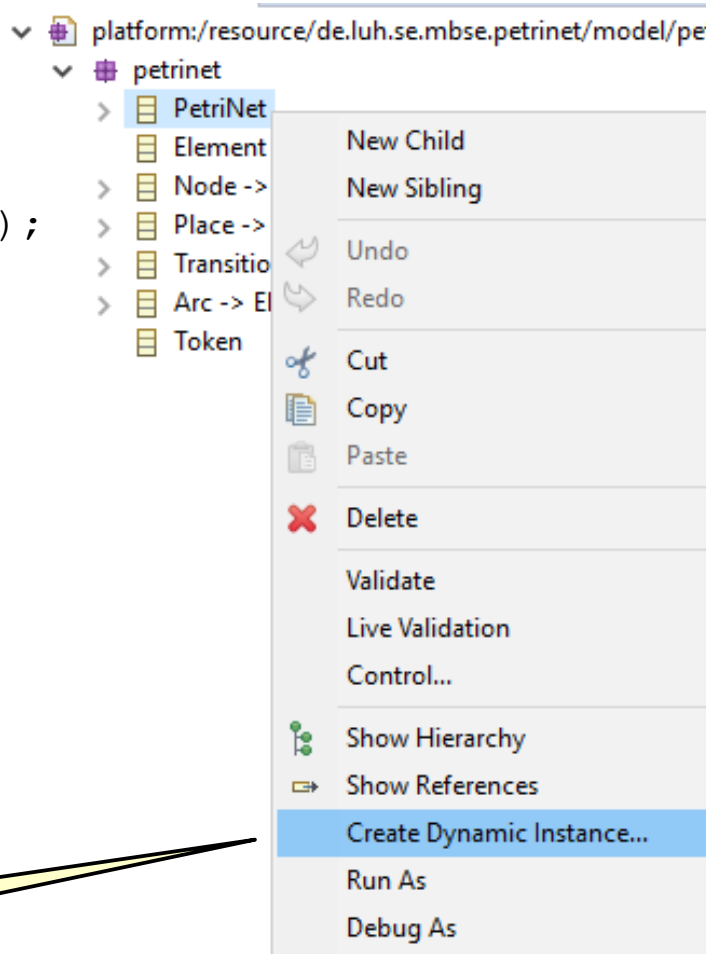
```
public Petrinet createPetrinet() {
    PetrinetImpl petrinet = new PetrinetImpl();
    return petrinet;
}
```

Dynamic EMF

- The generated code allows us to create instances of our Ecore models
  - for example:

```
public Petrinet createPetrinet() {
    PetrinetImpl petrinet = new PetrinetImpl();
    return petrinet;
}
```

- But EMF also supports working with **dynamic instances**
- EMF **interprets** the metamodels to allow us to work on instance models without code generation:



creating a dynamic object via the UI

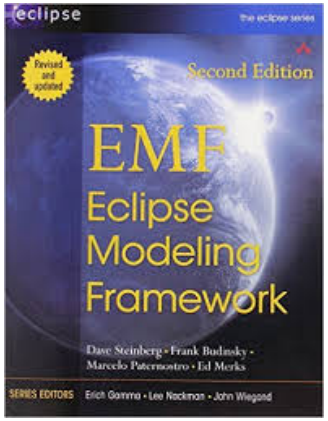
Dynamic EMF

- creating models and instances dynamically via the API:

```
// create package
EPackage petrinetPackage = EcoreFactory.eINSTANCE.createEPackage();

//create Place class
EClass placeClass = EcoreFactory.eINSTANCE.createEClass();
placeClass.setName("Place");
petrinetPackage.getEClassifiers().add(placeClass);

//create initialMarkings attribute and add it to the Place class
EAttribute initialMarkingsAttribute
                = EcoreFactory.eINSTANCE.createEAttribute();
initialMarkingsAttribute.setName("initialMarkings");
initialMarkingsAttribute.setEType(EcorePackage.eINSTANCE.getEInt());
placeClass.getEAttributes().add(initialMarkingsAttribute);

//create dynamic instance of Place class
EFactory petrinetFactory = petrinetPackage.getEFactoryInstance();
EObject place =  petrinetFactory.create(placeClass);
place.eSet(initialMarkingsAttribute, 2);
```

EMF Resources

- D. Steinberg, F. Budinski, M. Paternostro, E.Merks: EMF: Eclipse Modeling Framework, Addison Wesley, 2nd edition, 2008.



- Online resources
  - http://www.vogella.com/tutorials/EclipseEMF/article.html
  - http://eclipsesource.com/blogs/tutorials/emf-tutorial/
  - There are many more online resources...

Model-Based Software Engineering

Lecture 03 – Metamodeling cont., OCL

Prof. Dr. Joel Greenyer

April 18, 2016