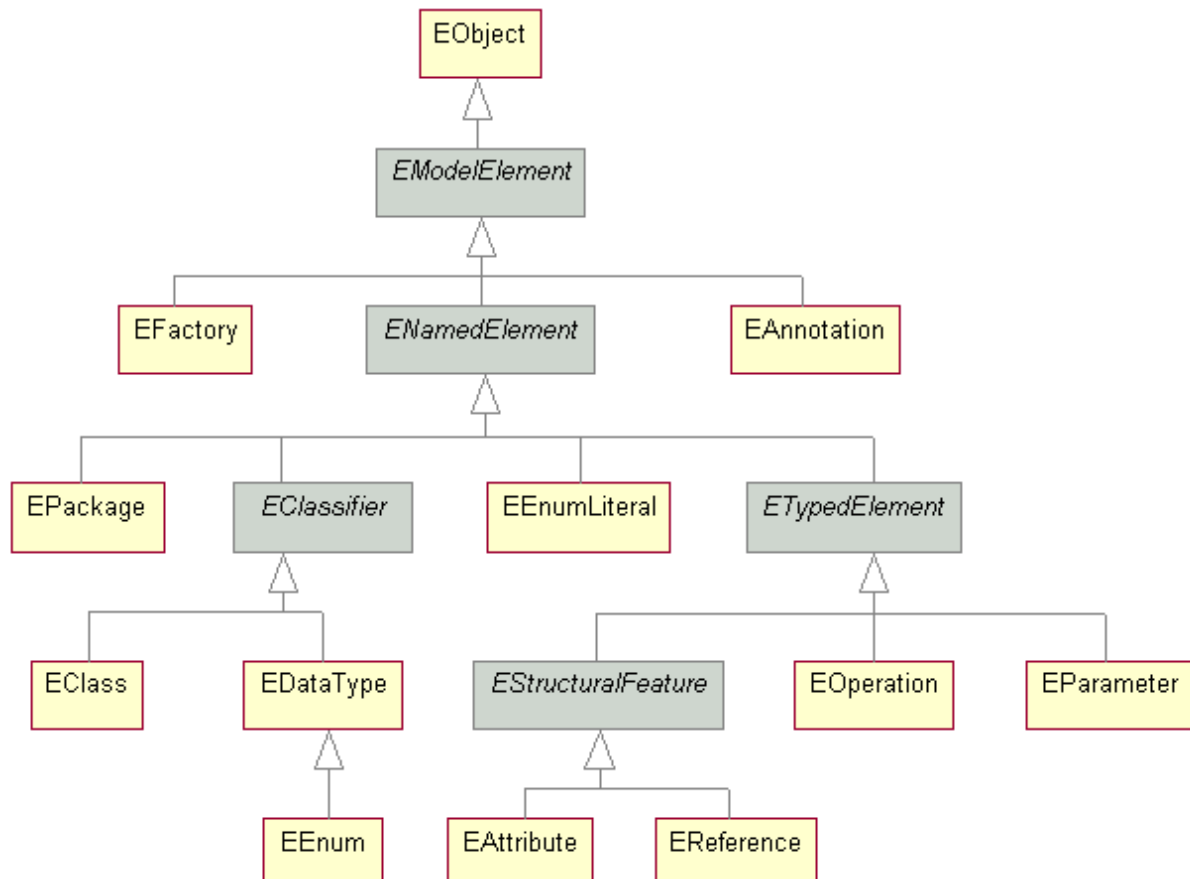


TP1 : PREMIERES MANIPULATIONS DE MODELES

Ce premier TP a pour but de voir en pratique comment créer un méta-modèle et éditer des modèles conformes à ce méta-modèle. Pour cela, nous nous appuyons sur le framework EMF et son méta-méta-modèle Ecore.

Le méta-modèle ECore est défini ci-dessus :



Il s'agit concrètement dans cette partie de réaliser le diagramme de classe Client/Compte défini ci-dessus en s'appuyant sur ECore. Les méta-éléments à instancier pour définir ce modèle sont :

- **EClass** pour définir une classe
- **EAttribute** pour définir un attribut d'une classe (un attribut est d'un type primitif entier, booléen, chaîne...)
- **EReference** pour définir une association entre 2 classes
- **EOperation** pour définir une opération d'une classe
- **EParameter** pour définir un paramètre d'opération

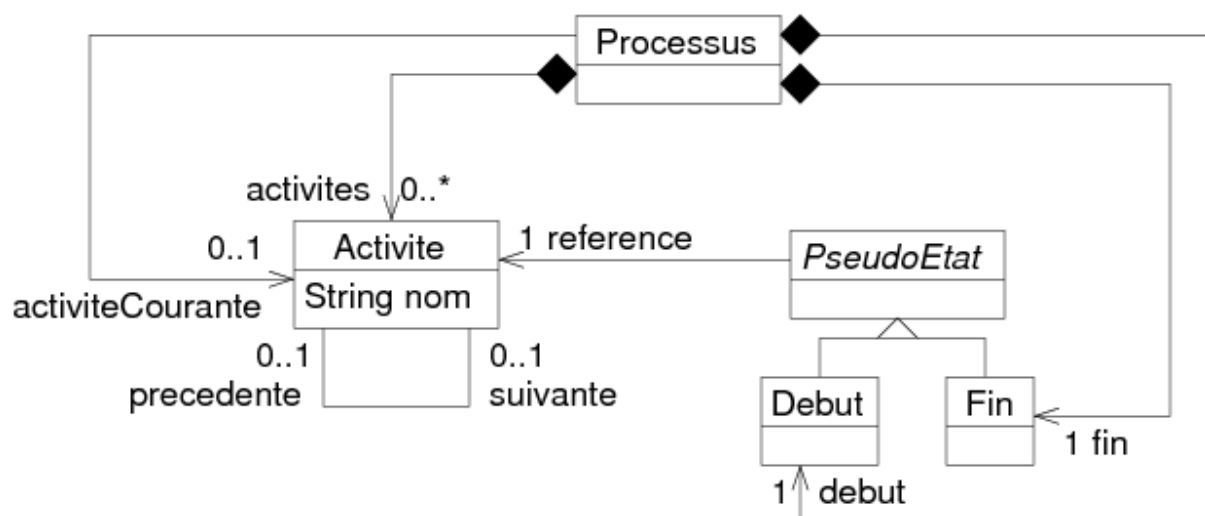
LANGAGE DE DESCRIPTION DE PROCESSUS

Nous allons dans ce TP implémenter un DSML (Domain Specific Modeling Language) pour définir des processus : LDP pour Langage de Définition de Processus. Pour cela, nous définirons son méta-modèle en Ecore.

Un processus est formé d'une séquence ordonnée d'activités, avec un début et une fin. A titre d'exemple, la figure ci-dessous donne le processus qui correspond au début du module d'ingénierie des modèles que vous êtes en train de suivre. Chaque carré aux coins arrondis correspond à une activité, ici à une partie de l'enseignement. Le point de départ du processus est précisé par un rond noir plein tandis que le point d'arrivée est un même rond noir mais entouré en plus d'un cercle. Nous ne développerons pas de syntaxe graphique concrète pour ce DSML mais nous pourrions définir un tel modèle de manière abstraite.

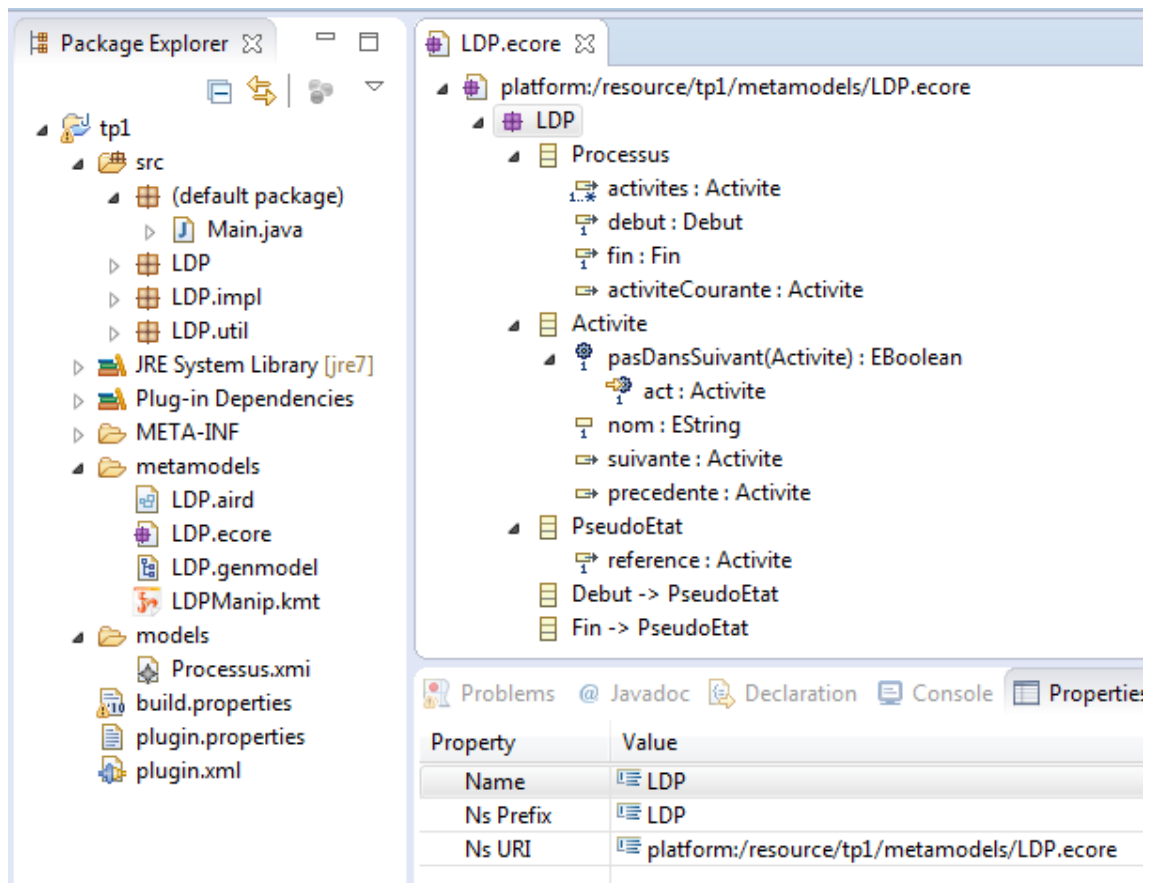


La figure ci-dessous représente le méta-modèle de LDP. Il contient un méta-élément *Processus* qui permet de définir les éléments du processus, à savoir sa liste d'activités et ses deux pseudo-états de *Debut* et *Fin* (la méta-classe *PseudoEtat* étant abstraite), chaque pseudo état référençant une activité (soit l'activité de début, soit la dernière du processus). Une *Activite*, à l'exception bien sûr de la dernière du processus, possède une activité *suivante*, ce qui permet de définir la séquence d'activités du processus. De même, une activité, sauf la première, possède une activité précédente. Si l'on est en train d'exécuter ce processus, *activiteCourante* référence, parmi les activités du processus, l'activité en cours du processus, sinon, elle n'est pas positionnée.



CREATION D'UN META-MODELE ECore

1. Créer un nouveau projet EMF vide : *File -> New -> Other -> Eclipse Modeling Framework -> Empty EMF Project* . Nommer le `tp1` pour reprendre directement les extraits de code de ce TP et du suivant.
2. Dans la hiérarchie du projet, créer un nouveau répertoire que vous nommerez `metamodels`
3. Sélectionner *metamodels* dans la hiérarchie et via le menu contextuel, créer un méta-modèle ECore : *New -> Other -> Eclipse Modeling Framework -> Ecore Model* . Nommer le fichier `LDP.ecore`. Un onglet central s'ouvre alors avec 2 lignes de texte qui génèrent des erreurs. Fermer l'onglet, sélectionner dans la hiérarchie le fichier Ecore et via le menu contextuel : *Open With -> Sample Ecore Model Editor* . Cela ouvre l'éditeur de méta-modèle en mode arbre qui est l'éditeur par défaut.
4. Dans la fenêtre centrale, ouvrir l'onglet du méta-modèle Ecore créé et remplir les champs du package créé par défaut et sans nom :
 - Name : le nom du package, mettre "LDP" par exemple
 - NS Prefix : mettre la même valeur que le nom du package
 - NS URI : mettre un URI de la forme suivante `platform:/resource/tp1/metamodels/LDP.ecore` si votre projet s'appelle "tp1" et votre fichier Ecore "LDP.ecore". Cet URI est l'adressage absolu de votre méta-modèle au sein de votre workspace Eclipse.
5. Ajouter, via *New Child* du menu contextuel, tous les éléments un à un et leurs sous-éléments pour créer le méta-modèle. Le méta-modèle à obtenir sous forme d'un arbre est celui-ci et correspond exactement au méta-modèle présenté plus haut :



Attention à bien positionner les cardinalités des références (un " * " se note par la valeur -1) et que les 2 références `precedente` et `suivante` soient bien l'opposée l'une de l'autre (réalisant une association bi-directionnelle). De même, attention à bien positionner la caractéristique de composition quand cela est requis. Les deux figures suivantes montrent précisément les propriétés à éditer pour tout cela :

The screenshot shows the Eclipse IDE interface. The top part displays a tree view of the Ecore model 'LDP.ecore'. The tree structure is as follows:

- platform:/resource/tp1/metamodels/LDP.ecore
 - LDP
 - Processus
 - activites : Activite (multiplicity 1..*)
 - debut : Debut (multiplicity 1)
 - fin : Fin (multiplicity 1)
 - activiteCourante : Activite (multiplicity 1)
 - Activite
 - pasDansSuivant(Activite) : EBoolean (multiplicity 1)
 - act : Activite (multiplicity 1)
 - nom : EString (multiplicity 1)
 - suivante : Activite (multiplicity 1)
 - precedente : Activite (multiplicity 1)

The bottom part of the screenshot shows the 'Properties' view for the 'activites' property. It is a table with two columns: 'Property' and 'Value'.

Property	Value
Changeable	true
Container	false
Containment	true
Default Value Literal	
Derived	false
EKeys	
EOpposite	
EType	Activite
Lower Bound	1
Name	activites
Ordered	true
Resolve Proxies	true
Transient	false
Unique	true
Unsettable	false
Upper Bound	-1
Volatile	false

- Il est possible d'afficher et d'éditer le modèle via une syntaxe graphique (à la UML). Pour cela : sélectionner le nom du modèle Ecore réalisé et choisir *Initialize ecore_diagram diagram files* dans le menu contextuel. Choisissez la représentation *Entities in a Class Diagram* et dans la palette à droite, cliquez sur *Add* dans *Existing elements*. Sélectionner toutes les classes et valider. Vous pouvez ensuite modifier la disposition des classes et de leurs relations.

On peut alors modifier le modèle selon l'une et l'autre des vues (arbre ou graphe UML) mais attention, c'est le même fichier Ecore derrière, il donc faut

sauvegarder une vue avant d'éditer par une autre sinon il y aura des incohérences de contenu.

Note : les relations de composition sont essentielles pour définir un méta-modèle Ecore et il faut respecter deux contraintes :

- A partir d'un méta-élément, s'il est associé à un autre via une relation de composition, on pourra directement instancier cet élément à l'intérieur d'une instance du premier via l'éditeur de modèles.
- En conséquence, pour pouvoir construire un modèle complet directement, tout méta-modèle doit contenir un élément dit racine à partir duquel on pourra instancier directement ou par transitivité via les liens de composition tous les autres éléments du méta-modèle. Cet élément racine sera de préférence un élément existant du méta-modèle mais si aucun ne peut convenir, on créera explicitement un élément racine. Ici, on a naturellement choisi le méta-élément `Processus` qui contiendra toutes les activités et les deux pseudo-états du processus.

CREATION ET EDITION BASIQUE D'UN MODELE

La création d'un modèle se fait en instanciant le méta-élément racine puis en créant ensuite les éléments qu'il contient :

1. Dans le méta-modèle, sélectionner `Processus` et via le menu contextuel, exécuter *Create Dynamic Instance*. Cela crée un fichier XML que vous pouvez placer dans le répertoire `models` de votre projet.
2. Sélectionner le processus créé dans le fichier XML et constater via le menu contextuel qu'il y a trois choix dans le menu *New Child* : *Activites*, *Debut* et *Fin*, chacun correspondant à une des trois associations en mode composition.
3. Créer un élément début, un élément fin et plusieurs activités. Positionner les propriétés de chacun des éléments pour former un modèle de processus, par exemple celui proposé ci-dessus. Noter que quand vous positionnez l'activité B comme suivante de A, A devient automatiquement l'activité précédente de B. Cela est dû à l'aspect bi-directionnelle de l'association.

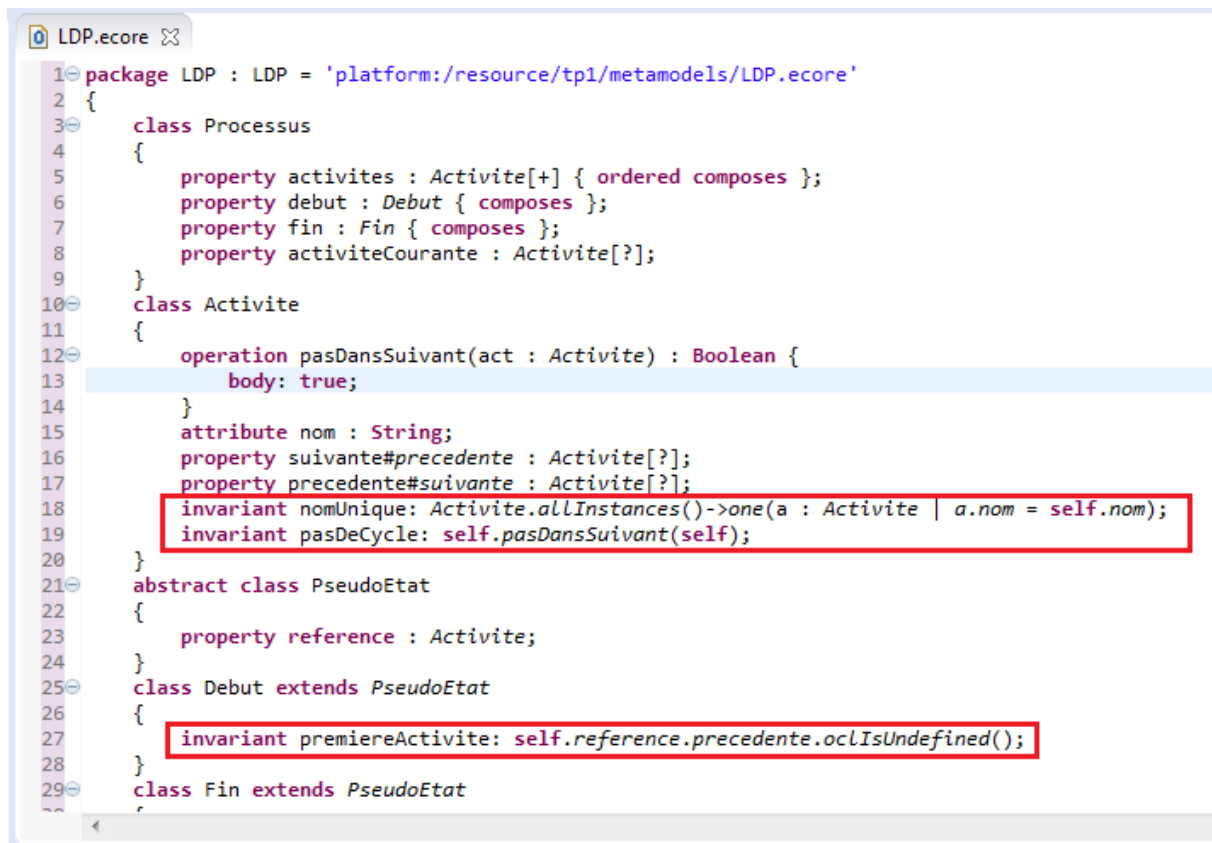
VALIDATION DE MODELE ET DE META-MODELE

Lors de l'édition d'un méta-modèle ou d'un modèle, le menu contextuel propose une option *Validation*. Celle-ci permet de vérifier structurellement qu'un modèle ou un méta-modèle est bien valide. Par exemple pour un méta-modèle, qu'on n'a pas oublié de préciser le type d'un attribut. Pour un modèle, les contraintes d'associations et de cardinalités entre éléments sont vérifiées. Par exemple pour le langage de processus, l'association `reference` de `PseudoEtat` a une cardinalité de 1, elle doit donc être forcément positionnée pour tout élément d'une sous-classe concrète de `PseudoEtat`.

Noter également que comme les associations `debut` et `fin` de `Processus` ont une cardinalité de 1, l'éditeur de modèle ne permet de créer qu'un exemplaire de chaque à partir de l'instance du processus.

Pour les modèles, il est également possible de rajouter des contraintes OCL sur les méta-éléments. La validation sur un modèle vérifiera alors en plus des contraintes structurelles que les invariants OCL sont bien respectés. Pour ajouter de l'OCL dans un méta-modèle, le plus simple est d'éditer le méta-modèle de manière textuelle vu qu'OCL est un langage textuel. Pour cela, sélectionner le fichier `Ecore` dans l'arborescence de fichiers de votre projet et l'ouvrir via *Open with -> OclInEcore Editor*

Vous obtenez alors la vue suivante sur le méta-modèle où on retrouve la même définition des méta-éléments mais dans une syntaxe textuelle :



```
1 package LDP : LDP = 'platform:/resource/tp1/metamodels/LDP.ecore'
2 {
3   class Processus
4   {
5     property activites : Activite[+] { ordered composes };
6     property debut : Debut { composes };
7     property fin : Fin { composes };
8     property activiteCourante : Activite[?];
9   }
10  class Activite
11  {
12    operation pasDansSuivant(act : Activite) : Boolean {
13      body: true;
14    }
15    attribute nom : String;
16    property suivante#precedente : Activite[?];
17    property precedente#suivante : Activite[?];
18    invariant nomUnique: Activite.allInstances()->one(a : Activite | a.nom = self.nom);
19    invariant pasDeCycle: self.pasDansSuivant(self);
20  }
21  abstract class PseudoEtat
22  {
23    property reference : Activite;
24  }
25  class Debut extends PseudoEtat
26  {
27    invariant premiereActivite: self.reference.precedente.oclIsUndefined();
28  }
29  class Fin extends PseudoEtat
```

1. Ajouter le contenu des deux cadres en rouge qui sont des invariants vérifiant qu'une activité a un nom unique et n'est pas dans un cycle (une activité ne se trouve pas elle-même dans la séquence de ses suivantes) ainsi que le pseudo-état de début référence bien une activité n'ayant pas de précédente. Ajouter aussi le "body" (corps de la méthode) de l'opération `pasDansSuivant` tel que présenté dans la ligne surlignée en bleu. Vérifier en modifiant votre modèle que les contraintes sont bien vérifiées en lançant la validation.

2. Ajouter l'invariant dans `Fin` qui assure que le pseudo-état de fin référence une activité sans suivante.
3. Ajouter un invariant qui assure qu'il n'existe qu'une seule activité qui n'a pas de suivante et qu'une seule activité qui n'a pas de précédente.
4. Ecrire le corps de la méthode `pasDansSuivant` pour qu'elle vérifie qu'une activité n'a pas dans ses suivantes celle qui est passée en paramètre.

Pour plus d'informations sur l'intégration des contraintes OCL en Ecore, voir ce [tutoriel](https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.ocl.doc%2Fhelp%2FTutorials.html) :(<https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.ocl.doc%2Fhelp%2FTutorials.html>) On y trouve notamment la définition de références ou attributs dérivés et de corps d'opération via une expression OCL.

MANIPULATION DES MODELES EN JAVA

Le framework EMF permet dans un programme Java de manipuler des modèles définis à partir d'un méta-modèle Ecore. Pour cela, il faut générer le code des classes Java qui correspondent aux méta-classes d'un méta-modèle Ecore ainsi que quelques classes utilitaires. Cela se fait de la manière suivante :

- Il faut commencer par créer l'équivalent "genmodel" du fichier Ecore. Sélectionner le fichier Ecore puis via le menu contextuel : *New -> Other -> Eclipse Modeling Framework -> EMF Generator Model*. Dans la liste de la fenêtre *Select a Model Importer* choisir *Ecore Model* et sélectionner le modèle Ecore du projet dans la fenêtre suivante, cliquer sur *Load* et finir.
- Ouvrir ce fichier "genmodel", se placer dans sa liste d'éléments sur le package principal et via le menu contextuel, générer le code du modèle via *Generate Model Code*.
- Ouvrir le fichier MANIFEST.MF du répertoire META-INF du projet : onglet *Dependencies*, dans la liste des *Required Plugs-ins*, rajouter les plugins suivants s'ils ne sont pas présents :
 - `org.eclipse.emf.ecore`
 - `org.eclipse.emf.ecore.xmi`

Une fois ce code généré pour le méta-modèle LDP, vous trouverez dans le répertoire `src` un package `LDP` contenant les interfaces Java `Processus`, `Activite`, `PseudoEtat`, `Debut` et `Fin`. Ouvrez les et constatez que vous y retrouvez des getters et setters pour les attributs et références de votre méta-modèle Ecore. Les deux autres interfaces `LDPFactory` et `LDPPackage` serviront au chargement de modèles et à la gestion de leur contenu.

La manipulation de modèles XML directement en Java est assez complexe concernant le chargement et la sauvegarde d'un modèle. Voici le code générique (indépendamment d'un méta-modèle particulier) qui permet de charger et d'enregistrer un modèle XML avec les imports requis :


```

import org.eclipse.emf.common.util.TreeIterator;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xml.XMLResource;
import org.eclipse.emf.ecore.xml.XMLResource.XMLMap;
import org.eclipse.emf.ecore.xml.impl.XMIResourceFactoryImpl;
import org.eclipse.emf.ecore.xml.impl.XMLMapImpl;

public void sauverModele(String uri, EObject root) {
    Resource resource = null;
    try {
        URI uriUri = URI.createURI(uri);
        Resource.Factory.Registry.INSTANCE.getExtensionToFactory
Map().put("xmi", new XMIResourceFactoryImpl());
        resource = (new
ResourceSetImpl()).createResource(uriUri);
        resource.getContents().add(root);
        resource.save(null);
    } catch (Exception e) {
        System.err.println("ERREUR sauvegarde du modèle : "+e);
        e.printStackTrace();
    }
}

public Resource chargerModele(String uri, EPackage pack) {
    Resource resource = null;
    try {
        URI uriUri = URI.createURI(uri);
        Resource.Factory.Registry.INSTANCE.getExtensionToFactory
Map().put("xmi", new XMIResourceFactoryImpl());
        resource = (new
ResourceSetImpl()).createResource(uriUri);
        XMLResource.XMLMap xmlMap = new XMLMapImpl();
        xmlMap.setNoNamespacePackage(pack);
        java.util.Map options = new java.util.HashMap();
        options.put(XMLResource.OPTION_XML_MAP, xmlMap);
        resource.load(options);
    }
    catch(Exception e) {
        System.err.println("ERREUR chargement du modèle : "+e);
        e.printStackTrace();
    }
    return resource;
}

```

Ensuite, pour charger le modèle LDP d'un fichier "Processus.xmi" se trouvant dans le répertoire "models" du projet courant, on exécutera le code suivant qui va récupérer l'instance (supposée unique) d'un processus :

```
Resource resource = chargerModele("models/Processus.xmi",
LDPPackage.eINSTANCE);
if (resource == null) System.err.println(" Erreur de
chargement du modèle");
TreeIterator it = resource.getAllContents();
Processus proc = null;
while(it.hasNext()) {
    EObject obj = it.next();
    if (obj instanceof Processus) {
        proc = (Processus)obj;
        break;
    }
}
```

Pour sauver une instance "proc" de `Processus` dans un fichier "Processus2.xmi" du répertoire "models", on exécutera le code suivant :

```
sauverModele("models/Processus2.xmi", (EObject)proc);
```

Une fois un processus chargé, on peut le manipuler, récupérer les éléments lui étant associés et changer les propriétés (attributs et associations) des éléments via l'appel des getter et setter dédiés. Si vous avez besoin de créer des instances de nouveaux éléments, il faut passer par la factory générée pour le méta-modèle. Par exemple, le code suivant crée une nouvelle instance de `Activite` :

```
Activite act = LDPFactory.eINSTANCE.createActivite();
```

1. Créer un programme Java qui affiche dans la console le contenu d'un modèle. Ne pas oublier de marquer l'éventuelle activité courante du processus.
2. Ajouter dans votre programme une méthode qui "exécute" le modèle pas-à-pas, c'est-à-dire qui modifie l'activité courante en la passant à l'activité suivante. Sauver le modèle après chaque pas d'exécution. Regarder le contenu des fichiers XMI générés pour constater que l'activité courante est positionnée ou modifiée.
3. Ajouter dans votre programme une méthode qui prend en paramètre deux noms et dont le but est de rajouter une activité dans la séquence du processus. Le premier nom est celui d'une activité existante et le second celui de l'activité à ajouter après celle existante.

Pour vous aider à implémenter tout cela, vous pouvez reprendre le fichier [LDPManipulation.java](#) qui contient tout le code de chargement/enregistrement des modèles et les signatures des méthodes à implémenter.