Génération de codes

à partir d'un modèle de Conception UML sous PowerAMC

Véronique Deslandres©, IUT,
Département Informatique
Université Lyon1

Introduction

- La génération de code, ça n'est pas immédiat : processus en 2 étapes
  - Vérifier le modèle
  - Générer le code

- Génération **totale** ou **restreinte**

Génération partielle de code

- Il est possible de restreindre la portée de la génération au contenu d'un **package particulier**

- C’est très utile lorsque les différents packages sont affectés à différents développeurs.
  - Chacun d'entre eux peut alors générer son package indépendamment des autres.
  - La génération depuis un package permet de produire un modèle indépendant.

Étape 1: Vérification du modèle

- Dans un AGL, la génération de code ou d'un modèle physique de données (MPD), commence toujours par une **vérification de la validité** du Modèle Orienté Objet (MOO)
  - Vérifier unicité des noms, pas de classe vide, etc.
- Vérifier le modèle (**F4** ou menu **Outils**)
- Si le modèle est incorrect ou si une erreur est détectée, la génération est interrompue

1

Avec PAMC

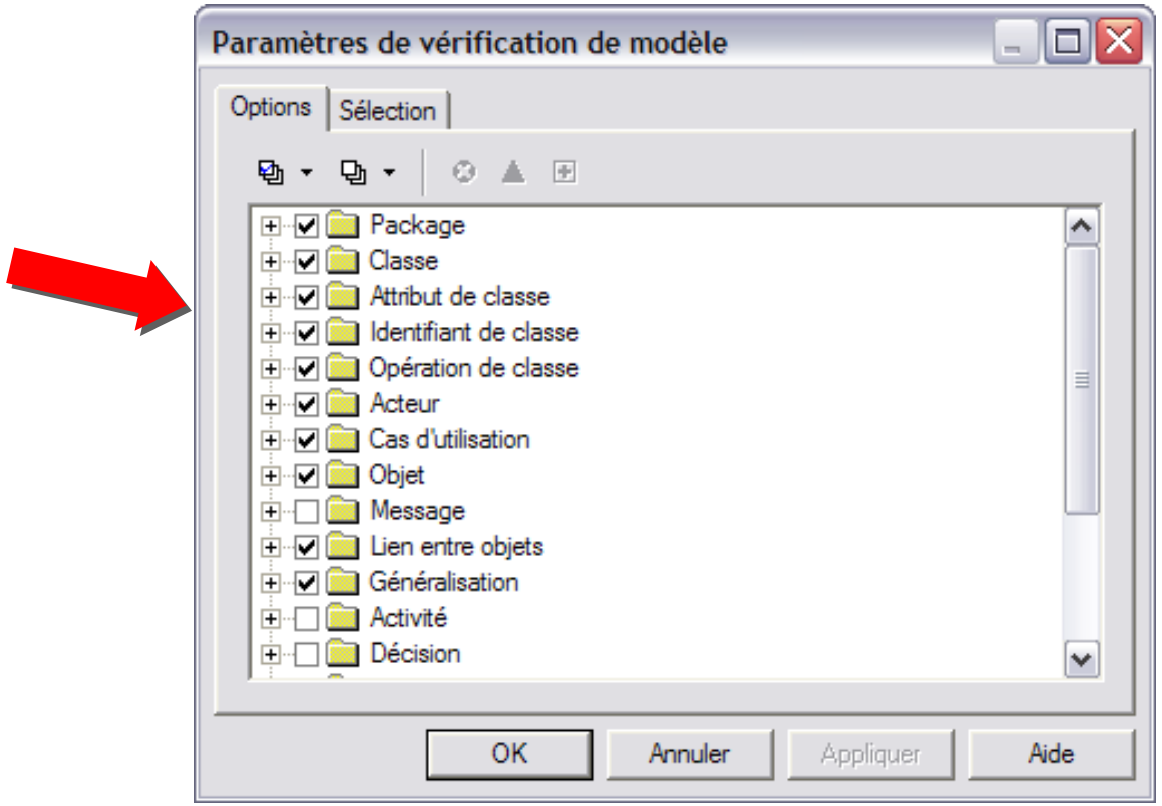
Vérification du modèle

Paramétrage de la vérification

On peut ajuster les erreurs / avertissements, affichés dans la fenêtre **Paramètres de vérification** du modèle :

Ainsi avant de lancer la vérification, vous pouvez définir :

- les **éléments** à vérifier
- le **degré de sévérité** pour les problèmes que la vérification détecte
  - et faire en sorte que certains problèmes soient corrigés automatiquement



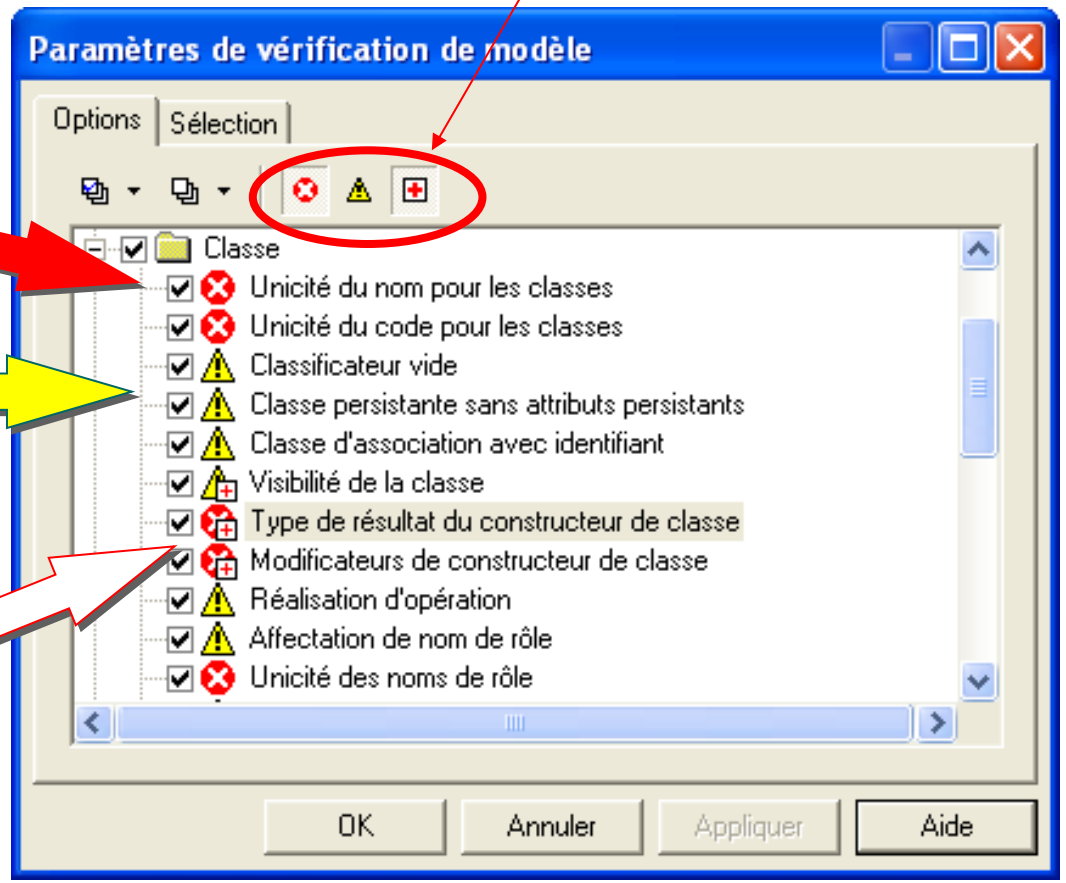
Degré de sévérité de la vérification

En cliquant sur les icônes, on peut ajuster la sévérité de la règle

erreur

avertissement

correction automatique par l'AGL



Règles de vérification de PowerAMC

Par exemple:

- Les noms et les codes de classe doivent être **uniques** dans le modèle ou le package (dans le même espace de nom)
  - Correction auto: modifie le nom ou le code de classe en y ajoutant un numéro
- Lorsqu'il existe une réalisation entre une classe et une interface (au sens java), on doit mettre en œuvre (**implémenter**) les opérations de l'interface au sein de la classe.
- Les fichiers externes incorporés dans les classes doivent avoir **un chemin d'accès** valide.
- Etc.

Règles de vérification de PowerAMC

- Pour voir toutes les règles de vérification du modèle MOO :
  - Regarder l'aide
  - Chapitre – Gestion des MOO
  - Sous-chapitre « Vérification d'un MOO »

Vérification du modèle : les erreurs courantes (1/2)

Existence d'attribut

Les identifiants doivent avoir au moins un attribut.

Correction manuelle	Correction automatique
Ajoutez un attribut à l'identifiant, ou supprimez l'identifiant	—

Inclusion d'identifiant

Deux identifiant ne doit pas utiliser le même attribut.

Correction manuelle	Correction automatique
Supprimez l'identifiant superflu	—

Unicité du nom et du code d'attribut

Les noms et les codes d'attribut doivent être uniques dans le modèle ou dans le package.

Correction manuelle	Correction automatique
Modifiez le nom/code d'attribut en double	Modifie le nom ou le code de l'attribut en y ajoutant un numéro

Diagramme de classes

Toutes les classes doivent avoir un nom de rôle !
Pas de rôle = NULL
→ le même pour toutes les classes sans rôle

Quelques erreurs courantes (2/2)

Message dépourvu de numéro d'ordre

Un message doit être doté d'un numéro d'ordre.

Correction manuelle	Correction automatique
Spécifiez un numéro d'ordre pour le message	—

Point de jonction incomplet

Un point de jonction représente une séparation ou une fusion de chemins de transition. Il doit donc compter au moins une transition entrante et une transition sortante.

Correction manuelle	Correction automatique
Ajouter les transitions manquantes au point de jonction	—

Transition entrante ou sortante absente

Chaque activité doit comporter au moins une transition entrante et une transition sortante.

Correction manuelle	Correction automatique
Ajoutez une transition liée à l'activité	—

Décision incomplète

Une décision représente une branche conditionnelle dans laquelle une transition entrante unique est scindée en plusieurs transitions sortantes, ou représente une fusion lorsque plusieurs transitions entrantes sont fusionnées en une transition sortante unique. Une décision doit donc comporter plusieurs transitions entrantes ou plusieurs transitions sortantes.

Correction manuelle	Correction automatique
Ajoutez les transitions manquantes sur la décision	—

Diagramme de collaboration ou de séquences

Diagramme d'activités

Cf Aide sur l'erreur ou l'avertissement

On peut aussi IGNORER certaines règles de vérification (ex. sur les activités, si on génère du code à partir du DCL)

2- Génération du code (Java)

Comment procéder avec PAMC

Étape 2 : Génération de codes

- Avec PAMC, on peut générer à la fois de **code JAVA** et de **scripts** de création de tables relationnelles

- Au fait : pourquoi une conception objet (UML) débouche-t-elle sur des *tables relationnelles* ?
  - Presque toutes les applications ont des données persistantes
    - (données conservées entre 2 exécutions de l'application)
  - Très souvent les données persistantes sont structurées.
  - Les SGBD relationnels sont la façon la plus efficace de stocker les informations.

Quelles données vers un SGBD, vers JAVA ?

- Seules les données **persistantes** sont converties en **tables relationnelles**
  - Le choix des données persistantes se fait en phase de conception
  - On décoche la case des classes non « persistantes »

- Les **classes JAVA** sont utilisées à l'exécution comme des *variables* : on y met les données récupérées des tables.

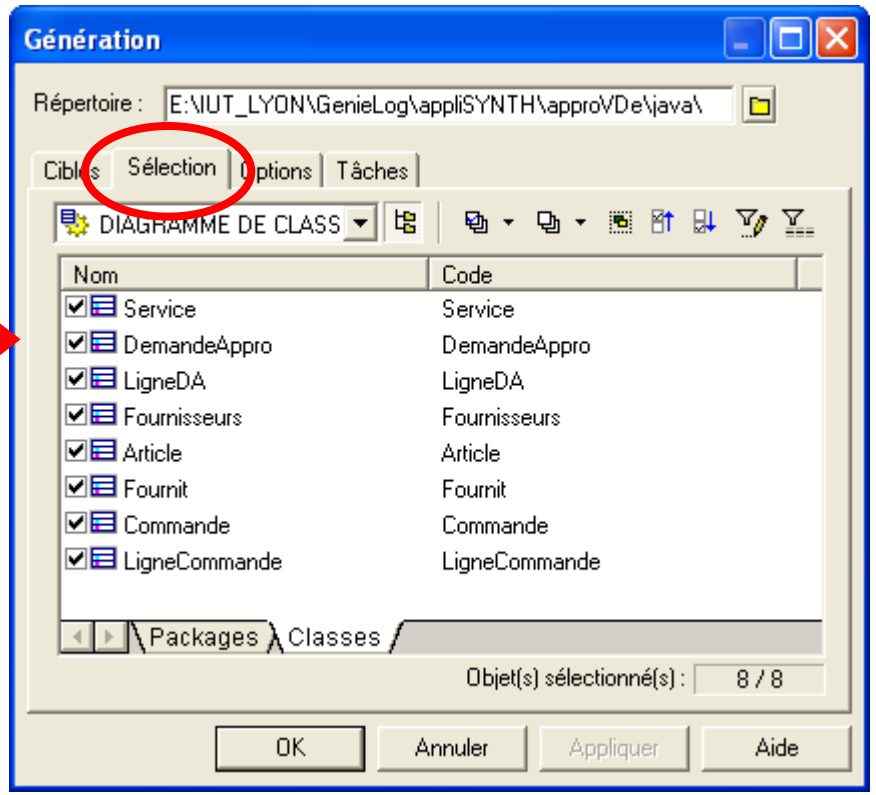
Génération de code JAVA

- Vous pouvez générer des fichiers source Java à partir des classes et interfaces d'un modèle
  - Classes que vous aurez sélectionnées
- Un fichier séparé, doté du suffixe .java, est généré pour chaque classe ou interface choisie
- Vous ne pouvez générer du code qu'à partir d'un **seul modèle à la fois**.

Pour générer le code Java

- Depuis le DCL conception :
- Sélectionnez **Langage-Générer du code Java** pour afficher la boîte de dialogue de génération

(Sélectionner les classes voulues en cas de génération restreinte)

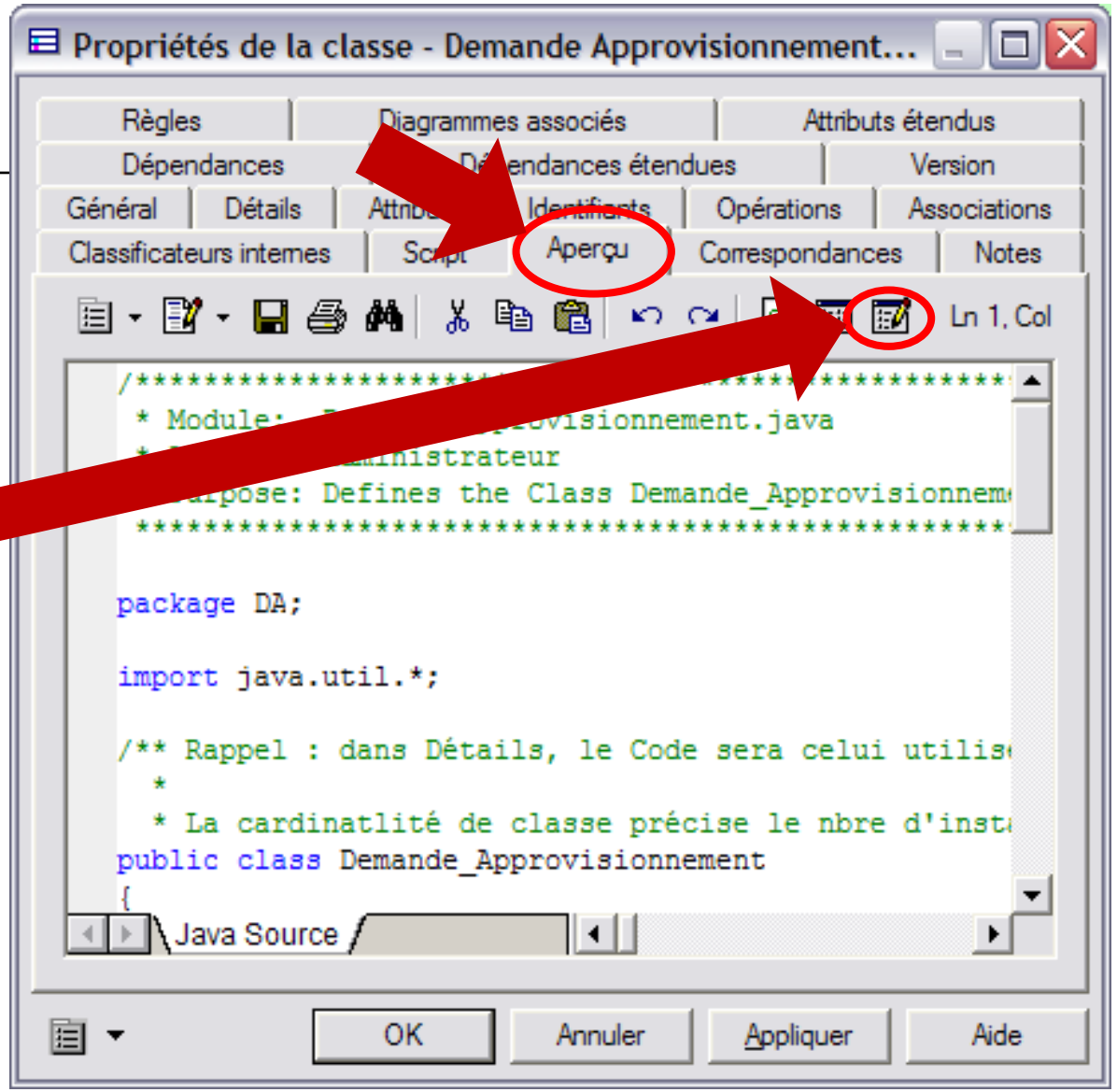


Si pas le bon langage :

menu Langage / Changer de langage objet courant…)

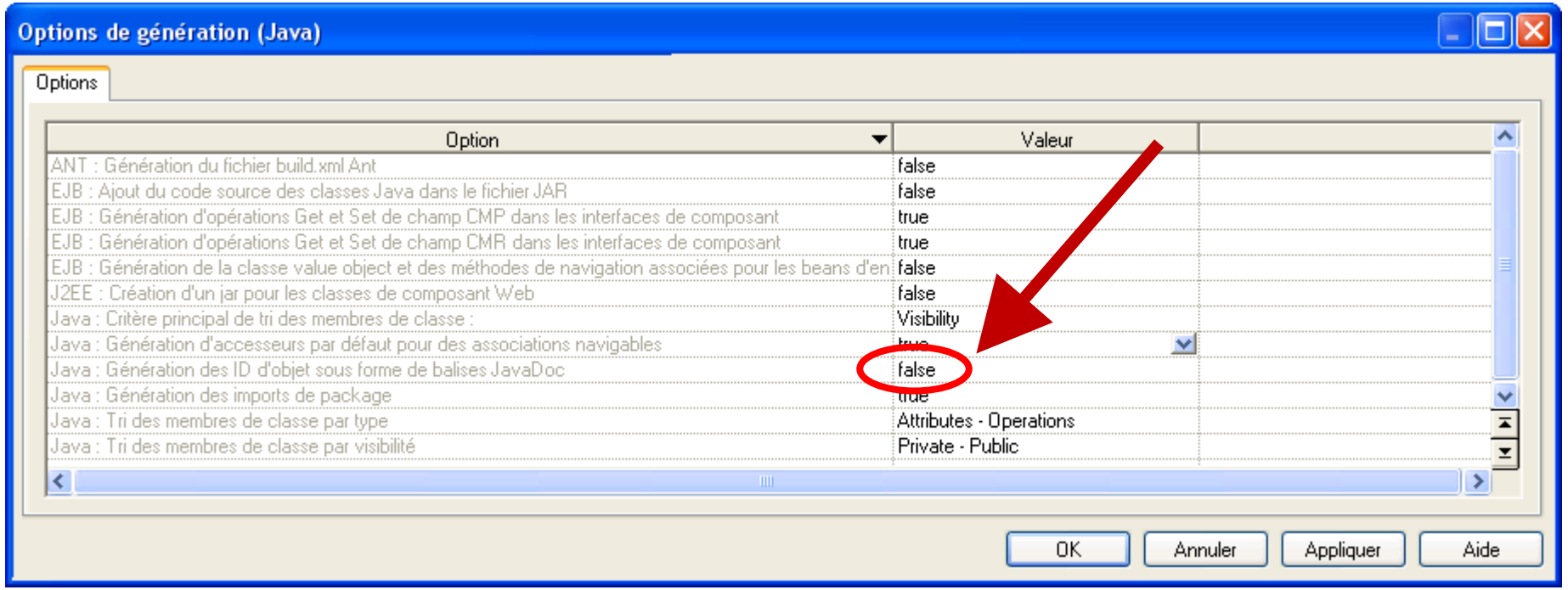
Aperçu du code d'une classe

On a la possibilité de **modifier les options de génération du code** avec l'outil « Affichage des options de génération »



Options de génération JAVA

- Pour supprimer les ID d'objets pour la Javadoc
  - Dans la fenêtre Génération, onglet Options, mettre à **false** la *Generation des ID d'objets balise Javadoc*
- Générer les imports de package

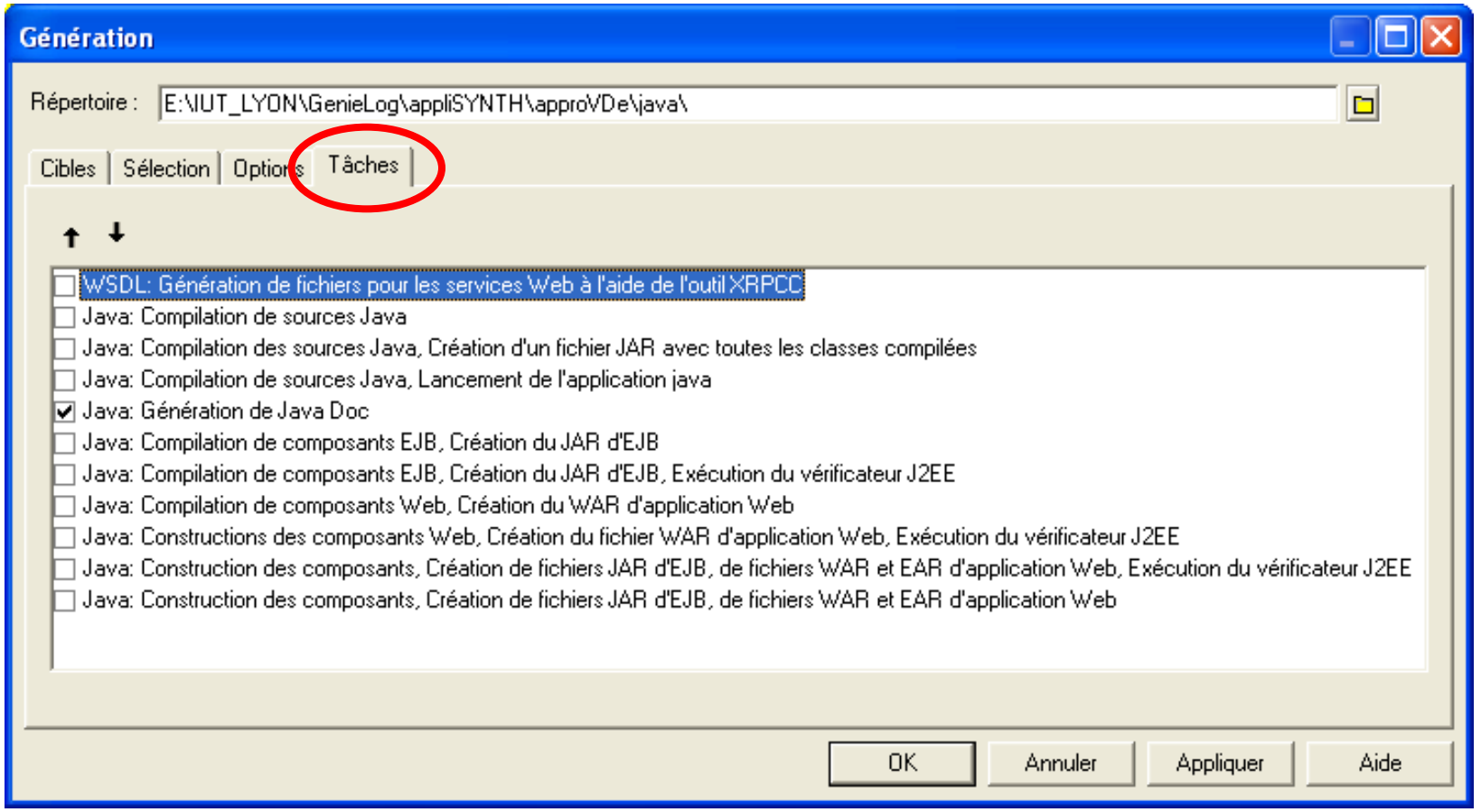


Options de génération pour Java

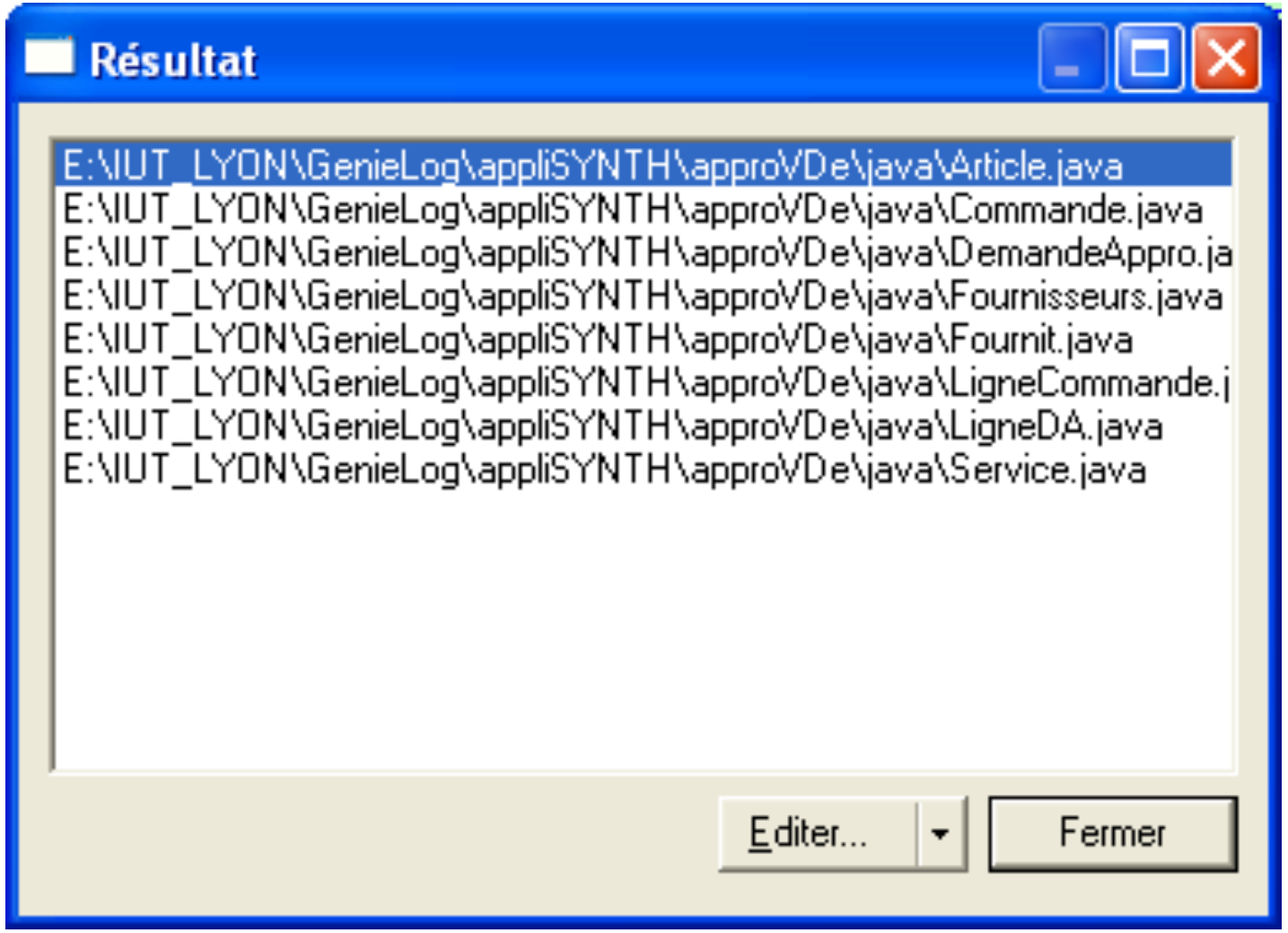
Options	Pour	Description
Génération des imports de package	Java	A utiliser pour déclarer l'importation de tout le package
Tri des membres de classe par type	Java	Trie les attributs et opérations par type d'abord
Tri des membres de classe par visibilité	Java	Trie les attributs et opérations par visibilité
Génération du fichier build.xml Ant	Java	Génère le fichier build.xml. (Vous pouvez utiliser ce fichier si vous avez installé Ant)
Génération d'opérations Get et Set de champ CMP dans les interfaces de composant	EJB	Génère des opérations Get et Set de champ CMP dans les interfaces EJB
Génération d'opérations Get et Set de champ CMR dans les interfaces de composant	EJB	Génère des opérations Get et Set de champ CMR dans les interfaces EJB
Ajout du code source des classes Java dans le fichier JAR Etc.	EJB	Inclut le code des classes Java dans le fichier JAR

Sélection des tâches

(tâches à exécuter lors de la génération)



Résultats


Résultat
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\Article.java
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\Commande.java
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\DemandeAppro.ja
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\Fournisseurs.java
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\Fournit.java
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\LigneCommande.j
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\LigneDA.java
E:\IUT_LYON\GenieLog\appliSYNTH\approVDe\java\Service.java
Editer...
Fermer

3- Génération des scripts SQL pour la création de tables

Comment procéder avec PAMC

Persistance :
génération de scripts pour un SGBD

Une fois le MOO complété et vérifié… deux alternatives :

Si on désire utiliser un SGBD

- Depuis le DCL de conception, générer d'abord un **Modèle Physique de Données** (MPD)
  - qui réalise la correspondance 'objet → relationnel'
  - on choisit le SGBD lors de la génération du MPD
- Générer ensuite le script SQL de création des tables

(Cette opération est détaillée ci-après)

Sérialisation avec des fichiers

Si on souhaite fonctionner avec des fichiers

On va sérialiser les objets créés lors de l'application Java.

Il suffit alors de générer directement les classes Java depuis le MOO Conception.

Correspondance Objet-Relationnel

- La correspondance O-R permet de gérer la **persistance** des objets dans une base de données relationnelle
  - Certains frameworks le font tout seul, ex. Spring, Hibernate
  - C'est ce qu'on appelle le **Mapping OR**

Correspondance Objet/Relationnel

MOO		MPD
Classe	→	Table
Attribut	→	Colonne
Opération	→	Requêtes SELECT ou UPDATE associées à une table
Association	→	Référence, table ou vue

- On peut aussi définir les requêtes SQL pour les opérations des classes concernées

- Deux types de correspondances O-R :
  - **Automatiques** (notre cas en CPOA)
  - Définies par l'utilisateur

Créer un MPD sous PAMC

- On génère, depuis le MOO, un **Modèle Physique de Données (MPD)**, puis le code SQL de création des tables à partir de ce MPD.
  - Se placer **sur le Diagramme de Classes**
  - Menu **Outil / Générer un MPD :** PAMC crée un nouveau modèle
  - Choisir le SGBD cible (ex. Oracle ou MySQL)
  - Dans l'onglet **Détails**, vous pouvez choisir (entre autre) le préfixe des tables
  - Dans l'onglet **Sélection**, **choisir les classes concernées**

Ex. : traduction d'une classe d'association

Les clefs étrangères seront automatiquement affectées

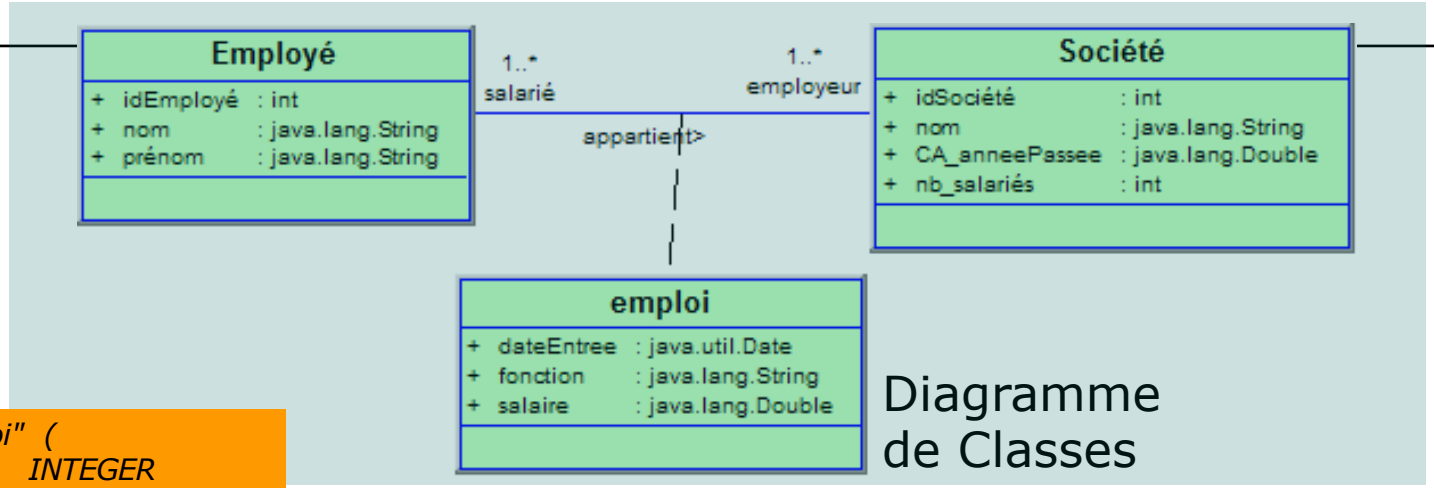
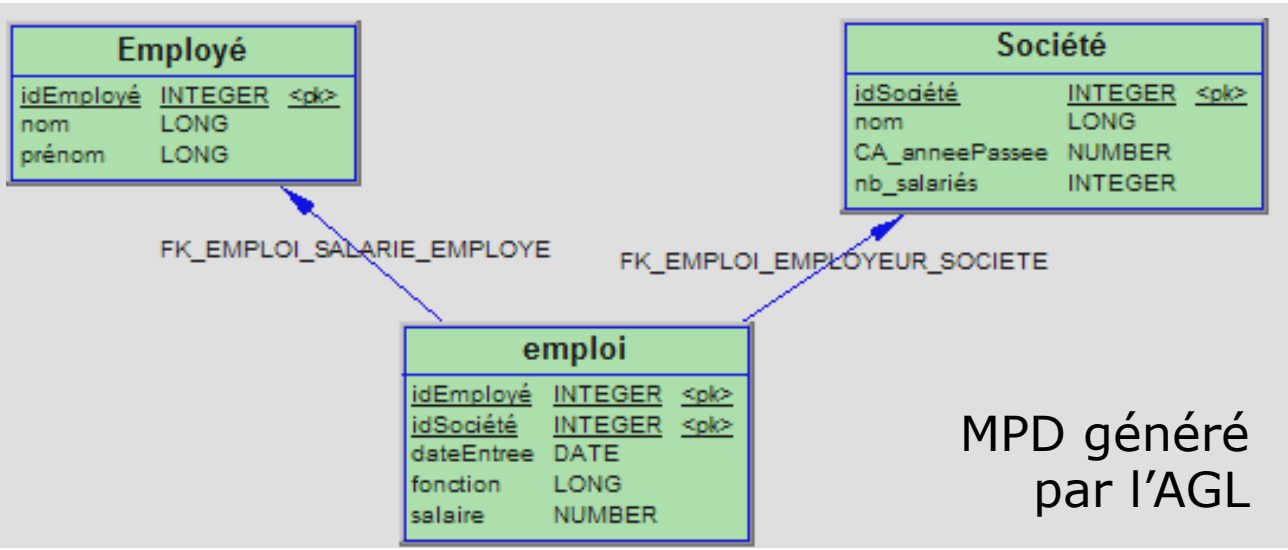


Diagramme de Classes

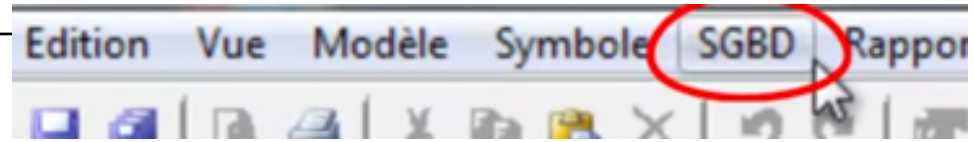
```
create table "emploi"  (
   "idEmploye"          INTEGER
not null,
   "idSociete"          INTEGER
not null,
   "dateEntree"         DATE,
   "fonction"           LONG,
   "salaire"            NUMBER,
   constraint PK_EMPLOI primary key
("idEmploye", "idSociete")
);

/* Index : emploi_PK */

create unique index """emploi""_PK"
on "emploi" (
   "idEmploye" ASC,
   "idSociete" ASC
);
```



MPD généré par l'AGL

Cf Aide, Chapitre « Mise en correspondance d'objets dans un MOO »

Générer le script SQL (1/2)



- Aller sur le modèle MPD généré
- Dans le nouveau menu *SGBD* proposé, choisir *Générer la base de données…*
  - Onglet **Options : cocher Clé étrangère**
  - **Sélectionner les classes** dont on veut le script SQL
    - Par défaut, aucune n'est sélectionnée
  - Onglet **Format** : spécifier le délimiteur d'identifiant (" par défaut, ` préférable ?), générer tout en minuscule (convention)
  - Préciser le **répertoire cible** pour sauvegarder le script *Crebas.sql*

Générer le script SQL (2/2)

Cela va créer un script **crebas.sql**

- Vous avez aussi un aperçu du script pour chaque classe du MPD dans l'onglet **Aperçu** (que vous pouvez modifier sous PAMC)

- En cas d'erreur lors de la vérification du MPD :
  - Tenter la correction automatique (clic droit)
  - Sinon corriger avec un **clic droit** sur l'erreur : corriger, aller dans l'onglet Aperçu du script SQL : on comprend souvent l'erreur mentionnée
- Tutoriels youTube associés :

https://www.youtube.com/watch?v=Ta1DhbNFFRU

https://www.youtube.com/watch?v=XvohJm2AqtI (à partir de 6'09)

Pour finir : quelques conseils…

- Bien **vérifier les identifiants** des classes **avant** la génération des tables

- Attention à l'ordre des commandes SQL quand on crée le script .sql !
  - Commandes de création, d'ajout des contraintes et d'insertion des données.
  - Problème de *synchronisation* si pas fait dans l'ordre

  - Idée : scinder le crebas en 3 fichiers .sql différents (création, contraintes, insertion) pour assurer une manipulation sécurisée des données.

Reverse Engineering

- Une fois l'application réalisée en JAVA, on peut reconstruire le modèle UML associé
  - par « Reverse Engineering » et comparer avec votre modèle initial !
  - Conseil : procéder par groupe de classes