



**Créer, configurer et exécuter une transformation de modèle à modèle**



---

## Table des matières

### **Créer, configurer et exécuter une transformation de modèle à modèle. . . . 1**

|  |    |
|--|----|
| Introduction - Créer, configurer et exécuter une transformation de modèle à modèle . . . . .                                     | 1  |
| Module 1 - Créer un projet de mappage pour transformation de modèle à modèle et raffiner le mappage de modèle à modèle . . . . . | 3  |
| Leçon 1 - Créer un projet de mappage pour transformation de modèle à modèle . . . . .  | 3  |
| Leçon 2 - Examiner le projet de mappage. . . . .   | 5  |
| Leçon 3 - Créer et raffiner une déclaration de mappage de classe à classe . . . . .  | 6  |
| Leçon 4 - Créer et raffiner une déclaration de mappage de classe à interface . . . . .   | 10 |
| Leçon 5 - Créer et raffiner les déclarations de mappage requises par la déclaration de mappage d'opération à opération . . . . . | 12 |

|   |    |
|---|----|
| Leçon 6 - Créer et raffiner une déclaration de mappage de package à package . . . . .         | 17 |
| Leçon 7 - Créer et raffiner une déclaration de mappage de modèle à modèle . . . . .           | 19 |
| Module 2 - Générer le code de la transformation et l'exécuter . . . . .                       | 21 |
| Leçon 1 - Générer et compiler le code source de transformation . . . . .                      | 22 |
| Leçon 2 - Configurer un plan de travail d'exécution . . . . .                                 | 23 |
| Leçon 3 - Créer une configuration de transformation . . . . .                                 | 25 |
| Leçon 4 - Exécuter la transformation . . . . .  | 26 |
| Récapitulatif - Créer, configurer et exécuter une transformation de modèle à modèle . . . . . | 27 |



---

# Créer, configurer et exécuter une transformation de modèle à modèle

Ce tutoriel décrit comment créer une transformation de modèle à modèle. Il est possible de créer ce type de transformation en créant un projet de mappage de transformation. Une transformation de modèle à modèle transforme un modèle en un autre modèle ayant un niveau d'abstraction différent. Ce tutoriel décrit également comment créer et raffiner les modèles de mappage dans un projet de mappage et comment générer et tester le code de transformation de modèle à modèle.

## Objectifs d'apprentissage

Vous devez suivre chaque leçon de ce tutoriel en respectant l'ordre indiqué. Vous y apprendrez à accomplir les tâches suivantes :

- Créer un projet de mappage pour transformation de modèle à modèle
- Créer et raffiner les modèles de mappage dans un projet de mappage pour transformation de modèle à modèle en utilisant les outils de mappage de transformation
- Générer et compiler le code source de transformation de modèle à modèle
- Configurer et exécuter la transformation de modèle à modèle dans un plan de travail d'exécution

## Durée

Environ 60 minutes

### Information associée

Création de transformations de modèle à modèle



Afficher la version PDF

Pour visualiser ce fichier, vous devez disposer d'Adobe Acrobat Reader sur votre système.

---

## Introduction - Créer, configurer et exécuter une transformation de modèle à modèle

Ce tutoriel décrit comment créer, configurer et exécuter une transformation de modèle à modèle en utilisant les outils de mappage disponibles dans les produits de modélisation IBM Rational. Après avoir créé un projet de mappage de transformation, puis créé et raffiné les modèles de mappage à inclure dans ce projet, vous pouvez générer le code d'une transformation de modèle à modèle. Une fois le code source de la transformation généré, vous pouvez configurer et exécuter la transformation dans un plan de travail d'exécution.

Dans ce tutoriel, vous créez une transformation de modèle à modèle qui transforme des classes dans un modèle source en interfaces et en classes d'implémentation dans un modèle cible. La classe d'implémentation générée requiert des copies des opérations trouvées dans la classe source, et l'interface générée requiert seulement des copies des opérations publiques de la classe source.

Ce tutoriel peut nécessiter certains composants dont l'installation est optionnelle. Assurez-vous qu'ils sont installés en consultant la liste Configuration requise.

Une transformation est un type de stratégie d'implémentation de pattern qui prend un ou plusieurs éléments source et les change en un ou plusieurs nouveaux éléments cible. Les transformations permettent de passer du modèle au code, mais aussi de convertir un modèle en un autre modèle

présentant un niveau d'abstraction différent. Les transformations que vous créez peuvent contenir des informations d'implémentation détaillées ou spécifier des relations, ou mappages, entre modèles ou métamodèles.

Un projet de mappage pour transformation de modèle à modèle vous permet de spécifier des métamodèles source et cible et de créer un modèle de mappage qui définit les relations entre les éléments de ces métamodèles. Le code source de transformation qui implémente ces relations est extensible et peut être généré graduellement. En travaillant à ce niveau d'abstraction, vous pouvez vous concentrer sur le domaine du problème plutôt que sur celui de la solution.

La création d'une transformation de modèle à modèle comprend les étapes principales suivantes :

1. Vous créez un projet de transformation avec mappage de modèle à modèle qui contient un modèle de mappage. Un projet de mappage peut contenir plusieurs modèles de mappage. Lorsque vous créez un projet de mappage, le service de transformation enregistre une transformation. Chaque transformation comporte un fournisseur de transformation, une transformée appelée MainTransform et une transformée pour chaque déclaration de mappage dans le projet.
2. Vous ajoutez des déclarations de mappage, également appelées "mappes", au modèle de mappage. Un modèle de mappage peut contenir une ou plusieurs déclarations de mappage.
3. Vous ajoutez des règles de mappage aux déclarations de mappage dans un modèle de mappage.
4. Vous générez le code source de transformation à partir du ou des modèles de mappage présents dans le projet de mappage. Les outils de création de transformation de modèle à modèle génèrent une transformation par modèle de mappage dans le projet. Pour chaque déclaration de mappage, les outils de création génère un fichier source Java qui implémente une transformée. Pour chaque règle de mappage de type Déplacer ou Personnalisé dans une déclaration de mappage, une règle est générée dans le code source de la transformée. Pour chaque règle de mappage de type Sous-mappe dans une déclaration de mappage, un extracteur de contenu est généré dans le code source de la transformée.

## Objectifs d'apprentissage

Ce tutoriel est composé de deux modèles que vous devez suivre dans l'ordre. Vous y apprendrez à accomplir les tâches suivantes :

- Créer un projet de transformation de modèle à modèle qui contient une infrastructure de transformation et un modèle de mappage
- Créer des déclarations de mappage dans le modèle de mappage
- Raffiner les déclarations de mappage en effectuant les tâches suivantes :
  - Spécifier un objet d'entrée et un objet de sortie pour chaque déclaration de mappage
  - Créer des règles de mappage en définissant les relations entre les attributs des objets d'entrée et de sortie dans une déclaration de mappage
- Générer et compiler le code source de transformation
- Configurer un plan de travail d'exécution en vue de tester la transformation de modèle à modèle
- Dans un plan de travail d'exécution, créer et appliquer une configuration de transformation qui exécute la transformation de modèle à modèle

## Durée

Ce tutoriel vous prendra environ 60 minutes. Comptez plus de temps si vous explorez d'autres concepts associés.

## Niveau de compétences

Avancé

## Public concerné

Ce tutoriel s'adresse aux développeurs.

## Configuration système requise

Pour suivre ce tutoriel, vous devez avoir le composant de création de transformations installé sur votre système.

Vous devez également activer la capacité Modélisation.

## Conditions préalables

Pour suivre ce tutoriel, vous devez être familiarisé avec les concepts suivants :

- Eclipse Modeling Framework (EMF)
- Projets de plug-in Eclipse
- Modèles Ecore
- Plan de travail Eclipse

---

## Module 1 - Créer un projet de mappage pour transformation de modèle à modèle et raffiner le mappage de modèle à modèle

Dans ce module, vous créez un projet de mappage pour transformation de modèle à modèle qui contient plusieurs déclarations de mappage. Dans chaque déclaration, vous créez des règles de mappage qui définissent comment sont reliées les fonctions des éléments d'entrée et de sortie de la déclaration. Vous découvrirez également les fichiers qui constituent le projet et la manière dont ils sont utilisés pour générer le code de transformation.

### Objectifs d'apprentissage

Les leçons de ce module décrivent les modèles de mappage, les déclarations de mappage et les règles de mappage et vous expliquent comment effectuer les tâches suivantes :

- Créer un projet de mappage
- Créer et raffiner des déclarations de mappage
- Gérer des règles de mappage

### Durée

Ce module vous prendra environ 40 minutes.

## Leçon 1 - Créer un projet de mappage pour transformation de modèle à modèle

Cette leçon vous apprend à créer un projet de mappage pour transformation de modèle à modèle.

Un projet de mappage pour transformation de modèle à modèle est un plug-in Eclipse standard qui spécifie un fournisseur de transformations, mécanisme au cœur de la définition des transformations. Un projet de mappage contient également au moins un fichier de mappage, autrement appelé modèle de mappage. Lorsque vous créez un projet de mappage, un modèle de mappage y est créé automatiquement.

**En savoir plus sur les projets de mappage de transformation :**

Les projets de mappage pour transformation de modèle à modèle, également appelés projets de mappage, sont des plug-ins Eclipse qui exploitent le point d'extension nommé `com.ibm.xtools.transform.core.transformationProviders`. La création d'une transformation de modèle à modèle dans un projet de mappage vous permet de vous concentrer sur la spécification de la relation qui existe entre les éléments des modèles source et cible (ou métamodèles) plutôt que sur la création du code qui représente les détails d'implémentation de la transformation.

Un projet de mappage peut contenir plusieurs fichiers de mappage que l'on appelle aussi modèles de mappage. Lorsque vous modifiez un modèle de mappage, vous pouvez générer à nouveau le code source de la transformation. Lorsque vous générez le code source de transformation, une transformée visible de l'extérieur et ayant pour nom `MainTransform` est enregistrée automatiquement, et le code source Java d'une autre transformée est générée pour chaque déclaration de mappage présente dans le modèle de mappage.

Lorsque vous créez un projet de mappage, vous pouvez spécifier un ou plusieurs métamodèles source et cible. Vous pouvez spécifier des métamodèles, dont l'extension de nom de fichier est `.ecore`, ou des profils UML, qui ont pour extension `.epx` ou `.uml`. Si vous spécifiez des métamodèles source et cible lorsque vous créez un projet, les dépendances nécessaires sont ajoutées automatiquement au fichier manifeste du plug-in. Si vous ajoutez les métamodèles en utilisant les commandes dans la zone des éditeurs après avoir créé le projet de mappage, vous devez ajouter chaque nouvelle dépendance requise au fichier manifeste du plug-in.

Pour créer un projet de mappage pour transformation de modèle à modèle :

1. Ouvrez la perspective Développement de plug-in : cliquez sur **Fenêtre → Ouvrir la perspective → Autre**. Dans la fenêtre Ouverture de la perspective, cliquez sur **Développement de plug-in**, puis sur **OK**.
2. Cliquez sur **Fichier → Nouveau → Projet**.
3. Dans l'assistant Nouveau projet, sur la page Sélection d'un assistant, cliquez sur **Création de transformation**, puis sur **Projet de transformation avec mappage modèle à modèle**.
4. Cliquez sur **Suivant**.
5. Sur la page Projet de plug-in, dans la zone **Nom du projet**, tapez `Generaliser les classes` (sans lettres accentuées). Conservez les valeurs par défaut des autres zones de la page.
6. Cliquez sur **Suivant**.
7. Examinez les valeurs affichées sur la page Contenu du plug-in et cliquez sur **Suivant**.
8. Sur la page Modèles, dans la liste **Modèles disponibles**, sélectionnez **Plug-in avec mappage de transformation**.
9. Cliquez sur **Suivant**.
10. Sur la page Nouveau mappage de transformation, dans la zone **Nom de la mappe**, tapez `Generaliser les classes` si ce nom n'est pas déjà affiché. Cette zone spécifie le nom du modèle de mappage, qui sera situé dans le dossier `model` du projet et aura l'extension de nom de fichier `.mapping`.
11. Dans la zone **Nom du package**, tapez `generaliser les classes` si ce nom n'est pas déjà affiché. Dans une prochaine leçon, lorsque vous générerez le code source de transformation, le code Java des transformées sera créé dans un dossier nommé `generaliser les classes.transforms`.
12. Pour spécifier les modèles d'entrée et de sortie, effectuez les étapes suivantes :
  - a. Sur la page Nouveau mappage de transformation, à droite de la liste Modèles d'entrée, cliquez sur **Ajouter un modèle**.
  - b. Dans la boîte de dialogue Chargement d'une ressource, cliquez sur le bouton approprié pour naviguer jusqu'à l'emplacement où se situent les modèles. Pour les besoins de ce tutoriel, cliquez sur **Parcourir les packages enregistrés**, sélectionnez la version la plus récente du modèle UML2 dont le nom est de la forme `http://www.eclipse.org/uml2/2.x.y/UML` et cliquez sur **OK**. Cette étape spécifie que la transformation accepte comme source un métamodèle UML.ecore.

- c. Sur la page Nouveau mappage de transformation, à droite de la liste Modèles de sortie, cliquez sur **Ajouter un modèle**.
- d. Répétez l'étape 12b pour spécifier que la sortie de la transformation est un modèle du type UML.ecore.metamodel. Cette étape spécifie que la sortie de la transformation est un métamodèle UML.ecore.
- e. Cliquez sur **OK**.

Si vous spécifiez les métamodèles d'entrée et de sortie au moment où vous créez le projet, les dépendances nécessaires sont ajoutées automatiquement au fichier manifeste du plug-in. Si vous ajoutez les métamodèles en utilisant les commandes dans la zone des éditeurs après avoir créé le projet de mappage, vous devez ajouter chaque nouvelle dépendance requise au fichier manifeste du plug-in.

13. Cliquez sur **Terminer**.

Un projet de mappage est créé dans l'espace de travail. Dans la leçon suivante, vous allez examiner la structure de ce projet.

## Leçon 2 - Examiner le projet de mappage

Après avoir créé le projet de mappage, vous pouvez utiliser la vue Explorateur de packages pour examiner sa structure.

Pour examiner le contenu du projet de mappage :

1. Dans la vue Explorateur de packages, développez la branche du projet de mappage **Generaliser les classes** et observez les fichiers générés.
2. Naviguez jusqu'au dossier model. Il contient un modèle de mappage portant le même nom que le projet de mappage de transformation. Remarquez également que le fichier de ce modèle porte l'extension .mapping. Un projet de mappage peut contenir plusieurs modèles de mappage. Plus tard dans ce tutoriel, vous ajouterez des déclarations de mappage à ce modèle de mappage.

### En savoir plus sur les modèles de mappage :

Les modèles de mappage, autrement appelés fichiers de mappage, sont des instances de métamodèles EMF (Eclipse Modeling Framework), également appelés modèles Ecore, qui contiennent des références aux métamodèles que vous mappez. Lorsque vous créez un projet de mappage, les outils y créent un modèle de mappage en utilisant le ou les modèles d'entrée et de sortie que vous spécifiez. Un modèle de mappage se caractérise par l'extension de nom de fichier .mapping.

Les modèles de mappage sont stockés et sérialisés sous forme de fichiers XML. La vue Erreurs présente des informations détaillées sur les éventuelles erreurs qui affectent les modèles de mappage. En faisant un double clic sur une entrée dans cette vue, vous ouvrez le modèle de mappage dans un éditeur de texte, calé sur la ligne où se situe l'erreur. Cette méthode de dépannage est souvent plus simple que celle qui consiste à visualiser le modèle de mappage dans la zone des éditeurs.

3. Dans la vue Explorateur de packages, développez la branche du dossier src. Le dossier generaliser\_les\_classes.transforms contient le code source des transformées générées. A ce stade, il n'existe qu'une seule transformée par défaut qui a pour nom MainTransform. Plus tard, vous allez créer des déclarations de mappage et régénérer le code source de transformation. Pour chaque déclaration de mappage présente dans le modèle de mappage, l'infrastructure de création de transformations génère une classe Java ayant pour nom *n*Transform, la variable *n* représentant ici le nom de la déclaration de mappage. Ensemble, ces classes Java forment le code de transformation.
4. Dans la vue Explorateur de packages, faites un double clic sur le modèle de mappage dans le dossier model. L'éditeur de mappage de transformation s'ouvre pour vous permettre de créer des déclarations de mappage et de raffiner les règles de mappage dans chacune d'elles. Vous allez effectuer ces tâches dans les leçons suivantes de ce module.

## Leçon 3 - Créer et raffiner une déclaration de mappage de classe à classe

Cette leçon vous apprend à créer une déclaration de mappage qui spécifie une classe UML à la fois pour l'objet d'entrée et pour l'objet de sortie. Cette déclaration de mappage de classe à classe contient des règles de mappage qui, lorsque vous exécutez la transformation générée, créent une copie de la classe et de ses opérations à partir du modèle source et placent cette copie dans le modèle cible. Vous allez également créer une déclaration de mappage d'opération à opération, qui sera invoquée par la sous-mappe dans la déclaration de mappage de classe à classe.

Une déclaration de mappage, également appelée "mappe", spécifie comment créer ou mettre à jour un objet de sortie pour un objet d'entrée particulier. Une déclaration de mappage vous permet de spécifier de quelle manière les attributs d'un objet d'entrée correspondent aux attributs d'un objet de sortie. Chaque déclaration de mappage spécifie un type d'entrée et un type de sortie, que vous sélectionnez à partir des métamodèles que vous ajoutez au modèle de mappage.

Pour chaque déclaration de mappage présente dans le modèle de mappage, l'infrastructure de création de transformations génère un fichier source Java ayant pour nom *n*Transform.java, la variable *n* représentant ici le nom de la déclaration de mappage. Ensemble, ces fichiers Java forment le code de transformation. En plus de générer le code d'implémentation de la transformation, l'infrastructure de création de transformations génère le code nécessaire à l'enregistrement de la transformation auprès du service de transformation. Après avoir créé une déclaration de mappage, vous pouvez graduellement y ajouter des règles de mappage et générer leur code source ou leur implémentation. Vous n'avez pas à définir toutes les règles de mappage avant de générer le code source.

### En savoir plus sur les déclarations de mappage :

Les déclarations de mappage obéissent généralement à une convention de dénomination de la forme *x2y*, où *x* représente le type d'objet d'entrée et *y*, le type d'objet de sortie. Par exemple, une déclaration de mappage nommée Package2EPackage spécifie un mappage ayant Package comme objet d'entrée et EPackage comme objet de sortie.

Une mappe reliant des éléments établit une correspondance entre leurs attributs, permettant ainsi l'échange de données entre eux. La plupart des mappes offrent la possibilité de manipuler davantage les données entre la source et la cible. Par exemple, vous pourriez choisir de spécifier des calculs ou d'apporter d'autres modifications aux données en créant un code personnalisé vous permettant d'affecter des valeurs à la cible.

Pour créer une déclaration de mappage de classe à classe dans le modèle de mappage :

1. Si ce n'est déjà fait, ouvrez le fichier .mapping dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages (dossier model du projet).
2. Dans l'éditeur de mappage de transformation, à la section Racine de mappage, cliquez avec le bouton droit sur **Generaliser les classes** et sélectionnez **Créer une mappe**.
3. Entrez ClasseAClasse dans la zone **Nom de la mappe** de la fenêtre Nouvelle mappe et cliquez sur **OK**. La mappe apparaît dans la vue Structure et s'ouvre dans l'éditeur, sous la section Racine de mappage.

### Ajouter des objets d'entrée et de sortie à la déclaration de mappage de classe à classe

Une fois votre déclaration de mappage créée, vous devez lui ajouter un objet d'entrée et un objet de sortie. Dans cette leçon, vous allez spécifier une classe UML à la fois pour l'objet d'entrée et pour l'objet de sortie.

**Remarque :** Lorsque vous créez des transformations de modèle à modèle, pour ajouter un type d'objet qui n'est pas disponible dans le métamodèle d'entrée ou de sortie que la mappe utilise, vous pouvez

ajouter les profils UML ou les métamodèles Ecore appropriés à la portée de la mappe. Dans la fenêtre Ajouter une entrée, cliquez sur **Ajouter un modèle** et spécifiez le modèle approprié.

Pour ajouter un objet d'entrée et un objet de sortie à la déclaration de mappage de classe à classe :

1. Cliquez sur l'icône **Ajouter un objet en entrée**, qui est la première icône de la barre d'outils de la mappe que vous éditez :



2. Dans le volet Élément de la fenêtre Ajouter une entrée, sélectionnez un objet de métamodèle. Le volet Élément contient les éléments présents dans le ou les métamodèles que vous spécifiez comme source ou cible du modèle de mappage. Pour ce tutoriel, développez la branche **uml** dans le volet Élément, cliquez sur **Class**, puis sur **OK**.
3. Cliquez sur l'icône **Ajouter un objet en sortie**, qui est la deuxième icône de la barre d'outils de la mappe que vous éditez :



4. Dans la fenêtre Ajouter une sortie, développez la branche **uml** dans le volet Élément, cliquez sur **Class**, puis sur **OK**.
5. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant définir des règles de mappage entre les attributs des objets d'entrée et de sortie Class.

## Définir des règles de mappage entre les attributs des objets d'entrée et de sortie Class

Après avoir ajouté les objets d'entrée et de sortie à la déclaration de mappage, vous pouvez définir les règles de mappage entre les attributs de ces objets. Les règles de mappage, également appelés mappages, spécifient comment affecter une valeur à un attribut d'un objet de sortie, en fonction des valeurs des attributs d'un objet d'entrée.

Dans la déclaration de mappage de classe à classe, effectuez les étapes suivantes :

- Créez une règle de mappage Déplacer afin de créer une classe dans le modèle cible.  
Vous créez une règle de mappage entre les attributs name respectifs des objets d'entrée et de sortie. La classe cible porte le même nom que la classe dans le modèle d'entrée ; cette règle revient plus ou moins à créer une copie de la classe. Dans une autre leçon, vous ajouterez les opérations de mappage à la règle de mappage.
- Créez une règle de mappage Sous-mappe qui, pour chaque opération dans la classe, crée une opération correspondante dans la classe du modèle cible.  
Vous créez une règle de mappage Sous-mappe entre les attributs ownedOperation respectifs des objets d'entrée et de sortie. Pour chaque opération dans la collection ownedOperation du modèle d'entrée, une opération est générée dans le modèle cible, avec le même nom et la même visibilité.

### En savoir plus sur les règles de mappage :

Les règles de mappage, également appelés mappages, spécifient comment affecter une valeur à un attribut d'un objet de sortie, en fonction des valeurs des attributs d'un objet d'entrée. Vous pouvez créer plusieurs règles de mappage entre les objets d'entrée et de sortie. Dans une future leçon de ce tutoriel, vous verrez également qu'il est possible de créer plusieurs règles de mappage portant sur le même attribut d'un objet d'entrée et d'un objet de sortie. Vous pouvez créer les types suivants de règles de mappage :

## Déplacer

Un déplacement, également appelé règle de mappage simple, est le type de règle de mappage le plus élémentaire. Les attributs d'entrée et de sortie doivent être des types de données compatibles. Soit les attributs d'entrée et de sortie ont tous les deux des valeurs multiples, soit aucun d'eux n'a de valeurs multiples. Par exemple, sélectionnez cette option si un attribut de l'objet de sortie est un type String et qu'un attribut de l'objet d'entrée peut être converti en String sans l'aide d'un code personnalisé. Ce type de règle de mappage permet de mettre en correspondance un élément ou attribut source et un élément ou attribut cible. Le code source de transformation généré pour une règle de mappage Déplacer implémente une fonction qui copie la valeur d'un attribut d'entrée vers un attribut de sortie.

## Sous-mappe

Une sous-mappe est une invocation d'une mappe depuis une autre mappe. La sous-mappe invoquée peut, sans obligation, être définie dans le même fichier de mappage que la mappe qui l'invoque. Une sous-mappe vous permet de mapper un type complexe dans le modèle d'entrée à un type complexe dans le modèle de sortie. La sous-mappe que vous créez peut invoquer une mappe existant dans n'importe quel fichier de mappage. La définition de sous-mappes dans des fichiers de mappage distincts favorise la réutilisation des mappes ; cependant, plus vous créez de fichiers de mappage, plus vous augmentez les besoins de maintenance du projet. Les sous-mappes peuvent elles-mêmes inclure d'autres sous-mappes, formant ainsi une structure hiérarchique.

Les règles Sous-mappe permettent de créer les types suivants de mappages :

- Mappages 1-à-1 entre des objets d'entrée et des objets de sortie, ou entre les attributs de ces objets.
- Mappages 1-à- $m$  ou  $m$ -à-1 entre les attributs d'objets d'entrée et de sortie.
- Mappages  $m$ -à- $n$  entre les attributs d'objets d'entrée et de sortie.

Si la règle Sous-mappe spécifie une relation 1-à- $m$ , la transformée générée ajoute à une liste un objet issu d'un attribut à une seule valeur ; dans le cas d'une relation  $m$ -à-1, la transformée extrait un objet d'une liste et l'insère dans un attribut à une seule valeur.

Vous pouvez aussi créer des règles de mappage Sous-mappe entre les objets d'entrée et les objets de sortie dans une déclaration de mappage.

Pour chaque sous-mappe présente dans une déclaration de mappage, un extracteur nommé `getFonctionEntréeToFonctionSortie_UsingMappe_Extractor` est généré dans la transformée conteneur. Ici, *FonctionEntrée* représente le nom de l'attribut d'entrée, *FonctionSortie* représente le nom de l'attribut de sortie et *Mappe* représente le nom de la déclaration de mappage.

## Personnalisé

Ce type de règle de mappage vous permet de spécifier un code personnalisé qui calcule la valeur d'une propriété de sortie. Par exemple, sélectionnez ce type de mappage pour donner à une propriété de l'objet de sortie une valeur égale à la concaténation de plusieurs propriétés de l'objet d'entrée.

Vous pouvez spécifier des raffinages sémantiques en utilisant l'API OCL (Object Constraint Language) fournie par Eclipse.

Les règles de mappage personnalisé permettent de créer les types suivants de mappages :

- Mappages 1-à- $n$  entre des objets d'entrée et des objets de sortie, ou entre les attributs de ces objets.
- Mappages  $m$ -à- $n$  entre des objets d'entrée et des objets de sortie, ou entre les attributs de ces objets.
- Mappages  $m$ -à-1, qui peuvent être de l'un des types suivants :
  - Un mappage entre un attribut d'entrée dont la multiplicité est réglée à  $m$  vers un attribut de sortie dont la multiplicité est réglée à 1.

- Un mappage entre plusieurs attributs d'entrée vers un attribut de sortie.
- Un mappage entre plusieurs objets d'entrée vers un objet de sortie.

### Mappes héritées

Cette règle de mappage ne peut être créée qu'entre un objet d'entrée et un objet de sortie.

Une déclaration de mappage héritière hérite des règles de mappage définies dans la déclaration de mappage héritée. Vous pouvez remplacer les règles héritées en définissant une règle de mappage ayant les qualités suivantes :

- La propriété objet d'entrée et la propriété objet de sortie sont les mêmes que celles de la règle de mappage héritée.
- Les règles de mappage remplaçante et héritée sont des mappages Sous-mappe ayant des extracteurs concordants, ou bien elles sont toutes les deux des règles de mappage Déplacer ou Personnalisé.

Si vous créez une règle de mappage Mappes héritées, vous devez spécifier la déclaration de mappage qui contient les règles de mappage à hériter. Vous ne pouvez pas spécifier plusieurs mappes héritées dans une déclaration de mappage.

Une règle de mappage héritée, et la règle ou l'extracteur générés à partir de ce mappage, conservent la même position relative dans l'ordre de traitement que la règle de mappage remplacée et sa règle ou son extracteur générés.

Pour chaque règle de mappage de type Déplacer ou Personnalisé dans une déclaration de mappage, une règle est ajoutée au code source de la transformation générée. Pour chaque règle de mappage de type Sous-mappe, un extracteur de contenu est généré dans le code source de la transformation. Lorsque vous créez une règle de mappage, son type est déterminé par les attributs d'entrée et de sortie que vous sélectionnez. Par exemple, si les attributs d'entrée et de sortie sont des types primitifs compatibles, tels que des chaînes et des entiers, une règle de mappage Déplacer est spécifiée. Si les attributs d'entrée et de sortie sont des types complexes, une règle Sous-mappe est spécifiée. Si aucun des types de règle de mappage Déplacer et Sous-mappe n'est approprié, une règle de mappage Personnalisé est spécifiée.

Pour définir les règles de mappage dans la déclaration de mappage ClasseAClasse :

1. Créez une règle de mappage Déplacer entre les attributs name respectifs des objets d'entrée et de sortie :
  - a. Dans l'éditeur, cliquez sur l'attribut name de l'objet d'entrée Class.
  - b. Faites glisser la poignée de l'attribut name jusqu'à l'attribut name de l'objet de sortie Class.
2. Créez une règle de mappage Sous-mappe entre les attributs ownedOperation respectifs des objets d'entrée et de sortie.
  - a. Dans l'éditeur, cliquez sur l'attribut ownedOperation de l'objet d'entrée Class.
  - b. Faites glisser la poignée de l'attribut ownedOperation jusqu'à l'attribut ownedOperation de l'objet de sortie Class. Etant donné que l'attribut ownedOperation est une collection, la règle créée par défaut est une sous-mappe.
3. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant créer la déclaration de mappage d'opération à opération.

### Créer une déclaration de mappage d'opération à opération

A ce stade de la leçon, la sous-mappe est ornée d'un X dans un cercle rouge, symbole qui signale une erreur. Placez le pointeur de la souris sur ce symbole pour faire apparaître le message d'erreur. Celui-ci indique que la règle de mappage Sous-mappe doit faire référence à une autre déclaration de mappage. Pour remédier à cette situation, vous allez créer une déclaration de mappage d'opération à opération.

Pour créer une déclaration de mappage d'opération à opération :

1. Si la vue Propriétés n'est pas ouverte, dans l'éditeur, cliquez avec le bouton droit sur l'élément Sous-mappe que vous avez créé à l'étape 2 de la section précédente, puis sélectionnez **Afficher dans les propriétés**.
2. Dans la vue Propriétés, sous l'onglet **Détails**, à droite de la zone **Mappe**, cliquez sur **Nouvelle**.
3. Entrez OperationA0operation (sans lettres accentuées) dans la zone **Nom de la mappe** de la fenêtre Nouvelle mappe.
4. Si la zone **Entrée** ne contient pas déjà la valeur Operation, cliquez sur **Parcourir** et, dans le volet Élément de la fenêtre Ajouter une entrée, cliquez sur **Operation**.
5. Si la zone **Sortie** ne contient pas déjà la valeur Operation, cliquez sur **Parcourir** et, dans le volet Élément de la fenêtre Ajouter une sortie, cliquez sur **Operation**.
6. Cliquez sur **OK**. Le symbole d'erreur disparaît et la nouvelle déclaration de mappage d'opération à opération figure maintenant dans la vue Structure. Dans une autre leçon, vous allez créer des règles de mappage dans cette déclaration de mappage.
7. Cliquez sur **Fichier** → **Sauvegarder**.

Dans la leçon suivante, vous allez définir une déclaration de mappage de classe à interface.

## Leçon 4 - Créer et raffiner une déclaration de mappage de classe à interface

Cette leçon vous montre comment créer une déclaration de mappage de classe à interface dans le modèle de mappage. Les règles de mappage contenues dans cette déclaration créent une interface dont le nom est dérivé de celui de la classe dans le modèle source et copient seulement les méthodes publiques de cette classe.

Pour créer une déclaration de mappage de classe à interface dans le modèle de mappage :

1. Si ce n'est déjà fait, ouvrez le fichier Generaliser\_les\_classes.mapping dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages.
2. Dans l'éditeur de mappage de transformation, à la section Racine de mappage, cliquez avec le bouton droit sur **Generaliser\_les\_classes** et sélectionnez **Créer une mappe**.
3. Entrez ClasseAInterface dans la zone **Nom de la mappe** de la fenêtre Nouvelle mappe et cliquez sur **OK**. La mappe apparaît dans la vue Structure et s'ouvre dans l'éditeur, sous la section Racine de mappage.

**Remarque :** Pour voir les détails d'une déclaration de mappage qui n'est pas ouverte dans l'éditeur, développez sa branche en cliquant sur le signe + correspondant dans la vue Structure. Pour ouvrir une déclaration de mappage différente dans l'éditeur, cliquez sur son nom dans la vue Structure.

## Ajouter des objets d'entrée et de sortie à la déclaration de mappage de classe à interface

Après avoir créé la déclaration de mappage de classe à interface, vous spécifiez une classe UML comme objet d'entrée et une interface UML comme objet de sortie.

Pour ajouter un objet d'entrée et un objet de sortie à la déclaration de mappage de classe à interface :

1. Cliquez sur l'icône **Ajouter un objet en entrée**, qui est la première icône de la barre d'outils de la mappe que vous éditez.
2. Dans la fenêtre Ajouter une entrée, développez la branche **uml** dans le volet Élément, cliquez sur **Class**, puis sur **OK**.
3. Cliquez sur l'icône **Ajouter un objet en sortie**, qui est la deuxième icône de la barre d'outils de la mappe que vous éditez.
4. Dans la fenêtre Ajouter une sortie, développez la branche **uml** dans le volet Élément, cliquez sur **Interface**, puis sur **OK**.

## 5. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant définir des règles de mappage entre les attributs de l'objet d'entrée Class et de l'objet de sortie Interface.

## Définir des règles de mappage entre les attributs des objets d'entrée et de sortie

Après avoir ajouté les objets d'entrée et de sortie à la déclaration de mappage de classe à interface, vous pouvez définir les règles de mappage entre les attributs de ces objets.

Dans cette leçon, vous allez créer une règle de mappage personnalisée qui créera une interface appelée *INomClasse*. Vous allez également créer une règle de mappage Sous-mappe qui appellera la sous-mappe opération à opération que vous avez créée dans la «Leçon 3 - Créer et raffiner une déclaration de mappage de classe à classe», à la page 6, et vous ajouterez un filtre d'entrée à cette règle pour spécifier que seules les opérations publiques doivent être transformées dans l'interface cible.

Pour définir les règles de mappage entre les attributs des objets d'entrée et de sortie :

1. Créez une règle de mappage Personnalisé entre les attributs name respectifs des objets d'entrée et de sortie :
  - a. Dans l'éditeur, cliquez sur l'attribut name de l'objet d'entrée Class.
  - b. Faites glisser la poignée de l'attribut name jusqu'à l'attribut name de l'objet de sortie Interface. Une règle de mappage Déplacer est créée.
  - c. Cliquez sur la flèche de l'élément **Déplacer** qui relie les deux attributs name et sélectionnez **Personnalisé**.
  - d. Cliquez avec le bouton droit sur l'élément **Personnalisé** et sélectionnez **Afficher dans les propriétés**.
  - e. Dans la vue Propriétés, sous l'onglet **Détails**, cliquez sur **En ligne** et, dans la zone de texte en dessous de la signature de méthode, tapez le code suivant :

```
Interface_tgt.setName("I"+Class_src.getName());
```

**Remarque :** Pour appeler l'assistant de contenu pendant que vous tapez le code, appuyez sur **Ctrl+Espace**.
  - f. Cliquez sur **Appliquer**.
2. Créez une règle de mappage Sous-mappe entre les attributs ownedOperation respectifs des objets d'entrée et de sortie.
  - a. Dans l'éditeur, cliquez sur l'attribut ownedOperation de l'objet d'entrée Class.
  - b. Faites glisser la poignée de l'attribut ownedOperation jusqu'à l'attribut ownedOperation de l'objet de sortie Interface. Etant donné que l'attribut ownedOperation est une collection, la règle créée par défaut est une sous-mappe.
  - c. Si la vue Propriétés n'est pas ouverte, cliquez avec le bouton droit sur le nouvel élément **Sous-mappe** et sélectionnez **Afficher dans les propriétés**.
  - d. Si ce n'est déjà fait, sélectionnez **OperationAOperation** dans la liste **Mappe**, sous l'onglet **Détails** de la vue Propriétés.
3. A la règle de mappage Sous-mappe que vous avez créée à l'étape 2, ajoutez un filtre d'entrée afin de ne transformer que les opérations publiques dans l'interface cible :
  - a. Dans la vue Propriétés, sous l'onglet **Filtre d'entrée**, cochez la case **Filtrer les éléments d'entrée** et cliquez sur l'option **En ligne**.
  - b. Dans la zone de texte située sous l'option **Code**, après la signature de la méthode, tapez le code suivant :

```
if (ownedOperation_src.getVisibility().equals(VisibilityKind.PUBLIC_LITERAL)) {  
    return true; } return false;
```

**Remarque :** Le code que vous spécifiez pour un filtre d'entrée ou de sortie doit retourner une valeur booléenne. Vous ne pouvez spécifier que le corps de la méthode ; sa signature est définie par l'infrastructure de création de transformations. Pour voir la liste des noms de variable valides, dans la zone de texte en dessous du bouton **En ligne**, appuyez sur **Ctrl+Espace**.

c. Cliquez sur **Appliquer**.

4. Cliquez sur **Fichier** → **Sauvegarder**.

Dans la prochaine leçon, vous allez créer plusieurs déclarations de mappage dans le modèle `Generaliser_les_classes.mapping` ; elles définiront de quelle manière les paramètres et les types primitifs seront transformés lors de l'exécution de la transformation générée.

## Leçon 5 - Créer et raffiner les déclarations de mappage requises par la déclaration de mappage d'opération à opération

Cette leçon vous montre comment créer les déclarations de mappage nécessaires à la déclaration de mappage d'opération à opération pour qu'elle puisse transformer les éléments dans une opération. Par exemple, vous créez une déclaration de mappage qui transforme les paramètres dans le modèle source en paramètres dans le modèle cible ; vous créez également une déclaration de mappage qui définit comment transformer les types primitifs dans le modèle source en types primitifs dans le modèle cible. Enfin, dans la déclaration de mappage d'opération à opération, vous créez les règles de mappage qui appellent les nouvelles déclarations de mappage que vous créez dans la présente leçon.

Le tableau suivant recense les déclarations de mappage et les règles de mappage que vous créez dans cette leçon :

| Déclaration de mappage | Type d'objet d'entrée et de sortie | Attributs mappés | Type de règle de mappage | Description de la règle de mappage  |
|------------------------|------------------------------------|------------------|--------------------------|---|
| PrimitifAPrimitif      | Type primitif UML                  | name             | Déplacer                 | Crée un type primitif dans l'élément dont la sous-mappe invoque la déclaration de mappage PrimitifAPrimitif |

| Déclaration de mappage | Type d'objet d'entrée et de sortie | Attributs mappés | Type de règle de mappage | Description de la règle de mappage   |
|------------------------|------------------------------------|------------------|--------------------------|--|
| ParametreAParametre    | Paramètre UML                      | name, visibility | Déplacer                 | Crée un paramètre dans l'élément dont la sous-mappe invoque la déclaration de mappage ParametreAParametre ; le paramètre généré a le même nom et la même visibilité que le paramètre dans l'élément source   |
|                        |                                    | type             | Sous-mappe               | Pour chaque type dans l'élément dont la sous-mappe invoque cette déclaration de mappage : <ul style="list-style-type: none"> <li>• Si le paramètre est d'un type primitif, cette règle invoque la déclaration de mappage PrimitifAPrimitif</li> <li>• Si le paramètre est d'un type classe, cette règle invoque les déclarations de mappage ClasseAClasse et ClasseAInterface</li> </ul> |
| OperationAOperation    | Opération UML                      | name, visibility | Déplacer                 | Crée une opération dans l'élément dont la sous-mappe invoque la déclaration de mappage OperationAOperation ; l'opération générée a le même nom et la même visibilité que l'opération dans l'élément source   |
|                        |                                    | ownedParameter   | Sous-mappe               | Pour chaque paramètre dans l'opération, cette sous-mappe invoque la déclaration de mappage ParametreAParametre   |

Dans ce tutoriel, la déclaration de mappage ParametreAParametre invoque la déclaration de mappage PrimitifAPrimitif pour créer des paramètres d'un type primitif.

Pour créer la déclaration de mappage de type primitif à type primitif dans le modèle de mappage :

1. Si ce n'est déjà fait, ouvrez le fichier `Generaliser_les_classes.mapping` dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages (dossier `model` du projet).
2. Dans l'éditeur de mappage de transformation, à la section Racine de mappage, cliquez avec le bouton droit sur **Generaliser\_les\_classes** et sélectionnez **Créer une mappe**.
3. Entrez `PrimitifAPrimitif` dans la zone **Nom de la mappe** de la fenêtre Nouvelle mappe et cliquez sur **OK**. La mappe apparaît dans la vue Structure et s'ouvre dans l'éditeur, sous la section Racine de mappage.

## Ajouter des objets d'entrée et de sortie à la déclaration de mappage `PrimitifAPrimitif`

Une fois votre déclaration de mappage `PrimitifAPrimitif` créée, vous devez lui ajouter un objet d'entrée et un objet de sortie. Dans cette leçon, vous allez spécifier un type primitif UML à la fois pour l'objet d'entrée et pour l'objet de sortie.

Pour ajouter un objet d'entrée et un objet de sortie à la déclaration de mappage `PrimitifAPrimitif` :

1. Cliquez sur l'icône **Ajouter un objet en entrée**, qui est la première icône de la barre d'outils de la mappe que vous éditez.
2. Dans la fenêtre Ajouter une entrée, développez la branche **uml** dans le volet Élément, cliquez sur **PrimitiveType**, puis sur **OK**.
3. Cliquez sur l'icône **Ajouter un objet en sortie**, qui est la deuxième icône de la barre d'outils de la mappe que vous éditez.
4. Dans la fenêtre Ajouter une sortie, développez la branche **uml** dans le volet Élément, cliquez sur **PrimitiveType**, puis sur **OK**.
5. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant définir une règle de mappage entre les attributs `name` respectifs des objets d'entrée et de sortie `PrimitiveType`.

## Définir des règles de mappage entre les attributs des objets d'entrée et de sortie `PrimitiveType`

Pour cette leçon, créez une règle de mappage Déplacer afin de créer un type primitif dans le modèle cible. Le type primitif généré porte le même nom que le type primitif dans le modèle d'entrée ; cette règle revient plus ou moins à créer une copie du type primitif.

Pour définir une règle de mappage Déplacer établissant une relation entre les attributs `name` respectifs des objets d'entrée et de sortie `PrimitiveType` :

1. Dans l'éditeur, cliquez sur l'attribut `name` de l'objet d'entrée `PrimitiveType`.
2. Faites glisser la poignée de l'attribut `name` jusqu'à l'attribut `name` de l'objet de sortie.
3. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant créer une déclaration de mappage de paramètre à paramètre.

## Créer une déclaration de mappage de paramètre à paramètre

Cette section vous apprend à créer une déclaration de mappage qui spécifie un paramètre UML à la fois pour l'objet d'entrée et pour l'objet de sortie. Cette déclaration de mappage de paramètre à paramètre contient des règles de mappage qui, à l'exécution de la transformation générée, créent dans le modèle cible un paramètre ayant les mêmes nom, visibilité et type que le paramètre dans l'élément dont la sous-mappe a invoqué cette déclaration de mappage. Dans le présent tutoriel, la déclaration de mappage `OperationAOperation` contient une règle de mappage Sous-mappe qui invoque cette déclaration de mappage de paramètre à paramètre.

Pour créer une déclaration de mappage de paramètre à paramètre :

1. Si ce n'est déjà fait, ouvrez le fichier `Generaliser_les_classes.mapping` dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages (dossier `model` du projet).
2. Dans l'éditeur de mappage de transformation, à la section Racine de mappage, cliquez avec le bouton droit sur **Generaliser\_les\_classes** et sélectionnez **Créer une mappe**.
3. Entrez `ParametreAParametre` (sans lettres accentuées) dans la zone **Nom de la mappe** de la fenêtre Nouvelle mappe, puis cliquez sur **OK**. La mappe apparaît dans la vue Structure et s'ouvre dans l'éditeur, sous la section Racine de mappage.

## Ajouter des objets d'entrée et de sortie à la déclaration de mappage `ParametreAParametre`

Une fois votre déclaration de mappage créée, vous devez lui ajouter un objet d'entrée et un objet de sortie. Dans cette leçon, vous allez spécifier un paramètre UML à la fois pour l'objet d'entrée et pour l'objet de sortie.

Pour ajouter un objet d'entrée et un objet de sortie à la déclaration de mappage `ParametreAParametre` :

1. Cliquez sur l'icône **Ajouter un objet en entrée**, qui est la première icône de la barre d'outils de la mappe que vous éditez.
2. Dans la fenêtre Ajouter une entrée, développez la branche **uml** dans le volet Élément, cliquez sur **Parameter**, puis sur **OK**.
3. Cliquez sur l'icône **Ajouter un objet en sortie**, qui est la deuxième icône de la barre d'outils de la mappe que vous éditez.
4. Dans la fenêtre Ajouter une sortie, développez la branche **uml** dans le volet Élément, cliquez sur **Parameter**, puis sur **OK**.
5. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant définir des règles de mappage entre les attributs des objets d'entrée et de sortie `Parameter`.

## Définir des règles de mappage dans la déclaration de mappage `ParametreAParametre`

Après avoir ajouté les objets d'entrée et de sortie à la déclaration de mappage, vous pouvez définir les règles de mappage entre les attributs de ces objets.

Dans cette section, vous créez des règles Sous-mappe et une règle Déplacer qui, à l'exécution de la transformation générée, créent dans le modèle cible un paramètre ayant les mêmes nom, visibilité et type que le paramètre dans un élément du modèle source. Dans le présent tutoriel, c'est la déclaration de mappage `OperationAOperation` qui invoque cette déclaration de mappage de paramètre à paramètre.

Pour définir les règles de mappage dans la déclaration de mappage `ParametreAParametre` :

1. Créez une règle de mappage Déplacer entre les attributs `name` respectifs des objets d'entrée et de sortie :
  - a. Dans l'éditeur, cliquez sur l'attribut `name` de l'objet d'entrée `Parameter`.
  - b. Faites glisser la poignée de l'attribut `name` jusqu'à l'attribut `name` de l'objet de sortie.
2. Répétez les étapes 1a et 1b, mais en remplaçant `name` par `visibility`.
3. Créez une règle de mappage Sous-mappe entre les attributs `type` respectifs des objets d'entrée et de sortie :
  - a. Dans l'éditeur, cliquez sur l'attribut `type` de l'objet d'entrée `Parameter`.
  - b. Faites glisser la poignée de l'attribut `type` jusqu'à l'attribut `type` de l'objet de sortie. Comme cet attribut est un type complexe, une sous-mappe est créée par défaut.

- c. Si la vue Propriétés n'est pas ouverte, cliquez avec le bouton droit sur le nouvel élément **Sous-mappe** dans l'éditeur et sélectionnez **Afficher dans les propriétés**.
- d. Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **PrimitifAPrimitif** dans la liste **Mappe**.

Cette règle de mappage génère un code qui invoque la transformée PrimitifAPrimitif lorsqu'un paramètre primitif est rencontré dans le modèle source. Si le paramètre dans le modèle source n'est pas un type primitif, il n'est pas transformé dans le modèle cible.

4. Répétez les étapes 3a, 3b et 3c, puis effectuez l'étape suivante :
  - Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **ClasseAClasse** dans la liste **Mappe**.

Cette règle de mappage Sous-mappe génère un code qui crée un paramètre du type classe dans un élément du modèle cible si un paramètre du type classe est rencontré dans le modèle source.

5. Répétez les étapes 3a, 3b et 3c, puis effectuez l'étape suivante :
  - Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **ClasseAInterface** dans la liste **Mappe**.

Cette règle de mappage Sous-mappe crée un paramètre du type interface dans un élément du modèle cible si un paramètre du type classe est rencontré dans le modèle source.

6. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant définir les règles de mappage dans la déclaration de mappage d'opération à opération que vous avez créée à la «Leçon 3 - Créer et raffiner une déclaration de mappage de classe à classe», à la page 6.

## Définir des règles de mappage dans la déclaration de mappage **OperationAOperation**

Dans la «Leçon 3 - Créer et raffiner une déclaration de mappage de classe à classe», à la page 6, vous avez vu comment créer une déclaration de mappage d'opération à opération. Vous êtes maintenant prêt à créer des règles de mappage dans cette déclaration de mappage. A l'exécution de la transformation générée, ces règles créeront dans le modèle cible une opération ayant les mêmes nom, visibilité et paramètres que l'opération dans le modèle source. Dans le présent tutoriel, cette déclaration de mappage d'opération à opération est invoquée par les déclarations de mappage **ClasseAClasse** et **ClasseAInterface**.

Pour définir les règles de mappage dans la déclaration de mappage **OperationAOperation** :

1. Si ce n'est déjà fait, ouvrez le fichier **Generaliser\_les\_classes.mapping** dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages (dossier **model** du projet).
2. Dans la vue Structure, cliquez sur la déclaration de mappage **OperationAOperation**.
3. Créez une règle de mappage **Déplacer** entre les attributs **name** respectifs des objets d'entrée et de sortie :
  - a. Dans l'éditeur, cliquez sur l'attribut **name** de l'objet d'entrée **Operation**.
  - b. Faites glisser la poignée de l'attribut **name** jusqu'à l'attribut **name** de l'objet de sortie.
4. Répétez les étapes 3a et 3b, mais en remplaçant **name** par **visibility**.
5. Créez une règle de mappage **Sous-mappe** entre les attributs **ownedParameter** respectifs des objets d'entrée et de sortie :
  - a. Dans l'éditeur, cliquez sur l'attribut **ownedParameter** de l'objet d'entrée **Operation**.
  - b. Faites glisser la poignée de l'attribut **ownedParameter** jusqu'à l'attribut **ownedParameter** de l'objet de sortie. Comme cet attribut est une collection, une sous-mappe est créée par défaut.
  - c. Si la vue Propriétés n'est pas ouverte, cliquez avec le bouton droit sur le nouvel élément **Sous-mappe** dans l'éditeur et sélectionnez **Afficher dans les propriétés**.
  - d. Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **ParametreAParametre** dans la liste **Mappe**.
6. Cliquez sur **Fichier** → **Sauvegarder**.

Dans la leçon suivante, vous allez créer une déclaration de mappage de package à package.

## Leçon 6 - Créer et raffiner une déclaration de mappage de package à package

Cette leçon vous montre comment créer une déclaration de mappage de package à package et plusieurs règles de mappage. Ces règles spécifient comment la transformée générée doit traiter les packages imbriqués ou les classes dans le package que contient le modèle source.

Les objets d'entrée et de sortie Package comportent une fonction appelée `packagedElement`. Il s'agit d'une collection qui contient différents types d'objets UML valides. Les règles de mappage que vous créez dans cette leçon définissent comment la transformation traite les éléments de la collection qui sont du type package ou classe.

Dans cette leçon, vous créez les règles de mappage suivantes :

- Une règle de mappage Déplacer qui crée un package dans le modèle cible ; ce package porte le même nom que le package dans le modèle source
- Une règle de mappage Sous-mappe qui invoque la transformée `ClasseAClasseTransform` si l'élément est du type classe
- Une règle de mappage Sous-mappe qui invoque la transformée `ClasseAInterfaceTransform` si l'élément est du type classe
- Une règle de mappage Sous-mappe qui invoque la transformée `PackageAPackageTransform` si l'élément est du type package

Dans la «Leçon 3 - Créer et raffiner une déclaration de mappage de classe à classe», à la page 6, vous avez vu que, pour chaque règle de mappage Déplacer définie dans une déclaration de mappage, une règle copiant une valeur d'attribut du modèle source au modèle cible est ajoutée au code source de la transformée générée. Pour chaque règle de mappage Sous-mappe, un extracteur est généré dans le code source de la transformée afin d'extraire les éléments dans la collection spécifiée. La règle Sous-mappe est appliquée à un objet si l'objet d'entrée courant est une instance du type d'entrée qui est défini dans la déclaration de mappage.

Lorsque vous exécutez la transformation générée, si le modèle source contient un package, la transformée `PackageAPackageTransform` est invoquée pour créer un package du même nom dans le modèle cible. La transformée `PackageAPackageTransform` traverse la collection de la fonction `packagedElement`. Pour chaque élément de cette collection qui est du type package (ce qui signifie que le modèle source contient des packages imbriqués), la transformée `PackageAPackageTransform` est invoquée. Pour chaque élément de la collection qui est du type classe, la transformée invoque les règles qui visent à transformer la classe en une classe et une interface correspondantes dans le modèle cible.

Pour créer une déclaration de mappage de package à package dans le modèle de mappage :

1. Si ce n'est déjà fait, ouvrez le fichier `Generaliser_les_classes.mapping` dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages.
2. Dans l'éditeur de mappage de transformation, à la section Racine de mappage, cliquez avec le bouton droit sur **Generaliser\_les\_classes** et sélectionnez **Créer une mappe**.
3. Entrez `PackageAPackage` dans la zone **Nom de la mappe** de la fenêtre Nouvelle mappe, puis cliquez sur **OK**. La mappe apparaît dans la vue Structure et s'ouvre dans l'éditeur, sous la section Racine de mappage.

## Ajouter des objets d'entrée et de sortie à la déclaration de mappage de package à package

Une fois votre déclaration de mappage créée, vous devez lui ajouter un objet d'entrée et un objet de sortie. Dans cette leçon, vous allez spécifier un package UML à la fois pour l'objet d'entrée et pour l'objet de sortie.

Pour ajouter un objet d'entrée et un objet de sortie à la déclaration de mappage de package à package :

1. Cliquez sur l'icône **Ajouter un objet en entrée**, qui est la première icône de la barre d'outils de la mappe que vous éditez.
2. Dans la fenêtre Ajouter une entrée, développez la branche **uml** dans le volet Élément, cliquez sur **Package**, puis sur **OK**.
3. Cliquez sur l'icône **Ajouter un objet en sortie**, qui est la deuxième icône de la barre d'outils de la mappe que vous éditez.
4. Dans la fenêtre Ajouter une sortie, développez la branche **uml** dans le volet Élément, cliquez sur **Package**, puis sur **OK**.
5. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant définir les règles de mappage entre les fonctions des objets d'entrée et de sortie.

## Définir des règles de mappage entre les attributs des objets d'entrée et de sortie Package

Après avoir ajouté les objets d'entrée et de sortie Package à la déclaration de mappage, vous pouvez créer les règles de mappage entre les attributs de ces objets.

**Remarque :** Lorsque vous générez le code source de transformation, une transformée est générée pour chaque déclaration de mappage. Chaque transformée générée contient une méthode `getAccept_Condition()`. Par défaut, cette méthode renvoie `true` si l'élément d'entrée courant est une instance de l'objet d'entrée qui est défini dans la déclaration de mappage. Si la méthode renvoie `false`, la transformée ne traite pas l'élément d'entrée courant. Dans le cadre de cette leçon, vous pouvez créer des filtres d'entrée qui vérifient également si l'objet d'entrée est une instance du type d'entrée. Ces filtres d'entrée sont redondants, mais ils sont inoffensifs et leur seul but est de montrer comment créer des filtres d'entrée. Les filtres d'entrée dans les règles de mappage Sous-mappe doivent être écrits en Java et renvoyer une valeur booléenne.

Pour créer les règles de mappage entre les attributs des objets d'entrée et de sortie Package :

1. Créez une règle de mappage Déplacer entre les attributs `name` respectifs des objets d'entrée et de sortie. Pour cela, faites glisser la poignée de l'attribut `name` de l'objet d'entrée jusqu'à l'attribut `name` de l'objet de sortie.
2. Créez une règle de mappage Sous-mappe entre les attributs `packagedElement` respectifs des objets d'entrée et de sortie, et spécifiez que cette sous-mappe invoque la transformée `ClasseAClasseTransform` si l'élément est du type classe :
  - a. Faites glisser la poignée de l'attribut `packagedElement` de l'objet d'entrée jusqu'à l'attribut `packagedElement` de l'objet de sortie. Etant donné que l'attribut `packagedElement` est une collection, la règle créée par défaut est une sous-mappe.
  - b. Dans l'éditeur, cliquez avec le bouton droit sur le nouvel élément **Sous-mappe** et sélectionnez **Afficher dans les propriétés**.
  - c. Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **ClasseAClasse** dans la liste **Mappe**.A l'exécution de la transformation générée, pour chaque élément rencontré dans le package d'entrée, la transformée `ClasseAClasseTransform` sera invoquée, mais seulement si l'élément en question est du type classe.
3. Facultatif : Dans la règle Sous-mappe que vous avez créée à l'étape 2, créez un filtre d'entrée qui vérifie si l'objet courant est une instance d'une classe :
  - a. Dans l'éditeur, cliquez avec le bouton droit sur l'élément **Sous-mappe** que vous avez créé à l'étape 2.
  - b. Dans la vue Propriétés, sous l'onglet **Filtre d'entrée**, cochez la case **Filtrer les éléments d'entrée**.
  - c. Cliquez sur **En ligne**.

- d. Dans la zone de texte située sous l'option **Code**, après la signature de la méthode, tapez le code suivant : `return packagedElement_src instanceof org.eclipse.uml2.uml.Class;`
- e. Cliquez sur **Appliquer**.
4. Créez une règle de mappage Sous-mappe entre les attributs `packagedElement` respectifs des objets d'entrée et de sortie, et spécifiez que si l'élément courant dans la collection `packagedElement` est du type classe, la transformée `PackageAPackageTransform` transforme cette classe en une interface dans le modèle cible.
  - a. Répétez l'étape 2a pour créer une autre sous-mappe entre les fonctions `packagedElement` respectives des objets d'entrée et de sortie.
  - b. Dans l'éditeur, cliquez avec le bouton droit sur le nouvel élément **Sous-mappe** et sélectionnez **Afficher dans les propriétés**.
  - c. Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **ClasseAInterface** dans la liste **Mappe**.
5. Facultatif : Dans la règle Sous-mappe que vous avez créée à l'étape 4, créez un filtre d'entrée qui vérifie si l'objet courant est une instance d'une interface :
  - a. Dans l'éditeur, cliquez avec le bouton droit sur l'élément **Sous-mappe** que vous avez créé à l'étape 4 et sélectionnez **Afficher dans les propriétés**.
  - b. Répétez les étapes 3b, 3c, 3d et 3e pour spécifier le même filtre d'entrée.
6. Créez une règle de mappage Sous-mappe pour traiter les packages imbriqués. Si l'élément courant dans la collection `packagedElement` est un package, la transformation invoque la transformée `PackageAPackageTransform`.
  - a. Répétez l'étape 2a pour créer une autre sous-mappe entre les fonctions `packagedElement` respectives des objets d'entrée et de sortie.
  - b. Dans l'éditeur, cliquez avec le bouton droit sur le nouvel élément **Sous-mappe** et sélectionnez **Afficher dans les propriétés**.
  - c. Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **PackageAPackage** dans la liste **Mappe**.
7. Facultatif : Dans la règle Sous-mappe que vous avez créée à l'étape 6, créez un filtre d'entrée qui vérifie si l'élément d'entrée courant est une instance d'un package :
  - a. Dans l'éditeur, cliquez avec le bouton droit sur l'élément **Sous-mappe** que vous avez créé à l'étape 6 et sélectionnez **Afficher dans les propriétés**.
  - b. Répétez les étapes 3b et 3c pour spécifier que cette règle comporte un filtre d'entrée.
  - c. Dans la zone de texte située sous l'option **Code**, après la signature de la méthode, tapez le code suivant : `return packagedElement_src instanceof org.eclipse.uml2.uml.Package;`
  - d. Cliquez sur **Appliquer**.
8. Cliquez sur **Fichier** → **Sauvegarder**.

La déclaration de mappage contient une règle Déplacer entre les attributs `name` et plusieurs règles Sous-mappe entre les attributs `packagedElement`.

## Leçon 7 - Créer et raffiner une déclaration de mappage de modèle à modèle

Cette leçon vous montre comment créer une déclaration de mappage de modèle à modèle dans le modèle de mappage. Cette déclaration de mappage contient une règle de mappage Personnalisé qui, à l'exécution de transformation générée, crée un modèle de sortie cible dont le nom est dérivé de celui du modèle d'entrée source. Dans cette leçon, vous découvrez également comment changer l'ordre dans lequel la transformation traite les déclarations de mappage dans le modèle de mappage.

Pour renommer le modèle cible généré par la transformation, vous pouvez créer une règle de mappage Personnalisé entre les objets d'entrée et de sortie au lieu de créer une règle de mappage entre les fonctions (attributs) de ces objets.

Pour créer une déclaration de mappage de modèle à modèle :

1. Si ce n'est déjà fait, ouvrez le fichier .mapping du modèle de mappage dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages (dossier model du projet).
2. Dans l'éditeur de mappage de transformation, à la section Racine de mappage, cliquez avec le bouton droit sur **Generaliser les classes** et sélectionnez **Créer une mappe**.
3. Entrez `ModeleAModele` (sans lettres accentuées) dans la zone **Nom de la mappe** de la fenêtre Nouvelle mappe, puis cliquez sur **OK**. La mappe apparaît dans la vue Structure et s'ouvre dans l'éditeur, sous la section Racine de mappage.

## Ajouter des objets d'entrée et de sortie à la déclaration de mappage de modèle à modèle

Une fois votre déclaration de mappage créée, vous devez lui ajouter un objet d'entrée et un objet de sortie. Dans cette leçon, vous allez spécifier un modèle à la fois pour l'objet d'entrée et pour l'objet de sortie.

Pour ajouter un objet d'entrée et un objet de sortie à la déclaration de mappage de modèle à modèle :

1. Cliquez sur l'icône **Ajouter un objet en entrée**, qui est la première icône de la barre d'outils de la mappe que vous éditez.
2. Dans la fenêtre Ajouter une entrée, développez la branche **uml** dans le volet Élément, cliquez sur **Model**, puis sur **OK**.
3. Cliquez sur l'icône **Ajouter un objet en sortie**, qui est la deuxième icône de la barre d'outils de la mappe que vous éditez.
4. Dans la fenêtre Ajouter une sortie, développez la branche **uml** dans le volet Élément, cliquez sur **Model**, puis sur **OK**.
5. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant définir les règles de mappage entre les objets d'entrée et de sortie.

## Définir les règles de mappage Personnalisé et Sous-mappe

Dans cette leçon, vous créez les règles de mappage suivantes :

- Une règle Personnalisé qui renomme le modèle cible généré par la transformation
- Une règle Sous-mappe qui invoque la transformée `PackageAPackageTransform` afin de transformer les packages dans le modèle

Pour créer les règles de mappage dans la déclaration de mappage :

1. Créez une règle Personnalisé qui renomme le modèle cible :
  - a. Dans l'éditeur, cliquez sur le compartiment supérieur de l'objet d'entrée Model. L'intégralité de l'objet est sélectionnée.
  - b. Faites glisser la poignée de l'objet d'entrée jusqu'à l'objet de sortie.
  - c. Cliquez sur la flèche du nouvel élément **Sous-mappe** et sélectionnez **Personnalisé**.
  - d. Cliquez avec le bouton droit sur l'élément **Personnalisé** et sélectionnez **Afficher dans les propriétés**.
  - e. Dans la vue Propriétés, sous l'onglet **Détails**, cliquez sur **En ligne**.
  - f. Dans la zone de texte située sous l'option **Code**, après la signature de la méthode, tapez le code suivant, qui implémente la règle de mappage Personnalisé :
 

```
Model_tgt.setName(Model_src.getName()+"ModeleCible");
```
  - g. Cliquez sur **Appliquer**.

**Remarque :** Pour renommer le modèle cible, au lieu de créer une règle de mappage entre les objets d'entrée et de sortie, vous pourriez créer une règle de mappage Personnalisé entre leurs attributs name respectifs et spécifier le même code sous l'onglet **Détails**.

2. Créez une règle de mappage Sous-mappe qui invoque le mappage PackageAPackage que vous avez créé à la «Leçon 6 - Créer et raffiner une déclaration de mappage de package à package», à la page 17 :
  - a. Faites glisser la poignée de l'attribut packagedElement de l'objet d'entrée jusqu'à l'attribut packagedElement de l'objet de sortie.
  - b. Cliquez avec le bouton droit sur le nouvel élément **Sous-mappe** et sélectionnez **Afficher dans les propriétés**.
  - c. Dans la vue Propriétés, sous l'onglet **Détails**, sélectionnez **PackageAPackage** dans la liste **Mappe**.
3. Cliquez sur **Fichier** → **Sauvegarder**.

Vous avez créé toutes les déclarations de mappage et toutes les règles de mappage nécessaires à ce tutoriel. Vous pouvez à présent spécifier l'ordre de traitement des déclarations de mappage.

## Changer l'ordre de traitement des déclarations de mappage

Vous pouvez définir l'ordre de traitement des déclarations de mappage dans un modèle de mappage. La vue Structure présente les déclarations de mappage dans l'ordre suivant lequel elles seront traitées à l'exécution de la transformation générée. En modifiant cet ordre, vous pouvez spécifier des instructions de traitement pour les objets d'entrée susceptibles d'être traités et consommés par une déclaration de mappage moins spécifique.

Pour ce tutoriel, vous allez spécifier l'ordre de traitement suivant pour les déclarations de mappage :

- ModeleAModele
- PackageAPackage
- ClasseAClasse
- ClasseAInterface
- OperationAOperation
- ParametreAParametre
- PrimitifAPrimitif

**Remarque :** Dans ce tutoriel, il est très important que la déclaration ModeleAModele soit traitée avant la déclaration PackageAPackage. En effet, comme un modèle est avant tout un package, la transformée PackageAPackage accepte aussi bien un modèle qu'un package comme objet d'entrée. Si la transformée PackageAPackage est en tête de liste dans la transformation, c'est elle qui traite et consomme l'objet d'entrée Model. Cet objet n'est donc pas traité par la transformée ModeleAModele comme il le devrait et la règle personnalisée que vous avez créée dans cette transformée n'est pas appliquée.

Pour changer l'ordre de traitement des déclarations de mappage :

1. Si la vue Structure n'est pas visible, sélectionnez **Fenêtre** → **Afficher la vue** → **Structure**.
2. Dans la vue Structure, cliquez avec le bouton droit sur la déclaration de mappage ModeleAModele, puis sélectionnez **Ordre d'exécution** et cliquez sur **Déplacer vers le haut**. Répétez cette étape jusqu'à ce que la déclaration de mappage ModeleAModele soit la première de la liste.
3. Répétez l'étape 2 pour les autres déclarations de mappage afin de les classer dans l'ordre indiqué plus haut.
4. Cliquez sur **Fichier** → **Sauvegarder**.

Vous êtes maintenant prêt à générer le code source de la transformation.

---

## Module 2 - Générer le code de la transformation et l'exécuter

Dans ce module, vous générez le code de la transformation de modèle à modèle, puis vous testez cette transformation en l'exécutant dans une instance séparée du plan de travail Eclipse.

## Objectifs d'apprentissage

Les leçons de ce module vous montrent comment effectuer les tâches suivantes :

- Générer et compiler le code source de transformation
- Configurer un plan de travail d'exécution
- Créer une configuration de transformation
- Exécuter le code de transformation dans un plan de travail d'exécution

## Durée

Ce module vous prendra environ 20 minutes.

## Conditions préalables

Vous devez savoir effectuer les tâches suivantes :

- Créer des modèles UML dans des projets
- Tâches élémentaires de modélisation UML

## Leçon 1 - Générer et compiler le code source de transformation

Cette leçon vous apprend à générer et compiler le code source de transformation.

Avant de générer le code source, examinez le contenu du dossier src dans le projet Generaliser les classes. Ces packages et certains fichiers ont été générés lorsque vous avez créé le projet ; d'autres fichiers ont été créés lorsque vous avez édité le fichier Generaliser\_les\_classes.mapping.

Vous pouvez graduellement ajouter des règles de mappage et générer leurs implémentations respectives dans une déclaration de mappage. Vous n'avez pas à définir toutes les règles de mappage avant de générer le code source de transformation. Par exemple, après avoir terminé ce tutoriel, vous pourriez poursuivre le développement de la transformation et ajouter des règles de mappage en vue de créer une relation de réalisation de la classe d'implémentation à l'interface.

Lorsque vous générez le code source de transformation, pour chaque déclaration de mappage présente dans le modèle de mappage, l'infrastructure de création de transformations génère un fichier source Java ayant pour nom *n*Transform.java, la variable *n* représentant ici le nom de la déclaration de mappage. Ensemble, ces fichiers Java forment le code de transformation. En plus de générer le code d'implémentation de la transformation, l'infrastructure de création de transformations génère le code nécessaire à l'enregistrement de la transformation auprès du service de transformation.

Pour générer et compiler le code source de transformation :

1. Si ce n'est déjà fait, ouvrez la perspective Développement de plug-in en sélectionnant **Fenêtre → Ouvrir la perspective → Autre → Développement de plug-in** et en cliquant sur **OK**.
2. Si ce n'est déjà fait, ouvrez le fichier Generaliser\_les\_classes.mapping dans l'éditeur en faisant un double clic dessus dans la vue Explorateur de packages.
3. Dans l'éditeur, cliquez avec le bouton droit sur la section Racine de mappage et sélectionnez **Générer un code source de transformation**.

**Conseil :** Vous pouvez aussi générer le code source de transformation en effectuant les étapes suivantes : dans la vue Explorateur de packages, cliquez avec le bouton droit sur le fichier .mapping file et sélectionnez **Transformer → Générer le code de transformation**.

Dans la vue Explorateur de packages, examinez les packages et les fichiers dans le dossier src. Les transformées générées se trouvent dans le sous-dossier src/generaliser\_les\_classes.transforms.

4. Pour compiler le code source généré, cliquez sur le projet **Generaliser les classes** dans la vue Explorateur de packages et, sur la barre de menus, sélectionnez **Projet → Générer le projet**. Par défaut,

les projets Eclipse sont générés (compilés) automatiquement lorsque vous sauvegardez les modifications apportées à un projet. Si vous n'avez rien changé aux préférences de génération pour l'espace de travail ou le projet, vous n'avez pas à effectuer cette étape.

**Conseil :** Pour changer les préférences de génération des projets, sélectionnez **Fenêtre → Préférences**, développez la catégorie **Général** et cliquez sur **Espace de travail**. Une fois les préférences modifiées, cliquez sur **OK**.

5. Examinez le contenu du dossier src pour voir si des erreurs sont signalées.
6. Si une erreur de compilation est signalée dans le code de la transformée ClasseAInterface, cela signifie peut-être que vous devez importer le package VisibilityKind :
  - a. Dans la vue Explorateur de packages, développez la branche du fichier src/generaliser\_les\_classes.transforms/ClasseAInterfaceTransform.java et faites un double clic sur la méthode ornée du symbole d'erreur.
  - b. Dans l'éditeur, cliquez avec le bouton droit sur le symbole d'erreur dans la marge gauche, sélectionnez **Correctif rapide** et faites un double clic sur **Importer "VisibilityKind" (org.eclipse.uml2.uml)**.
  - c. Cliquez sur **Fichier → Sauvegarder**.
7. Dans la vue Explorateur de packages, cliquez sur le projet **Generaliser les classes** et, sur la barre de menus, sélectionnez **Projet → Générer le projet**.

Examinez les fichiers dans le dossier src/generaliser\_les\_classes.transforms du projet. Une transformée Java est générée pour chaque déclaration de mappage définie dans le modèle de mappage. Dans le fichier MainTransform.java, la méthode MainTransform ajoute une instance de chaque transformée générée en respectant l'ordre que vous avez défini dans la vue Structure.

Vous pouvez à présent configurer un plan de travail d'exécution.

## Leçon 2 - Configurer un plan de travail d'exécution

Cette leçon vous montre comment configurer un plan de travail d'exécution et y créer des modèles UML source et cible.

Vous pouvez créer et invoquer un plan de travail d'exécution pour tester et déboguer une transformation, ce qui signifie que le plug-in de la transformation n'a pas besoin d'être mis en package avant d'être testé.

Après avoir créé et ouvert le plan de travail d'exécution, vous devez y créer les modèles source et cible qui serviront à tester le code de transformation. Ce code transforme les éléments que vous créez dans le modèle source et génère la sortie correspondante dans le modèle cible.

Pour configurer un plan de travail d'exécution :

1. Ouvrez la perspective Développement de plug-in : cliquez sur **Fenêtre → Ouvrir la perspective → Développement de plug-in**.
2. Sur la barre de menus, sélectionnez **Exécuter → Ouvrir la boîte de dialogue Exécuter**.
3. Dans le volet gauche de la fenêtre Exécuter, cliquez sur **Application Eclipse**, puis sur l'icône **Nouvelle configuration de lancement** (la première icône de la barre d'outils en partant de la gauche).
4. Cliquez sur l'onglet **Configuration**.
5. Cliquez sur **Utiliser un fichier config.ini existant comme modèle** et conservez la valeur par défaut dans la zone **Emplacement**. Vous spécifiez ainsi que l'instance d'exécution est une instance du produit que vous utilisez, et non une instance Eclipse par défaut. Une instance d'exécution Eclipse par défaut n'offre pas suffisamment de fonctionnalités pour permettre le test d'une transformation.
6. Cliquez sur **Appliquer**.
7. Cliquez sur **Exécuter**.

**Remarque :** Si des projets dans l'espace de travail contiennent des erreurs, une boîte de dialogue s'affiche pour vous en informer. Pour poursuivre l'ouverture du plan de travail d'exécution, cliquez sur **Continuer** (ou Commencer).

Le nouveau plan de travail d'exécution s'ouvre.

**Remarque :** L'ouverture du plan de travail d'exécution peut prendre quelques minutes, selon les ressources système disponibles.

## Créer un projet de test dans le plan de travail d'exécution

Après avoir configuré et ouvert le plan de travail d'exécution, vous devez y créer un projet contenant les éléments suivants :

- Un modèle UML source à transformer par la transformation ; il contient un package qui lui-même contient une classe.
- Un modèle cible vide, dans lequel la transformation génère sa sortie.

Pour créer le projet qui contient les modèles source et cible :

1. Dans le plan de travail d'exécution, ouvrez la perspective Modélisation : cliquez sur **Fenêtre → Ouvrir la perspective → Modélisation**.
2. Créez un projet de modélisation UML appelé TestTransformation et un modèle UML appelé ModeleSource :
  - a. Cliquez sur **Fichier → Nouveau → Projet**, développez la catégorie **Modélisation** et cliquez sur **Projet UML**, puis sur **Suivant**.
  - b. Sur la page Créer un projet de modèle, dans la zone **Nom du projet**, tapez TestTransformation. Conservez les valeurs par défaut des autres zones et cliquez sur **Suivant**.
  - c. Sur la page Créer un modèle, cliquez sur la catégorie **Général** si elle n'est pas déjà sélectionnée.
  - d. Dans le volet Canevas, cliquez sur **Modèle vierge**.
  - e. Dans la zone **Nom de fichier**, tapez ModeleSource (sans lettre accentuée).
  - f. Cliquez sur **Terminer**.
  - g. Si vous êtes invité à basculer vers la perspective Modélisation, cliquez sur **Oui**.
3. Dans le projet TestTransformation, créez un modèle UML appelé ModeleCible :
  - a. Dans la vue Explorateur de projets, cliquez avec le bouton droit sur le projet TestTransformation et sélectionnez **Nouveau → Modèle UML**.
  - b. Dans l'assistant Modèle UML, sur la page Créer un modèle, conservez les valeurs par défaut et cliquez sur **Suivant**.
  - c. Sur la deuxième page Créer un modèle, cliquez sur la catégorie **Général** si elle n'est pas déjà sélectionnée.
  - d. Dans le volet Canevas, cliquez sur **Modèle vierge**.
  - e. Dans la zone **Nom de fichier**, tapez ModeleCible (sans lettre accentuée).
  - f. Cliquez sur **Terminer**.

Le projet TestTransformation contient maintenant le modèle source qui sera traité par la transformation et le modèle cible dans lequel la transformation placera son résultat.
4. Dans le modèle ModeleSource, créez un package nommé ClassesMetier, contenant une classe appelée Employe avec trois opérations privées et une opération publique.
  - a. Dans la vue Explorateur de projets, cliquez avec le bouton droit sur le modèle **ModeleSource** et sélectionnez **Ajouter une entité UML → Package**.
  - b. Nommez le package ClassesMetier.
  - c. Cliquez avec le bouton droit sur le package **ClassesMetier** et sélectionnez **Ajouter une entité UML → Classe**.
  - d. Nommez la classe Employe.

- e. Cliquez avec le bouton droit sur la classe **Employe** et sélectionnez **Ajouter une entité UML** → **Opération**.
  - f. Nommez l'opération `lireEmail`.
  - g. Dans la vue Propriétés, sous l'onglet **Général**, dans la section **Visibilité**, cliquez sur **Privée**.
  - h. Répétez l'étape 4e et nommez la nouvelle opération `repondreTelephone`.
  - i. Répétez l'étape 4g pour spécifier que l'opération `repondreTelephone` est privée.
  - j. Répétez l'étape 4e et nommez la nouvelle opération `effectuerTravail`.
  - k. Répétez l'étape 4g pour spécifier que l'opération `effectuerTravail` est privée.
  - l. Répétez l'étape 4e et nommez la nouvelle opération `rapportResponsable(name:String)`.
5. Cliquez sur **Fichier** → **Sauvegarder**.

Vous pouvez maintenant créer une configuration de transformation, que vous utiliserez pour exécuter la transformation.

### Leçon 3 - Créer une configuration de transformation

Cette leçon vous apprend à créer une configuration de transformation. Ce type de configuration spécifie les informations nécessaires à une transformation pour qu'elle génère le résultat attendu.

Vous pouvez appliquer une configuration de transformation pour transformer le contenu spécifié en un contenu différent dans le modèle cible. Lorsque la source spécifiée est un modèle, la transformation interprète les éléments qu'il contient et génère sa sortie dans la cible spécifiée. Dans la plupart des cas, le modèle source reste inchangé. La transformation génère un nouveau jeu de fichiers ou modifie un jeu de fichiers existant.

Pour simplifier l'utilisation du fichier de configuration de transformation, il est préférable de le sauvegarder dans le projet qui contient la source de la transformation. Dans cette leçon, vous sauvegardez le fichier de configuration de transformation dans le plan de travail d'exécution, dans le projet `TestTransformation`. Les fichiers de configuration de transformation portent l'extension de nom de fichier `.tc`.

Pour créer une configuration de transformation :

1. Dans le plan de travail d'exécution, sélectionnez **Fichier** → **Nouveau** → **Configuration de transformation**. Si **Configuration de transformation** ne figure pas dans les options de menu proposées, cliquez sur **Fichier** → **Nouveau** → **Autre** → **Configuration de transformation**.
2. Dans l'assistant Nouvelle configuration de transformation, sur la page Nom et transformation, spécifiez les valeurs suivantes :
  - a. Dans la zone **Nom**, tapez `PremiereConfiguration`.
  - b. Dans la liste **Transformation de réacheminement**, développez la branche **Generaliser\_les\_classes** et sélectionnez **Generaliser\_les\_classes Transform**.
  - c. Dans la zone **Destination du fichier de configuration**, tapez `/TestTransformation` si cette valeur n'est pas déjà affichée.
  - d. Cliquez sur **Suivant**.
3. Sur la page Source et cible, spécifiez le modèle source qui sera traité par la transformation et le modèle cible dans lequel la transformation placera son résultat :
  - a. Dans le volet Source sélectionnée, cliquez sur **TestTransformation** → **Modèles** → **ModeleSource**.
  - b. Dans le volet Cible sélectionnée, cliquez sur **TestTransformation** → **Modèles** → **ModeleCible**.
4. Cliquez sur **Suivant** et acceptez les valeurs par défaut sur les pages restantes de l'assistant.
5. Cliquez sur **Terminer**.

Un fichier de configuration de transformation est créé dans le plan de travail d'exécution, dans le projet TestTransformation. Il est affiché dans l'éditeur de configuration de transformation. Vous êtes maintenant prêt à exécuter la transformation.

## Leçon 4 - Exécuter la transformation

Cette leçon vous montre comment exécuter la transformation dans le plan de travail d'exécution. Lorsque vous appliquez une configuration de transformation, une instance de la transformation est créée et exécutée avec les propriétés que vous avez spécifiées dans la configuration.

Lorsque vous exécutez une transformation, elle crée un modèle temporaire et le compare au modèle cible qui est spécifié dans la configuration de transformation. Les différences entre les deux modèles sont affichées dans la fenêtre Fusionner.

Pour plus d'informations sur l'exécution de transformations et la spécification de stratégies de fusion, consultez les rubriques connexes citées plus bas.

Pour appliquer la configuration de transformation qui invoque la transformation Generaliser\_les\_classes :

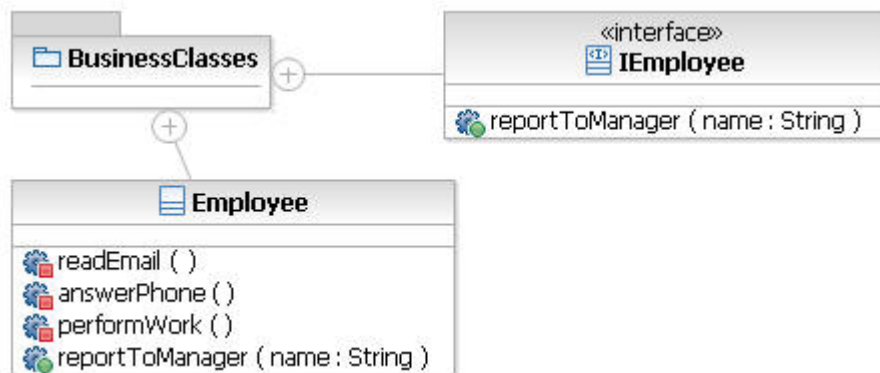
1. Si ce n'est déjà fait, ouvrez la configuration de transformation dans le plan de travail d'exécution. Pour cela, dans la vue Explorateur de projets, développez la branche du projet TestTransformation et faites un double clic sur le fichier **PremiereConfiguration.tc**. Ensuite, sur la page Principales de l'éditeur de configuration de transformation, cliquez sur **Exécuter**.

**Conseil :** Vous pouvez aussi cliquer avec le bouton droit sur **PremiereConfiguration.tc** dans le projet TestTransformation, puis sélectionner **Transformer** → **Generaliser\_les\_classes Transform**.

2. Pendant l'exécution de la transformation, en fonction des options de fusion par défaut spécifiées par le fournisseur de transformation, des messages peuvent vous demander d'accepter les modifications apportées aux fichiers dans le modèle cible. Cliquez sur **OK**.
3. En réponse aux messages affichés, cliquez sur **OK**.
4. Dans la fenêtre Fusionner, passez en revue les modifications proposées pour le modèle cible, cochez ou décochez celles que vous acceptez ou refusez, puis cliquez sur **OK**.
5. En réponse aux messages affichés, cliquez sur **OK**.

**Conseil :** Pour exécuter à nouveau cette transformation, cliquez sur **Modélisation** → **Transformer** → **Exécuter la dernière transformation**. La transformation utilise ainsi les mêmes éléments source que lors de sa dernière exécution.

Vous pouvez maintenant explorer les résultats de la transformation dans le modèle ModeleCible. L'image suivante montre la représentation visuelle des éléments dans le modèle de sortie généré (ModeleSourceModeleCible). Ce modèle contient un package nommé ClassesMetier, qui contient une classe Employee et une interface IEmployee.



Rubriques connexes :

Ajouter le support de fusion pour les modèles générés par les projets de mappage de transformation de modèle à modèle

Exécuter et réexécuter des transformations

---

## Récapitulatif - Créer, configurer et exécuter une transformation de modèle à modèle

Ce tutoriel vous a montré comment créer, configurer et exécuter une transformation de modèle à modèle simple.

### Ce que vous avez appris

Ce tutoriel vous a expliqué les concepts des transformations de mappage de modèle à modèle et vous a montré comment effectuer les tâches suivantes :

- Créer un projet de transformation de modèle à modèle qui contient une infrastructure de transformation et un modèle de mappage
- Créer des déclarations de mappage dans le modèle de mappage
- Raffiner les déclarations de mappage en effectuant les tâches suivantes :
  - Spécifier un objet d'entrée source et un objet de sortie cible pour chaque déclaration de mappage
  - Créer des règles de mappage en définissant les relations entre les attributs des objets d'entrée et de sortie dans une déclaration de mappage
- Générer et compiler le code source de transformation
- Configurer un plan de travail d'exécution en vue de tester la transformation de modèle à modèle
- Dans un plan de travail d'exécution, créer et appliquer une configuration de transformation qui exécute la transformation de modèle à modèle

### Autres ressources

Pour parfaire vos connaissances des sujets traités dans ce tutoriel, consultez les sources d'informations suivantes :

- Aide en ligne des outils de création de transformations et des extensions de transformations