



モデルからモデルへの変換の作成、構成、および実行



---

## 目次

### モデルからモデルへの変換の作成、構成、 および実行 . . . . . 1

概要: モデルからモデルへの変換の作成、構成、および実行 . . . . . 1

モジュール 1: モデルからモデルへの変換マッピング・プロジェクトの作成およびモデルからモデルへのマッピングの洗練 . . . . . 3

演習 1: モデルからモデルへの変換マッピング・プロジェクトの作成 . . . . . 3

演習 2: マッピング・プロジェクトの検査 . . . . . 5

演習 3: クラスからクラスへのマッピング宣言の作成と洗練 . . . . . 6

演習 4: クラスからインターフェースへのマッピング宣言の作成と洗練 . . . . . 10

演習 5: 操作から操作へのマッピング宣言が必要とするマッピング宣言の作成と洗練 . . . . . 12

演習 6: パッケージからパッケージへのマッピング宣言の作成と洗練 . . . . . 17

演習 7: モデルからモデルへのマッピング宣言の作成と洗練 . . . . . 20

モジュール 2: 変換コードの生成と変換の実行 . . . . . 22

演習 1: 変換ソース・コードの生成とコンパイル . . . . . 23

演習 2: ランタイム・ワークベンチの構成 . . . . . 24

演習 3: 変換構成の作成 . . . . . 26

演習 4: 変換の実行 . . . . . 27

要約: モデルからモデルへの変換の作成、構成、および実行 . . . . . 28



---

## モデルからモデルへの変換の作成、構成、および実行

このチュートリアルではモデルからモデルへの変換の作成の方法を説明します。変換マッピング・プロジェクトを作成することで、モデルからモデルへの変換を作成することができます。モデルからモデルへの変換により、1 つのモデルは別の抽象化レベルを持つモデルに変換されます。また、このチュートリアルでは、マッピング・プロジェクトのマッピング・モデルを作成および洗練する方法、およびモデルからモデルへの変換コードを作成およびテストする方法についても説明します。

### 学習目標

指定された手順でこのチュートリアルの各演習を実行してください。このチュートリアルでは以下のタスクを実行する方法を説明します。

- モデルからモデルへの変換マッピング・プロジェクトを作成します。
- 変換マッピング・ツールを使用して、モデルからモデルへの変換マッピング・プロジェクトのマッピング・モデルを作成して洗練します
- モデルからモデルへの変換ソース・コードを生成してコンパイルします
- ランタイム・ワークベンチでモデルからモデルへの変換を構成して実行します

### 所要時間

約 60 分

#### 関連情報

モデルからモデルへの変換のオーサリング



PDF 版の表示

このファイルを表示するには、ご使用のシステムに Adobe® Acrobat® Reader がインストールされていなければなりません。

---

## 概要: モデルからモデルへの変換の作成、構成、および実行

このチュートリアルでは、IBM® Rational® モデリング製品で使用可能なマッピング・ツールを使用して、モデルからモデルへの変換の作成、構成、および実行する方法を説明します。変換マッピング・プロジェクトを作成し、プロジェクト内にマッピング・モデルを作成して洗練した後、モデルからモデルへの変換用のコードを生成することができます。変換ソース・コードを生成した後、ランタイム・ワークベンチで変換を構成し実行することができます。

このチュートリアルでは、ソース・モデルのクラスをターゲット・モデルのインターフェース・クラスおよび実装クラスに変換する、モデルからモデルへの変換を作成します。生成された実装クラスにはソース・クラスの操作のコピーが必要ですが、生成されたインターフェースにはソース・クラスの public 操作のみのコピーが必要です。

このチュートリアルを使用するためには、オプションでインストール可能ないくつかのコンポーネントが必要な場合があります。適切なオプション・コンポーネントがインストールされていることを確認するには、システム要件リストを参照してください。

変換とは、ソース要素 (複数可) を取得して、それらを新規のターゲット要素 (複数可) に変更するという、パターン実装戦略の一種です。変換を実行することにより、モデル/コード間のほか、さまざまな抽象化レベルのモデル間で切り替えを行えます。変換を作成すること (変換オーサリングとも呼ぶ) は、詳細な実装情報を持つ変換を作成したり、モデル間またはメタモデル間のマッピング関係を指定する変換を作成したりすることを可能にするモデル駆動型プロセスです。

モデルからモデルへの変換マッピング・プロジェクトを使用すると、ソースおよびターゲットのメタモデルを指定し、メタモデルの要素間の関係を定義するマッピング・モデルを作成できます。それらの関係を実装する拡張可能な変換ソース・コードを、段階的に生成できます。このレベルの抽象化を使用することにより、ソリューション・ドメインではなく、問題ドメインに焦点を置くことができます。

モデルからモデルへの変換オーサリング・プロセスは、主に以下のステップから構成されています。

1. モデルからモデルへの変換マッピング・プロジェクト マッピング・モデルを含んだ、1 つのマッピング・プロジェクトに複数のマッピング・モデルを含めることができます。マッピング・プロジェクトを作成すると、変換サービスに 1 つの変換が登録されます。それぞれの変換に、1 つの変換プロバイダー、MainTransform という名前の変換式、および 1 つの変換式 (プロジェクト内の各マッピング宣言に 1 つずつ) が含まれています。
2. マッピング・モデルにマッピング宣言 (マップとも呼ぶ) を追加します。1 つのマッピング・モデルに 1 つ以上のマッピング宣言を含めることができます。
3. マッピング・モデル内のマッピング宣言にマッピング規則を追加します。
4. マッピング・プロジェクト内のマッピング・モデル (複数可) から変換ソース・コードを生成します。モデルからモデルへの変換オーサリング・ツールによって、マッピング・プロジェクト内のマッピング・モデルごとに 1 つの変換が生成されます。オーサリング・ツールは、マッピング宣言ごとに、1 つの変換式を実装した 1 つの Java™ ソース・ファイルを生成します。マッピング宣言内の 1 つの移動マッピング規則またはカスタム・マッピング規則ごとに、変換式のソース・コード内に 1 つの規則が生成されます。マッピング宣言内の 1 つのサブマップ・マッピング規則ごとに、変換式のソース・コード内に 1 つのコンテンツ抽出が生成されます。

## 学習目標

このチュートリアルは、順序正しく完了する必要のある 2 つのモジュールから構成されています。これらのモジュールでは、以下のタスクを実行します。

- 変換フレームワークおよびマッピング・モデルを含む、モデルからモデルへの変換オーサリング・プロジェクトを作成します
- マッピング・モデル内にマッピング宣言を作成します
- 以下のタスクを実行して、マッピング宣言を洗練します。
  - マッピング宣言ごとに、入力オブジェクトと出力オブジェクトを指定します
  - マッピング宣言の入力オブジェクトおよび出力オブジェクトの属性間の関係を定義して、マッピング規則を作成します
- 変換ソース・コードを生成してコンパイルします
- モデルからモデルへの変換をテストするために、ランタイム・ワークベンチを構成します
- ランタイム・ワークベンチで、モデルからモデルへの変換を実行する変換構成を作成して適用します。

## 所要時間

このチュートリアルは、終了するのに約 60 分かかります。このチュートリアルに関連した他の概念も検討する場合、実行するのにさらに長い時間がかかる可能性があります。

## スキル・レベル

拡張

## 対象読者

このチュートリアルの対象読者は開発者です。

## システム要件

このチュートリアルを実行するには、変換オーサリングのコンポーネントがインストールされている必要があります。

また、モデリング機能も使用可能にする必要があります。

## 前提条件

このチュートリアルを実行するには、以下の概念に精通していることが必要です。

- Eclipse モデリング・フレームワーク (EMF)
- Eclipse プラグイン・プロジェクト
- Ecore モデル
- Eclipse ワークベンチ

---

## モジュール 1: モデルからモデルへの変換マッピング・プロジェクトの作成 およびモデルからモデルへのマッピングの洗練

このモジュールでは、複数のマッピング宣言を含むモデルからモデルへの変換マッピング・プロジェクトを作成します。それぞれのマッピング宣言で、マッピング宣言の入力要素と出力要素のフィーチャーの関連の仕方を定義するマッピング規則を作成します。また、プロジェクトのファイル、およびプロジェクトのファイルが変換コードを生成するために使用される方法について学習します。

### 学習目標

このモジュールの演習では、マッピング・モデル、マッピング宣言、およびマッピング規則を説明し、以下のタスクを実行する方法を示します。

- マッピング・プロジェクトを作成します
- マッピング宣言を作成して洗練します
- マッピング規則を管理します

### 所要時間

このモジュールを完了するには、約 40 分を要します。

## 演習 1: モデルからモデルへの変換マッピング・プロジェクトの作成

この演習では、モデルからモデルへの変換マッピング・プロジェクトを作成する方法を示します。

モデルからモデルへの変換マッピング・プロジェクトは、変換定義メカニズムである変換プロバイダーを指定する、標準の Eclipse プラグインです。また、マッピング・プロジェクトには、少なくとも 1 つのマッピング・ファイル (マッピング・モデルとも呼ぶ) を組み込みます。マッピング・プロジェクトを作成すると、プロジェクト内にマッピング・モデルが自動的に作成されます。

## 変換マッピング・プロジェクトについてさらに学習したい方に:

モデルからモデルへの変換マッピング・プロジェクト (マッピング・プロジェクトとも呼ぶ) は、`com.ibm.xtools.transform.core.transformationProviders` という名前の拡張ポイントを拡張する Eclipse プラグインです。変換マッピング・プロジェクトにモデルからモデルへの変換を作成すると、変換の実装の詳細を表すコードを作成する代わりに、選択したソースおよびターゲットのモデルまたはメタモデル内の要素の関連の仕方を指定することに集中できます。

1 つのマッピング・プロジェクトに複数のマッピング・ファイル (マッピング・モデルとも呼ぶ) を含めることができます。マッピング・モデルを変更すると、変換ソース・コードを繰り返し生成できます。変換ソース・コードを生成すると、外部から認識可能な `MainTransform` という名前の 1 つの変換式が自動的に登録されます。また、マッピング・モデル内のマッピング宣言ごとに変換式の Java ソース・コードが生成されます。

マッピング・プロジェクトを作成する際には、1 つ以上のソース・メタモデルおよびターゲット・メタモデルを指定できます。`.ecore` というファイル名拡張子を持つメタモデルや、`.epx` または `.uml` というファイル名拡張子を持つ UML プロファイルを指定できます。プロジェクトの作成時にソース・メタモデルおよびターゲット・メタモデルを指定すると、必要な依存関係がプラグイン・マニフェスト・ファイルに自動的に追加されます。マッピング・プロジェクトの作成後に、エディター領域でコマンドを使用してメタモデルを追加する場合、必要なすべての新規依存関係を、プラグイン・マニフェスト・ファイルに追加する必要があります。

モデルからモデルへの変換マッピング・プロジェクトを作成する手順は、以下のとおりです。

1. 「プラグイン開発」パースペクティブを開き、「ウィンドウ」→「パースペクティブを開く」→「その他」とクリックします。「パースペクティブを開く」ウィンドウで、「プラグイン開発」をクリックして、「OK」をクリックします。
2. 「ファイル」→「新規」→「プロジェクト」をクリックします。
3. 「新規プロジェクト」ウィザードの「ウィザードを選択」ページで、「変換オーサリング」をクリックして、「モデル間マッピング変換プロジェクト」をクリックします。
4. 「次へ」をクリックします。
5. 「プラグイン・プロジェクト」ページの「プロジェクト名」フィールドに、`Generalize Classes` と入力します。このページの他のフィールドについては、デフォルト値を採用します。
6. 「次へ」をクリックします。
7. 「プラグイン・コンテンツ」ページで値を確認し、「次へ」をクリックします。
8. 「テンプレート」ページの「使用可能なテンプレート」リストから、「変換マッピング付きプラグイン」を選択します。
9. 「次へ」をクリックします。
10. 「新規変換マッピング」ページの「マップ名」フィールドに、値がまだ存在しない場合は、`Generalize_Classes` と入力します。このフィールドには、マッピング・モデルの名前を指定します。マッピング・モデルは、プロジェクトの `model` フォルダにあり、ファイル名拡張子が `.mapping` です。
11. 「パッケージ名」フィールドに、名前がまだ存在しない場合は、`generalize_classes` と入力します。後の演習では、変換ソース・コードを生成すると、`generalize_classes.transforms` という名前のフォルダに、変換用の Java ソース・コードが作成されます。
12. 入出力モデル (複数可) を指定するには、以下のステップを実行します。

- a. 「新規変換マッピング」ページの、入力モデル領域の横にある「**モデルの追加**」をクリックします。
- b. 「リソースのロード」ダイアログ・ボックスで該当するボタンをクリックし、モデル (複数可) を参照します。このチュートリアルでは、「登録済みパッケージの参照」をクリックして、  
<http://www.eclipse.org/uml2/2.x.y/UML> という命名規則を持つ最新版の UML2 モデルを選択し、「**OK**」をクリックします。このステップでは、変換によって UML.ecore メタモデルが変換ソースとして受け入れられるように指定します。
- c. 「新規変換マッピング」ページの、出力モデル領域の横にある「**モデルの追加**」をクリックします。
- d. ステップ 12b を繰り返して、変換出力が UML.ecore.metamodel タイプのモデルとなるように指定します。このステップでは、変換出力が UML.ecore メタモデルとなるように指定します。
- e. 「**OK**」をクリックします。

プロジェクトの作成時に入力メタモデルおよび出力メタモデルを指定すると、必要な依存関係がプラグイン・マニフェスト・ファイルに自動的に追加されます。マッピング・プロジェクトの作成後に、エディター領域でコマンドを使用してメタモデルを追加する場合、必要なすべての新規依存関係を、プラグイン・マニフェスト・ファイルに追加する必要があります。

13. 「終了」をクリックします。

マッピング・プロジェクトがワークスペースに作成されました。次の演習では、マッピング・プロジェクトの構造を検討します。

## 演習 2: マッピング・プロジェクトの検査

マッピング・プロジェクトを作成した後、「パッケージ・エクスプローラー」ビューを使用して、プロジェクトの構造を調べることができます。

マッピング・プロジェクトの内容を調べる手順は、以下のとおりです。

1. 「パッケージ・エクスプローラー」ビューで、「**Generalize Classes**」マッピング・プロジェクトを展開し、生成されたファイルを監視します。
2. 「モデル」フォルダーにナビゲートします。このフォルダーに変換マッピング・プロジェクトと同じ名前を持つマッピング・モデルの含まれ方と、またファイルがファイル名拡張子に `.mapping` を持つことに注目してください。1 つのマッピング・プロジェクトに複数のマッピング・モデルを含めることができます。このチュートリアルの後の方では、マッピング宣言をこのマッピング・モデルに追加します。

**マッピング・モデルについてさらに学習したい方に:**

マッピング・モデル (マッピング・ファイルとも呼ぶ) は、Eclipse モデリング・フレームワーク (EMF) のメタモデル (Ecore モデルとも呼ぶ) のインスタンスであり、マップされるメタモデルへの参照を含んでいます。マッピング・プロジェクトを作成すると、指定した入出力モデル (複数可) を使用して、オーサリング・ツールによってプロジェクト内にマッピング・モデルが作成されます。マッピング・モデルのファイル名拡張子は `.mapping` です。

マッピング・モデルは、XML ファイルとして保管され、直列化されます。「問題」ビューには、マッピング・モデルに関する詳細なエラー情報が表示されます。このビューでは、項目をダブルクリックしてテキスト・エディターでマッピング・モデルを開いたり、エラーのある行を表示します。この方法は、エディター領域内にマッピング・モデルを表示してトラブルシューティングを行うよりも簡単な場合がよくあります。

3. 「パッケージ・エクスプローラー」ビューで、「src」フォルダーを展開します。  
「generalize\_classes.transforms」フォルダーに生成済みの変換式のソース・コードが含まれます。この時点で、MainTransform という 1 つのデフォルトの変換式のみが存在します。このチュートリアルの後の方では、マッピング宣言を作成し変換ソース・コードを再生成します。変換オーサリング・フレームワークは、マッピング・モデル内のマッピング宣言ごとに、*n* Transform という名前の Java クラスを生成します (この *n* はマッピング宣言の名前を表します)。これらの Java クラスによって集会的に変換コードが構成されます。
4. 「パッケージ・エクスプローラー」ビューで、「モデル」フォルダーのマッピング・モデルをダブルクリックします。変換マッピング・エディターが開き、それによってマッピング宣言を作成し、それぞれのマッピング宣言のマッピング規則を洗練できるようになります。このモジュールの以下の演習で、これらのタスクを実行します。

### 演習 3: クラスからクラスへのマッピング宣言の作成と洗練

この演習では、入力オブジェクトおよび出力オブジェクトとして UML クラスを指定するマッピング宣言を作成する方法を示します。このクラスからクラスへのマッピング宣言には、生成済みの変換を実行したときに、ソース・モデルにクラスとその操作のコピーを作成し、ターゲット・モデルにそのコピーを置くマッピング規則を含みます。また、クラスからクラスへのマッピング宣言のサブマップが呼び出す、操作から操作へのマッピング宣言を作成することもできます。

マッピング宣言 (マップとも呼ぶ) は、指定された入力オブジェクトに基づいて出力オブジェクトを作成または更新する方法を指定します。マッピング宣言を使用することで、入力オブジェクトの属性を出力オブジェクトの属性に対応させる方法を指定することができます。それぞれのマッピング宣言では入力タイプおよび出力タイプを指定し、その入出力タイプは、マッピング・モデルに追加するメタモデルから選択します。

変換オーサリング・フレームワークは、マッピング・モデル内のマッピング宣言ごとに、*n* Transform.java という名前の Java ソース・ファイルを生成します (この *n* はマッピング宣言の名前を表します)。これらの Java ファイルによって集会的に変換コードが構成されます。変換オーサリング・フレームワークは、変換用の実装コードを生成するほか、変換サービスに変換を登録するためのコードも生成します。マッピング宣言を作成した後は、マッピング規則を段階的に追加し、マッピング規則に対してソース・コードまたは実装を生成できます。ソース・コードを生成する前に、すべてのマッピング規則を定義しておく必要はありません。

#### マッピング宣言についてさらに学習したい方に:

通常、マッピング宣言は *x2y* という命名規則に従っており、*x* は入力オブジェクト・タイプを、*y* は出力オブジェクト・タイプを表します。例えば、Package2EPackage という名前のマッピング宣言は、入力オブジェクトとして Package を、出力オブジェクトとして EPackage を含むマッピング宣言を指定します。

要素間のマップでは双方の属性間の対応が確立され、それにより相互間でのデータ交換が可能になります。ほとんどのマップでは、ソースとターゲット間でさらに進んだデータ操作を行えます。例えば、ターゲットへの値の割り当てを可能にするカスタム・コードを作成することによって、データに対して計算を指定したり、その他の変更を行ったりすることができます。

マッピング・モデルでクラスからクラスへのマッピング宣言を作成する手順は、以下のとおりです。

1. まだ開かれていない場合は、「パッケージ・エクスプローラー」ビューで、model フォルダーの .mapping ファイルをダブルクリックします。
2. 変換マッピング・エディターの「マッピング・ルート」セクションで、「Generalize\_Classes」を右クリックしてから「マップの作成」をクリックします。

3. 「新規マップ」ウィンドウの「マップ名」フィールドに Class2Class と入力してから、「OK」をクリックします。「アウトライン」ビューにマップが表示され、エディター領域内の「マッピング・ルール」セクションの下にそのマップが開きます。

## クラスからクラスへのマッピング宣言への入出力オブジェクトの追加

マッピング宣言を作成した後、入力オブジェクトおよび出力オブジェクトをそのマッピング宣言に追加する必要があります。この演習では、入出力オブジェクトとして UML クラスを指定します。

注: モデルからモデルへの変換を作成するとき、該当する UML プロファイルまたは Ecore メタモデルをマップの範囲に追加することができます。。「入力の追加」ウィンドウで、「モデルの追加」をクリックして該当するモデルを指定します。

入力オブジェクトと出力オブジェクトをクラスからクラスへのマッピング宣言に追加する手順は、以下のとおりです。

1. 編集しているマップのツールバーの左端にあるアイコン、「入力オブジェクトの追加」アイコン (



) をクリックします。

2. 「入力の追加」ウィンドウの「要素」ペインで、メタモデル・オブジェクトを選択します。「要素」ペインに、マッピング・モデルのソースまたはターゲットとして指定したメタモデル (複数可) の要素が表示されます。このチュートリアルでは、「要素」ペインで「uml」を展開し、「クラス」をクリックして、「OK」をクリックします。

3. 編集しているマップのツールバーの左から 2 番目のアイコン、「出力オブジェクトの追加」アイコン (



) をクリックします。

4. 「出力の追加」ウィンドウの「要素」ペインで、「uml」を展開し、「クラス」をクリックして、「OK」をクリックします。

5. 「ファイル」 → 「保存」をクリックします。

これでクラス入力オブジェクトとクラス出力オブジェクトの属性の間に、マッピング規則を定義できるようになります。

## クラス入力オブジェクトとクラス出力オブジェクトの属性間のマッピング規則の定義

入力オブジェクトと出力オブジェクトをマッピング宣言に追加した後、属性間にマッピング規則を定義することができます。マッピング規則 (マッピングとも呼ぶ) では、入力オブジェクトの属性値に応じて、出力オブジェクトの属性に値を割り当てる方法を指定します。

クラスからクラスへのマッピング宣言で、以下のステップを実行します。

- ターゲット・モデルにクラスを作成する移動マッピング規則を作成します。

入力オブジェクトと出力オブジェクトの name 属性の間にマッピング規則を作成します。ターゲット・クラスは入力モデルのクラスと同じ名前を持ちます。これは、クラスのコピーの作成と考えることもできます。後の演習では、マッピング操作をマッピング規則に追加します。

- クラス内の操作ごとに、対応する操作をターゲット・モデルのクラスに作成するサブマップ・マッピング規則を作成します。

入力オブジェクトと出力オブジェクトの `ownedOperation` 属性の間にサブマップ・マッピング規則を作成します。`ownedOperation` コレクションのそれぞれの操作に対して、ターゲット・モデルの生成された操作は入力モデルの操作と同じ名前と可視性を持ちます。

### マッピング規則についてさらに学習したい方に:

マッピング規則 (マッピングとも呼ぶ) では、入力オブジェクトの属性値に応じて、出力オブジェクトの属性に値を割り当てる方法を指定します。入力オブジェクトおよび出力オブジェクトの間に複数のマッピング規則を作成することができます。また、このチュートリアル後の演習で示されるように、入出力オブジェクトの属性間に複数のマッピング規則を作成することもできます。以下のタイプのマッピング規則を作成することができます。

**移動** 移動 (シンプル・マッピング規則とも呼ぶ) は、最も基本的なタイプのマッピング規則です。入力属性と出力属性のデータ型には互換性が必要です。入力属性と出力属性は、その両方が複数の値を持っているか、または両方とも持っていないかのどちらかです。例えば、出力オブジェクトの属性が文字列であり、入力オブジェクトの属性がカスタム・コードなしで文字列に変換できるものである場合に、このオプションを選択します。このタイプのマッピング規則は、1 つのソース要素または属性と 1 つのターゲット要素または属性の間のマッピングをサポートしています。移動マッピング規則用に生成される変換ソース・コードは、1 つの入力属性の値を 1 つの出力属性にコピーする規則を実装します。

### サブマップ

サブマップは、あるマップの内部から別のマップを呼び出します。呼び出されるサブマップは、それを呼び出すマップ自身のマッピング・ファイルに定義できます (定義しなくても構いません)。サブマップを使用すると、入力モデルの複合型を出力モデルの複合型にマップすることができます。作成するサブマップによって、任意のマッピング・ファイルに存在するマップを起動できます。別々のマッピング・ファイルにサブマップを定義すると、マップの再利用が促進されますが、複数のマッピング・ファイルを作成することで、プロジェクトの保守に要する時間が増える可能性があります。サブマップに他のサブマップをインクルードして、階層構造にすることもできます。

サブマップ・マッピング規則では、以下のタイプのマッピングがサポートされています。

- 入力オブジェクトおよび出力オブジェクト間、または入力オブジェクトおよび出力オブジェクトの属性間の 1 対 1 のマッピング
- 入力オブジェクトおよび出力オブジェクトの属性間の 1 対  $m$  または  $m$  対 1 のマッピング
- 入力オブジェクトおよび出力オブジェクトの属性間の  $m$  対  $n$  のマッピング

サブマップ規則が 1 対  $m$  のマッピングを指定した場合、生成された変換式によりオブジェクトが `singleton` 属性からリストに追加されます。 $m$  対 1 のマッピングを指定した場合は、変換式によりリストからオブジェクトが抽出され、`singleton` 属性に挿入されます。

また、マッピング宣言内の入力オブジェクトおよび出力オブジェクトの間に、サブマップ・マッピング規則を作成することもできます。

マッピング宣言のそれぞれのサブマップには、`getInputFeature ToOutputFeature_UsingMap_Extractor` という名前の抽出が、その中に含まれている変換式に生成されます。この `InputFeature` は入力属性の名前を表し、`OutputFeature` は出力属性の名前を表し、そして `Map` はマッピング宣言の名前を表します。

### カスタム

このマッピング規則タイプを使用することによって、出力プロパティの値を計算するカスタム・

コードを指定できます。例えば、出力オブジェクトのプロパティ値を、入力オブジェクトの複数のプロパティの連結値と等しく設定する場合は、このマッピング・タイプを選択します。

Eclipse に提供されているオブジェクト制約言語 (OCL) の API を使用することで、意味構造の調整を指定できます。

カスタム・マッピング規則では、以下のタイプのマッピングがサポートされています。

- 入力オブジェクトおよび出力オブジェクト間、または入力オブジェクトおよび出力オブジェクトの属性間の 1 対  $n$  のマッピング
- 入力オブジェクトおよび出力オブジェクト間、または入力オブジェクトおよび出力オブジェクトの属性間の  $m$  対  $n$  のマッピング
- $m$  対 1 のマッピング。この  $m$  対 1 は以下のマッピングの 1 つを表します。
  - 単一の入力属性 (その多重度が  $m$  に設定されている) から単一の出力属性 (その多重度が 1 に設定されている) へのマッピング
  - 複数の入力属性から単一の出力属性へのマッピング
  - 複数の入力オブジェクトから単一の出力オブジェクトへのマッピング

### 継承マップ

入力オブジェクトと出力オブジェクトの間にのみこのマッピング規則を作成することができます。

継承元のマッピング宣言は、継承先のマッピング宣言で定義されるマッピング規則を継承します。以下の品質を持つマッピング規則を定義することで、継承先のマッピング規則をオーバーライドできます。

- 入力オブジェクト・プロパティおよび出力オブジェクト・プロパティは、継承先のマッピング規則と同じです。
- オーバーライドおよび継承されたマッピング規則は、一致した抽出を持つサブマップ・マッピングです。あるいは、オーバーライドおよび継承されたマッピング規則は両方とも、移動マッピングであるか、またはカスタム・マッピングです。

継承マップ・マッピング規則を作成する場合は、継承するマッピング規則を含むマッピング宣言を指定する必要があります。マッピング宣言では、複数の継承マップは継承できません。

継承されたマッピング規則、およびそのマッピングから生成された規則または抽出は、オーバーライドされたマッピング規則、およびそのマッピングから生成された規則または抽出と同じ処理順序の相対位置を維持します。

マッピング宣言内の 1 つの移動マッピング規則またはカスタム・マッピング規則ごとに、生成された変換ソース・コード内に 1 つの規則が生成されます。サブマップ・マッピング規則ごとに 1 つのコンテンツ抽出が、変換ソース・コード内に生成されます。マッピング規則を作成すると、そのタイプは選択した入力属性によって決まります。例えば、入力属性と出力属性が、文字列や整数のような互換性のある基本タイプである場合、移動マッピング規則が指定されます。入力属性と出力属性が複合型である場合、サブマップ規則が指定されます。マッピング規則のタイプとして、移動もサブマップも適切ではない場合、カスタム・マッピング規則が指定されます。

Class2Class マッピング宣言でマッピング規則を定義する手順は、以下のとおりです。

1. 入力オブジェクトと出力オブジェクトの `name` 属性の間に移動マッピング規則を作成します。
  - a. エディター領域内で、クラス入力オブジェクト内にある `name` 属性をクリックします。
  - b. `name` 属性のハンドルを、クラス出力オブジェクト内の `name` 属性にドラッグします。
2. 入力オブジェクトと出力オブジェクトの `ownedOperation` 属性の間にサブマップ・マッピング規則を作成します。

- a. エディター領域内で、クラス入力オブジェクト内にある `ownedOperation` 属性をクリックします。
  - b. `ownedOperation` 属性のハンドルを、クラス出力オブジェクト内の `ownedOperation` 属性にドラッグします。`ownedOperation` 属性はコレクションであるため、デフォルトで、サブマップ・マッピング規則が作成されます。
3. 「ファイル」 → 「保存」をクリックします。

これで操作から操作へのマッピング宣言を作成できるようになります。

## 操作から操作へのマッピング宣言の作成

演習のこの時点では、エディター領域内のサブマップにエラーを示す赤い丸で囲まれた X の装飾が表示されています。この装飾の上にマウス・ポインターを置いて、エラー・メッセージを表示します。エラー・メッセージは、サブマップ・マッピング規則が呼び出すマッピング宣言を選択しなければならないことを示しています。このエラーを解決するには、操作から操作へのマッピング宣言を作成します。

操作から操作へのマッピング宣言を作成する手順は、以下のとおりです。

1. 「プロパティ」ビューが開いていない場合は、エディター領域にある前のセクションのステップ 2 で作成したサブマップ要素を右クリックしてから、「**プロパティで表示**」をクリックします。
2. 「プロパティ」ビューの「詳細」タブの「マップ」フィールドの横の「**新規**」をクリックします。
3. 「新規マップ」ウィンドウの「**マップ名**」フィールドに `Operation2Operation` と入力します。
4. 「**入力**」フィールドに `Operation` の値が含まれない場合、「参照」をクリックして、「入力の追加」ウィンドウの「要素」ペインで、「**操作**」をクリックします。
5. 「**出力**」フィールドに `Operation` の値が含まれない場合、「参照」をクリックして、「出力の追加」ウィンドウの「要素」ペインで、「**操作**」をクリックします。
6. 「**OK**」をクリックします。エラーを示す装飾が除去され、操作から操作へのマッピング宣言が「アウトライン」ビューに表示されます。後の演習では、このマッピング宣言にマッピング規則を作成します。
7. 「ファイル」 → 「保存」をクリックします。

次の演習では、クラスからインターフェースへのマッピング宣言を定義します。

## 演習 4: クラスからインターフェースへのマッピング宣言の作成と洗練

この演習では、マッピング・モデル内にクラスからインターフェースへのマッピング宣言を作成する方法を示します。このマッピング宣言には、名前がソース・モデルのクラスの名前から派生するインターフェースを作成し、ソース・モデルのクラスの `public` メソッドのみをコピーする、マッピング規則が含まれます。

マッピング・モデルにクラスからインターフェースへのマッピング宣言を作成する手順は、以下のとおりです。

1. `Generalize_Classes.mapping` ファイルがマッピング・エディターで開かれていない場合は、「パッケージ・エクスプローラー」ビューでそのファイルをダブルクリックします。
2. 変換マッピング・エディターの「マッピング・ルート」セクションで、「**Generalize\_Classes**」を右クリックしてから「**マップの作成**」をクリックします。
3. 「新規マップ」ウィンドウの「**マップ名**」フィールドに `Class2Interface` と入力してから、「**OK**」をクリックします。「アウトライン」ビューにマップが表示され、エディター領域内の「マッピング・ルート」セクションの下にそのマップが開きます。

注: エディター領域内に開かれていないマッピング宣言の詳細を表示するには、「アウトライン」ビューでマッピング宣言の名前を展開します。別のマッピング宣言を開くには、「アウトライン」ビューでそのマッピング宣言の名前をクリックします。

## クラスからインターフェースへのマッピング宣言への入出力オブジェクトの追加

クラスからインターフェースへのマッピング宣言を作成した後、UML クラスを入力オブジェクトとして指定し、UML インターフェースを出力オブジェクトとして指定します。

入力オブジェクトと出力オブジェクトをクラスからインターフェースへのマッピング宣言に追加する手順は、以下のとおりです。

1. 編集しているマップのツールバーの左端にある「入力オブジェクトの追加」アイコンをクリックします。
2. 「入力の追加」ウィンドウの「要素」ペインで、「uml」を展開し、「クラス」をクリックして、「OK」をクリックします。
3. 編集しているマップのツールバーの左から 2 番目にあるアイコンである「出力オブジェクトの追加」アイコンをクリックします。
4. 「出力の追加」ウィンドウの「要素」ペインで、「uml」を展開し、「インターフェース」をクリックして、「OK」をクリックします。
5. 「ファイル」→「保存」をクリックします。

これでクラス入力オブジェクトとインターフェース出力オブジェクトの属性の間に、マッピング規則を定義できるようになります。

## 入出力オブジェクトの属性間のマッピング規則の定義

入出力オブジェクトをクラスからインターフェースへのマッピング宣言に追加した後、入力オブジェクトおよび出力オブジェクトの属性の間にマッピング規則を定義することができます。

この演習では、`IClassName` というインターフェースを作成するカスタム・マッピング規則を作成します。また、6 ページの『演習 3: クラスからクラスへのマッピング宣言の作成と洗練』で作成した操作から操作へのサブマップを呼び出すサブマップ・マッピング規則を作成し、`public` 操作のみがターゲット・インターフェースに変換されることを指定するために、入力フィルターをサブマップ・マッピング規則に追加することができます。

入力オブジェクトおよび出力オブジェクトの属性の間にマッピング規則を定義する手順は、以下のとおりです。

1. 入力オブジェクトと出力オブジェクトの `name` 属性の間にカスタム・マッピング規則を作成します。
  - a. エディター領域内で、クラス入力オブジェクト内にある `name` 属性をクリックします。
  - b. `name` 属性のハンドルを、インターフェース出力オブジェクト内の `name` 属性にドラッグします。移動マッピング規則が作成されます。
  - c. `name` 属性に接続している「移動」要素で、下矢印をクリックして、「カスタム」をクリックします。
  - d. 「カスタム」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - e. 「プロパティー」ビューの「詳細」タブで、「インライン」をクリックして、メソッド・シグニチャーの下にあるテキスト域に以下のコードを入力します: `Interface_tgt.setName ("I"+Class_src.getName());`

注: コンテンツ・アシスト機能呼び出すには、コードを入力するときに **Ctrl + スペース**を押します。

- f. 「適用」をクリックします。
2. 入力オブジェクトと出力オブジェクトの `ownedOperation` 属性の間にサブマップ・マッピング規則を作成します。
  - a. エディター領域内で、クラス入力オブジェクト内にある `ownedOperation` 属性をクリックします。
  - b. `ownedOperation` 属性のハンドルを、インターフェース出力オブジェクト内の `ownedOperation` 属性にドラッグします。`ownedOperation` 属性はコレクションであるため、デフォルトで、サブマップ・マッピング規則が作成されます。
  - c. 「プロパティー」ビューが開いていない場合は、新規の「サブマップ」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - d. まだ選択されていない場合は、「プロパティー」ビューの「詳細」タブの「マップ」リストから、「**Operation2Operation**」を選択します。
3. ステップ 2 で作成したサブマップ・マッピング規則に、`public` 操作のみをターゲット・インターフェースに変換する入力フィルターを追加します。
  - a. 「プロパティー」ビューの「入力フィルター」タブで、「入力要素のフィルター」をクリックして、「インライン」をクリックします。
  - b. 「コード」オプションの下テキスト・エリアのメソッド・シグニチャーの下に、以下のコードを入力します: 

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

注: 入力または出力フィルターに指定するコードは、ブール値を戻すものでなければなりません。指定できるのは、メソッド本文のみです。変換オーサリング・フレームワークがメソッド・シグニチャーを定義します。有効な変数名のリストを表示するには、「インライン」ボタンの下にあるテキスト域で **Ctrl + スペース**を押します。

- c. 「適用」をクリックします。
4. 「ファイル」 → 「保存」をクリックします。

次の演習では、`Generalize_Classes.mapping` モデルでいくつかのマッピング宣言を作成します。これらのマッピング宣言では、生成済みの変換を実行するときにパラメーターおよび基本タイプが変換される方法を定義します。

## 演習 5: 操作から操作へのマッピング宣言が必要とするマッピング宣言の作成と洗練

この演習では、操作の要素を変換するために操作から操作へのマッピング宣言が必要とするマッピング宣言を作成する方法を示します。たとえば、ソース・モデルのパラメーターをターゲット・モデルのパラメーターに変換するマッピング宣言を作成します。また、ソース・モデルの基本タイプをターゲット・モデルの基本タイプに変換する方法を定義するマッピング宣言を作成します。この演習ではまた、この演習で作成したマッピング宣言を呼び出す操作から操作へのマッピング宣言のマッピング規則も作成します。

以下の表は、この演習で作成するマッピング宣言とマッピング規則のリストです。

マッピング宣言	入出力オブジェクト・タイプ	マップされた属性	マッピング規則のタイプ	マッピング規則の説明
Primitive2Primitive	UML 基本タイプ	name	move	サブマップが Primitive2Primitive マッピング宣言を呼び出す、要素の基本タイプを作成します。
Parameter2Parameter	UML パラメーター	name、visibility	move	サブマップが Parameter2Parameter マッピング宣言を呼び出す、要素のパラメーターを作成します。生成されたパラメーターはソース入力要素のパラメーターと同じ名前と可視性を持ちます。
		type	サブマップ	サブマップがこのマッピング宣言を呼び出す要素のタイプごとに、結果は以下のようになります。 <ul style="list-style-type: none"> <li>パラメーターが基本タイプの場合、この規則は Primitive2Primitive マッピング宣言を呼び出します。</li> <li>パラメーターがクラス・タイプの場合、この規則は Class2Class および Class2Interface マッピング宣言を呼び出します。</li> </ul>
Operation2Operation	UML 操作	name、visibility	move	サブマップが Operation2Operation マッピング宣言を呼び出す、要素の操作を作成します。生成された操作はソース入力要素の操作と同じ名前と可視性を持ちます。
		ownedParameter	サブマップ	操作のそれぞれのパラメーターに対して、このサブマップは Parameter2Parameter マッピング宣言を呼び出します。

このチュートリアルでは、基本タイプのパラメーターを作成するために、Paramater2Parameter マッピング宣言が Primitive2Primitive マッピング宣言を呼び出します。

マッピング・モデルに基本タイプから基本タイプへのマッピング宣言を作成する手順は、以下のとおりです。

1. マッピング・モデルが開かれていない場合は、「パッケージ・エクスプローラー」ビューで、model フォルダの Generalize\_Classes.mapping ファイルをダブルクリックします。
2. 変換マッピング・エディターの「マッピング・ルート」セクションで、「Generalize\_Classes」を右クリックしてから「マップの作成」をクリックします。
3. 「新規マップ」ウィンドウの「マップ名」フィールドにマッピングの名前として Primitive2Primitive と入力してから、「OK」をクリックします。「アウトライン」ビューにマップが表示され、エディター領域内の「マッピング・ルート」の下にそのマップが開きます。

## Primitive2Primitive マッピング宣言への入出力オブジェクトの追加

Primitive2Primitive マッピング宣言を作成した後、入力オブジェクトと出力オブジェクトをそのマッピング宣言に追加する必要があります。この演習では、UML 基本タイプ入出力オブジェクトとして指定します。

入力オブジェクトと出力オブジェクトを Primitive2Primitive マッピング宣言に追加する手順は、以下のとおりです。

1. 編集しているマップのツールバーの左端にある「入力オブジェクトの追加」アイコンをクリックします。
2. 「入力の追加」ウィンドウの「要素」ペインで、「uml」を展開し、「PrimitiveType」をクリックして、「OK」をクリックします。
3. 編集しているマップのツールバーの左から 2 番目にあるアイコンである「出力オブジェクトの追加」アイコンをクリックします。
4. 「出力の追加」ウィンドウの「要素」ペインで、「uml」を展開し、「PrimitiveType」をクリックして、「OK」をクリックします。
5. 「ファイル」→「保存」をクリックします。

これで、PrimitiveType 入力オブジェクトおよび出力オブジェクトの name 属性の間にマッピング規則を定義できるようになります。

## PrimitiveType 入力オブジェクトおよび出力オブジェクトの属性間のマッピング規則の定義

この演習では、ターゲット・モデルに基本タイプを作成する移動マッピング規則を作成します。生成された基本タイプは入力モデルの基本タイプと同じ名前を持ちます。これは、基本タイプのコピーの作成と考えることもできます。

PrimitiveType 入力オブジェクトおよび出力オブジェクトの name 属性の間に関係を定義する移動マッピング規則を定義する手順は、以下のとおりです。

1. エディター領域内で、PrimitiveType 入力オブジェクト内にある name 属性をクリックします。
2. name 属性のハンドルを、出力オブジェクト内の name 属性にドラッグします。
3. 「ファイル」→「保存」をクリックします。

これでパラメーターからパラメーターへのマッピング宣言を作成できるようになります。

## パラメーターからパラメーターへのマッピング宣言の作成

このセクションでは、入力オブジェクトおよび出力オブジェクトとして UML パラメーターを指定するマッピング宣言を作成する方法を示します。このパラメーターからパラメーターへのマッピング宣言には、生成済みの変換を実行したときに、ターゲット・モデルにパラメーターを作成するマッピング規則が含まれます。このときのパラメーターの名前、可視性、およびタイプは、サブマップがこのマッピング宣言を呼び出す要素のパラメーターと同じです。このチュートリアルでは、Operation2Operation マッピング宣言にはこのマッピング宣言を呼び出すサブマップ・マッピング規則が含まれます。

パラメーターからパラメーターへのマッピング宣言を作成する手順は、以下のとおりです。

1. マッピング・モデルが開かれていない場合は、「パッケージ・エクスプローラー」ビューで、model フォルダーの **Generalize\_Classes.mapping** ファイルをダブルクリックします。
2. 変換マッピング・エディターの「マッピング・ルート」セクションで、「**Generalize\_Classes**」を右クリックしてから「**マップの作成**」をクリックします。
3. 「新規マップ」ウィンドウの「**マップ名**」フィールドに **Parameter2Parameter** と入力してから、「**OK**」をクリックします。「アウトライン」ビューにマップが表示され、エディター領域内の「マッピング・ルート」の下にそのマップが開きます。

## Parameter2Parameter マッピング宣言への入出力オブジェクトの追加

マッピング宣言を作成した後、入力オブジェクトおよび出力オブジェクトをそのマッピング宣言に追加する必要があります。この演習では、UML パラメーターを入出力オブジェクトとして指定します。

入力オブジェクトと出力オブジェクトを **Parameter2Parameter** マッピング宣言に追加する手順は、以下のとおりです。

1. 編集しているマップのツールバーの左端にある「**入力オブジェクトの追加**」アイコンをクリックします。
2. 「入力の追加」ウィンドウの「要素」ペインで、「**uml**」を展開し、「**パラメーター**」をクリックして、「**OK**」をクリックします。
3. 編集しているマップのツールバーの左から 2 番目にあるアイコンである「**出力オブジェクトの追加**」アイコンをクリックします。
4. 「出力の追加」ウィンドウの「要素」ペインで、「**uml**」を展開し、「**パラメーター**」をクリックして、「**OK**」をクリックします。
5. 「**ファイル**」 → 「**保存**」をクリックします。

これでパラメーター入力オブジェクトとパラメーター出力オブジェクトの属性の間に、マッピング規則を定義できるようになります。

## Parameter2Parameter マッピング宣言でのマッピング規則の定義

入力オブジェクトおよび出力オブジェクトをマッピング宣言に追加した後、属性間にマッピング規則を定義することができます。

このセクションでは、生成済みの変換を実行したときに、ターゲット・モデルにパラメーターを作成するサブマップ規則および移動マッピング規則を作成します。このときのパラメーターの名前、可視性、およびタイプは、ソース・モデルの要素のパラメーターと同じです。このチュートリアルでは、Operation2Operation マッピング宣言がこのマッピング宣言を呼び出します。

Parameter2Parameter マッピング宣言でマッピング規則を定義する手順は、以下のとおりです。

1. 入力オブジェクトと出力オブジェクトの **name** 属性の間に移動マッピング規則を作成します。
  - a. エディター領域内で、パラメーター入力オブジェクト内にある **name** 属性をクリックします。
  - b. **name** 属性のハンドルを、出力オブジェクト内の **name** 属性にドラッグします。
2. ステップ 1a と 1b を繰り返しますが、**name** を **visibility** に置き換えます。
3. 入力オブジェクトと出力オブジェクトの型属性の間にサブマップ規則を作成します。
  - a. エディター領域内で、パラメーター入力オブジェクト内にある **type** 属性をクリックします。
  - b. **type** 属性のハンドルを、出力オブジェクト内の **type** 属性にドラッグします。この属性が複合型であるため、サブマップはデフォルトで作成されます。
  - c. 「プロパティ」ビューが開いていない場合は、エディター領域内の新規の「サブマップ」要素を右クリックしてから、「**プロパティで表示**」をクリックします。
  - d. 「プロパティ」ビューの「詳細」タブの「マップ」リストから、「**Primitive2Primitive**」を選択します。

このマッピング規則は、基本パラメーターがソース・モデル内に見つかったときに、**Primitive2Primitive** 変換式を呼び出すコードを生成します。ソース・モデル内のパラメーターが基本タイプでない場合、このパラメーターはターゲット・モデルに変換されません。

4. ステップ 3a、3b、および 3c を繰り返してから、以下のステップを実行します。
  - 「プロパティ」ビューの「詳細」タブの「マップ」リストから、「**Class2Class**」を選択します。このサブマップ・マッピング規則は、ソース・パラメーターがクラス・タイプの場合、ターゲット・モデルの要素にクラス・タイプのパラメーターを作成するコードを生成します。
5. ステップ 3a、3b、および 3c を繰り返してから、以下のステップを実行します。
  - 「プロパティ」ビューの「詳細」タブの「マップ」リストから、「**Class2Interface**」を選択します。このサブマップ・マッピング規則は、ソース・パラメーターがクラス・タイプの場合、ターゲット・モデルの要素にインターフェース・タイプのパラメーターを作成します。
6. 「ファイル」→「保存」をクリックします。

これで 6 ページの『演習 3: クラスからクラスへのマッピング宣言の作成と洗練』で作成した操作から操作へのマッピング宣言にマッピング規則を定義できるようになります。

## Operation2Operation マッピング宣言でのマッピング規則の定義

6 ページの『演習 3: クラスからクラスへのマッピング宣言の作成と洗練』では、操作から操作へのマッピング宣言を作成する方法を示しました。これでこのセクションで、このマッピング宣言にマッピング規則を作成する用意ができました。生成済みの変換を実行すると、これらのマッピング規則によって名前、可視性、およびパラメーターがソース・モデルの操作と同じである操作がターゲット・モデルに作成されます。このチュートリアルでは、**Class2Class** および **Class2Interface** マッピング宣言がこのマッピング宣言を呼び出します。

Operation2Operation マッピング宣言でマッピング規則を定義する手順は、以下のとおりです。

1. マッピング・モデルが開かれていない場合は、「パッケージ・エクスプローラー」ビューで、**model** フォルダーの **Generalize\_Classes.mapping** ファイルをダブルクリックします。
2. 「アウトライン」ビューで、「**Operation2Operation**」マッピング宣言をクリックします。
3. 入力オブジェクトと出力オブジェクトの **name** 属性の間に移動マッピング規則を作成します。

- a. エディター領域内で、操作入力オブジェクト内にある `name` 属性をクリックします。
- b. `name` 属性のハンドルを、出力オブジェクト内の `name` 属性にドラッグします。
4. ステップ 3a と 3b を繰り返しますが、`name` を `visibility` に置き換えます。
5. 入力オブジェクトと出力オブジェクトの `ownedParameter` 属性の間にサブマップ規則を作成します。
  - a. エディター領域内で、操作入力オブジェクト内にある `ownedParameter` 属性をクリックします。
  - b. `ownedParameter` 属性のハンドルを、出力オブジェクト内の `ownedParameter` 属性にドラッグします。この属性はコレクションであるため、サブマップはデフォルトで作成されます。
  - c. 「プロパティ」ビューが開いていない場合は、エディター領域内の新規の「サブマップ」要素を右クリックしてから、「**プロパティで表示**」をクリックします。
  - d. 「プロパティ」ビューの「詳細」タブの「マップ」リストから、「**Parameter2Parameter**」を選択します。
6. 「ファイル」→「保存」をクリックします。

次の演習では、パッケージからパッケージへのマッピング宣言を作成します。

## 演習 6: パッケージからパッケージへのマッピング宣言の作成と洗練

この演習では、パッケージからパッケージへのマッピング宣言と複数のマッピング規則を作成する方法を示します。マッピング規則は、生成済みの変換式がソース・モデルに含まれるパッケージのネストされたパッケージまたはクラス要素を処理する方法を指定します。

パッケージの入出力オブジェクトには、`packagedElement` というフィーチャーが含まれます。このフィーチャーは、有効な UML オブジェクトのさまざまなタイプを含むコレクションです。この演習で作成するマッピング規則は、変換がパッケージ・タイプまたはクラス・タイプであるコレクション要素を処理する方法を定義します。

この演習では、以下のマッピング規則を作成します。

- ターゲット・プロジェクトにパッケージを作成する移動マッピング規則。このパッケージはソース・モデル内のパッケージと同じ名前を持っています。
- 要素がクラス・タイプの場合、`Class2ClassTransform` 変換式を呼び出すサブマップ・マッピング規則
- 要素がクラス・タイプの場合、`Class2InterfaceTransform` 変換式を呼び出すサブマップ・マッピング規則
- 要素がパッケージ・タイプの場合、`Package2PackageTransform` 変換式を呼び出すサブマップ・マッピング規則

6 ページの『演習 3: クラスからクラスへのマッピング宣言の作成と洗練』では、マッピング宣言内の 1 つの移動マッピング規則ごとに、ソース・モデルからターゲット・モデルへ属性値をコピーする 1 つの規則が、生成された変換式のソース・コードに追加されることを示しました。サブマップ・マッピング規則ごとに、指定されたコレクションに要素を取り出す 1 つの抽出が、変換式のソース・コード内に生成されます。サブマップ規則は、現在の入力オブジェクトがマッピング宣言で定義される入力タイプのインスタンスである場合に、オブジェクトに適用されます。

生成済みの変換を実行したときに、ソース・モデルにパッケージが含まれる場合、`Package2PackageTransform` 変換式が呼び出され、ターゲット・モデルに同じ名前を持つパッケージを作成します。`Package2PackageTransform` 変換式は、`packagedElement` フィーチャーのコレクションをトラバースします。パッケージ・タイプ (ソース・モデルにネストされたパッケージが含まれることを意味する) であるコレクション要素ごとに、変換式は `Package2PackageTransform` 変換式を呼び出します。クラス・タイプであるコレクション要素ごとに、クラスをターゲット出力モデル内の対応するクラスとインターフェースに変換するために、変換式はこれらの規則を呼び出します。

マッピング・モデルにパッケージからパッケージへのマッピング宣言を作成する手順は、以下のとおりです。

1. **Generalize\_Classes.mapping** ファイルが変換マッピング・エディターで開かれていない場合は、「パッケージ・エクスプローラー」ビューでそのファイルをダブルクリックします。
2. 変換マッピング・エディターの「マッピング・ルート」セクションで、「**Generalize\_Classes**」を右クリックしてから「**マップの作成**」をクリックします。
3. 「新規マップ」ウィンドウの「**マップ名**」フィールドに **Package2Package** と入力してから、「**OK**」をクリックします。「アウトライン」ビューにマップが表示され、エディター領域内の「マッピング・ルート」の下にそのマップが開きます。

## パッケージからパッケージへのマッピング宣言への入出力オブジェクトの追加

マッピング宣言を作成した後、入力オブジェクトおよび出力オブジェクトをそのマッピング宣言に追加する必要があります。この演習では、UML パッケージを入出力オブジェクトとして指定します。

入力オブジェクトおよび出力オブジェクトをパッケージからパッケージへのマッピング宣言に追加する手順は、以下のとおりです。

1. 編集しているマップのツールバーの左端にある「**入力オブジェクトの追加**」アイコンをクリックします。
2. 「入力の追加」ウィンドウの「要素」ペインで、「**uml**」を展開し、「**パッケージ**」をクリックして、「**OK**」をクリックします。
3. 編集しているマップのツールバーの左から 2 番目にあるアイコンである「**出力オブジェクトの追加**」アイコンをクリックします。
4. 「出力の追加」ウィンドウの「要素」ペインで、「**uml**」を展開し、「**パッケージ**」をクリックして、「**OK**」をクリックします。
5. 「**ファイル**」 → 「**保存**」をクリックします。

これで、入力オブジェクトおよび出力オブジェクトのフィーチャーの間にマッピング規則を定義できるようになります。

## パッケージ入出力オブジェクトの属性間のマッピング規則の定義

パッケージの入力オブジェクトおよび出力オブジェクトをマッピング宣言に追加した後、入力オブジェクトおよび出力オブジェクトの属性の間にマッピング規則を作成することができます。

**注:** 変換ソース・コードを生成すると、マッピング宣言ごとに変換式が生成されます。それぞれの生成された変換式には、`getAccept_Condition()` メソッドが含まれます。デフォルトでは、現在の入力要素がマッピング宣言で定義される入力オブジェクトのインスタンスである場合、このメソッドは `true` を返します。メソッドが `false` を返す場合、変換式は現在の入力要素を処理しません。この演習では、入力オブジェクトが入力タイプのインスタンスであるかどうかの検証も行う入力フィルターを作成することができます。これらの入力フィルターは、冗長ではありますがまた無害でもあり、入力フィルターを作成する方法を示すだけのものです。サブマップ・マッピング規則の入力フィルターは、`Java` で書き、ブール値を返す必要があります。

パッケージの入力オブジェクトおよび出力オブジェクトの属性の間にマッピング規則を作成する手順は、以下のとおりです。

1. 入力オブジェクトの `name` 属性のハンドルを、出力オブジェクト内の `name` 属性にドラッグすることで、`name` 属性の間に移動マッピング規則を作成します。

2. 入出力オブジェクトの `packagedElement` 属性の間にサブマップ・マッピング規則を作成し、要素がクラス・タイプの場合にこのサブマップが `Class2ClassTransform` 変換式を呼び出すように指定します。
  - a. 入力オブジェクトの `packagedElement` 属性のハンドルを、出力オブジェクト内の `packagedElement` 属性にドラッグします。 `packagedElement` 属性はコレクションであるため、マッピング・エディターによってサブマップ・マッピング規則がデフォルトで作成されます。
  - b. エディター領域にある新規の「サブマップ」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - c. 「プロパティー」ビューの「詳細」タブの「マップ」リストから、「**Class2Class**」を選択します。  
生成済みの変換を実行すると、入力パッケージの要素ごとに、変換により `Class2ClassTransform` 変換式が呼び出されます。ただし、これは、要素がクラス・タイプの場合のみです。
3. オプション: ステップ 2 で作成したサブマップ規則に、現在のオブジェクトがクラスのインスタンスであるかどうかを検証する入力フィルターを作成します。
  - a. エディター領域にあるステップ 2 で作成した「サブマップ」要素を右クリックします。
  - b. 「プロパティー」ビューの「入力フィルター」タブで、「入力要素のフィルター」チェック・ボックスを選択します。
  - c. 「インライン」をクリックします。
  - d. 「コード」フィールドの下テキスト域のメソッド・シグニチャーの下に、以下のコードを指定します: `return packagedElement_src instanceof org.eclipse.uml2.uml.Class;`
  - e. 「適用」をクリックします。
4. 入出力オブジェクトの `packagedElement` フィーチャーの間にサブマップ・マッピング規則を作成し、`packagedElement` コレクションの現在の要素がクラス・タイプの場合に、`Package2PackageTransform` という変換式がクラスをターゲット・モデル内のインターフェースに変換します。
  - a. ステップ 2a を繰り返して、`packagedElement` フィーチャーの間に別のサブマップを作成します。
  - b. エディター領域にある新規の「サブマップ」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - c. 「プロパティー」ビューの「詳細」タブの「マップ」リストから、「**Class2Interface**」を選択します。
5. オプション: ステップ 4 で作成したサブマップ規則に、現在のオブジェクトがインターフェースのインスタンスであるかどうかを検証する入力フィルターを作成します。
  - a. エディター領域にある 4 で作成した「サブマップ」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - b. ステップ 3b、3c、3d、および 3e を繰り返して、同じ入力フィルターを指定します。
6. ネストされたパッケージを処理するサブマップ・マッピング規則を作成します。`packagedElement` コレクションの現在の要素がパッケージの場合、変換によって `Package2PackageTransform` 変換式が呼び出されます。
  - a. ステップ 2a を繰り返して、`packagedElement` フィーチャーの間に別のサブマップを作成します。
  - b. エディター領域にある新規の「サブマップ」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - c. 「プロパティー」ビューの「詳細」タブの「マップ」リストから、「**Package2Package**」を選択します。
7. オプション: ステップ 6 で作成したサブマップ規則に、現在の入力要素がパッケージのインスタンスであるかどうかを検証する入力フィルターを作成します。

- a. エディター領域にある 6 で作成した「サブマップ」要素を右クリックしてから、「プロパティーで表示」をクリックします。
- b. ステップ 3b と 3c を繰り返して、この規則に入力フィルターが含まれるように指定します。
- c. 「コード」フィールドの下テキスト・エリアのメソッド・シグニチャーの下に、以下のコードを指定します: `return packagedElement_src instanceof org.eclipse.uml2.uml.Package;`
- d. 「適用」をクリックします。

8. 「ファイル」 → 「保存」をクリックします。

マッピング宣言には、name 属性間に 1 つの移動マッピング規則が含まれ、packagedElement 属性間に複数のサブマップ規則が含まれます。

## 演習 7: モデルからモデルへのマッピング宣言の作成と洗練

この演習では、マッピング・モデル内にモデルからモデルへのマッピング宣言を作成する方法を示します。このマッピング宣言には、生成済みの変換を実行したときに、名前がソース入力モデルから派生するターゲット出力モデルを作成するカスタム・マッピング規則が含まれます。また、この演習では、変換によってマッピング・モデル内のマッピング宣言が処理される順序を変更する方法も示します。

変換によって生成されるターゲット・モデルの名前を変更するには、入力オブジェクトおよび出力オブジェクトのフィーチャーの間にマッピング規則を作成する代わりに、入力オブジェクトおよび出力オブジェクトの間にカスタム・マッピング規則を作成することができます。

モデルからモデルへのマッピング宣言を作成する手順は、以下のとおりです。

1. マッピング・モデルが変換マッピング・エディターで開かれていない場合は、「パッケージ・エクスプローラー」ビューで、model フォルダの .mapping ファイルをダブルクリックします。
2. 変換マッピング・エディターの「マッピング・ルート」セクションで、「Generalize\_Classes」を右クリックしてから「マップの作成」をクリックします。
3. 「新規マップ」ウィンドウの「マップ名」フィールドに Model2Model と入力してから、「OK」をクリックします。「アウトライン」ビューにマップが表示され、エディター領域内の「マッピング・ルート」の下にそのマップが開きます。

## モデルからモデルへのマッピング宣言への入出力オブジェクトの追加

モデルからモデルへのマッピング宣言を作成した後、入力オブジェクトおよび出力オブジェクトをマッピング宣言に追加する必要があります。この演習では、モデルを入出力オブジェクトとして指定します。

入力オブジェクトと出力オブジェクトをモデルからモデルへのマッピング宣言に追加する手順は、以下のとおりです。

1. 編集しているマップのツールバーの左端にある「入力オブジェクトの追加」アイコンをクリックします。
2. 「入力の追加」ウィンドウの「要素」ペインで、「uml」を展開し、「モデル」をクリックして、「OK」をクリックします。
3. 編集しているマップのツールバーの左から 2 番目にあるアイコンである「出力オブジェクトの追加」アイコンをクリックします。
4. 「出力の追加」ウィンドウの「要素」ペインで、「uml」を展開し、「モデル」をクリックして、「OK」をクリックします。
5. 「ファイル」 → 「保存」をクリックします。

これで入力オブジェクトおよび出力オブジェクトの間にマッピング規則を定義できるようになります。

## カスタム・マッピング規則およびサブマップ・マッピング規則の定義

この演習では、以下のマッピング規則を作成します。

- 変換によって生成されるターゲット・モデルの名前を変更するカスタム・マッピング規則
- モデル内のパッケージを変換するために、Package2PackageTransform 変換式を呼び出すサブマップ・マッピング規則

マッピング宣言でマッピング規則を作成する手順は、以下のとおりです。

1. ターゲット・モデルの名前を変更するカスタム・マッピング規則を作成します。
  - a. エディター領域内で、モデル入力オブジェクトの上部区画をクリックします。モデル入力オブジェクト全体が強調表示されます。
  - b. モデル入力オブジェクトのハンドルをモデル出力オブジェクトにドラッグします。
  - c. 新規の「サブマップ」要素で、下矢印をクリックして「カスタム」をクリックします。
  - d. 「カスタム」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - e. 「プロパティー」ビューの「詳細」タブで、「インライン」をクリックします。
  - f. 「コード」フィールドの下テキスト・エリアのメソッド・シグニチャーの下に、変換を実行するときにカスタム・マッピング規則を実装する以下のコードを指定します: `Model_tgt.setName(Model_src.getName()+"TgtModel");`
  - g. 「適用」をクリックします。

注: ターゲット・モデルの名前を変更するには、入出力オブジェクトの間にマッピング規則を作成する代わりに、入力オブジェクトおよび出力オブジェクトの `name` 属性の間にカスタム・マッピング規則を作成して、「詳細」タブで同じコードを指定することもできます。

2. 17 ページの『演習 6: パッケージからパッケージへのマッピング宣言の作成と洗練』で作成した Package2Package マッピングを呼び出すサブマップ・マッピング規則を作成します。
  - a. packagedElement フィーチャーのハンドルを、モデル出力オブジェクト内の packagedElement 属性にドラッグします。
  - b. 新規の「サブマップ」要素を右クリックしてから、「プロパティーで表示」をクリックします。
  - c. 「プロパティー」ビューの「詳細」タブの「マップ」リストから、「Package2Package」を選択します。
3. 「ファイル」→「保存」をクリックします。

このチュートリアルに必要なすべてのマッピング宣言とマッピング規則を作成しました。これでマッピング宣言の処理順序を指定できるようになります。

## マッピング宣言の処理順序の変更

マッピング・モデル内のマッピング宣言の処理順序を変更できます。生成済み変換コード実行時の処理順序に従って、「アウトライン」ビューにマッピング宣言がリストされます。処理順序を変更することによって、固有性の低いマッピング宣言が処理および消費する可能性のある入力オブジェクトの処理命令を指定することができます。

このチュートリアルでは、生成済みの変換が以下の順序でマッピング宣言を処理するように指定してください。

- Model2Model

- Package2Package
- Class2Class
- Class2Interface
- Operation2Operation
- Parameter2Parameter
- Primitive2Primitive

注: このチュートリアルでは、Model2Model および Package2Package マッピング宣言の順序が最も重要です。生成済みの変換を実行するとき、モデルがパッケージであるため、Package2Package 変換式はモデルまたはパッケージを受け入れます。Package2Package 変換式が変換でリストされている最初の変換式の場合、この変換式はモデル入力オブジェクトを処理し消費します。このシナリオでは、Model2Model 変換式はモデル入力オブジェクトを処理しません。したがって Model2Model 変換式で作成したカスタム規則は稼動しません。

マッピング宣言の処理順序を変更する手順は、以下のとおりです。

1. 「アウトライン」ビューが表示されていない場合は、「ウィンドウ」→「ビューの表示」→「アウトライン」とクリックします。
2. 「アウトライン」ビューで Model2Model マッピング宣言を右クリックし、「実行順序」をクリックして、「上へ移動」をクリックします。Model2Model マッピング宣言がリストの先頭になるまで、このステップを繰り返します。
3. マッピング宣言が上記のリストと同じ順序になるまで、それぞれのマッピング宣言に対してステップ 2 を繰り返します。
4. 「ファイル」→「保存」をクリックします。

これで変換ソース・コードを生成できるようになります。

---

## モジュール 2: 変換コードの生成と変換の実行

このモジュールでは、モデルからモデルへの変換用のコードを生成し、それを Eclipse ワークベンチの個別のインスタンスで実行することで、変換のテストを行います。

### 学習目標

このモジュールの演習では、以下のタスクを実行する方法を示します。

- 変換ソース・コードを生成してコンパイルします
- ランタイム・ワークベンチを構成します
- 変換構成を作成します
- ランタイム・ワークベンチで変換コードを実行します

### 所要時間

このモジュールを完了するには、約 20 分を要します。

### 前提条件

以下のタスクを実行する方法を知っておく必要があります。

- プロジェクトに UML モデルを作成します

- 基本 UML モデリングを実行します

## 演習 1: 変換ソース・コードの生成とコンパイル

この演習では、変換ソース・コードを生成してコンパイルする方法を示します。

変換ソース・コードを生成する前に、Generalize Classes プロジェクトの src フォルダの内容を調べます。これらのパッケージとファイルは、プロジェクトを作成したときに生成されましたが、他のファイルは Generalize\_Classes.mapping ファイルを編集するときに作成されます。

マッピング規則を段階的に追加したり、マッピング宣言のマッピング規則に対して実装を生成することができます。変換ソース・コードを生成する前に、すべてのマッピング規則を定義する必要はありません。例えば、このチュートリアルを完了した後に、実装クラスからインターフェースへの実現関係を作成するマッピング規則を追加することができます。

変換ソース・コードを生成するとき、変換オーサリング・フレームワークは、マッピング・モデル内のそれぞれのマッピング宣言に対して、*n*Transform.java という名前の Java ソース・ファイルを生成します (この *n* はマッピング宣言の名前を表します)。これらの Java ファイルによって集成的に変換コードが構成されます。変換オーサリング・フレームワークは、変換用の実装コードを生成するほか、変換サービスに変換を登録するためのコードも生成します。

変換ソース・コードを生成してコンパイルする手順は、以下のとおりです。

1. まだ開かれていない場合は、「プラグイン開発」パースペクティブを開き、「ウィンドウ」→「パースペクティブを開く」→「その他」→「プラグイン開発」とクリックしてから、「OK」をクリックします。
2. Generalize\_Classes.mapping ファイルがマッピング・エディターで開かれていない場合は、「パッケージ・エクスプローラー」ビューでそのファイルをダブルクリックします。
3. マッピング・エディター領域の「マッピング・ルート」セクションを右クリックし、「変換ソース・コードの生成」をクリックします。

**ヒント:** また、次のステップを完了することで、変換ソース・コードを生成することもできます: すなわち「パッケージ・エクスプローラー」ビューで .mapping ファイルを右クリックしてから、「変換」→「変換コードの生成」とクリックします。

「パッケージ・エクスプローラー」ビューで、「src」フォルダのパッケージとファイルを調べます。生成された変換式が「src/generalize\_classes.transforms」フォルダに表示されます。

4. 生成されたソース・コードをコンパイルするには、「パッケージ・エクスプローラー」ビューで、「Generalize Classes」プロジェクトをクリックして、「プロジェクト」→「プロジェクトのビルド」とクリックします。デフォルトでは、変更をプロジェクトに保管するときに、Eclipse プロジェクトが自動的にビルドします。ワークスペースまたはプロジェクトのビルド設定を変更しなければ、このステップを実行する必要はありません。

**ヒント:** プロジェクトのビルドの設定を変更するには、「ウィンドウ」→「設定」とクリックして、「一般」を展開し、「ワークスペース」をクリックします。設定を変更した後は、「OK」をクリックします。

5. エラーがあるかどうか、「src」フォルダの内容を調べます。
6. コンパイル・エラーが Class2Interface 変換コードに発生した場合、VisibilityKind パッケージをインポートする必要がある可能性があります。
  - a. 「パッケージ・エクスプローラー」ビューで、「src/generalize\_classes.transforms/Class2InterfaceTransform.java」ファイルのエラーの装飾のあるメソッドをダブルクリックします。

- b. エディター領域内の左マージンにあるエラーの装飾を右クリックしてから、「クイック・フィックス」をクリックして、「**"VisibilityKind" (org.eclipse.uml2.uml)** のインポート (Import **"VisibilityKind" (org.eclipse.uml2.uml)**)」をダブルクリックします。
  - c. 「ファイル」 → 「保存」をクリックします。
7. 「パッケージ・エクスプローラー」ビューで、「**Generalize Classes**」プロジェクトをクリックして、「プロジェクト」 → 「プロジェクトのビルド」とクリックします。

プロジェクトの「src/generalize\_classes.transforms」フォルダーのファイルを確認します。マッピング・モデル内のマッピング宣言ごとに 1 つの Java 変換式が生成されます。「MainTransform.java」ファイルでは、「MainTransform」メソッドにより、「アウトライン」ビューで指定した順序で、生成された変換式ごとにインスタンスが追加されます。

これでランタイム・ワークベンチを構成できるようになりました。

## 演習 2: ランタイム・ワークベンチの構成

この演習では、ランタイム・ワークベンチを構成する方法と、その中にソースおよびターゲットの UML モデルを作成する方法を示します。

ランタイム・ワークベンチを作成して呼び出し、変換のテストおよびデバッグを行うことができます。それはテストする前に変換プラグインをパッケージにする必要がないことを意味します。

ランタイム・ワークベンチを作成して開いた後、変換コードをテストするために、ソース・モデルとおよびターゲット・モデルを作成する必要があります。変換によってソース・モデルで作成した要素が変換され、ターゲット・モデルに出力が生成されます。

ランタイム・ワークベンチを構成する手順は、以下のとおりです。

1. 「プラグイン開発」パースペクティブを開き、「ウィンドウ」 → 「パースペクティブを開く」 → 「プラグイン開発」とクリックします。
2. 「実行」 → 「実行ダイアログを開く」とクリックします。
3. 「実行」ウィンドウの左側で「Eclipse アプリケーション」をクリックし、「新規の起動構成」アイコン (ツールバーの一番左側にあるアイコン) をクリックします。
4. 「構成」タブをクリックします。
5. 「テンプレートとして既存の **config.ini** ファイルを使用」をクリックして、「ロケーション」フィールドにデフォルト値を採用します。これらの値により、ランタイム・インスタンスがデフォルトの Eclipse インスタンスの代わりに、実行している製品のインスタンスとなるように指定されます。デフォルトの Eclipse ランタイム・インスタンスでは、変換をテストするのに十分な機能は提供されていません。
6. 「適用」をクリックします。
7. 「実行」をクリックします。

注: ワークスペースのプロジェクトにエラーが含まれる場合、プロジェクトがリストされるダイアログ・ボックスが表示されます。ランタイム・ワークベンチを開き続けるには、「続行」をクリックします。

新規のランタイム・ワークベンチが開きます。

注: 使用可能なシステム・リソースに応じて、ランタイム・ワークベンチを開くのに数分かかる場合があります。

## ランタイム・ワークベンチにテスト・プロジェクトを作成

ランタイム・ワークベンチを構成して開いた後、以下の項目を含むプロジェクトを作成してください。

- 変換式への変換用のソース UML モデル。このモデルには 1 つのクラスを含むパッケージを含みます。
- 変換によって出力が生成される空のターゲット・モデル。

ソース・モデルおよびターゲット・モデルを含むプロジェクトを作成する手順は、以下のとおりです。

1. ランタイム・ワークベンチで、「モデリング」パースペクティブを開き、「ウィンドウ」→「パースペクティブを開く」→「モデリング」とクリックします。
2. TransformationTest という UML モデリング・プロジェクトと、SourceModel という UML モデルを作成します。
  - a. 「ファイル」→「新規」→「プロジェクト」とクリックして、「モデリング」を展開し、「UML プロジェクト」をクリックしてから、「次へ」をクリックします。
  - b. 「モデル・プロジェクトの作成」ページの「プロジェクト名」フィールドに、TransformationTest と入力します。他のフィールドについては、デフォルト値を採用して、「次へ」をクリックします。
  - c. 「モデルの作成」ページで、まだ選択されていない場合は、「カテゴリー」ペインで、「一般」をクリックします。
  - d. 「テンプレート」ペインで、「ブランク・モデル」をクリックします。
  - e. 「ファイル名」フィールドに、SourceModel と入力します。
  - f. 「終了」をクリックします。
  - g. 「モデリング」パースペクティブへ切り替えるプロンプトが出された場合、「はい」をクリックします。
3. TransformationTest プロジェクトで、TargetModel という UML モデルを作成します。
  - a. 「プロジェクト・エクスプローラー」ビューで、TransformationTest プロジェクトを右クリックしてから、「新規」→「UML モデル」とクリックします。
  - b. 「UML モデル」ウィザードの「モデルの作成」ページで、デフォルト値を採用し、「次へ」をクリックします。
  - c. 2 番目の「モデルの作成」ページで、まだ選択されていない場合は、「カテゴリー」ペインで、「一般」をクリックします。
  - d. 「テンプレート」ペインで、「ブランク・モデル」をクリックします。
  - e. 「ファイル名」フィールドに、TargetModel と入力します。
  - f. 「終了」をクリックします。

これで TransformationTarget プロジェクトに、生成済みの変換によって変換されるソース・モデル、および変換によって出力が生成されるターゲット・モデルが含まれるようになります。
4. SourceModel モデルで、BusinessClasses というパッケージを作成します。そのパッケージには、3 つの private 操作と 1 つの public 操作を持つ、Employee という 1 つのクラスが含まれます。
  - a. 「プロジェクト・エクスプローラー」ビューで、「SourceModel」モデルを右クリックしてから、「UML の追加」→「パッケージ」とクリックします。
  - b. パッケージ名を BusinessClasses と指定します。
  - c. 「BusinessClasses」パッケージを右クリックしてから、「UML の追加」→「クラス」とクリックします。
  - d. クラス名を Employee と指定します。
  - e. 「Employee」クラスを右クリックしてから、「UML の追加」→「操作」とクリックします。

- f. 操作名を readEmail と指定します。
  - g. 「プロパティ」ビューの「一般」タブの「可視性」領域で、「プライベート (Private)」をクリックします。
  - h. ステップ 4e を繰り返し、新規操作名を answerPhone と指定します。
  - i. ステップ 4g を繰り返し、answerPhone 操作が private であることを指定します。
  - j. ステップ 4e を繰り返し、新規操作名を performWork に指定します。
  - k. ステップ 4g を繰り返し、performWork 操作が private であることを指定します。
  - l. ステップ 4e を繰り返し、新規操作名を reportToManager(name:String) に指定します。
5. 「ファイル」 → 「保存」をクリックします。

これで、変換を実行するために使用する変換構成を作成できるようになりました。

### 演習 3: 変換構成の作成

この演習では、変換構成を作成する方法を示します。変換構成は予期した出力を生成するために変換が必要とする情報を指定します。

指定された内容をターゲット・モデルの別の内容に変換するために、変換構成を適用できます。指定されたソースがモデルのとき、変換はソース・モデルの要素を解釈し、指定されたターゲットに出力を生成します。ほとんどの場合、ソース・モデルは未変更のままです。変換によって一式の新規のファイルが生成されるか、または一式の既存のファイルが変更されます。

変換構成ファイルに関する作業を簡単にするには、変換のソースを含む構成ファイルをプロジェクトに保管する必要があります。この演習では、TransformationTest プロジェクトのランタイム・ワークベンチに変換構成ファイルを保管します。変換構成ファイルには .tc というファイル名拡張子が付いています。

変換構成を作成するには、次のようにします。

1. ランタイム・ワークベンチで、「ファイル」 → 「新規」 → 「変換構成」とクリックします。「変換構成」がメニュー項目でない場合は、「ファイル」 → 「新規」 → 「その他」 → 「変換構成」とクリックします。
2. 「新規変換構成」ウィザードの「名前と変換」ページで、以下の値を指定します。
  - a. 「名前」フィールドに、FirstConfiguration と入力します。
  - b. 「前方変換」リストから、「Generalize\_Classes」を展開し、「Generalize\_Classes 変換 (Generalize\_Classes Transform)」を選択します。
  - c. 「構成ファイル宛先」フィールドにまだ指定されていなければ、/TransformationTest と入力します。
  - d. 「次へ」をクリックします。
3. 「ソースとターゲット」ページで、変換によって変換されるソース・モデル、および変換によって出力が生成されるターゲット・モデルを指定します。
  - a. 「選択済みソース」ペインで、「TransformationTest」 → 「モデル」 → 「SourceModel」とクリックします。
  - b. 「選択済みターゲット」ペインで、「TransformationTest」 → 「モデル」 → 「TargetModel」とクリックします。
4. 「次へ」をクリックして、ウィザードの残りのページでデフォルト値を採用します。
5. 「終了」をクリックします。

変換構成ファイルがランタイム・ワークベンチで作成され、TransformationTest プロジェクトの変換構成エディターに、変換構成ファイルが表示されます。これで変換を実行できるようになりました。

## 演習 4: 変換の実行

この演習では、ランタイム・ワークベンチで変換を実行する方法を示します。変換構成を適用すると、変換のインスタンスが作成され、変換構成で指定したプロパティーを使用して変換が実行されます。

変換を実行すると、一時モデルが作成され、このモデルと変換構成で指定したターゲット・モデルを比較します。2 つのモデル間の違いは、「マージ」ウィンドウに表示されます。

変換の実行およびマージ戦略の指定についての詳細は、下の関連トピックを参照してください。

Generalize\_Classes 変換を呼び出す変換構成を適用する手順は、以下のとおりです。

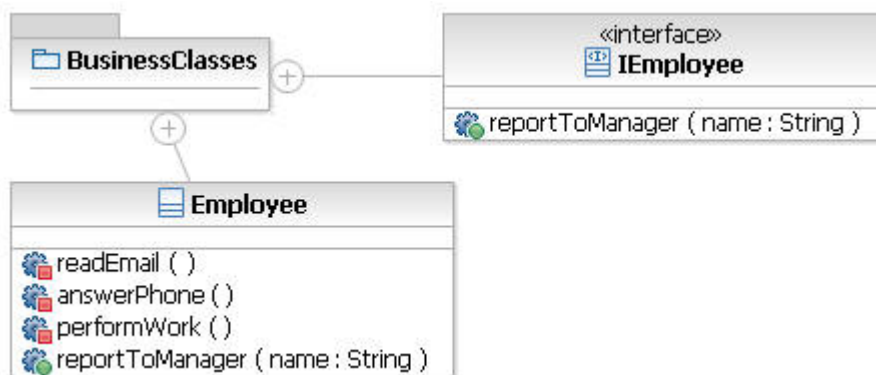
1. まだ開かれていない場合は、ランタイム・ワークベンチの「プロジェクト・エクスプローラー」ビューで、TransformationTest プロジェクトの「**FirstConfiguration.tc**」をダブルクリックしてから、変換構成エディターのメインページで「実行」をクリックします。

**ヒント:** また、TransformationTest プロジェクトで、「**FirstConfiguration.tc**」を右クリックしてから、「変換」→「**Generalize Classes 変換 (Generalize Classes Transform)**」とクリックすることもできます。

2. 変換プロバイダーが指定したデフォルトのマージ・オプションに基づいて、変換が実行している間、ターゲット・モデルのファイルへの変更を受け入れるかどうかのプロンプトが出される可能性があります。「**OK**」をクリックします。
3. 表示されているメッセージに応答して、「**OK**」をクリックします。
4. 「マージ」ウィンドウで、ターゲット・モデルへの指定された更新を表示し、チェック・ボックスを選択またはクリアして、指定された変更を受け入れるかまたは拒否し、「**OK**」をクリックします。
5. 表示されているメッセージに応答して、「**OK**」をクリックします。

**ヒント:** この変換を再び実行するには、「モデリング」→「変換」→「最後に実行された変換を実行」をクリックします。最後に変換を実行した時と同じソース要素を使用して変換が行われます。

これで TargetModel モデル内の変換出力を検討できるようになりました。下の画像は生成された SourceModelTgtModel 出力モデルの要素のビジュアル表示を示しています。このモデルには BusinessClasses というパッケージが含まれ、そのパッケージには Employee というクラスと IEmployee というインターフェースが含まれています。



関連情報:

モデルからモデルへの変換マッピング・プロジェクトで生成されたモデルでのマージ・サポートの追加  
(Adding merge support for models generated by model-to-model transformation mapping projects)

変換の実行および再実行 (Running and rerunning transformations)

---

## 要約: モデルからモデルへの変換の作成、構成、および実行

このチュートリアルでは、モデルからモデルへの単純なマッピング変換の作成、構成、および実行する方法を示しました。

### 学習した内容

このチュートリアルでは、モデルからモデルへのマッピング変換の概念について説明し、また以下のタスクを実行する方法を示しました。

- 変換フレームワークおよびマッピング・モデルを含む、モデルからモデルへの変換オーサリング・プロジェクトを作成します
- マッピング・モデル内にマッピング宣言を作成します
- 以下のタスクを実行して、マッピング宣言を洗練します。
  - それぞれのマッピング宣言に対して、ソース入力オブジェクトとターゲット出力オブジェクトを指定します
  - マッピング宣言の入力オブジェクトと出力オブジェクトの属性間の関係を定義して、マッピング規則を作成します
- 変換ソース・コードを生成してコンパイルします
- モデルからモデルへの変換をテストするために、ランタイム・ワークベンチを構成します
- ランタイム・ワークベンチで、モデルからモデルへの変換を実行する変換構成を作成して適用します

### 追加リソース

このチュートリアルのトピックについてさらに詳しく学習したい方は、以下のソースを検討してください。

- オーサリング変換および変換拡張のオンライン・ヘルプ