ENTERPRISE ARCHITECT

Code generation from Activity/Sequence diagrams

by SparxSystems Japan

アクティビティ図・シーケンス図からのコード生成 機能ガイド

(2022/03/10 最終更新)

SPARX SYSTEMS

目次

1 はじめに 3
2 サンプルを利用してコード生成までの手順を確認する 3
3 アクティビティ図からのコード生成 5
3.1 処理の呼び出し（振る舞い呼び出しアクション) 6
3.2 if 文（ガード条件) 8
3.3 while 文, for 文 8
3.4 並列処理 10
3.5 アクション要素(通常) 11
3.6 操作呼び出しアクション 12
3.7 アクティビティ図からのコード生成のメリットとデメリット 13
3.8 コード生成のためのアクティビティ図の追加方法 14
4 シーケンス図からのコード生成 15
4.1 ライフラインとメッセージ 17
4.2 シーケンス図からのコード生成のメリットとデメリット 18
4.3 コード生成のためのシーケンス図の追加方法 19
4.4 シーケンス図作成時の注意事項 21
4.5 振る舞い図から生成される操作と、クラス図での操作の関係 22

1 はじめに

このドキュメントでは、アクティビティ図やシーケンス図からソースコード生成を行うための手順の概要を紹介しています。

この機能は、ユニファイド版あるいはアルティメット版で利用できます。プロフェッショナル版・コーポレート版ではこの機能は利用できません。

アクティビティ図やシーケンス図からコード生成ができる言語は、C 言語, C++, Java, C#, VB.NET です。なお、C 言語の場合には、オプションの「オブジェクト指向のサポート」を「True」に設定しなければなりません。
(オプション画面の「C」グループに、「オブジェクト指向のサポート」の項目があります。)

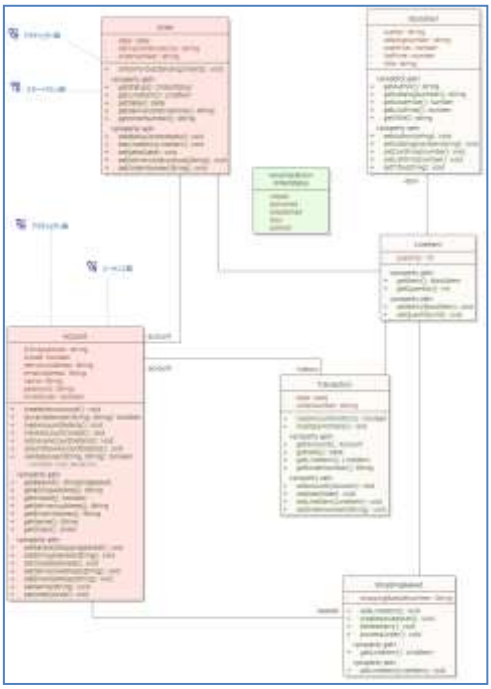
このドキュメントは、Enterprise Architect 16.0 ビルド 1603 を利用して作成しました。それ以外のビルドの場合、サンプルの内容や動作結果が異なる場合があります。

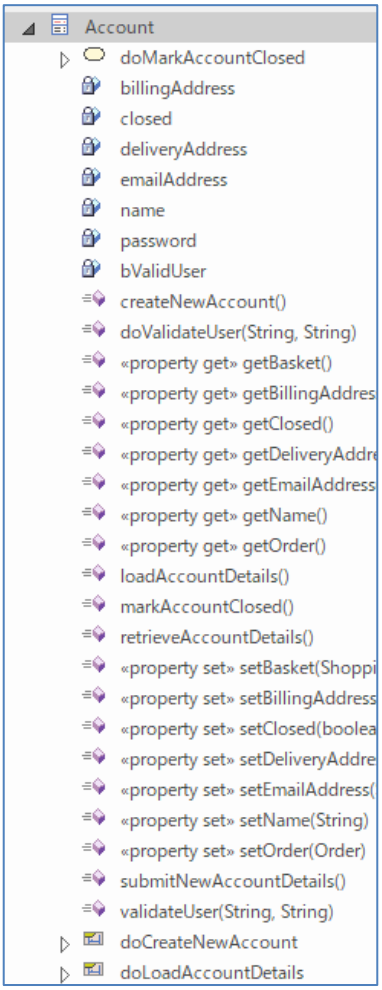
2 サンプルを利用してコード生成までの手順を確認する

このソースコードの生成の作業の流れを理解するための一番良い方法は、サンプルを利用することです。この章では、このサンプルからソースコードを生成するまでの手順を紹介します。

まず、Enterprise Architect のサンプルプロジェクトファイル「EAExample.qea」を開きます。このプロジェクトは、「ホーム」リボン内の「ヘルプ」パネルにある「ヘルプ」ボタンを押すと表示されるメニューから「サンプルプロジェクトを開く」を実行すると開くことができます。

このファイルを開くと自動的にダイアグラムが開きますので、「ソフトウェアモデリング」→「振る舞い図からのソースコード生成」とたどると、「Account」クラスが配置されたダイアグラムが開きます。





この「Account」クラスの子要素として、「doMarkAccountClosed」というアクティビティ要素と、「doCreateNewAccount」「doLoadAccountDetails」という相互作用要素があります。

なお、「Order」クラスにも、アクティビティ図とステートマシン図が定義されています。

このように、クラス要素の子要素として、アクティビティ要素あるいは相互作用要素を作成してその中にモデルを作成することで、クラス要素の操作として出力することができます。
(ステートマシン図についても同様です。PDF ドキュメント「ステートマシン図からのコード生成 機能ガイド」にて説明しています。)

ソースコードの生成方法は、通常のソースコード生成と同じです。つまり、対象のクラス要素をダイアグラム内で選択した状態で、「コード」リボン内の「ソースコード」パネルにある「生成」ボタンを押してください。メニューが表示されますので、「選択した要素」を選択します。なお、シーケンス図・アクティビティ図(およびステートマシン図)からソースコード生成を行う場合に既にソースファイルが存在する場合、必ず「上書き」にする必要があります。既存のソースコードに対して、内容を同期させることはできません。

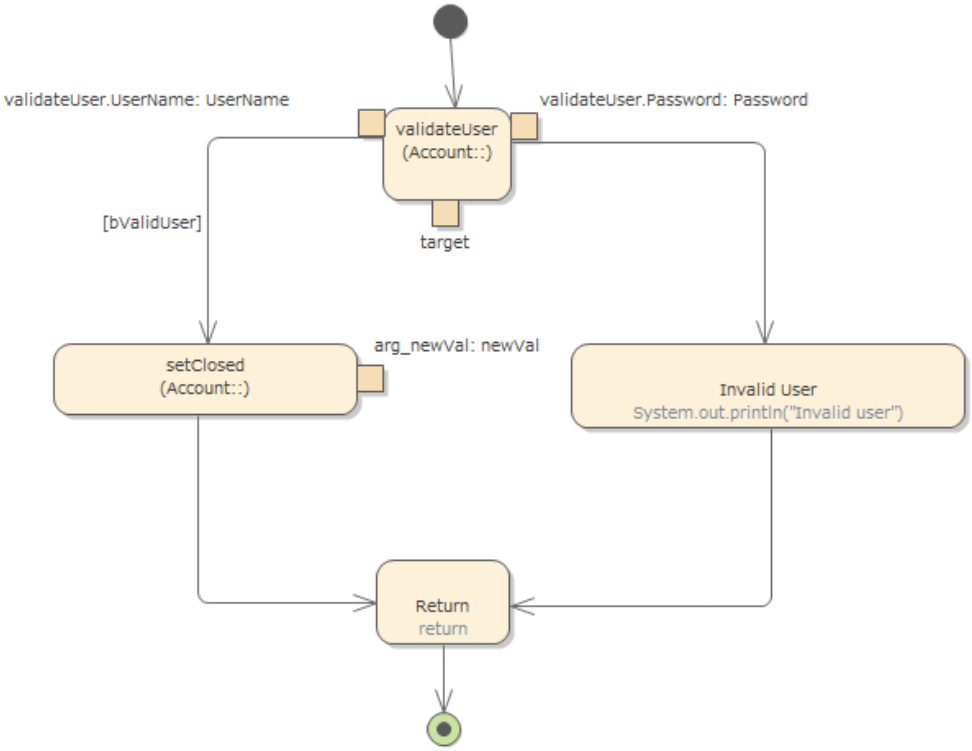
（「上書き」にする設定は、「プロジェクト」リボン内の「ツール」パネルにある「オプション」ボタンを押してプロジェクトのオプション画面を表示し、「ソースコードの生成と読み込み」グループにある「既存のファイルを上書き」を選択してください。）

また、既存のソースコードからシーケンス図・アクティビティ図・ステートマシン図を生成(リバース)したり、ソースコードを編集した結果からモデルを同期更新したりすることもできません。

また、モデルの内容については、不適切な内容があったとしてもエラーが表示されることはありません。不適切な内容のソースコードが生成されますので、確認が必要です。

3　アクティビティ図からのコード生成

次に、アクティビティ図からのコード生成の結果を確認します。先ほどのサンプルのアクティビティ図と、その図から生成したソースコードは以下の通りです。



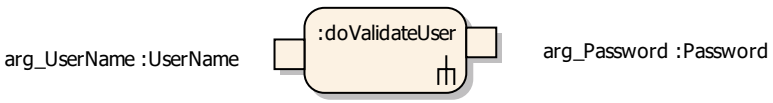

```
public void doMarkAccountClosed(String name, String password)
{
```

```
        // behavior is a Activity
        validateUser(name,password);
        if (bValidUser)
        {
                setClosed(true);
        }
        else
        {
                System.out.println("Invalid user");
        }
        return;
}
```

このように、アクティビティ図の内容を解釈して、ソースコードを生成します。なお、アクティビティ図の中で出力対象として処理されるのは、アクション要素のみです。アクティビティ要素をアクティビティ図に配置しても無視されます。

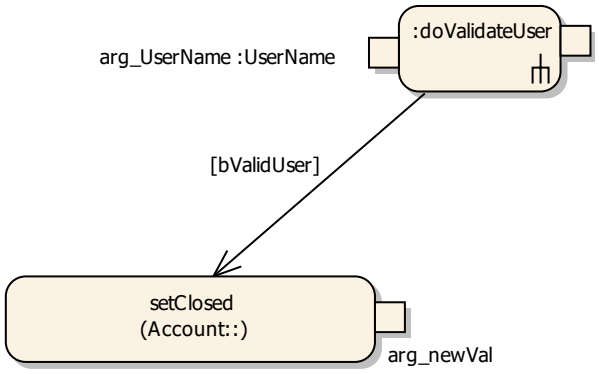
それぞれの内容の概要を説明します。

3.1 処理の呼び出し (振る舞い呼び出しアクション)



これは、「振る舞い呼び出しアクション」です。他のアクション要素をアクティビティ図にドロップすることでも作成できます。新規に作成する場合には、関連する振る舞い要素(アクティビティなど)を指定します。

左右にある四角形の「アクションピン」は、操作の引数を示します。次の例では、引数が 2 つある操作を示しています。



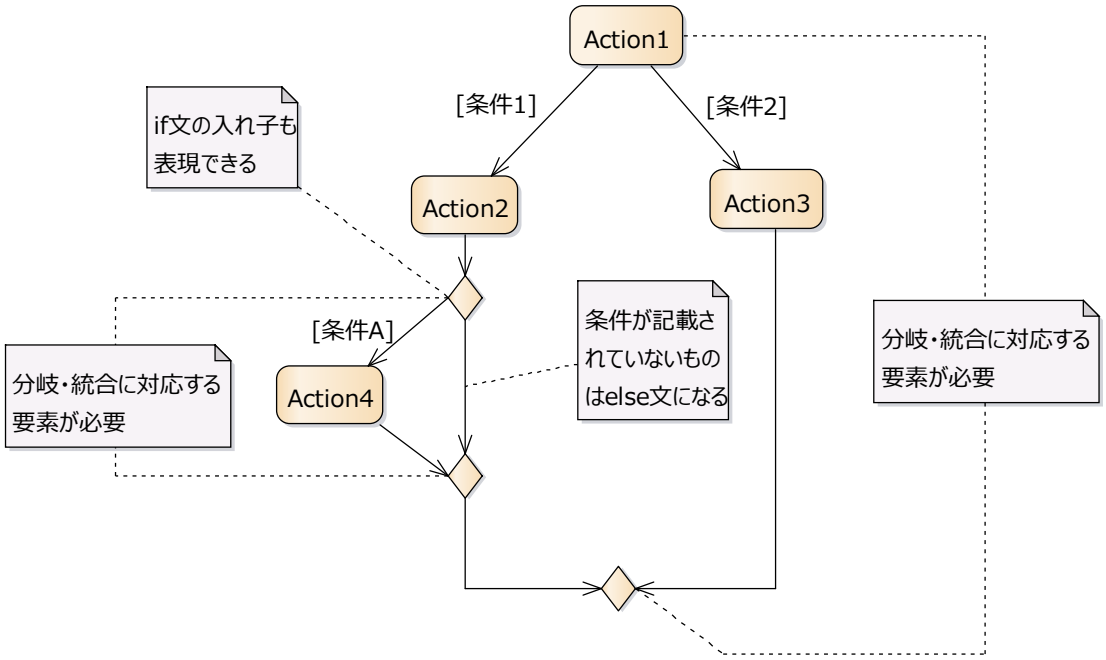
この処理の呼び出しで最も多い活用状況は、既存のクラス要素が持つ、操作を呼び出す場合です。この場合には、次のような手順で作成します。

1. モデルブラウザ内の、対象の操作をアクティビティ図内にドラッグ&ドロップします。自動的に振る舞い呼び出しアクション要素が作成されます。
2. 操作にパラメータ(引数)がある場合には、アクションピンを追加します。プロパティサブウィンドウが表示されている状態で作成したアクション要素を選択すると、プロパティサブウィンドウにはアクションの情報が表示されます。「振る舞い」タブにある「パラメータと同期」ボタンを押しますと、アクションピン要素が追加されます。複数のアクションピンがある場合には同じ位置に配置されますので、ダイアグラム内の配置を調整してください。
   同期後、「引数」タブからそのパラメータについてコード生成する場合に、実際にソースコードに出力される値や変数を指定できます。
3. 操作に戻り値がある場合にその戻り値を変数に格納したい場合も、アクションピンを追加する必要があります。プロパティサブウィンドウの「振る舞い」タブにある「結果」の枠にある「追加」ボタンを押すとピンの選択画面が表示されますので、画面左下の「新規追加」ボタンを押し、戻り値を格納するアクションピンを追加します。
   追加したアクションピンをダイアグラム内に表示するためには、アクション要素を右クリックして「属性・操作と付属要素」→「付属要素」を選択し、追加したアクションピンを表示するようにチェックを入れて下さい。
   追加後は、アクションピン要素の名前を変更することで、出力先となる変数名を指定できます。

3.2 if文 (ガード条件)

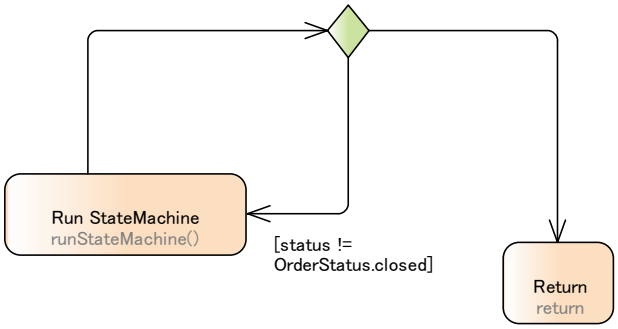
上記の図のように、コントロールフローがデシジョン要素によって分岐し、コントロールフローに「ガード条件」がある場合には、そのまま if 文の条件になります。ガード条件を指定しない場合には、else 文になります。

if 文となる内容で分岐する場合には、その分岐の対応を明確にするために、統合する箇所にマージ要素が必要です。また、if 文を入れ子にすることもできます。以下の図がその例です。

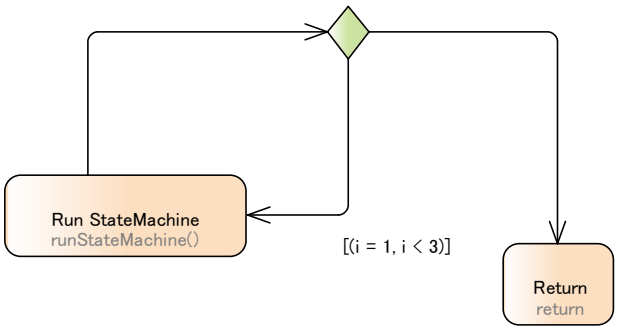


3.3 while文, for文

デシジョン要素を利用して以下のような構成にすることで、条件を満たすまで処理を繰り返す while 文を生成することができます。デシジョン要素を利用する場合には、下の図のように出力するフローを 2 つ作成し、そのうちの 1 つにはガード条件がない(デシジョンでの処理が終了した後の処理先となる)ようにする必要があります。



for 文を生成する場合にも、同様にデシジョン要素を利用します。その際には、ガード条件を「(初期設定,条件)」と表現する必要があります。例えば、以下の図のように「(i = 1, i < 3)」と設定した場合には、「for (i = 1; i < 3; ;)」と生成されます。
(for 文の 3 つ目のパラメータを設定することはできませんので、ループ内で加算などの処理をするようなアクションを追加するか、コード生成テンプレートをカスタマイズして下さい。)



なお、この for 文の生成については、既定のコード生成テンプレートでは出力されないようになっています。for 文の生成を利用する場合には、コード生成テンプレートの「Action Loop」テンプレート内に以下のような内容がありますので、変更して下さい。
(下のテンプレートの例は Java の例です。他の言語の場合でも同様の修正をする必要があります。)

既定の内容:

```
%if $expression != ""%
while ($expression)¥n
%elseIf $lower != ""%
for ($lower; $upper; ;)¥n
%else%
```

```
while (true)¥n
%endIf%
```

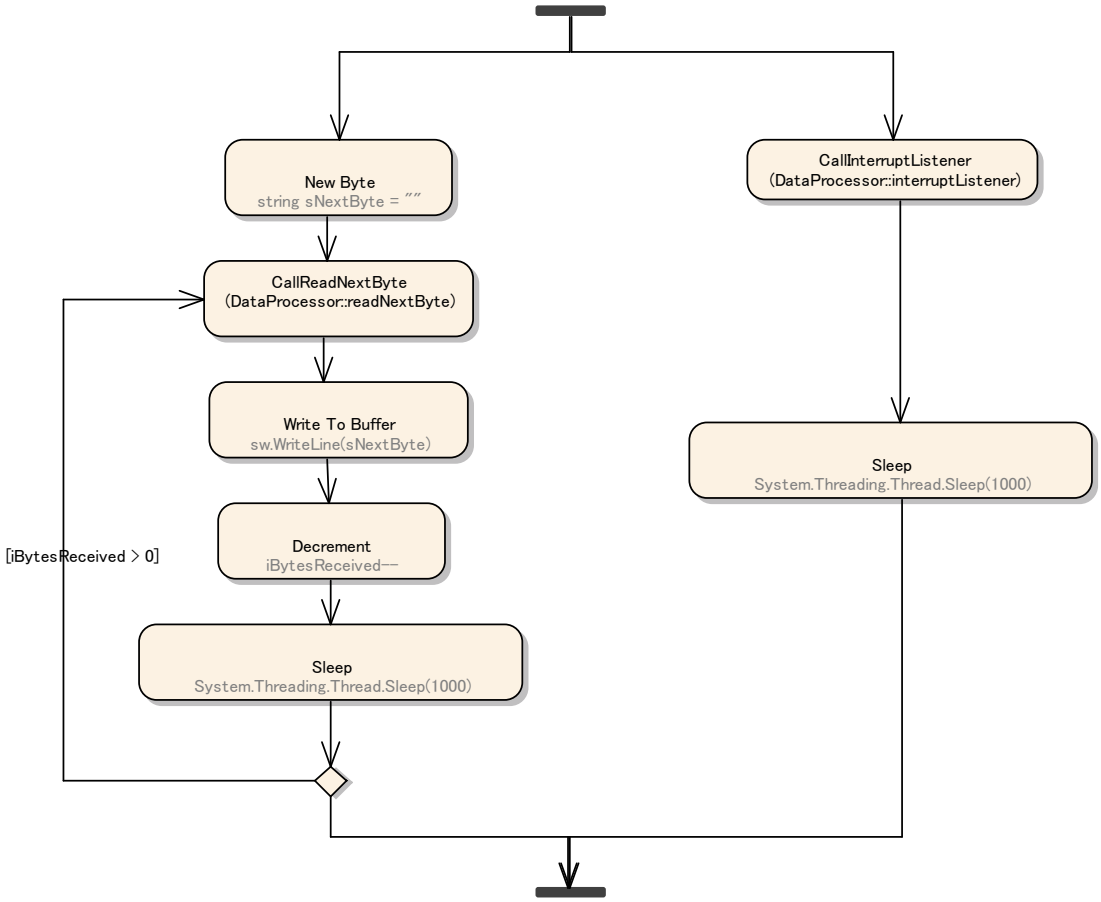
変更後の内容:

```
%if $lower != ""%
for ($lower; $upper; ;)¥n
%elseIf $expression != ""%
while ($expression)¥n
%else%
while (true)¥n
%endIf%
```

コード生成テンプレートおよびそのカスタマイズについては、PDF ドキュメント「コードテンプレートフレームワーク(CTF) 機能ガイド」やヘルプファイルをご覧下さい。
https://www.sparxsystems.jp/products/EA/ea_documents.htm

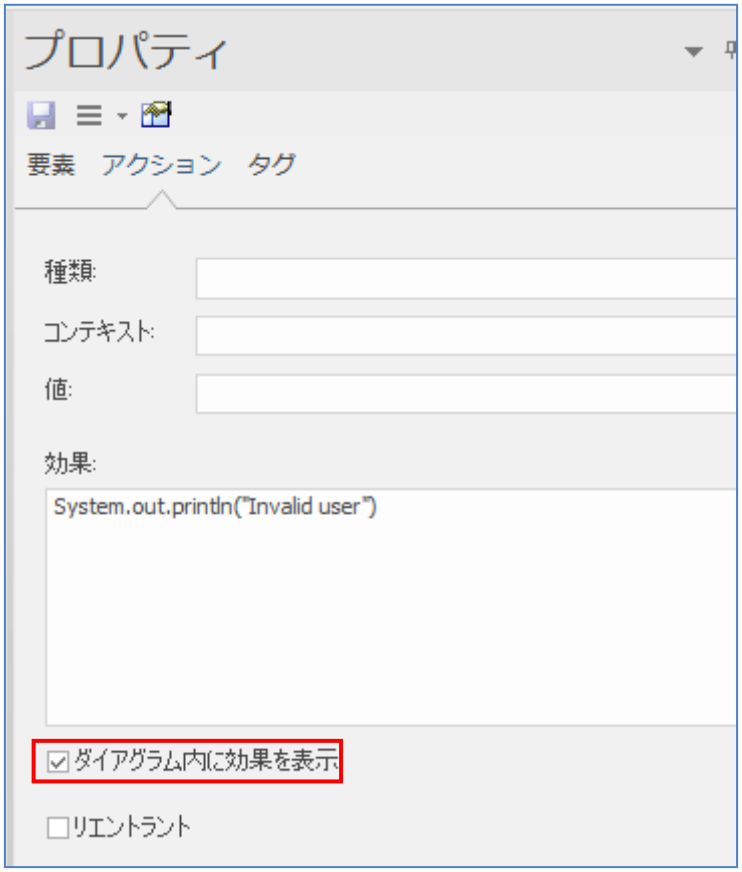
3.4 並列処理

以下のように「フォーク/ジョイン」要素を利用することで、挟まれた部分について並列処理のコードを生成することができます。ただし、C#/Java のみの対応です。



3.5 アクション要素(通常)

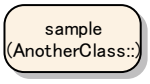
通常のアクション要素を配置した場合には、プロパティサブウィンドウの「アクション」タブの「効果」欄に、処理内容を記述します。「ダイアグラム内に効果を表示」にチェックを入れることで、上記のように処理内容をダイアグラム内で確認することができます。



3.6 操作呼び出しアクション

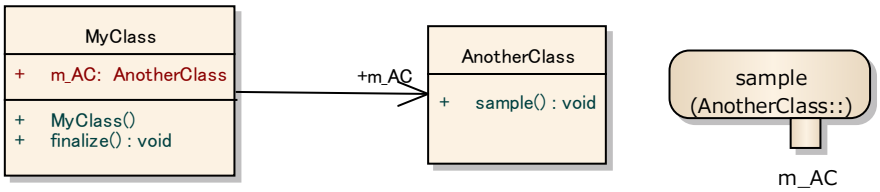
クラスの操作を呼び出す場合には、「操作呼び出しアクション」を作成します。プロジェクトブラウザにある「操作」をドロップしても作成できます。この状態のままでは、どのインスタンス(属性)の操作を呼び出すかが関連づけられていません。この関連づけの部分は手作業で行う必要があります。

まず、プロジェクトブラウザから操作をアクティビティ図にドロップし、操作呼び出しアクションを作成します。以下の例は、AnotherClass の sample 操作をドロップした場合の例です。



その後、このアクションをダブルクリックして、名前として、クラスを保持している属性の名前を入力してください。例えば、以下のようにアクティビティ図を持つクラス

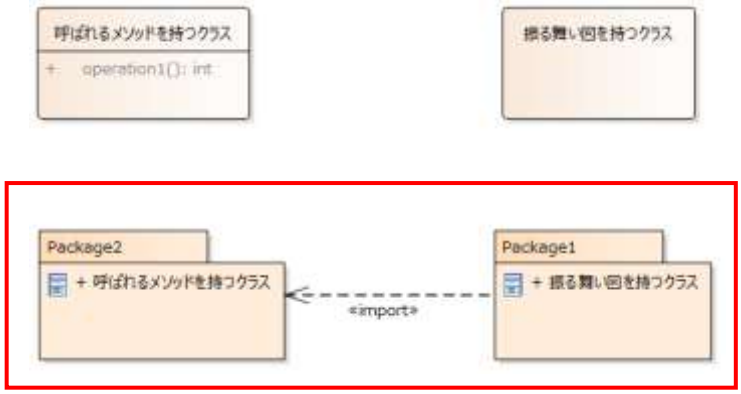
(MyClass)が、AnotherClass を m_AC という属性で保持している場合には、アクションに付属するピンの名前を「m_AC」に変更します。(既定値は target という名前になっています。)



これで、以下のような出力になります。

```
public void Activity()
{
        // behavior is a Activity
        m_AC.sample();
}
```

なお、異なるパッケージに含まれるクラス要素が持つ操作の呼び出しの場合には、以下のようなパッケージ図を作成し、「インポート」の関係で結んでください。
(赤枠部分のみが必要です。上のクラス要素は、参考のために記載しています。)



3.7 アクティビティ図からのコード生成のメリットとデメリット

このように、アクティビティ図での操作の定義は簡単です。このアクティビティ図を利用して操作(メソッド) の処理内容を定義するメリットは、次の通りです。

- 全体の流れ(特に分岐)がわかりやすい
- 他の操作やアクションをドロップして配置するだけで、他の操作の呼び出しを定義できる
- 内容に変更がある場合には、要素の追加や削除・フローの付け替えといった操作でグラフィカルに変更できる
- あるいは、コード生成テンプレートをカスタマイズすることで、モデルの内容をいっさい変更することなく、ソースコードの生成結果を変更できる。
  (クラス要素やアクティビティ図が多数ある場合に、一貫性があり確実な変更が可能)
- 他の設計要素とのトレーサビリティを確保できる
  (トレーサビリティについては、他のドキュメントや動画デモをご覧ください。)

一方で、デメリットもあります。多くの場合には、モデルを書いてからソースコード生成するよりは、直接ソースコードを書く方が早いです。また、書ける内容にも制限があって自由に記述することができません。そのため、この機能を使う目的や意味をきちんと定義し、それが関係者に周知されていなければ、効果を発揮することはできません。

具体的には、設計者の人数が多い場合に、「ソースコードを書く人によって内容が変わる」「決まった形のみのソースコードを強制したい」などの目的がある場合には、この機能が有用です。一方で、少人数での設計・実装の場合や、実装(ソースコード)に各実装者の裁量を認める方が効率的な場合には、この機能は使うべきではないでしょう。

3.8 コード生成のためのアクティビティ図の追加方法

クラス要素に対して、新規にアクティビティ図を追加する場合の手順は次の通りです。

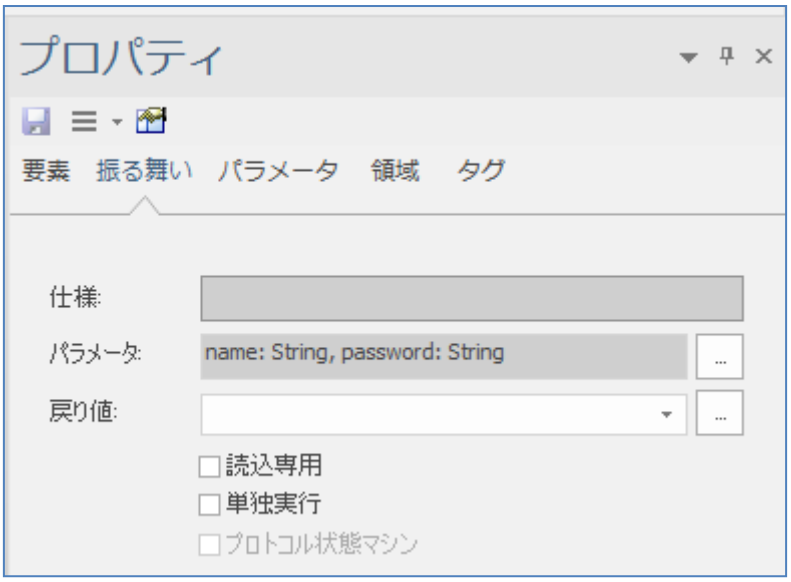
まず、対象のクラス要素をダイアグラム内で右クリックして、「子ダイアグラムの追加」→「アクティビティ」→「アクティビティ図の追加」を選択してください。

すると、下の図のようにアクティビティ要素とアクティビティ図が追加されます。

このアクティビティ要素の名前は、そのままクラス要素の操作の名前になりますので、名前を変更してください。図の名前はコード生成には影響しませんので、変更しなくても構いません。

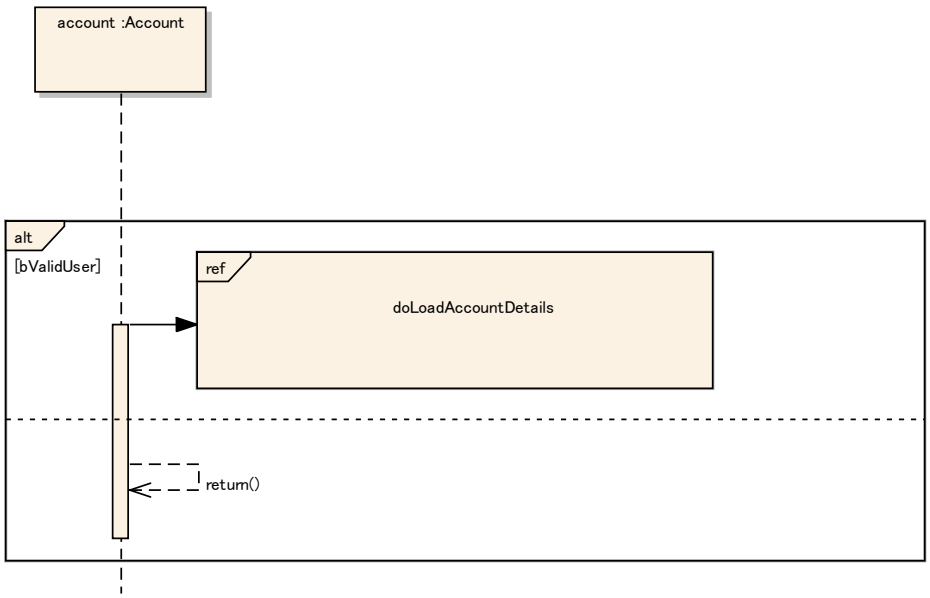
これで準備は完了です。あとは、追加されたアクティビティ図で処理内容をモデリングしてください。「開始」「終了」の要素をそれぞれ 1 つずつ追加してください。内部では、アクション要素のみがソースコードの出力対象になります。アクティビティ要素が含まれる場合には無視されます。
(アクティビティ図にアクティビティ要素を配置するのは UML の仕様としては誤りです。)

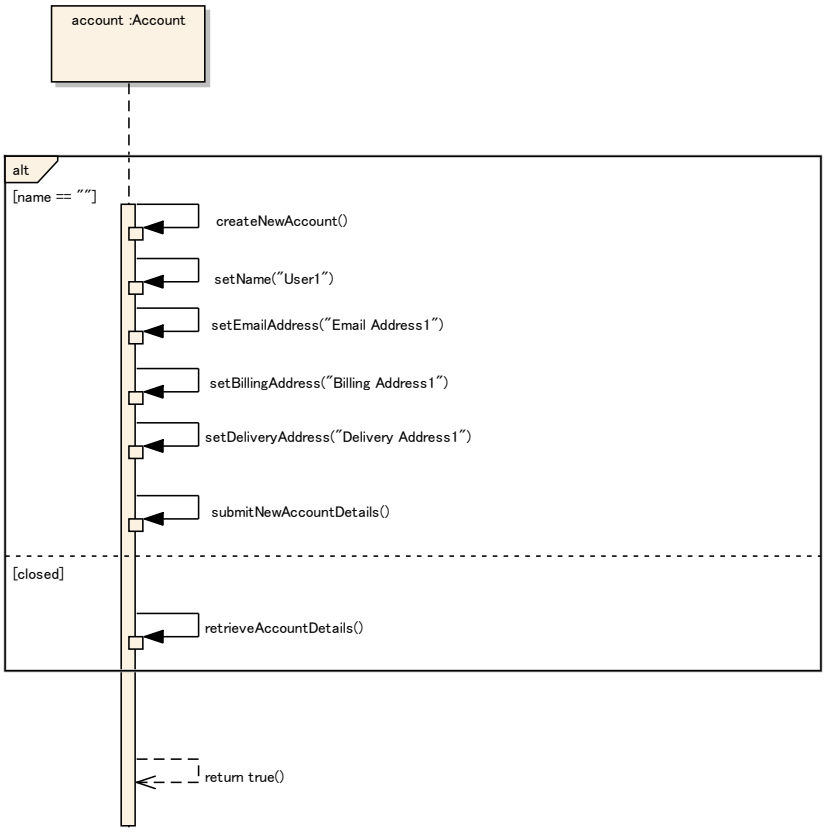
また、必要に応じて、(操作として生成される)アクティビティ要素のパラメータ(引数)や戻り値を指定してください。これらの情報は、アクティビティ図の親になっているアクティビティ要素を選択するとプロパティサブウィンドウに表示される「振る舞い」グループで指定することができます。



4 シーケンス図からのコード生成

次に、シーケンス図からのコード生成の概要を説明します。こちらも、サンプルのシーケンス図と、生成結果を示します。






```
public void doCreateNewAccount()
{
        // behavior is a Interaction
```

```
        if (bValidUser)
        {
                doLoadAccountDetails();
        }
        else
        {
        }
        return;
}
public void doLoadAccountDetails()
{
        // behavior is a Interaction
        if (name == "")
        {
                this.createNewAccount();
                this.setName("User1");
                this.setEmailAddress("Email Address1");
                this.setBillingAddress("Billing Address1");
                this.setDeliveryAddress("Delivery Address1");
                this.submitNewAccountDetails();
        }
        else if (closed)
        {
                this.retrieveAccountDetails();
        }
        return true;
}
```

4.1 ライフラインとメッセージ

このように、シーケンス図ではそれぞれのメッセージが、クラスの操作(メソッド)を示します。モデルブラウザからクラスをインスタンスとして配置し、メッセージを追加します。メッセージに対応する操作やパラメータ(引数)の情報は、メッセージのプロパティ画面で指定します。次の例は、実パラメータ(実引数)を指定している例です。

4.2 シーケンス図からのコード生成のメリットとデメリット

シーケンス図を利用して操作(メソッド)の処理内容を定義するメリットは、次の通りです。

- 複合フラグメントを利用することで、分岐処理やループ処理を表現できる
- 相互作用の利用(ref のフラグメント)を利用することで、他のシーケンス図で定義した内容を呼び出すことができる
- 処理の表現がソースコードに近いので、理解しやすい

このように、アクティビティ図でのモデリングに比べれば、シーケンス図を利用するメリットは薄いです。特に、シーケンス図の場合、自分のクラス要素にのみ関係のあるシーケンス図となりますので、設計段階で広く使われているような複数のライフラインが配置

され、ライフライン間でメッセージがやりとりされるようなシーケンス図とは異なる、ソースコードを生成するためだけのシーケンス図を作成する必要があります。
(アクティビティ図のような「モデル」としてのメリットは、アクティビティ図を利用する方が上です。)

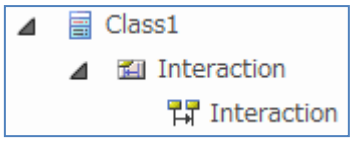
前の章でアクティビティ図を利用することのデメリットとして挙げた項目は、そのままシーケンス図を利用することのデメリットにも適用できます。

4.3 コード生成のためのシーケンス図の追加方法

クラス要素に対して、新規にシーケンス図を追加する場合の手順は次の通りです。

まず、対象のクラスをダイアグラム内で右クリックして、「子ダイアグラムの追加」→「相互作用」→「シーケンス図の追加」を選択してください。

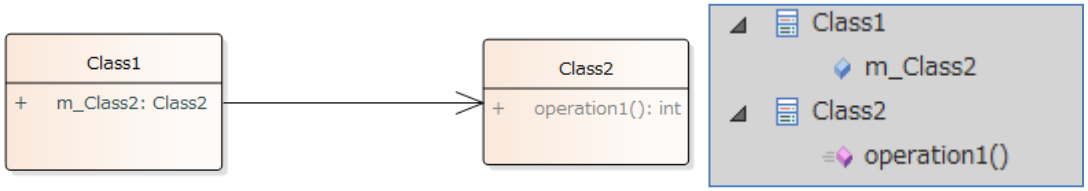
すると、下の図のように相互作用要素とシーケンス図が追加されます。



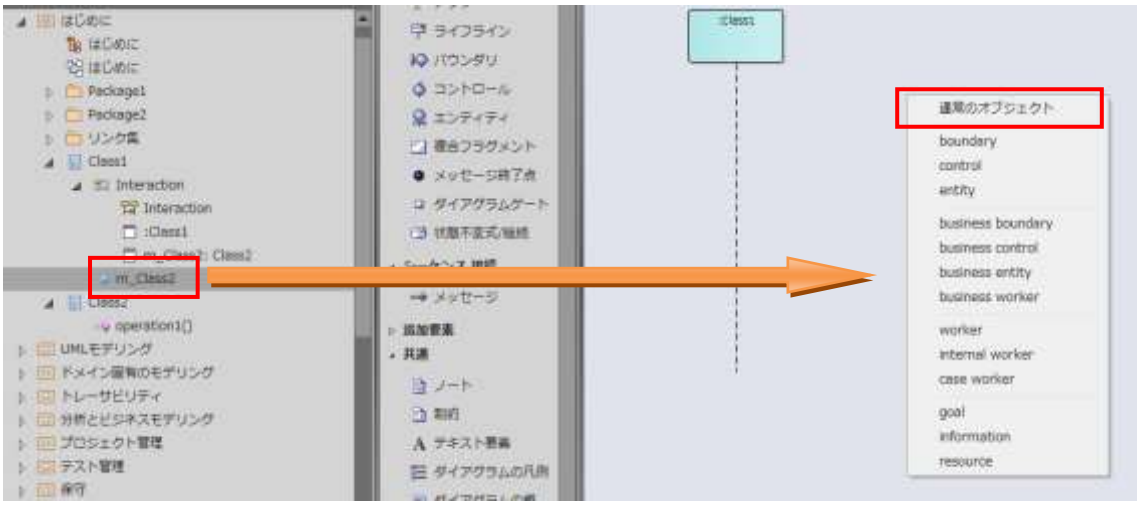
この相互作用要素の名前は、そのままクラスの操作の名前になりますので、名前を変更してください。図の名前は、変更しなくても構いません。

これで準備は完了です。あとは、追加されたシーケンス図にて処理内容をモデリングしてください。

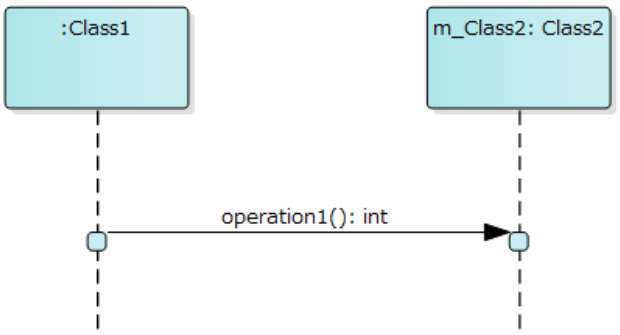
他のクラスの操作を呼び出す場合には、プロジェクトブラウザから対象のクラスまたは属性をドロップします。一般的には、呼び出す先のクラスを属性として保持しているはずですので、属性をシーケンス図にドロップします。下の例では、クラス Class1 が、属性 m_Class2 としてクラス Class2 を保持している場合で、この Class2 が持つ操作 Operation2 を呼び出す、というような場合です。



このような場合には、まず Class1 に対して前述のように「シーケンス図の追加」を実行します。そのシーケンス図に、Class1 および、Class1 が持つ属性 m_Class2 をドロップします。属性 m_Class2 をドロップしたときには以下のようにメニューが表示されますが、「通常のオブジェクト」を選択してください。



その後、ライフライン間にメッセージを作成し、メッセージのプロパティ画面でクラス Class2 の操作「Operation2」を選択します。



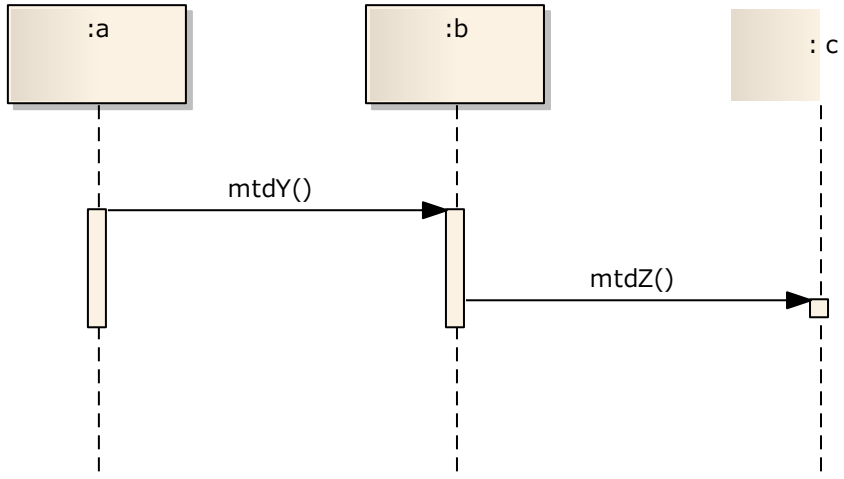
この状態でソースコードの生成を実行すると、以下のようなコードを生成できます。

```
public void Interaction()
{
        // behavior is a Interaction
        m_Class2.operation2();
}
```

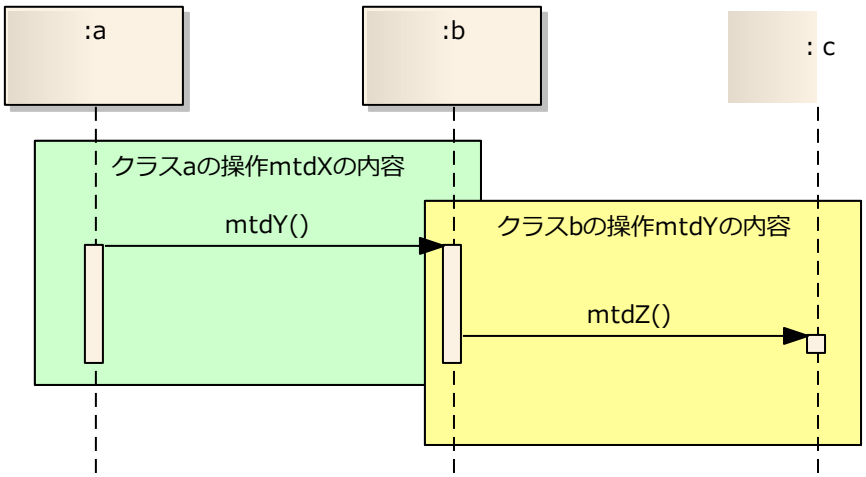
4.4 シーケンス図作成時の注意事項

Enterprise Architect のシーケンス図の作成の機能は、あるクラスの操作(メソッド)の内容を記述するものになります。このシーケンス図には、別の操作や別のクラスの内容は含まれません。

設計でのシーケンス図では、以下のように複数のインスタンスの間のやりとりを表現することが多くあります。この例では、クラス a の操作 mtdX が呼ばれたときの処理内容を記載しているものとします。クラス a の操作 mtdX の内部で、クラス b の操作 mtdY が呼ばれ、クラス b の操作 mtdY の中から、クラス c の操作 mtdZ が呼ばれている様子を示しています。
(クラス a の操作 mtdX が呼ばれる、という内容はこのシーケンス図には含まれていません。クラス a の操作 mtdX が呼ばれた場合の処理内容を示す図であるからです。)



ソースコードを生成するためのシーケンス図は、このような内容にはなりません。その意味を、以下の図に示しました。



このように、上のシーケンス図の黄色の枠の部分は、クラス b の内容になりますので、別途クラス b の操作 mtdY のためのシーケンス図をクラス b の下に作成しなければなりません。クラス a の操作 mtdX のシーケンス図には、クラス a が直接関係する緑の枠の部分のみを記載しなければなりません。

このように、ソースコードを生成するためのシーケンス図は、設計者が設計を行うためのシーケンス図とは目的・記述範囲が異なりますので、ご注意ください。

また、異なるパッケージに含まれるクラスの操作を呼び出すようなソースコードを生成するためには、パッケージ図での定義が必要です。アクティビティ図に関する説明と同じです。

4.5 振る舞い図から生成される操作と、クラス図での操作の関係

このドキュメントで説明している振る舞いからのソースコード生成の場合には、クラスの操作としては表現されません。つまり、クラス図には操作として表示されませんので、ご注意ください。

クラスの操作としても定義したい場合(=クラス要素に操作として表現したい場合)には、アクティビティ要素あるいは相互作用要素と同名の操作を作成後、その操作を選択した状態でプロパティサブウィンドウの「関係する振る舞い」の欄に、アクティビティ要素あるいは相互作用要素を指定してください。このように設定することで、クラスの操作として表示されるが、出力される操作が重複しないようにすることができます。

○ 改版履歴

2009/03/31 初版
2010/04/16 Enterprise Architect8.0 のリリースに伴い、内容を更新。
2010/06/28 操作としての表現についての注記を追加。
2010/09/08 アクティビティ図やシーケンス図の作成で、他のクラスのメソッドを呼び出す方法について説明。クラス図の操作として表現し、アクティビティ図やシーケンス図からのコード生成も行う方法について追記。全体を整形。
2010/09/13 アクティビティ図についての補足を追加。
2010/10/22 シーケンス図について、設計のためのシーケンス図とソースコード生成のためのシーケンス図は記述範囲が異なることを追記。
2010/02/23 C 言語についての補足を追加。その他、内容の微修正。
2011/05/18 Enterprise Architect9.0 のリリースに伴い、内容を更新。
2011/12/21 Enterprise Architect9.2 のリリースに伴い、内容を更新。過去のバージョンの制限になっていた項目が解消したため、一部の内容を削除。
2012/12/14 Enterprise Architect10.0 のリリースに伴い、内容を更新。
2014/04/22 Enterprise Architect11.0 のリリースに伴い、内容を更新。
2014/10/15 4.4 章に補足事項を追記。
2015/07/02 アクティビティ図で for 文を生成するための方法について追記。
2015/12/01 Enterprise Architect12.1 のリリースに伴い、内容を更新。振る舞い呼び出しアクションの利用方法について追記。
2016/06/02 デシジョン要素を利用する際の条件について、3.2 章および 3.3 章の内容を変更。
2016/10/07 Enterprise Architect13.0 のリリースに伴い、内容を更新。
2018/05/16 Enterprise Architect14.0 のリリースに伴い、内容を更新。
2019/08/22 Enterprise Architect15.0 のリリースに伴い、内容を更新。
2022/03/10 Enterprise Architect16.0 の情報を追加。この機能を利用することのデメリットや使いどころを追記。