学習目標

今回の授業では，設計レベルのクラス図を描けるようにするために，ある程度複雑なクラス図の表記方法を理解します．また，シーケンス図とは何かを理解し，シーケンス図を描けるようになります．具体的には次のようなことができるように学習しましょう．

- クラスのステレオタイプとは何か説明できる
- 典型的なステレオタイプの意味を説明できる
- パッケージ，オブジェクト，ノートの目的と表記法を説明できる
- ある程度複雑なクラス図に対応するJavaコードおよびJavaコードに対応するクラス図を記述できる
- シーケンス図の目的を説明できる
- シーケンス図におけるライフライン，メッセージの記法と意味を説明できる
- 与えられた情報システムの説明に適するシーケンス図を，ツールを使って描くことができる

クラス図の利用場面

クラス図は様々な場面で使うことができます．

まず，モデリングの初期段階で，情報システムが対象とする問題領域にどのような概念があって，それらがどう関連するかを表すときに，利用できます．「概念レベルのクラス図」と言うことができます．問題領域を理解し，共通の用語で話ができるようにするためには，「用語集」を作ることが多いのですが，それらの関連を明らかにしておきたいときに，クラス図が有効です． 今回の授業で扱っている問題のように，対象となる問題領域が単純であれば，作成する必要はないでしょう．

次に，もう少しモデリングが進み，ユースケース図などを作成してシステムへの要件が明らかになった後に分析モデルを記述するために利用できます．これが「分析レベルのクラス図」です．たとえば，「スケジュールを管理する」といったユースケースがある場合に，「スケジュール」はどのような属性から構成されるかを考えることになります． １つのスケジュールの項目としては，年月日，開始時間，終了時間，予定の名称，予定の詳細などを記述しておく必要があるということになれば，それらを属性としたクラス「スケジュール項目」を定義するといった感じで，クラス図を記述していきます． こうした成果をもとに，データベースの設計に進むことになります．

さらに，モデリングが進み，設計段階になると，開発するシステムがどのような構造になるかを表すクラス図を記述できます．これが「設計レベルのクラス図」です． オブジェクト指向でシステムを開発する場合には，クラス図に書かれている構造が，そのままソフトウェアの構造になります．ソフトウェアの構造をクラス図で表すことで，ソースコードを読むのに比べて，ソフトウェアを全体的に捉えることができます． そのためには，設計レベルのクラス図には厳密性が求められるし，ソースコードで書かれた内容と一致していなければなりません．これらが設計レベルのクラス図の難しい点です．

以上のような利用場面のうち，概念レベルのクラス図，分析レベルのクラス図を記述するための表記は，ほぼ，第3回の授業で学習しました．今回は，設計レベルのクラス図を記述するために必要な記法についていくつか学習します．

ステレオタイプ

ステレオタイプは，**UML**の要素**(**例，クラス図などでモデル化したクラスなど**)**に，意味付けを行なって，分類するために使います．ここではクラス図でのステレオタイプの使い方を見ていきますが，ステレオタイプはクラス図だけで使われるわけではなく，UMLの他のモデルでも使われます．ステレオタイプを使うことで，同じ種類，機能を持つ要素は同じステレオタイプに分類することができます．クラス図では，クラスを種類や目的，役割などで分類するために使います．ステレオタイプは，1つの要素に複数付けることができます．

ステレオタイプは，キーワードを << >> で囲んで書きます．(<< >>をギルメットと呼びます．) クラス図では，図1のように，クラス名の上に付けます．

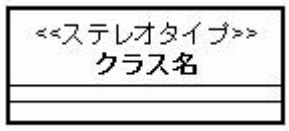

図1 ステレオタイプの書き方

たとえば，entityというステレオタイプを持つ学生クラスは，図2のようになります．

図2 ステレオタイプの例 **(entity**というステレオタイプを持つ学生クラス**)**

ステレオタイプは，UMLで定義されていますが，独自に定義して用いることもできます．
表1に，UMLで定義されているステレオタイプのうちのいくつかについて説明します．

表1 UMLで定義されているステレオタイプの例

entity	データ(情報)の保持や管理を行なう役割を持ちます．
control	ある程度まとまった処理手順を表します．entity(データ)を利用したり処理したりする手順を実装する役割を持ちます．
boundary	システムとその外側(システムとシステム，システムとユーザなど)をつなぐ役割を持ちます．
exception	例外を表します．例外処理を実装する役割を持ちます．
interface	インターフェースを表します．外部から見たとき，クラスなどがどのように振舞うかを示します．操作の宣言する役割を持ちますが，実装は他のクラスに任せます．

パッケージ

パッケージは，**UML**モデルの要素をまとめるために使います．クラス図では，クラスをまとめてグループ化するために使います．Javaプログラミングで使った，packageと同じものと考えて差し支えありません．

パッケージは，四角形の左上に，タブ(小さな四角形)を付けた図形で表します．(図3)

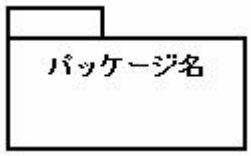

図3 パッケージの描きかた

パッケージの四角形の中に，このパッケージに属するクラスを配置します．パッケージの中にパッケージが含まれていてもかまいません．パッケージを階層化して表示する場合もあります．(図4)

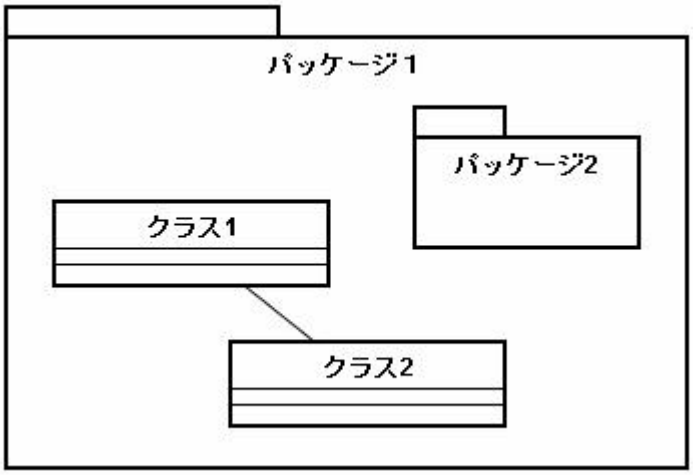

図4 パッケージの例

パッケージとパッケージの間には，「依存関係」と「汎化関係」を記述することができます．図5では，3DCGソフトパッケージは，画像描画パッケージを使っているので依存関係を持つことをあらわしています．また，図6では，ねこパッケージは動物パッケージと汎化の関係を持つことを表しています．

図5 依存関係を持つパッケージ

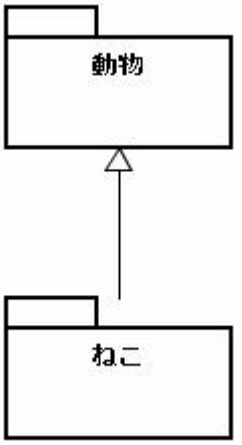

図6 汎化関係を持つパッケージ

オブジェクト

オブジェクトは，物理的・観念的な「もの」「ひと」「こと」を意味しています．オブジェクトは，属性(データ,情報)と操作(振舞い)を備えています．クラスが「もの」「ひと」「こと」を抽象的に表したものであるのに対して，一般に，オブジェクトは具体的な「もの」「ひと」「こと」を指しています．オブジェクト指向では，クラスもオブジェクトですが，UMLでは，オブジェクトと言うとほとんどの場合，インスタンスを指します．

クラスが雛形なら，オブジェクト(インスタンス)は，その型を元に生成した実体といえます．オブジェクトは，生成されて，役目が終わると消滅するまで，いくつかの状態を持っています．

UMLでオブジェクトを記述する場合，長方形を使って，クラスと同じように表記します．オブジェクト名には下線を引いて表します．「オブジェクト名」のみ，「オブジェクト名：クラス名」，「：クラス名」のみ，の3通りの表記かできます．オブジェクトの元となったクラスがパッケージに属することを明示することもできます．「オブジェクト名：クラス名：：パッケージ名」と書きます．オブジェクトに属性とその値を記入することもできます．(図7)

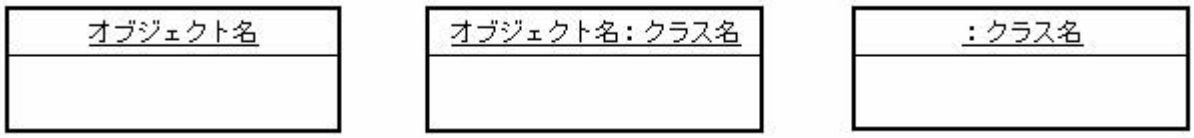

図7 オブジェクトの描きかた

ノート

ノートは，補足情報を付け加えるために使われます．注釈，制約，コメントなどを入れるために使います．

ノートは，右上を折り曲げた長方形で表し，コメントの対象となる要素から点線を引いて配置します．(図8)

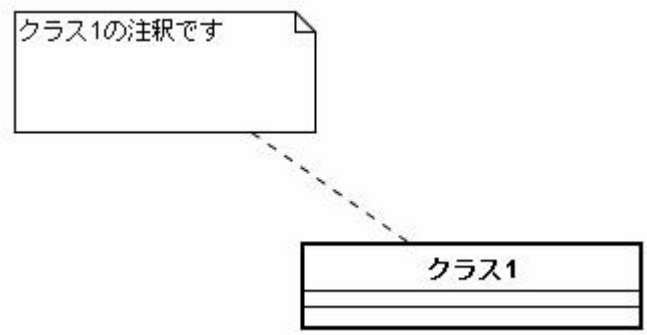

図8 ノートの例

ある程度複雑なクラス図 セルフテスト

次のページに移動して，「クラス図の応用」に関するセルフテストを受験して，理解を確認しましょう．

シーケンス図

シーケンス図は，相互作用図の一つです．相互作用図には，シーケンス図とコミュニケーション図があります．相互作用図は，「もの」や「ひと」同士のメッセージのやり取りを表現するために使われます．シーケンス図は，システムに要求される機能を実現するためのオブジェクト間の相互作用を時系列に沿って表現するために用いられます．一方，コミュニケーション図は，オブジェクト同士の関連を表現するために使います．

シーケンス図を描くことにより，オブジェクト同士がどのように相互作用するのかを，どのようなメッセージを受け取ってどのように振舞うのかを明確にすることができます．このように， クラス図がシステムの静的な側面であるのに対し，シーケンス図は，システムの動的な側面をあらわすために用いられます．

システムの具体的な動作をシナリオとして作成しますので，シーケンス図は，シナリオに沿って作成します．

シーケンス図は，主にライフラインと，メッセージ，実行指定で構成されます．(図9)

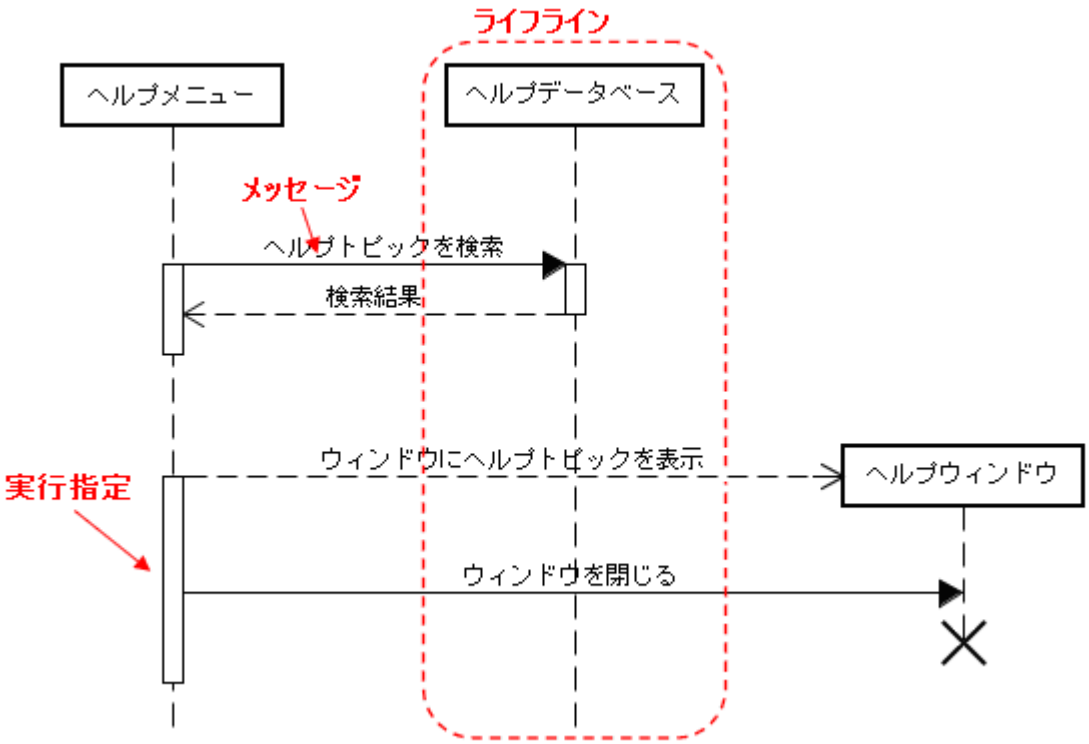

図9 シーケンス図

シーケンス図では，システムの処理の流れを，オブジェクトの間でやり取りされるメッセージのやり取りとして，上から下へ，時系列に沿って記述します．ライフラインは，相互作用に参加する要素(「もの」「ひと」など)を表現しています．

次に，シーケンス図の描きかたを，具体的に見ていきます．

ライフライン

ライフラインは，相互作用に参加する要素を表します．ライフラインは，長方形と，長方形から下に伸びる点線で表します．長方形と点線をあわせたものがライフラインです．点線のみではありませんので注意してください．点線は，ライフラインが存在している期間を表します．(図10) また，メッセージを送信するのがアクターである場合も考えられます．その場合は，アクターから点線を引いたものをライフラインとして用います．

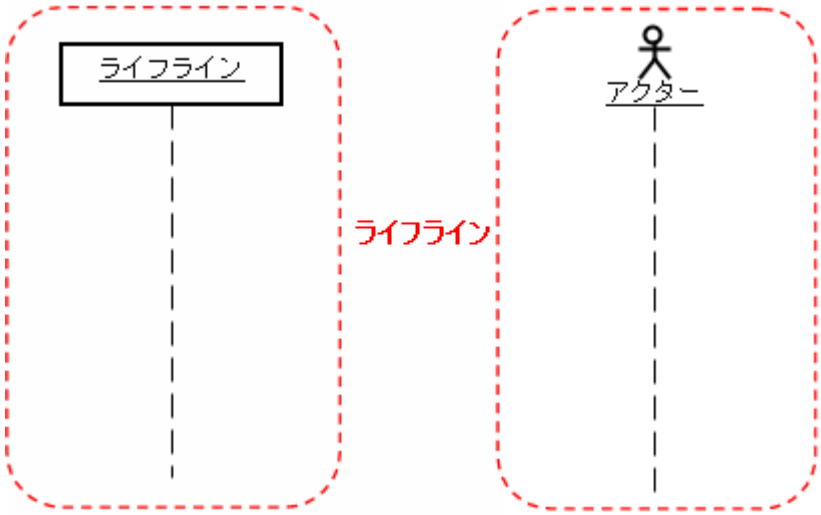

図10 ライフライン

ライフラインは，長方形の中に，相互作用の参加者である「もの」「ひと」などの名前を記述します．ライフラインに書く名前をライフライン名と呼びます．ライフライン名は，「役割名とクラス名」という書式で書きますが，役割名あるいはクラス名を省略して書くこともできます．図11のように，役割名とクラス名は「：(コロン)」で区切ります．クラス名を省略して役割名のみ書く場合は，役割名だけを書きますが，役割名を省略してクラス名のみ書く場合は，クラス名の前に：(コロン)をつけて，クラスであること示します．

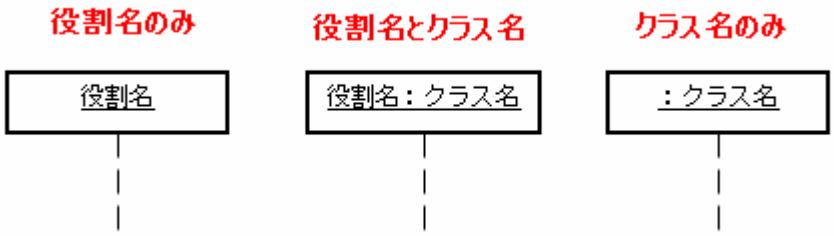

図11 ライフライン名の書き方

ライフラインの長方形あるいはアクターと点線は，ライフラインの生存期間を表しているので，あるメッセージによりライフラインが生成される場合は，ライフラインの長方形に対して矢印を引きます．(図12)

生成のメッセージにも，同期(矢印の先端が塗りつぶし)と非同期(矢印の先端が「く」の字)があり，それぞれ矢印の先端を塗りつぶす，あるいは「く」の字で描きます．また，ライフラインが消滅する場合は，ライフラインの点線の終端に×印を描きます．

ライフラインの生成と消滅は，特に明示する必要がない場合は指定しなくてもかまいません．

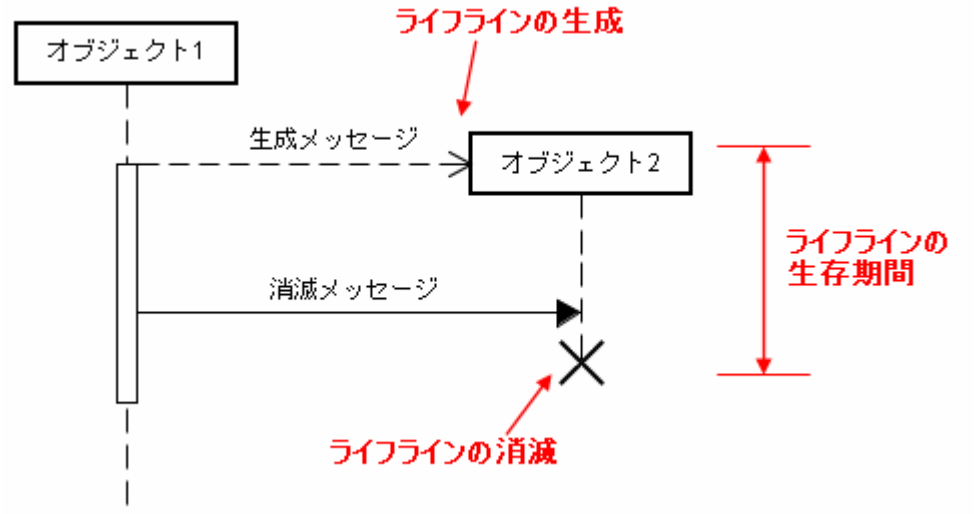

図12 ライフラインの生成と消滅

実行指定

実行指定は，メッセージの呼び出しの制御を表しています．また，メッセージの実期間も表しています．他のライフラインの実行指定の位置(始まり)や長さから，メッセージを実行する相対的な位置(始まり)や期間がわかります．

実行指定は，ライフラインの点線の上に，長方形で描きます．実行指定の長さは，メッセージの実行期間を表しています．(図13)

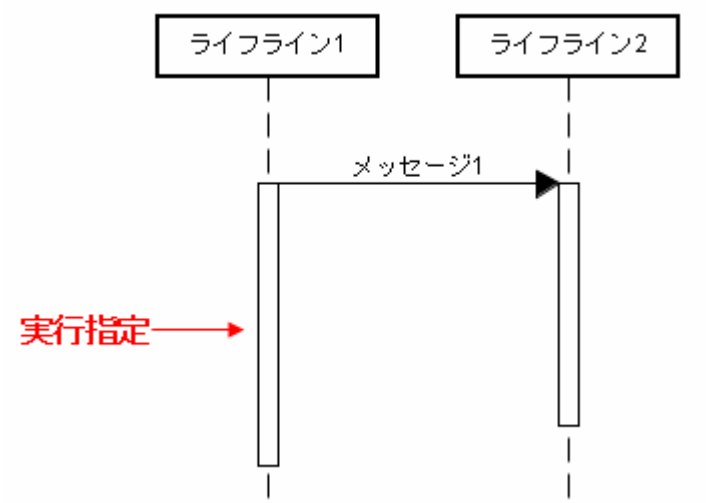

図13 実行指定

実行指定は，メッセージの呼び出しの制御を表します．図14では，メッセージ1が呼び出されることで，メッセージ2とメッセージ3が呼び出されています．すなわち，メッセージ1がメッセージ2とメッセージ3の呼び出しを制御していることを表しています．

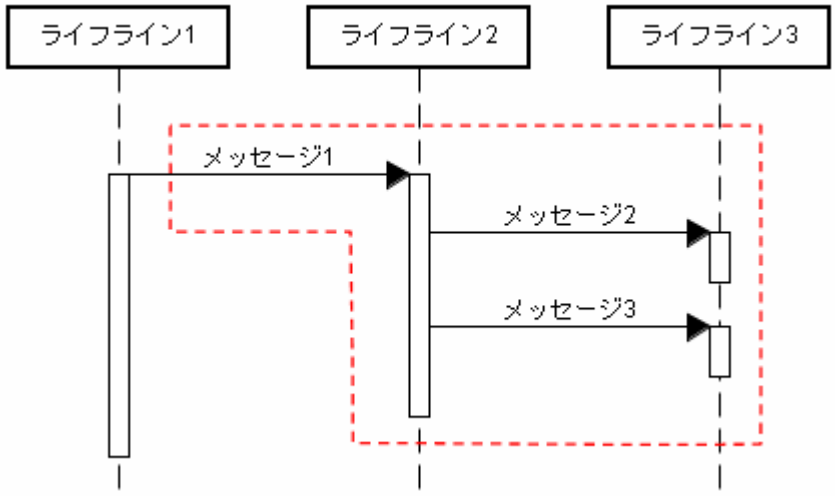

図14 実行指定によるメッセージ呼び出しの制御

メッセージ

シーケンス図で，ライフライン同士の相互作用を記述するのがメッセージです．メッセージは，ライフラインからライフラインへの矢印で表します．ライフライン間でやり取りされるメッセージには，同期メッセージ，(同期メッセージの)戻り，非同期メッセージの3種類があります．

同期メッセージ

同期メッセージは，先に呼び出されたメッセージの処理が終了して，そのメッセージに対する戻りのメッセージを受け取った後で，次のメッセージを送る，ということを意味しています．メッセージの戻りを受け取るまで，次のメッセージ送信に進みません．同期メッセージは，実線に，先端を塗りつぶした矢印で描きます．実線の上にメッセージ名を書きます．(図15)

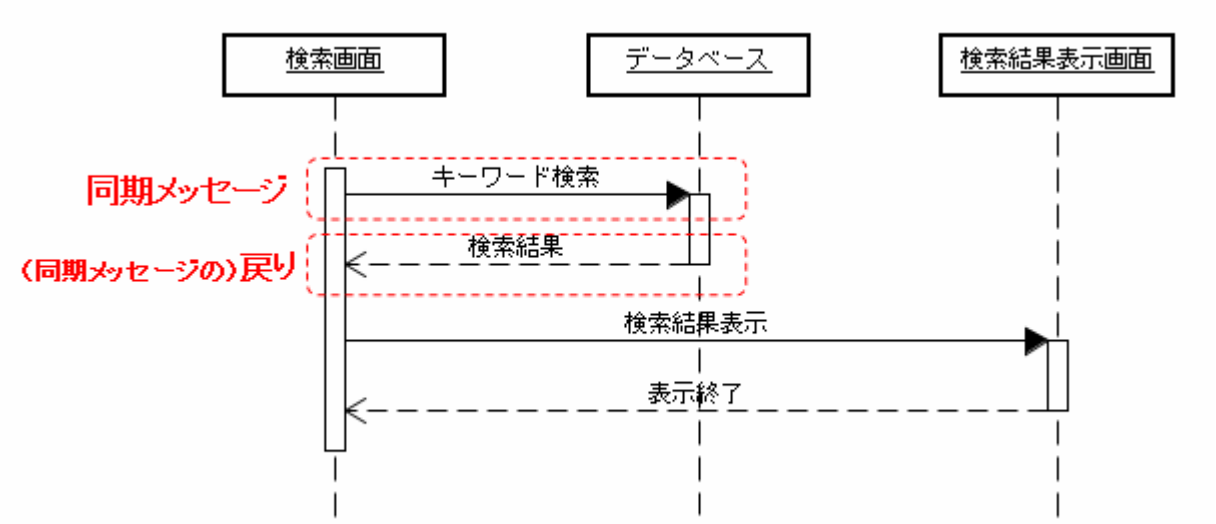

図15 同期メッセージ

同期メッセージの戻り

同期メッセージの戻りは，点線の矢印で描きます．矢印の先端は，塗りつぶさない矢印を使います．(図15)

※点線矢印は，ライフラインを生成するメッセージにも使われます．

非同期メッセージ

非同期メッセージは，先により出されたメッセージの処理の終了を待たずに，次のメッセージを送ることができる，ということを意味しています．(図16)

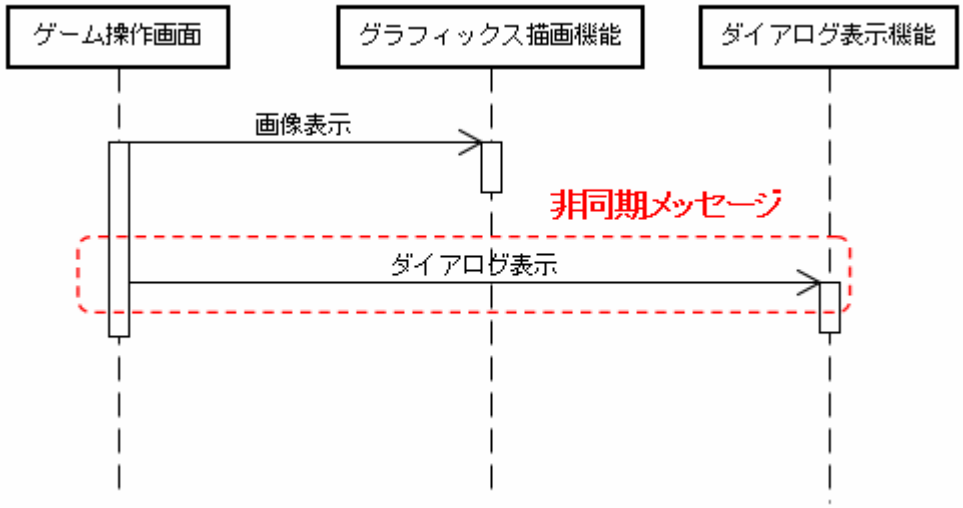

図16 非同期メッセージ

メッセージは，パラメータを持つこともできます．パラメータは，メッセージ名に”( )”をつけて書きます．パラメータが複数ある場合は，”,”(カンマ)で区切って指定します．(図17)

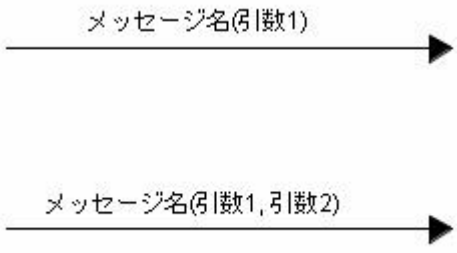

図17 パラメータを持つメッセージ

【コラム】astahでのシーケンス図の表記

astahでのシーケンス図の表記は，UML2.0で定められている表記とは，次の点が異なります．

- ライフライン名に下線が引かれる
- ライフラインの生成の矢印が実線になる
- ライフラインの生成と消滅に実行指定が付く

図18にUML2.0の表記法で描いたシーケンス図を，図19にastahで描いたシーケンス図を示します．
図19の赤丸で囲んだ部分の表記が異なります．

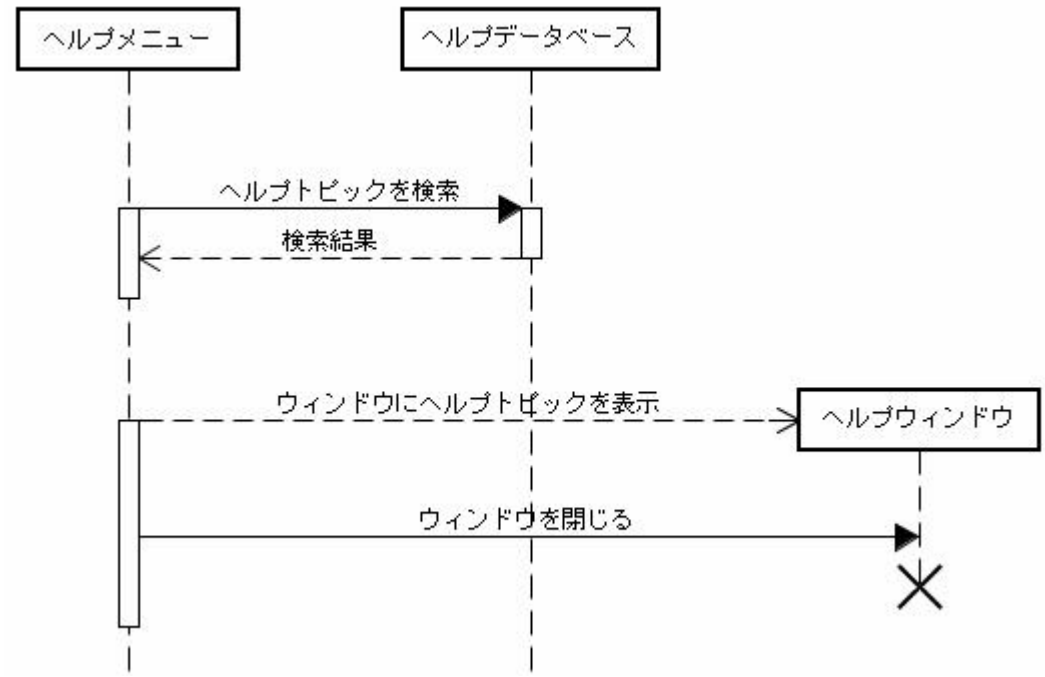

図18 UML2.0での表記

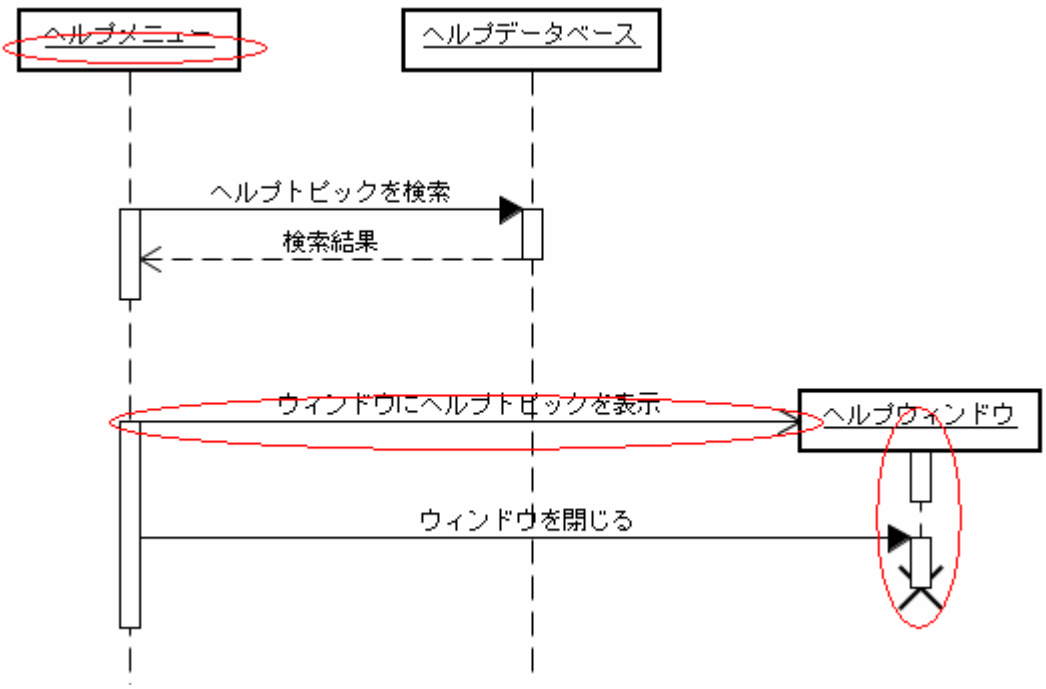

図19 astahでの表記

シーケンス図 セルフテスト

次のページに移動して，「シーケンス図」に関するセルフテストを受験して，理解を確認しましょう．

シーケンス図の基礎を確かめるセルフテストと，シーケンス図モデリングについてのセルフテストの二つがありますので，どちらとも受験してください．

まとめ

このモジュールでは，以下のことを学びました.

- クラスのステレオタイプの記法と意味
- 典型的なステレオタイプの意味
- パッケージ，オブジェクト，ノートの目的と表記法
- ある程度複雑なクラス図に対応するJavaコードおよびJavaコードに対応するクラス図の記述
- シーケンス図の目的と記法
- シーケンス図におけるライフライン，メッセージの記法と意味
- 与えられた情報システムの説明に適するシーケンス図の記述

例題実習1 astahを使ってシーケンス図を描く

次のシーケンス図と同じものを，astahで描いてみましょう．

例題1 簡単なシーケンス図

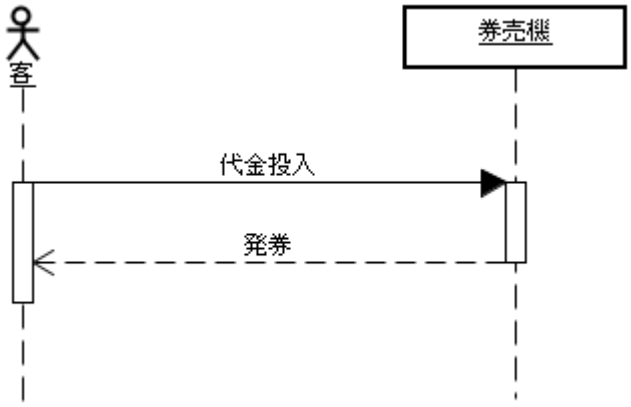

図1 簡単なシーケンス図(例題1)

例題2 少し複雑なシーケンス図

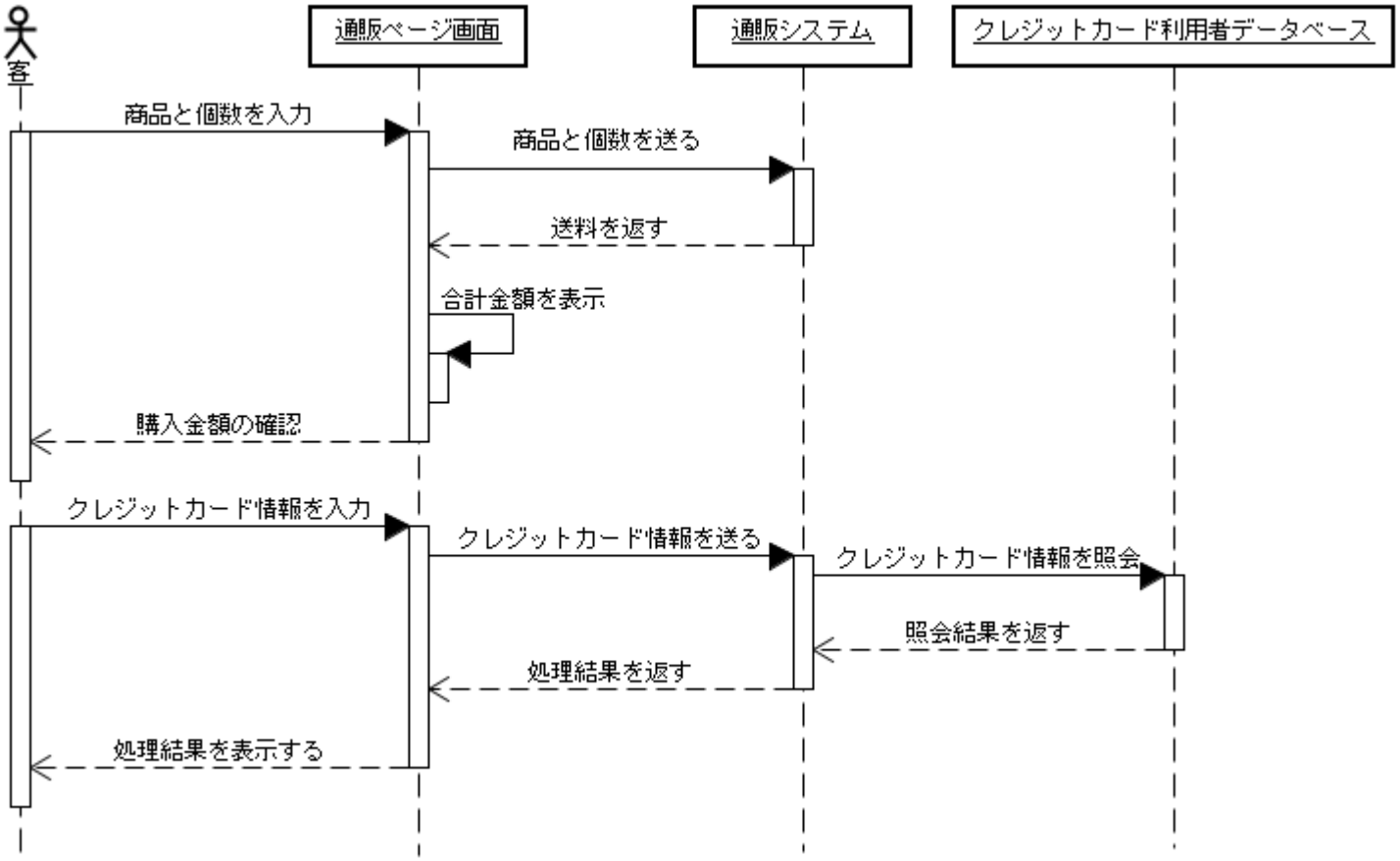

図2 少し複雑なシーケンス図(例題2)

例題実習2 記述に対するシーケンス図を描く

例題3 通帳に記帳するときのシーケンス図

以下の記述を読み，銀行の記帳端末で，通帳に記帳するときのシーケンス図を描きなさい．

「記帳したい客は，記帳端末に通帳を挿入する．記帳端末は，挿入された通帳から口座番号を読み込む．次に，その口座の利用情報を預金口座データベースに問い合わせる．預金口座データベースは，その口座の利用情報を記帳端末に返す．記帳端末は，取得した口座利用情報を通帳に記入する．記帳端末は，記入が終わったら，通帳を客に返却する．」

解説

ライフラインとして，「客」，「記帳端末」，「預金口座データベース」が登場しています．

ライフライン間でやり取りされるメッセージを抽出して番号を振ると次のようになります．まず客から記帳端末に(1)「通帳を挿入する」メッセージが送られます．記帳端末は，挿入された通帳から口座番号を読み込みますが，これは，記帳端末から記帳端末へ(2)「口座番号を読み込む」メッセージを送ることになります．

次に，記帳端末から預金口座データベースへ(3)「口座利用情報を問い合わる」メッセージが送られます．預金口座データベースは，記帳端末へ(4)「口座利用情報を返す」メッセージを送ります．記帳端末は，口座利用情報を通帳に記入しますが，このとき，記帳端末から記帳端末へ，(5)「口座利用情報を記入する」メッセージが送られます．記入が終わったら，記帳端末は，客に(6)「通帳を返却する」メッセージを送ります．

メッセージの流れとして考えると，(1)の「通帳を挿入する」メッセージを受け取ってから(2)の「口座番号を読み込む」メッセージが送られるので，「通帳を挿入する」は同期メッセージとなります．同様に「口座番号を読み込む」メッセージが送られてから，(3)の「口座情報を問い合わせる」メッセージが送られるので，「口座情報を読み込む」メッセージも同期メッセージとなります．同様に，(5)の「口座利用情報を記入する」も同期メッセージとなります．(4)の「口座利用情報を返す」は(3)の「口座利用情報を問い合わせる」の戻りとなり，(6)の「通帳を返却する」は(1)の「通帳を挿入する」の戻りとなります．

このメッセージの流れをシーケンス図で描くと，図3のようになります．

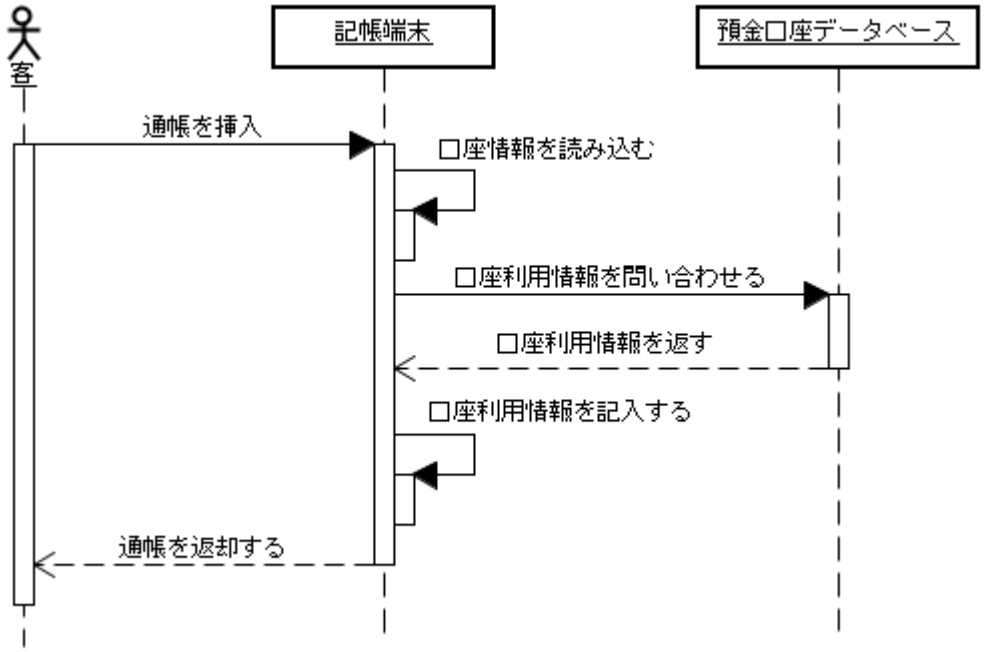

図3 記帳端末で記帳をする際のシーケンス図

シーケンス図を描く際のポイント

(1)シナリオあるいはユースケース記述を作成する
相互作用図であるシーケンス図は，シナリオあるいはユースケース記述として書く内容が明確になっている必要があります．シーケンス図を描く前にこれらのどちらかは作成しておくようにしましょう．

(2)必要に応じて複数のシーケンス図を使って表現する
シーケンス図はあまり複雑になりすぎると読みにくくなります．シーケンス図は必要に応じて複数用いてかまいませんので，1つのシナリオは1枚のシーケンス図にまとめるようにするといいでしょう．また，シナリオが長くなる場合は，シーケンス図を分割して，1枚のシーケンス図が複雑になり過ぎないようにしましょう．

※複数のシナリオを1枚のシーケンス図に収めるためには，分岐という処理をもちい，シーケンス図を分割するためには相互作用使用をもちいますが，分岐や相互作用使用についてはここでは説明していません．興味があったら調べてみましょう．

(3)メッセージの名前はメッセージを受け取る側の視点でつける
シーケンス図のメッセージのやり取りは，Javaなどのプログラムでは，クラスやオブジェクト間でのメソッドの呼び出しとして実装されます．このとき，メソッドを呼び出すのがメッセージを送る側で，メソッドをメンバとして持つはメッセージを受け取る側となります．このため，特に，具体的なプログラムの設計のためのシーケンス図を描く場合は，メッセージを受け取る側の視点でメッセージ名をつけておくと，メッセージ名と同じ意味のメソッド名としてプログラムで実装することができます．