13 章 オブジェクト指向機能

この章では、COBOL2002 のオブジェクト指向機能について説明します。
COBOL2002 では、COBOL にとってまったく新しい概念であるオブジェクト指向機能を、従来の機能と完全な互換を保ちながら取り込んでいます。そのため、既存資産を活かしたオブジェクト指向開発が可能です。その一方で、オブジェクト指向プログラミングに必要な機能を一通りサポートしており、他のオブジェクト指向言語と比較しても遜色のないものになっています。

13.1 クラスとオブジェクト

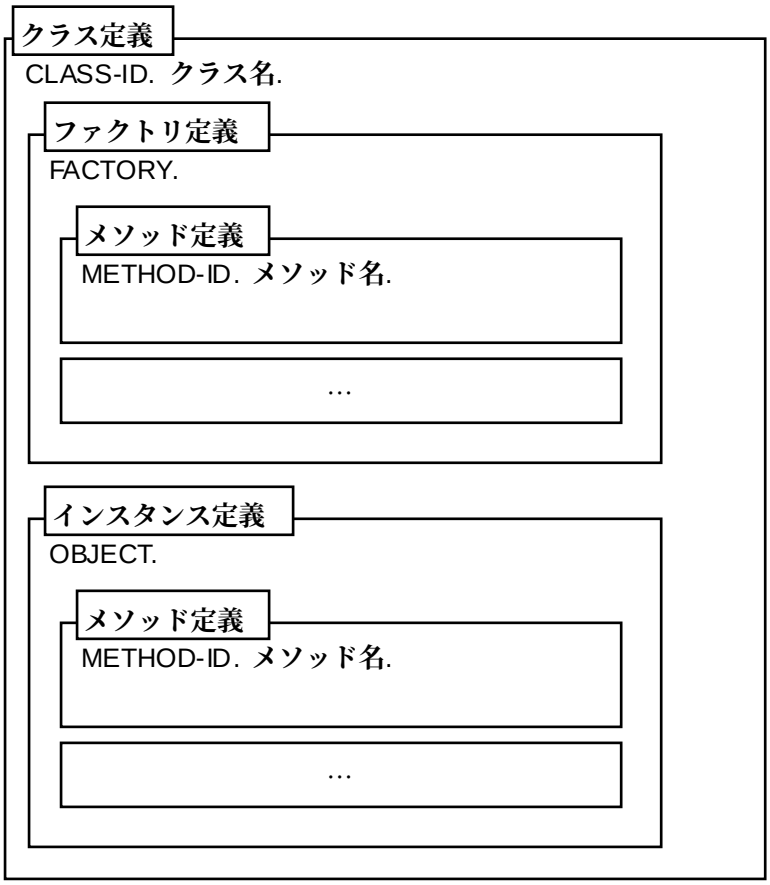
オブジェクト指向プログラミングで扱うオブジェクトは、現実世界の“ もの ”あるいは“ 概念 ” を計算機上でモデル化したものです。オブジェクト指向プログラミングは、システムをオブジェクトの集まりと考え、それらを相互に連携させながらシステムを開発する方法です。
オブジェクトは、それ自身の内部にデータや手続きのコードを持っていますが、それらは外部からは一切見えません。外部から見えるのは、公開された手続き（メソッド）のインタフェースだけです。つまり、オブジェクト内部の実現方法は外部から完全に隠蔽されており、公開されたメソッドを通じてだけアクセスできます。
オブジェクトの共通の性質（データレイアウトと手続き）はクラスにより定義されます。あるクラスに属するオブジェクト、すべて同じデータレイアウトと手続きを持ちます。
たとえば、銀行業務アプリケーションでは、たくさんの預金口座オブジェクトを扱います。
それぞれの預金口座オブジェクトは様々なデータ（口座番号、名義人、残高など）を持ち、外部からアクセスするための手続き（預け入れ、払戻し、残高照会など）を持っています。
すべての預金口座オブジェクトは、同じデータレイアウトと手続きを持っており、それらを定義するのが預金口座クラスです。

13.1.1 クラス

クラスは、各々のオブジェクトが持っているデータのレイアウト及び、手続きであるメソッドを定義したものです。同じクラスに所属するオブジェクトは、すべて同じデータレイアウトを持ち、同じメソッドを持っています。
COBOL のクラス定義は、次の二つの部分から構成されています。

・インスタンス定義
・ファクトリ定義

インスタンス定義では、そのクラスに属するインスタンスオブジェクトを定義し、ファクトリ定義は各クラスに一つだけ存在するファクトリオブジェクトを定義します。これらのオブジェクトの違いについては、“13.1.2 インスタンスオブジェクト”と“13.1.3 ファクトリオブジェクト”で説明します。
COBOL のクラス定義は、次のような構造になっています。クラス定義はファクトリ定義とインスタンス定義を含み、それぞれの定義はメソッド定義を含んでいます。



13.1.2 インスタンスオブジェクト

クラスに属するオブジェクトをインスタンスオブジェクトまたは単にインスタンスと呼びます。
同じクラスに属するインスタンスはインスタンス定義で定義されたデータ定義を共有します。しかし、それぞれのインスタンスはぞれぞれ固有のデータ値を持っています。また、同じクラスに属するインスタンスはメソッド定義も共有します。
インスタンスオブジェクトは、そのクラスのファクトリオブジェクトによって生成されます。インスタンス生成時には、インスタンス定義中のデータ記述項にしたがってオブジェクトの領域を割りつけます。インスタンスのデータ領域は、そのクラスのインスタンスメソッドからしかアクセスできません。そのため、インスタンスデータを外部から完全に隠すことができます。
たとえば、銀行業務アプリケーションでは、それぞれの預金口座の情報は預金口座クラスのインスタンスである預金口座インスタンスが持っています。また、それぞれの預金口座インスタンスは、名義人、残高などの口座に関する情報をもっています。これらの領域は、預金口座クラスのインスタンス定義で定義されています。
次の例では、預金口座インスタンスは、残高と名義人の二つのインスタンスデータを持ち、預け入れ、引出し、残高照会の三つのインスタンスメソッドを持っています。

```
CLASS-ID. 預金口座 INHERITS BASE.
…
FACTORY.
…
END FACTORY.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 残高          PIC S9(16)V9（２）.
01 名義人        PIC N(40).
PROCEDURE DIVISION.
METHOD-ID. 預け入れ.
…
END METHOD 預け入れ.
METHOD-ID. 引出し.
…
END METHOD 引出し.
METHOD-ID. 残高照会.
…
END METHOD 残高照会.
END OBJECT.
END CLASS 預金口座.
```

13.1.3 ファクトリオブジェクト

各クラスは、ファクトリオブジェクトと呼ばれる特殊なオブジェクトを一つだけ持っています。ファクトリオブジェクトは、オブジェクトインスタンスを生成したり、クラス共通の情報を管理したりするために使います。
ファクトリオブジェクトは、インスタンスオブジェクトと同様に、データ（ファクトリデータ）とメソッド(ファクトリメソッド）を持ちます。ファクトリデータは、クラス内で共通なデータを格納するのに使い、ファクトリメソッドはオブジェクトインスタンスの生成などに使います。
たとえば、預金口座クラスは、預金口座ファクトリオブジェクトと呼ばれるファクトリオブジェクトを持っています。新しい預金口座インスタンスを作るには、預金口座ファクトリオブジェクトのファクトリメソッドを使います。
次の例では、預金口座ファクトリオブジェクトは、新しい預金口座インスタンスを生成するための新規口座メソッドを持っています。

```
CLASS-ID. 預金口座 INHERITS BASE.
…
FACTORY.
PROCEDURE DIVISION.
METHOD-ID. 新規口座.
DATA DIVISION.
LINKAGE SECTION.
01 作成口座      USAGE OBJECT REFERENCE ACTIVE-CLASS.
PROCEDURE DIVISION   RETURNING 作成口座.
…
END METHOD 新規口座.
END FACTORY.
OBJECT.
…
END OBJECT.
END CLASS 預金口座.
```

13.1.4 定義済みオブジェクト

COBOL では、次のオブジェクトが予め用意されており、オブジェクトを使用できる場所で使うことができます。

- EXCEPTION-OBJECT： 例外オブジェクト
- NULL： NULL オブジェクト（何も指していない状態を表す）
- SELF： メソッドを実行中のオブジェクト
- SUPER：メソッドの探索をスーパークラスから始める（メソッド呼出し時に使用）

13.2 継承と多態

オブジェクト指向の特徴の一つとして、アプリケーションの一部を簡単に再利用できることが挙げられます。継承は、アプリケーションの再利用を行うための機構です。
多態とは、“一つのものが複数の形態をとれる”ことです。オブジェクト指向プログラミングでは、多態を継承の機構を使って実現しています。

13.2.1 適合

二つのクラス A, B があり、クラス A のオブジェクトがクラス B のオブジェクトとして振る舞える場合を、“A は B に適合する”と言います。
クラス A のオブジェクトがクラス B のオブジェクトとして振る舞えるとは、クラス A のオブジェクトが、クラス B のオブジェクトが受け入れることができる要求をすべて受け入れられるということです。具体的には、クラス A のオブジェクト上で、クラス B のオブジェクト上で呼び出せるメソッドと同名かつ同じシグニチャのメソッドをすべて呼び出せるということです。
A が B に適合していれば、実際のオブジェクトがクラス A のものであっても、クラス B のオブジェクトだと思って使うことができます。クラス A のオブジェクトに対してクラス B のインタフェースでメソッドを呼び出しても、A のオブジェクトは必ず同名・同シグニチャのメソッドを実行できるからです。

13.2.2 継承

継承の機構により、既存のクラスをベースに、新しいデータを追加し、新しいメソッドを追加したり、既存のメソッドを置き換えたりすることにより、新しいクラスを定義することができます。つまり、既存のクラスとの差分をコーディングするだけで、新しいクラスを定義することができます。このとき、既存のクラスをスーパークラス、新しいクラスをサブクラスと呼びます。
スーパークラスで定義されたデータは、サブクラスにも引き継がれます。ただし、スーパークラスで定義したデータには、サブクラスのメソッドからはアクセスできません。また、スーパークラスのすべてのメソッドは、サブクラスのメソッドとしても使うことができます。
クラスを継承した場合、サブクラスは、スーパークラスに必ず適合しています。したがって、サブクラスのオブジェクトは、必ずスーパークラスのオブジェクトとして振舞うことができます。適合の要件を満たすために、継承したクラスはスーパークラスで定義されたメソッドのインタフェースを削除・変更することはできません。

継承により、クラスを階層化することができます。あるクラスのオブジェクトを参照するために宣言されたデータ項目は、そのクラスを継承したすべてのクラスのオブジェクトを参照できます。

継承の例として、銀行業務アプリケーションについて考えてみます。銀行業務で扱う預金口座は一種類だけではありません。例えば、普通預金、定期預金、当座預金などがあります。しかしそれらはすべて“預金口座”であることに変わりはありません。たとえば、当座預金と普通預金を比べる、それらは多くの共通点（たとえば、名義人と口座残高の情報を持つ）といくつかの相違点（たとえば、当座預金は小切手を発行できるが利息はつかない）があります。
このような場合、ベースとなる預金口座クラスを作り、当座預金と普通預金を定義するために継承を使います。預金口座クラスはすべての預金口座に共通な部分を定義し、当座預金クラスには当座預金に固有の部分を、普通預金クラスには普通預金に固有の部分を定義します。
このとき、当座預金クラスのコードは次のようになります。当座預金クラスは、預金口座クラスを継承しています。そのため、当座預金クラスのオブジェクトは、預金口座クラスで定義されたインスタンスデータ（残高、名義人）とインスタンスメソッド（預け入れ、引出し、残高照会）も持っています。また、それ以外に当座預金クラスで新しく定義されたインスタンスメソッド“小切手発行”を持っています。

```
CLASS-ID. 当座預金 INHERITS 預金口座.
…
FACTORY.

END FACTORY.
OBJECT.
DATA DIVISION.
…
PROCEDURE DIVISION.
METHOD-ID. 小切手発行.
…
END METHOD 小切手発行.
END OBJECT.
END CLASS 預金口座.
```

13.2.3 多態

多態とは、“一つのものが複数の形態をとれる”ことです。
銀行業務アプリケーションの例では、普通預金クラスも当座預金クラスもどちらも預金口座クラスのサブクラスです。そのため、どちらも預金口座クラスの性質を引き継いでおり、預金口座オブジェクトとしても振舞うことができます。逆に言えば、プログラム上は預金口座オブジェクトと書かれていても、実際のオブジェクトは普通預金オブジェクトであったり、当座預金オブジェクトであったりするわけです。
たとえば、預金口座クラスが解約処理メソッドを持っている場合、それを継承した普通預金クラスも当座預金クラスも、同じインタフェースの解約処理メソッドを持っています。このような場合、プログラム上で“預金口座オブジェクト対し、解約処理を呼び出す”とさえ記述しておけば、実際のオブジェクトが普通預金オブジェクトであれば普通預金用の解約処理を、当座預金オブジェクトであれば当座預金用の解約処理を呼び出すことができます。

13.2.4 インタフェース

それぞれのクラスは二つのインタフェースを持っています。一つはインスタンスオブジェクトのインタフェースであり、すべてのインスタンスメソッド（継承されたメソッドも含む）から構成されます。もう一つはファクトリオブジェクトのインタフェースであり、すべてのファクトリメソッド（継承されたメソッドも含む）から構成されます。
オブジェクトは、それぞれのクラスで定義したインタフェースを持っています。インスタンスオブジェクトはインスタンス定義で定義したインタフェース持ち、ファクトリオブジェクトはファクトリ定義で定義したインタフェースを持ちます。
COBOL では、クラスとは独立にインタフェースだけを定義することができます。 オブジェクトを参照するデータ項目を定義する際にインタフェースを指定すると、そのデータ項目からはそのインタフェースを実装するオブジェクトなら、クラス階層に関係なく参照することができます。このようなデータ項目では、代入の可否やメソッド呼出し時のパラメタの妥当性はすべてインタフェースを実装しているか否かによりチェックされます。
たとえば、銀行業務アプリケーションは、 オブジェクトの情報を印刷するクラスがたくさんあります。たとえば、預金口座オブジェクトは名義人と残高を印刷し、顧客オブジェクトは顧客の名前と住所を印刷します。両者に共通の印刷ルーチンを作成する場合、印刷関連のメソッドを含むインタフェースを定義し、そのインタフェースを通して両者の印刷関連メソッドを呼び出します。両者の間に継承関係はありません、印刷のための同じインタフェースを実装しています。これにより、このインタフェースを実装するオブジェクトは、すべてこのルーチンで印刷可能になります。

```
*> 印刷処理インタフェースの定義
INTERFACE-ID. 印刷処理.
…
PROCEDURE DIVISION.
METHOD-ID. 印刷.
…
END METHOD 印刷.
END INTERFACE 印刷処理.

*> 預金口座クラスの定義
*> 預金口座インスタンスは、印刷処理インタフェースを実装する。
CLASS-ID. 預金口座 INHERITS BASE.
…
OBJECT IMPLEMENTS 印刷処理.
…
PROCEDURE DIVISION.
METHOD-ID. 印刷.
…
END METHOD 印刷.
END OBJECT.
END CLASS 預金口座.

*> 顧客クラスの定義
*> 顧客インスタンスも、印刷処理インタフェースを実装する。
CLASS-ID. 顧客 INHERITS BASE.
…
OBJECT IMPLEMENTS 印刷処理.
…
PROCEDURE DIVISION.
METHOD-ID. 印刷.
…
END METHOD 印刷.
END OBJECT.
END CLASS 顧客.

*> 上記のクラス/インタフェースを使うプログラム
PROGRAM-ID. 印刷ルーチン.
…
DATA DIVISION.
WORKING-STORAGE SECTION.
01 口座          USAGE OBJECT REFERENCE 預金口座.
01 名義人        USAGE OBJECT REFERENCE 顧客.
01 印刷オブジェクト      USAGE OBJECT REFERENCE 印刷処理.
…
PROCEDURE DIVISION.
```

```
...
    INVOKE 預金口座 "新規口座" RETURNING 口座.
    INVOKE 顧客 "New" RETURNING 顧客情報.
    ...
    SET 印刷オブジェクト TO 口座.
    INVOKE 印刷オブジェクト "印刷".
    ...
    SET 印刷オブジェクト TO 顧客情報.
    INVOKE 顧客情報 "印刷".
    ...
END PROGRAM 印刷ルーチン.
```

13.3 オブジェクト参照

それぞれのオブジェクトには、実行時システムが割り当てた参照値を持っています。参照値はオブジェクトへのポインタと考えることができ、オブジェクトを特定する場合には必ずこの値を使います。オブジェクトの参照値は、オブジェクト参照と呼ばれるデータ項目に格納します。

13.3.1 オブジェクト参照の定義

オブジェクト参照は、USAGE 句に OBJECT　REFERENCE を指定して定義します。

預金口座クラスまたはそのサブクラスのオブジェクトを参照できるデータ項目

```
01　オブジェクト　　USAGE OBJECT REFERENCE 預金口座.
```

預金口座クラスのオブジェクトしか参照できないデータ項目

```
01　オブジェクト　　USAGE OBJECT REFERENCE 預金口座 ONLY.
```

印刷処理インタフェースを実装するオブジェクトを参照できるデータ項目

```
01　オブジェクト　　USAGE OBJECT REFERENCE 印刷処理.
```

メソッドを実行しているオブジェクトのクラスのインスタンスオブジェクトを参照するデータ項目

```
01　オブジェクト　　USAGE OBJECT REFERENCE ACTIVE-CLASS
```

また、上記のほかに、参照できるオブジェクトに一切制約を設けない普遍オブジェクト参照を使用することができます。USAGE 句の OBJECT　REFERENCE 指定で何も指定しなければ、普遍オブジェクト参照になります。

どのようなオブジェクトでも参照できるデータ項目

```
01　オブジェクト　　USAGE OBJECT REFERENCE.
```

13.3.2 適合チェック

オブジェクト参照は、データ記述項での指定を満たすオブジェクトしか参照できません。この規則は代入とパラメタの受渡しに適用されます。
たとえば、銀行業務アプリケーションでは、次の二つのオブジェクト参照を定義します。

```
01 口座－１　　USAGE OBJECT REFERENCE 預金口座.
01 口座－２　　USAGE OBJECT REFERENCE 普通預金.
```

口座－１は、預金口座クラスとそのサブクラスのインスタンスを参照できます。たとえば、

普通預金クラスは預金口座クラスのサブクラスなので、口座－１は普通預金インスタンスを参照できる。逆に、口座－２は普通預金インスタンスは参照できますが、その他のクラスのインスタンスは参照できません。口座－２に預金口座インスタンスを代入しようとすると、コンパイラがエラーを検出します。

13.3.3 手続き部での扱い

オブジェクト参照の代入には、MOVE 文ではなく SET 文を使います。このとき、送り側項目のクラスは受取り側項目のクラスと同じかまたはサブクラスでなければなりません。また、比較は = と NOT = しか使用できません。比較は、オブジェクト参照の値しか比較しません。したがって、同じオブジェクトを指していれば等しいとみなされます。

13.3.4 オブジェクトビュー

オブジェクト参照がスーパークラスのインスタンスを参照するように定義されていても、実行時にサブクラスのインスタンスを参照することがあります。たとえば、オブジェクト参照が預金口座インスタンスを参照するように宣言されていても、実行に普通預金インスタンスへの参照が格納されることがあります。このような場合には、普通預金インスタンスを参照するオブジェクト参照に代入できる必要があります。
このような場合、COBOL ではオブジェクトビューを使います。オブジェクトビューを使うことにより、預金口座インスタンスを参照するオブジェクト参照を、普通預金インスタンスを参照するオブジェクト参照とみなすことができます。
たとえば、

```
01  口座－１      USAGE IS OBJECT REFERENCE 預金口座.
02  口座－２      USAGE OBJECT REFERENCE 普通預金.
```

と宣言した場合、

```
SET 口座－１ TO 口座－２
```

は書けますが、

```
SET 口座－２ TO 口座－１
```

はエラーになります。普通預金クラスは預金口座クラスのサブクラスですが、預金口座クラスは普通預金クラスのサブクラスではないからです。

しかし、口座－１に普通預金インスタンスへの参照が入っていることもあり、口座－２への代入が必要になることがあります。このような場合、オブジェクトビューを使って、次のように書きます。

```
SET 口座－２ TO 口座－１ AS 普通預金
```

このように書くと、口座－１が参照するオブジェクトが普通預金クラスのオブジェクトとみなされるので、代入が可能になります。ただし、口座－１が普通預金クラスおよびそのサブクラスの以外のオブジェクトを参照していた場合は実行時エラーになります。

13.4 メソッド

メソッドは、そのクラスのオブジェクト上で実行される手続きコードです。オブジェクトの内部には、メソッドを呼び出すことによってアクセスできます。
たとえば、預金口座クラスは、預け入れ、引出し、残高照会などの情報をもっています。アプリケーションが預金口座インスタンスの残高を求めたい場合には、残高照会メソッドを呼び出します。

```
CLASS-ID. 預金口座 INHERITS BASE.
…
FACTORY.
…
END FACTORY.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  残高          PIC S9(18).
…
PROCEDURE DIVISION.
METHOD-ID. 預け入れ.
…
END METHOD 預け入れ.
METHOD-ID. 引出し.
…
END METHOD 引出し.
METHOD-ID. 残高照会.
DATA DIVISION.
LINKAGE SECTION.
01  現在残高      PIC S9(18).
PROCEDURE DIVISION  RETURNING 現在残高.
    MOVE 残高 to 現在残高.
    EXIT METHOD.
END METHOD 残高照会
…
END OBJECT.
END CLASS 預金口座.

*> 残高照会メソッドを呼び出すプログラム
PROGRAM-ID. 残高チェック.
…
DATA DIVISION.
WORKING-STORAGE SECTION.
…
01  口座          USAGE OBJECT REFERENCE 預金口座.
```

```
01  現在残高       PIC S9(18).
…
PROCEDURE DIVISION.
    …
    INVOKE 口座 "残高照会"      RETURNING 現在残高.
    …
END PROGRAM 残高チェック.
```

13.4.1 メソッドの一意性

メソッドは、名前だけで区別されます。そのため、同じクラスには同じ名前のメソッドを複数定義することはできません。継承したクラスが継承されたクラスの中のメソッドと同じ名前のメソッドを定義した場合、継承したクラスのメソッドで上書きされます。

13.4.2 メソッド呼出し

メソッドは、オブジェクトとメソッド名を指定することにより呼び出します。メソッドはそのオブジェクトのクラスで定義されているかも知れないし、直接又は間接的に継承したクスラで定義されているかも知れません。
メソッド呼出しには、行内呼出しと、 INVOKE 文を使う方法があります。行内呼出しは一意名の書けるところに指定することができ、実行時にメソッドの復帰値に置き換えられます。
たとえば、預金口座からある金額をひき出す処理は、次のように書きます。

```
INVOKE 口座 "引出し" USING 金額
```

また、残高紹介は次のように書きます。

```
INVOKE 口座 "残高照会" RETURNING 金額
```

さらに、行内呼出しでは次のようになります。

```
MOVE 口座:: "残高照会"  TO 金額
```

また、パラメタを伴う行内呼出しは、次のように書きます。

```
MOVE 口座:: "残高照会－２" (パラメタ)  TO 金額
```

13.5 オブジェクトプロパティ

COBOL では、オブジェクトのデータは外部から完全に隠蔽されています。オブジェクトデータにアクセスするためには、そのためのメソッドを呼び出さなければなりません。
しかし、オブジェクトのデータを参照するのに一々メソッド呼出しの文を書いたりメソッドを定義したりするのは面倒です。そのため、特殊な構文のメソッド（プロパティメソッド）を定義することにより、通常の OF による修飾と同じ構文でメソッドの呼出しを可能にしています。また、データ記述項に PROPERTY 句を書くことにより、プロパティメソッドを自動的に生成することもできます。

13.5.1 オブジェクトプロパティの設定/参照

一意名の書ける所に "プロパティ名 OF 一意名" と書くと、次のメソッド呼出しと同じになります。

[受取側の場合]

```
送り側項目 を temp-data へ代入
INVOKE 一意名 "プロパティ設定メソッド" USING temp-data.
```

[送り側項目の場合]

```
INVOKE 一意名 "プロパティ参照メソッド" RETURNING temp-data
temp-data を 受取側項目 へ代入
```

13.5.2 PROPERTY 句

ファクトリ定義およびインスタンス定義の作業場所節のデータ記述項に PROPERTY 句を指定することにより、そのデータ項目に値を設定/参照するプロパティメソッドを自動的に生成することができます。
たとえば、

```
01  残高 PIC S9(18) PROPERTY.
```

と書くと、次の二つのプロパティメソッドが生成されます。

```
METHOD-ID. GET PROPERTY 残高.
LINKAGE SECTION.
01  現在残高       PIC S9(18).
PROCEDURE DIVISION RETURNING 現在残高.
    MOVE 残高 TO 現在残高
    EXIT METHOD.
END METHOD.
```

```
METHOD-ID. SET PROPERTY 残高.
LINKAGE SECTION.
01  現在残高       PIC S9(18).
PROCEDURE DIVISION USING 現在残高.
    MOVE 現在残高 TO 残高
    EXIT METHOD.
END METHOD.
```

13.6 オブジェクトの寿命

インスタンスオブジェクトは、実行時に割り当てられ、オブジェクト参照を通して参照されます。インスタンスオブジェクトは、ファクトリオブジェクトのメソッドを呼び出す事によって生成されます。インスタンスオブジェクトは、それ以上アクセスされなくなると、必要に応じて自動的に削除されます。

13.7 リポジトリ

クラスを使用するプログラムを翻訳する場合、適合チェックなどのために、プログラムで使用しているクラスの情報が必要になります。COBOL では、これらの情報をやり取りするために、外部リポジトリを使用します。

13.7.1 クラス/インタフェースの情報

COBOL コンパイラは、クラスのソースを翻訳する際に、クラス参照時に必要となる情報を外部リポジトリに格納します。そして、クラスを参照しているソースを翻訳する際に、リポジトリからクラス情報を取り出し、使用します。
さらに、COBOL では、クラスと同じようにインタフェースも独立して定義することができます。そのため、外部リポジトリにはクラスの情報だけでなく、インタフェースの情報も格納されます。

13.7.2 REPOSITORY 段落

翻訳時に外部リポジトリからクラス・インタフェース情報を取り出すために、環境部の構成節の REPOSITORY 段落に、ソース中で参照するクラス・インタフェースをすべて宣言します。
たとえば、BASE クラス、預金口座クラスと印刷処理インタフェースを参照している場合、次のように宣言します。

```
REPOSITORY.
    CLASS BASE
    CLASS 預金口座
    INTERFACE 印刷処理.
```

13.8 その他のオブジェクト指向機能

13.8.1 パラメタ化クラス

パラメタ化クラスは、あらかじめクラスの骨組みを用意しておき、利用時にパラメタを渡すことによって目的のクラスを作成するための機能です。オブジェクトの集合を扱うコレクションクラスを定義する際に､格納するオブジェクトのクラスをパラメタにしておけば、様々なクラスのコレクションを作成することができます。

13.8.2 例外オブジェクト

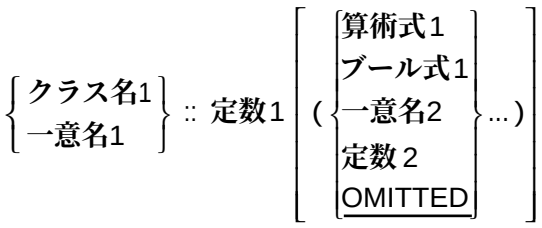
COBOL2002 では例外名による例外処理が追加されますが、オブジェクト指向機能を使った場合は、例外名の代わりに例外オブジェクトを使用できます。例外オブジェクトは、オブジェクト内に例外に関する詳細な情報を持たせることができるので、きめ細かい例外処理が可能です。

13.8.3 クラスライブラリ

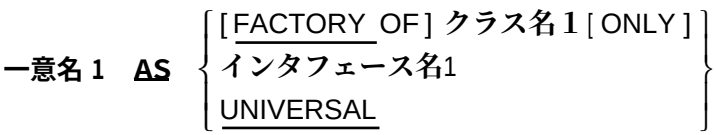
オブジェクト作成、初期化および他の便利な基本的な機能を含む BASE クラスを提供します。インスタンスオブジェクトを作成する NEW メソッドも、BASE クラスのメソッドです。

13.9 オブジェクト指向関連の構文

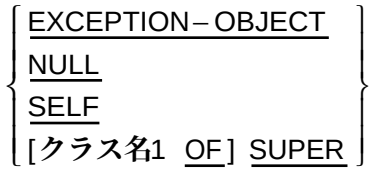
【行内メソッド呼出し】



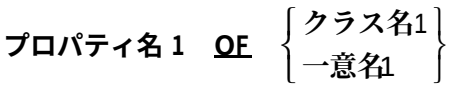
【オブジェクトビュー】



【定義済みオブジェクト参照】



【オブジェクトプロパティ】



【クラス定義】

[IDENTIFICATION DIVISION.]

CLASS-ID. クラス名 1 [AS 定数 1] [IS FINAL]

[INHERITS FROM { クラス名 2} ...]

[USING { パラメタ名 1 } ...] .

[オプション段落]

[環境部]

[ファクトリ定義]

[インスタンス定義]

END CLASS クラス名 1.

【インタフェース定義】

[IDENTIFICATION DIVISION.]

INTERFACE-ID. インタフェース名 1 [AS 定数 1]

[INHERITS FROM { クラス名 2} ...]

[USING { パラメタ名 1 } ...] .

[オプション段落]

[環境部]

[手続き部]

END INTERFACE インタフェース名 1.

【ファクトリ定義】

[IDENTIFICATION DIVISION.]

FACTORY. [IMPLEMENTS { インタフェース名 1 } ... .]

[オプション段落]

[環境部]

[データ部]

PROCEDURE DIVISION.

[{ メソッド定義 } ...]

END FACTORY.

【インスタンス定義】

[IDENTIFICATION DIVISION.]

OBJECT. [IMPLEMENTS { インタフェース名 2 } ... .]

[オプション段落]

[環境部]

[データ部]

PROCEDURE DIVISION.

[{ メソッド定義 } ...]

END OBJECT.

【メソッド定義】

[IDENTIFICATION DIVISION.]

METHOD-ID. { メソッド名1 [AS 定数1] / { GET / SET } PROPERTY プロパティ名1 }

[OVERRIDE] [IS FINAL].

[オプション段落]

[環境部]

[データ部]

[手続き部]

END METHOD [メソッド名 1].

【リポジトリ段落】

REPOSITORY.

[{ クラス指定子 / インタフェース指定子 / 関数指定子 / プログラム指定子 / プロパティ指定子 }...]

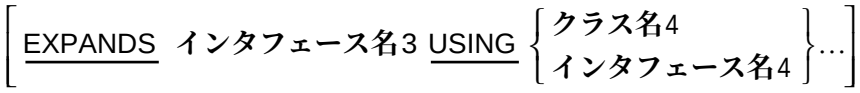
【クラス指定子】

CLASS クラス名 1 [AS 定数 1]

[EXPANDS クラス名2 USING { クラス名3 / インタフェース名1 }...]

【インタフェース指定子】

INTERFACE インタフェース名 2 [AS 定数 2]

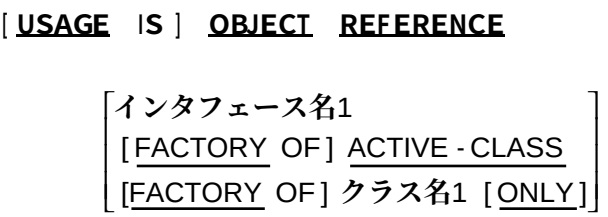


【プロパティ指定子】

PROPERTY プロパティ名1 [AS 定数5]

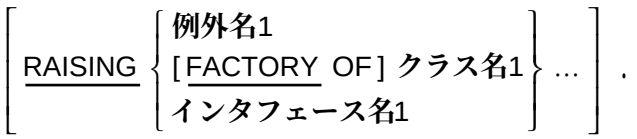
【PROPERTY句】

【USAGE句】

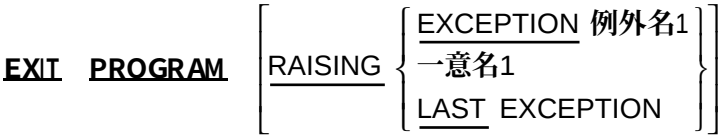


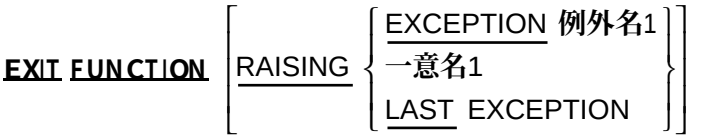
【手続き部の見出し】

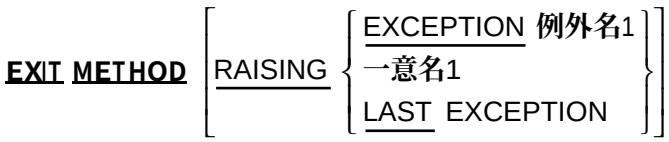
PROCEDURE DIVISION [USING指定] [RETURNING データ名2]



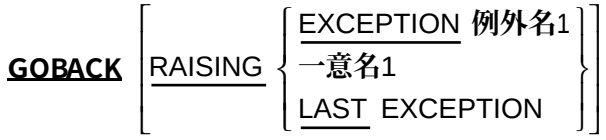
【EXIT文】



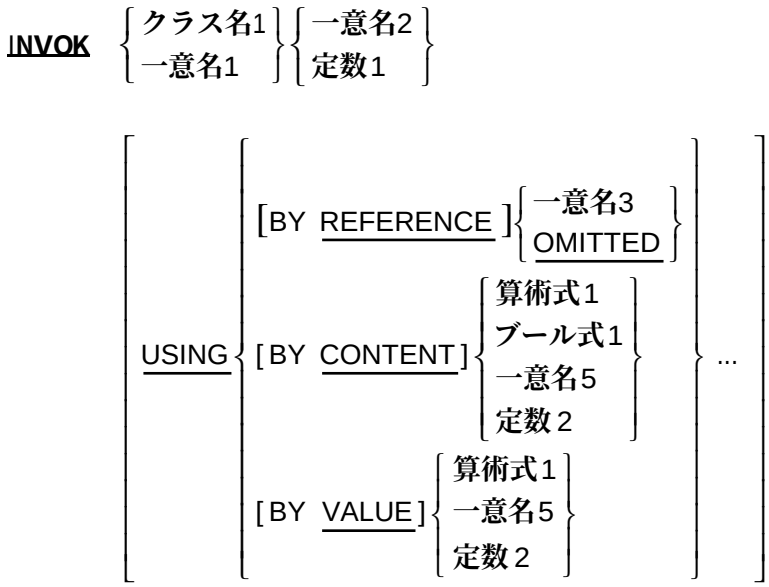



EXIT METHOD
RAISING
EXCEPTION 例外名1
一意名1
LAST EXCEPTION

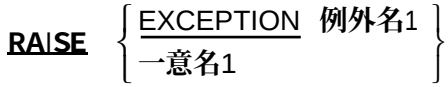
【GOBACK 文】



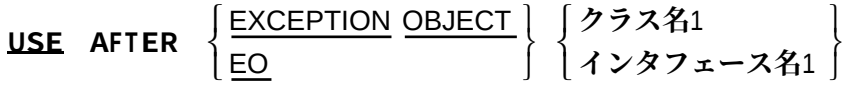
【INVOKE 文】



【RAISE 文】



【USE 文】



【SET 文】