ソフトウェア第三
— オブジェクト指向プログラミング —

機械情報工学科　水内郁夫

http://www.jsk.t.u-tokyo.ac.jp/~ikuo/lec/soft3/
(ID:soft3, pass:MechanoI)

2008 年 10 月 6 日 (月) その 2

オブジェクト指向プログラミングについて本講義では後日取り扱うが，本日午後の機械情報工学科の演習に少し登場するので，概要を今日触れておく．

1　オブジェクト指向プログラミング

1.1　抑えるべきキーワード

クラス，メソッド（メンバ関数），スロット変数（メンバ変数），インスタンス，コンストラクタ，デストラクタ，継承，スーパークラス，サブクラス，ポリモーフィズム，オーバーライド，オーバーロード，仮想関数，フレンドクラス，スコープ

1.2　オブジェクト指向プログラミングの背景

オブジェクト指向プログラミングという考え方が生まれた背景にはこの数十年で計算機の性能が爆発的に向上したことにより，従来より大規模なソフトウェアが書かれるようになってきた．大規模なソフトウェア開発が乱発，複雑化したために，ソフトウェア開発コストが上昇し，1960 年代にはソフトウェア危機という言葉も登場するようになった．そこでソフトウェアの再利用，部品化といったような概念をベースにしたソフトウェア工学が誕生することとなった．

プログラムを構成するコードとデータのうち，コードについては手続き，関数といった構成によって全体をブラックボックスとすることで再利用性の向上が提唱された．このようなスタイルは構造化プログラミングと呼ばれ 1967 年にエドガー・ダイクストラらによってまとめられた．

このように，オブジェクト指向プログラミングは，大規模なソフトウェア開発には欠かすことのできない概念であり，バグの対策や，複数人でのプロジェクト管理にも役立つ概念である．そのベースとなっている概念は，カプセル化と抽象化による，モジュールのブラックボックス化と，機能の明瞭化といえる．

1.3　クラスとインスタンス

オブジェクト指向プログラミングでは，従来までに学んだ基本的な言語（例えば C 言語や BASIC）などでは，関数を定義し，その関数を次々と呼び出すようなスタイルのプログラミングであった．

しかしながら，大規模なソフトウェアを構築する場合や，複数人のチームでソフトウェアを構築する場合，必ずしもこのようなスタンダードなスタイルが有効であるとは限らない．他人の書いたソフトウェアを理解し，自分の書くソフトウェアをそれと同調させることは一般的に難しい．オブジェクト指向プログラミングとは，機能，内部状態，が定義されたオブジェクトと呼ばれる単位のモジュールを使って，ソフトウェアを構築しやすくする技法の一つである．

オブジェクトの型に相当するものがクラスであり，内部状態に相当するものがスロット変数（メンバ変数）機能に相当するものがメソッド（メンバ関数）となる．オブジェクト同士の間で,「機能」に対するメッセージ通信を行うことでのみ内部状態を読み込んだり，変更したりすることができる．逆に言えば，許可なく相手のオブジェクトの内部状態を変更することはできない．このこのように必要最小限の情報だけを外部に見せ，複雑で外部に必要ではない部分を隠蔽するスタイルをカプセル化と呼ぶ．これはオブジェクト指向プログラミングの特徴の一つとなっている．

1.4 インスタンス

オブジェクトは実態のない宣言のようなもので，C 言語で言えば構造体の定義分に相当する．具体的なオブジェクトの実態はインスタンスと呼ばれ，コンストラクタと呼ばれるメソッドで構築される．逆にインスタンスはデストラクタによって削除される．例えて言うと，クラスは概念を表す一般名詞（例えばイヌ）で，インスタンスは文脈の中で現れた固有名詞（例えばポチ）と考えると分かりやすい．当然のことながらインスタンスは適宜複数個作られ，必要に応じて削除されて行く．

1.5 継承，スーパークラス，サブクラス

いくつかのクラスを使ってプログラミングを進めて行くうちに，かなり似た「機能」や「内部状態」を持つケースが出てくることがある．そのような「似たような部分が沢山存在する」場合，従来の関数定義型のプログラミング言語では，サブルーチンやライブラリを構築することで対応してきた．しかしながら，オブジェクト指向プログラミングの場合はクラス自体に重なりがあるため，サブルーチンの概念を適用することはできない．そこで，オブジェクト指向プログラミングの特徴の一つでもある，継承を用いることとなる．

継承とは，あるクラスのメソッドやスロット変数をそのまま「継承」し，新たにメソッドやスロット変数を追加することで新しいクラスを簡単に定義する概念である．例えば，イヌというクラスを継承して盲導犬というクラスを作ったり，学生というクラスを継承して高校生や大学生というクラスを作ったりすることに相当する．この場合，継承元になるクラス（学生）をスーパークラス（あるいは親クラス），継承したクラス（大学生）をサブクラス（あるいは小クラス）と呼ぶ．

1.6 ポリモーフィズム

ポリモーフィズム (polymorphism) とは，日本語で「多様性」のことを意味する．オブジェクトの機能を使用する立場（ユーザの立場）で見ると，異なるクラスに存在するそれぞれの異なる機能（メソッド）にある共通性がある場合がある．ユーザからすればなるべくシンプルな形で機能を使用したいのだが，クラスごとに異なるメソッドが用意されていると，面倒なことが起こる場合がある．この場合にポリモーフィズムの概念を用いたクラスとメソッドの定義を行う場合がある．

例えば，コンビニエンスストアで，荷物を送る場合を考える．ユーザの立場ではダンボール箱を持ってコンビニに行き,「宅配をお願いします」と店員に声をかければよい．しかしながら，実際に

は,「セブンイレブンでは宅急便のみを受け付ける」「ampm ではペリカン便のみ受け付ける」「ファミリーマートではゆうパックのみ受け付ける」というような制限がある.（もちろん例えばの話）しかしながらユーザは「宅配を依頼する」というインタフェースで目的を達成することが日常生活では当たり前になっている．これがポリモーフィズムの概念であり，異なるクラス（コンビニ）に共通するインタフェース（宅配）を親クラスのレベル（コンビに）で定義し，各サブクラス（セブンイレブン，ampm）では，そのインタフェースの実際の挙動を決めるメソッド（宅急便，ペリカン便）を各自定義しなおすことを行う．この定義しなおすことを，オーバーライドと呼び，親クラスに共通のレベルで定義するメソッドを仮想関数と呼ぶ.

似た用語として，オーバーロードがある．これは同じ関数名にも関わらず，引数の型の違いによって異なる挙動をさせたい場合に，複数の同じ名前の関数の定義をすることである．感覚的には上記のポリモーフィズムと同じような概念のように思えるが，このオーバーロードは同じクラス内で複数の同じ名前の関数を定義することになるので，全く違う概念である．STL ではテンプレートと呼ばれる概念でこのオーバーロードを簡略化することも可能である.

1.7 フレンドクラス，多重継承

C++ や Java などでは，public や private などの修飾子を用いてカプセルの内部なのか外部なのかを区別する．private で定義されたメソッドやスロット変数は外部から直接アクセスすることができなくなる．public で定義された場合はいつでも誰からでもアクセス可能である．フレンド関数は，基本的には外部からアクセスできないメソッドを半ば強引にアクセスさせることができる．フレンドクラスはクラスの全てのメソッドに対してフレンド関数を適用することと同等の概念である.

このほかにも，複数のクラスのメソッドを同時に使いたいという場合には，多重継承と呼ばれる手法も存在する．これは二つのクラスを同時に継承することに相当する．ただし，C++では可能であるが，Java では禁止されている継承のスタイルなので，注意が必要である.

このように異なる複数のクラスを考慮しながらプログラミングを進めていくと，同じスロット変数やメソッドが重複して登場することがある．基本的にはある名前のメソッドを呼ぶ場合には自分のクラスのメソッドが呼ばれるが，親クラスやフレンドクラスのメソッドを故意に呼びたい場合にはスコープ解決を行う必要がある．C++の場合には，[クラス名]::[メソッド名] という具合に，::でスコープを指定することができる.