オブジェクト

工学部情報工学科　1031060322　松本 亮介

平成 17 年 10 月 24 日

1　オブジェクトとは

1.1　オブジェクトとは何か

「もの」「対象」「目的」といった抽象的な意味を持つ言葉．大きく分けて,「物理的なオブジェクト」と「概念的なオブジェクト」に分けられる．一般的に，目に見える，触れる事のできるものを物理的なオブジェクトと呼ばれ，行為などの目に見えない触れる事のできないものを概念的なオブジェクトという.（ただし，一部の行為には視覚で認識できる概念的オブジェクトもある.）例えば，プレゼンテーションやキャンペーンは，行動や行為としては認識する事はできるが，物理的な「もの」としては認識できない.

1.2　視点による認識の違い

※スライド参照.

1.3　オブジェクトの構成要素

オブジェクトには，それ自身を特徴づける情報や動作がある．一般的に，オブジェクトを特徴付ける情報の事を属性（プロパティ），オブジェクトの動作の事を振る舞い（メソッド）という.

1.3.1　属性（プロパティ）

属性（プロパティ）は，オブジェクトを利用したり，識別するための情報である．テレビにとっては，チャンネルや音量など，利用者が任意の値を設定できるものもあれば，製造メーカー名や，テレビの画面の大きさ，重さなど，利用者には設定できないものもある.

また，プレゼンテーションのような概念的なオブジェクトにおいても，属性は重要な役割を果たす．開催日時，場所，テーマという情報がなければ，いつどこでどんな内容のプレゼンテーションが行われるのかわからない.

1.3.2　振る舞い（メソッド）

振る舞い（メソッド）は，オブジェクトを特徴づける動作である．テレビの場合，チャンネルを変更するという操作をすると，受信周波数が選択されたチャンネルに変更され，画面のその指定されたチャンネルの映像が映し出される．音量を調節すれば，それに応じて音量が大きくなったり小さくなったりする．これらを，オブジェクト指向における用語で言い換えると「テレビというオブジェクトは，利用者が『チャ

ンネルを変更する』という操作をすると,『チャンネル』属性が変化し，それによって『周波数を変更して画面の映像を変える』という振る舞いをする」となる．

概念的なオブジェクトであるプレゼンテーションの例で考えてみると，属性としてテーマ，講師，開催日時，場所などの情報があり，それらの属性に従って,「プレゼンテーションをする」という行為が振る舞いになる．

1.4 メッセージ

オブジェクトのプロパティを変え，メソッドを動作させるには,「きっかけ」が必要である．テレビの場合，電源を入れるとテレビが動き出し，リモコンなどでチャンネルを変更すると，選んだチャンネルの番組が映る．このように，オブジェクトの動作を開始するためには，オブジェクトを操作しなくてはならない．その操作がきっかけとなって，オブジェクトの属性が変更されたり，特定のメソッドが動作したりする．このような，オブジェクトへの働きかけの事をメッセージという．

1.5 UML におけるオブジェクトの記述

オブジェクトやそれに関する様々な要素や動作を表現する手法として，UML（Unified Mdeling Language）がある．UML には，システムやビジネスの流れなど要素をオブジェクト指向で表現するための図（daiagram）が数種類用意されている．オブジェクトを UML であらわすには，オブジェクト図（object diagram）という記述方法を用いる．

オブジェクトは，内部を二つの区画に分けた長方形で表す．上の区画には，オブジェクト名を記述する．「オブジェクト名:クラス名」という形式や,「オブジェクト名」または「:クラス名」という形式で表すことが出来る．下の区画には，オブジェクトの属性やその値を記述できるが，省略も可能である．UML では，オブジェクト名には必ず下線をつけるというルールがあるので注意しなければならない．属性（プロパティ）の記述に関しては，属性の名称だけでなく,「＝」を使って具体的な値を記述する事も出来る．オブジェクトが存在する事，あるいは必要であることのみを表現したい場合は，下の区画を省力する事ができる．

1.6 オブジェクトのリンク

オブジェクトの存在意義は，互いに関係しあう事で明確になる．例えば，テレビ，PC，自動車などのオブジェクトも，それらを利用する人がいてはじめて存在意義を持つ．テレビや PC などは，利用者がメッセージを送る，つまりは特定の振る舞いを引き起こさせなければ利用する事ができない．このようなオブジェクトの関わりをリンクという．

2 クラス

2.1 クラスという概念

クラスとは，類似するオブジェクトが持つ共通の特性（属性，振る舞い，関連性）を抽象的に表したものである．PC を例に例えてみると，PC にはデスクトップ型，ノートブック型，ポケット PC 型など様々な種類のものがある．しかし，どのような形状であっても，キーボードやマウスを使う点などは共通している．といったように，見た目が違っていても PC としての基本特性は同じだといえる．オブジェクトの属性や振る舞いをこのような共通特性としてみれば，オブジェクトをある程度グループ化することができ

る．このようなグループ化の単位をクラスという．クラスとは，オブジェクトの概念的な原型（設計図・雛形）と考える事もできる．

プログラムの世界でのクラスを考えてみる．属性（プロパティ）とは変数のようにある情報を保持するもの，メソッド（振る舞い）とは，関数や手続きと同じものと考えられる．プログラムの世界では，一定の大きさを示すには型を使うため，オブジェクトを特徴づけるためにクラスに定義された，プロパティやメソッドの大きさをあわせたものがオブジェクトの大きさになる．クラスとは，オブジェクトの要素（形や大きさ）を表す原型・設計図であると考えられる．

プログラムは，メモリにロードされて実行される．オブジェクトを作成すると，まず，メモリのそのオブジェクトの基となるクラスの大きさが割り当てられる．これによって，情報を保持する事が可能になる．割り当てられたメモリ空間は，クラス（型）と同じ構造を持つため，クラスを使って利用していく．オブジェクト思考とは，このように割り当てられたメモリ空間を高度に効率よく利用するための技術である．

2.2　クラスとインスタンス

クラスに対して，インスタンスとは,「オブジェクト」という言葉が抽象的な概念としての意味合いが強いのに対して，明確に実在するオブジェクトの事を言う．なかには，特定のインスタンスをオブジェクトと呼ぶ事もできる．ただし，インスタンスは漠然としたものを指して使う事はできない．インスタンスはそれだけが単独に存在する事はできない．インスタンスを生み出す何かが必要であり，インスタンスを生み出すための基本情報や雛形がクラスになる．インスタンスはいわば部品であり，部品の型さえ作っておけば，部品の生産は楽になる．さらには，色々な場所で使われている部品は，その安全性や機能性が保障されているため，この部品の組み合わせで何かを作る事で，より生産性の工場も見込める．

オブジェクト指向にそった，JAVA のプログラミングでは，

１，ソフトウェアを構成する部品の型であるクラスを定義する
２，クラスからインスタンスである実際の部品を生産する
３，それらの部品を組み合わせてソフトウェアを構築する

という流れになる．

2.3　オブジェクトの作成

オブジェクトの作成には，そのクラス自身のメソッドを使う．このメソッドのことを「コンストラクタ」という．コンストラクタは，クラスの内容をメモリに展開して,「属性の初期化」や「オブジェクトの作成に関する処理」などを行う．

※スライドの図より

クラス名と同じ名前をつけたメソッドがコンストラクタになる．スライドの図では，チャンネルを表す変数を０にするコードがコンストラクタに書かれているので，オブジェクトを作成した時，これらの変数には０が格納される．

オブジェクトの作成する時の作業の手順としては，

１，コンストラクタが呼び出されクラスがメモリ空間の中に読み込まれる
２，コンストラクタに書かれているメソッドを実行して（この場合はチャンネルを０へ初期化）オブジェ

クトを作成する
3，メモリ空間に読み込まれたオブジェクトにアクセスするための情報（メモリアドレスの値など）を返す

である．

このように，メモリに読み込まれたオブジェクトにアクセスするための情報の事をインスタンスという．クラスを型として作成した変数（この場合は「t」）にインスタンスを代入する事で，この変数を使ってオブジェクトが利用できるようになる．インスタンスを受け取る変数のこと，をインスタンス変数（この場合は「t」）と呼ぶ．このように，コンストラクタの役割は，オブジェクトの作成とメンバ（クラスに記述された要素）の初期化を行い，作成したオブジェクトにアクセスするためのインスタンスを返す事である．

2.4　オブジェクトの破棄

プログラム中のオブジェクトも不要になればメモリ空間から破棄する．オブジェクトを作成するメソッドであるコンストラクタに対し，オブジェクトをメモリ空間から破棄するメソッドをデストラクタという．しかし，JAVA ではプログラムで使わなくなったオブジェクトを自動的に破棄する機能をもっており，手動でメモリから破棄する必要がない．このようにオブジェクトを自動的に破棄するしくみの事をガーベジコレクションという．

3　オブジェクト指向のテクニック

オブジェクト指向には特徴的な機能がある．そのひとつとして，カプセル化というものを紹介する．オブジェクトには，利用する側に見える部分と見えない部分がある．自動車にはハンドル，アクセル，ブレーキなどがあるが，これらは私たちは見たり触ったりする事ができる．しかし，エンジン，シャーシなどは，ボンネットを開かない限りは目にすることができないし，それらを自分で改造，交換といった作業ができるのは専門的な知識を持っている人だけにしかできない．

同じように，プログラムのオブジェクトでも，利用するために外部から見える部分と，操作に関係ないため隠されている部分が存在する．このように，操作に必要な部分だけを外から見えるようにして，それ以外の部分は見えないようにクラスを設計する事を，オブジェクト指向の世界で「カプセル化」と呼ばれている．

さらに例を挙げて考えてみる．CD プレイヤーを利用する時は CD をセットする．CD を取り出す時は，エジェクトボタンを押す．CD を入れると，内部の装置によって音楽を再生する事ができるが，これは CD をセットする事によって内部の装置が作動し，CD からの情報の読み出しという振る舞い（メソッド）が行われる事を意味している．読み込みを行う箇所は CD プレイヤーというオブジェクトにとって大変重要な部分であるため，むやみに手を加えてしまうと故障や誤作動の原因になりかねない．そのために，カバーにより厳重に保護され，外部から隠されている．

このように，重要な機能を持つ部分を保護するために外部から隠す事をカプセル化といい，CD プレイヤーオブジェクトは CD から情報を読み出す機能をカプセル化したオブジェクトであり，外部との窓口としてトレイやイジェクトボタン，再生ボタンなどをもっている．カプセル化した機能を利用するためにの機能はユーザーに公開されている部分であり，外部からのメッセージを受け取って振る舞いをおこす窓口になる．

3.1 アクセス制御

プログラムにおいて，オブジェクトの利用側がしようすべきでない，または，使用してほしくない属性（プロパティ）や振る舞い（メソッド）を隠して（隠蔽），利用して欲しい属性や振る舞いのみ利用側に見せる（公開）ことができる．オブジェクト指向のプログラム言語には，機能の公開や隠蔽を行うためにアクセス修飾子という機能がある．

「アクセス修飾子」
private-プライベート-同じクラスの中でしか利用できない
public-パブリック-オブジェクトの利用側がどのクラスであっても自由に利用できる

※スライドのプログラムの例
スライドの例では，Television クラスの属性とメソッドの定義にアクセス修飾子を付加している．プロパティには全て private，メソッドには public をつけている．このクラス内の変数は全て private になっているので，これらの変数をしようできるのは Television クラスのメソッドのみであり，オブジェクトを利用する側からは認識する事も使う事もできない．メソッドには public をつけているので，こちらはクラスの内部からも外部からも自由に利用する事ができる．

このようにして，private を指定した機能はカプセル化され，public を指定した機能は公開される．つまり，この例では「プロパティは全てカプセル化されているが，メソッドは全て公開されている」表現できる．

3.2 UML におけるアクセス制御の記述

UML では，クラス図に定義した属性やメソッドのアクセス制御を非常に簡単な記号で表すことができる．アクセス制御を表すには，クラス図に定義されたメンバの前にそれを表す記号をつける．
「-」をつけると private,「+」をつけると public なメンバや属性を意味する．

3.3 アクセサメソッド

アクセス制御で設定したように，プロパティが全て private になっていると，利用者側から認識する事ができない．プロパティには，オブジェクト内だけで必要なものとオブジェクトを利用する側でも必要なものがある．例えば，Television オブジェクトの場合，オブジェクトの利用側が「製品名」や「メーカー」を所得したくても，private がついているため所得することができない．

このように，オブジェクトを利用する側から参照したり，書き換えたりしたいプロパティがある場合，それらの変数には public アクセス修飾子をつけて公開すべきなのか．その答えは「ノー」である．変数には，恒に private アクセス修飾子をつけてそのオブジェクト内部でのみ利用するようにすべきである.「製品名」や「メーカー名」などは，そのオブジェクトを表す重要な情報であるため，オブジェクトの利用側がどのクラスであっても直接的に自由に利用できるというのは，好ましくないからである．さらには，値の妥当性のチェックやソフトウェアとしての保守性というメリットもある．

では実際に，情報を外部から参照したり書き換えを行うメソッドを作成して公開する例を考えてみる．このようなメソッドのことをアクセサメソッドと読んでいる．JAVA の場合は，属性を参照するメソッドの名前は属性名の前に get をつけたもの，属性を変更するメソッドの名前は属性名の前に set をつけたものとする．属性を参照するメソッドの名前をゲッター，属性を変更するメソッドをセッターという．これによって，プロパティを間接的に利用できるだけでなく，同時に関連する処理を行えるなどのメリットがある．

※スライドのプログラムの例

Television クラスの maker プロパティと name プロパティにはセッターがなく，ゲッターしかない．したがって，maker と name は読み取り専用プロパティといえる．また channel プロパティにはゲッターとセッターの両方があり，値の参照だけでなく，変更も可能になっている．

3.4　カプセル化の有用性のまとめ

前節で述べた，カプセル化の有用性をまとめてみる．カプセル化の有用性としては，

・メソッド内で値の妥当性チェック
・ソフトウェアとしての保守性が高まる

が挙げられる．属性（プロパティ）に値を設定する際に，メソッドを介すことで，その値の妥当性チェックができる．たとえば，チャンネル属性のように負の値を設定してはいけないような属性であれば，セッターメソッド内で渡された値が正かどうかをチェックする処理を付け加えることで，妥当性のチェックが可能になる．

また，ソフトウェアとしての保守性が高まるとは,「クラスの実装を変更したとしても，影響を及ぼす範囲を小さく抑える事ができる」ということである．例えば，クラスに設定されている属性の中から最初に指定していた属性とは違う属性を参照したい場合などに，アクセサメソッドを介していればアクセサメソッド内のコードを変更するだけですみ，そのクラスを利用しているほかのクラスのコードを変更する必要がなくなる．つまり，カプセル化を進めることによりオブジェクト内部の仕様変更が外部に影響しなくなり，ソフトウェアの保守性や開発効率が高まり，プログラムの部分的な再利用が容易になる．

4　まとめ－オブジェクト指向の有用性

以上で，オブジェクト指向における基本的な内容の説明は終わりであるが，オブジェクト指向の有用性をまとめておきたいと思う．ここでは，オブジェクト指向の特徴的な機能として「カプセル化」を取り上げたが，他にも「継承」,「インターフェイスと抽象クラス」,「ポリモーフィズム」といったものもある．これらをふまえた上で，オブジェクト指向の有用性を簡単に挙げてみると，

・大規模なソフトウェアの開発に向いている
・内部動作の詳細を覆い隠し，利用しやすくする

が挙げられる.「大規模なソフトウェアの開発に向いている」というのは，オブジェクト指向とは，関連するデータの集合と，それに対する手続き（メソッド）を「オブジェクト」というひとつのまとまりとして管理し，その組み合わせによってソフトウェアを構築していく．そのため，すでに存在するオブジェクトについては，利用に関してその詳細な内部構造や動作原理を知る必要はなく，外部からなんらかのメッセージを送れば機能するため，大規模なソフトウェアの開発において有効な考え方とされている．

「内部動作の詳細を覆い隠し，利用しやすくする」とは，例えば「テレビ」というオブジェクトは自身の内部を構成する電子回路を動作させる手続きが決まっているため，リモコンなどで適切なメッセージを与えるだけで動作する．このように，個々の操作対象に対して，個々の操作方法を設定する事で，詳細を

知ることなく利用しやすくしようとする考え方といえる.

このような考え方を応用したプログラミング技法を総称して「オブジェクトプログラミング（OPP)」と読んでいる.

参考文献

[1] 先生にお借りした資料

[2] 米山 学　JAVA 教科書　SHOEISHA .

[3] http://e-words.jp/w/E382AAE38396E382B8E382A7E382AFE38388E68C87E59091.html

[4] http://e-words.jp/w/E382ABE38397E382BBE383ABE58C96.html