オブジェクト指向とは何か？

高根晴

アウトライン

はじめに

　オブジェクト指向(object-orientation)とは直訳すると「モノ中心」や「モノ指向」と訳される.これだけでは、それがいったい何を表現しているのかわからないが、簡単に要約すると object(物)が持つ性質とそれどうしのやり取りに着目する思想であると捉えることができる.これまで,オブジェクト指向は情報科学をはじめ,情報実務,マーケティングなどのビジネス分野など様々な分野で用いられてきた.

　本稿では,オブジェクト指向の系譜の中でも 1960 年ごろから始まり,80 年頃から盛んとなったオブジェクト指向を反映したプログラム(OOP: object-oriented-programing)をことわりがない限りオブジェクト指向とし,その基本概念、機能的利点、オブジェクト指向をき

っかけに発展した技術や開発プロセスについてまとめていきたい.

1. オブジェクト指向の基本的概念-

1-1.オブジェクトとは何か

そもそもオブジェクト指向でいう「オブジェクト(object)」とは何なのか.oxford learner's dictionaries によると object とは大きく分けて三つの意味で構成されている.一つ目が“a thing that can be seen and touched, but is not alive”というもので,生命はないけれど,見えて触れることができるものの総称を指す言葉のようである.二つ目の意味が, “object of desire, study, attention”であり,これは私たちが目的や着目する対象を指すものであるようだ.それに関連して三つ目は “an aim or a purpose” というものであり,これもまた目的となる物や対象を指す概念であるようである.これらの意味からするに,オブジェクト(object)とは大まかに私たちが見たり触れたりすることができる「物」,そして私たちが対象とする「物」を表す概念のようである.

オブジェクト指向のオブジェクトは「物」を表す言葉であることがわかったが,「物」そのものではわかることやできることは少ない.私たちは,そのものについてそれが何であるのか？どのような機能を持つのか？どのように使うのか？を知ることによってそれを理解し,用いることができる.例えば,哲学者のヴィトゲンシュタインは『論理哲学論考』の中で,「世界は事実の総体であり,物の総体ではない」という言葉に代表されるように,我々の認識している世界は物そのものとして存在しているのではなく,物の振る舞い,物がとのやり取りから生まれる事実や実態によって構成されているのだとしている.つまるところ,「物」というのが意味を持ち認識されうるためには,振る舞いや機能それとのやり取りが付加される必要がある.

その付加される情報を指すものがオブジェクトの概念と並んで,柱となる概念と言われるクラスとインスタンスである.クラスとは,オブジェクト指向プログラミングにおいて,ほぼオブジェクトと同義であることが多い,実際多くのオブジェクト指向を反映したプログラミング言語においては,最上位のクラスは,オブジェクトクラスとなっており,その同義性がうかがえる.

ただもう少し踏み込んで考えれば,クラスはオブジェクトという抽象概念そのものを指すよりも,セグメンテーションないし,分類された後のオブジェクトであるといえる.例えば,

我々の多くは普段「りんご」を「物」とは呼ばないが,「りんご」は「物」である.オブジェクトとクラスの関係はこのりんごと物の関係性に似ていて,オブジェクトという上位概念を,何らかの基準に基づいてより分割したり,セグメンテーションした後の「物」の姿をクラスと言ったりすることが多い.この何らかの基準というのが重要で,その基準によってクラスの取られ方は異なる.例えば,オブジェクト単体しか必要ない場合、「オブジェクトクラス」の存在だけで良い.しかしながら我々は自らの基準や要望に基づいて,そのオブジェクトを「果物」とそれ以外というクラスにも分割して認識することができるし,「りんご」、「バナナ」、「電車」などより細かなクラスにも分割して認識することができる.このようにオブジェクトとクラスはほとんど同義的な概念として捉えられながらも,クラスはオブジェクトよりもより可変的で具体的である.

翻ってインスタンスとは,オブジェクトやクラスが持つ具体的な振る舞い（機能）やデータを指す.前述にあるとおりオブジェクトはそれそのものでは何ら知識や理解をもたらさず,それが何らかの機能ないし,ふるまいを持つことによって我々に知識や理解をもたらす。したがって個々のオブジェクトには何らかの情報や機能、ふるまいが付加される必要がある.その付加される情報や機能、ふるまいの総称を一般的にはインスタンスと呼ぶのである.例えば,「桜」というクラスがあったとして,それは「ソメンヨシノ」という名前の情報をもち,「薄いピンク」という色の情報をもち,そして「3 月後半〜4 月初旬に開花」という開花の機能を持つとする,ここでいう名前,色、開花と言った情報、機能をまとめてインスタンスと呼ぶ.この桜クラスが,「御衣黄」,「薄い緑」,「4 月中旬〜下旬」というインスタンスを持つ可能性もある.同じ桜クラスでも持つインスタンスによってもたらされる知識やふるまいが異なる.つまるところ,オブジェクトやクラスはインスタンスを持つことによって,はじめてその実態,役割,を帯びることになる.またクラスがインスタンスを帯びることはクラス間のやり取りを可能にする.例えば,「花見」というオブジェクトの,見る対象となる花が「桜」であるとすれば,「ソメンヨシノ」を見るのか,はたまた「御衣黄」を見るのかによって,実施場所や開催地といったインスタンスは異なってくるだろう.このようにクラスがインスタンスを持つことによって,クラス間のやり取りも可能となってくるのである.

1-2.継承,多相性,カプセル化

オブジェクトやクラス、それ自体が独自に持つ情報や機能,データをインスタンスと呼ぶが,そのインスタンスの機能をより効率的に活用しようとする仕組みがオブジェクト指向には見られる.大きく分けられると,三つのものがあげられ,継承,多相性,カプセル化である.

継承とは,あるクラスが持つ全てのインスタンスを直接的に引き継ぐことのできる機能である.この時,インスタンスを継承する側をスーパークラス,される側をサブクラスという.例えば,人間クラスは日本人クラスクラスやアメリカ人クラスに対して継承可能であろう.人間が持つ,手や足の機能といったインスタンスはより具体化したクラスである,日本人やアメリカ人に対して広く適用できるので,人間が持つ基礎的なインスタンスを継承することが可能である.そうすることによって,プログラムであれば,クラスを書くたびに多くのクラスに共通して装備されるインスタンスを書く必要がなくなり効率性がもたらされる.ちなみに,スーパークラスから継承されたインスタンスをサブクラスで書き換えることも,多くの場合可能であり,それをオーバライドという.

次に,多相性(polymorphism)であるが,これは,同一名のインスタンスでも,そのクラスによって機能が異なることを表す.例えば,挨拶インスタンスを持つ言語クラスが存在し,英語や中国語などの各言語クラスに継承され,オーバーライドされたとする.その挨拶インスタンスは,同じ挨拶インスタンスであっても英語クラスに属すれば, “hello”と振る舞うし,中国語クラスに属すれば “你好” と振る舞うだろう.ここでは,つまりインスタンスの名前が同じでもクラスによって振る舞いが異なることにより,クラスが独自で持つ機能や役割が描写されている.このような,同一名のインスタンスであっても,そのクラスが違うことによって振る舞いが異なることは,継承と組み合わせると効果を発揮する.例えば,共通するインスタンスを継承しつつ.書き換えが必要な機能だけ,継承先で書き換えることによって,プログラマーはサブクラスのインスタンスを全て書く必要性はなくなるし,一部機能をオーバーライドすることによって,サブクラスの独自性,識別を明記することが可能となるのである.

ただ,そのインスタンスについては全てが独自性を持つことが許されたり,他のクラスとのやりとりが可能であったり,オーバーライドされてしまってはインスタンスの破壊的な改変や思わぬ改変が起きることとなるだろう. 例えば,パスワードなどの情報は外部から無闇にアクセスしたり,継承できてしまうと,改変されたり,思わぬ変更がある可能性があり危険である.したがって,それを未然に防ぐ術が必要とされるだろう.オブジェクト指向ではそのようなクラスが持つインスタンスについて,外部からのアクセスを制限したい場合や改変を制限したい場合,制御することが可能となっている.多くの場合,private などの修飾子を設定

することにより,インスタンスを外部のアクセスや改変から保護することができる,この機能をカプセル化といい,クラス間のやり取りを中心とするオブジェクト指向において,そのやりとりが持つ脆弱性を克服する重要な役割を果たしている.

1-3.小括

ここまでは,オブジェクト指向で共通する基本的概念についてまとめてきた.簡単に要約すると,まず,オブジェクト指向では,オブジェクトとそれが分割されたクラスが存在する.そして,そのクラスは独自の振る舞いや機能,情報であるインスタンスを持ち,そこではじめてクラスの機能や役割,が備わり,すなわち意味をなし,そして他のクラスとのやりとりが可能となる.そのインスタンスはより特化したクラスに継承することができる.また,同じ名前のインスタンスであっても,それが属するクラスによって,振る舞いや持つ情報などは異なる多相性という働きが供えられていて,同じクラスをスーパークラスにもつサブクラスであってもそれぞれが独自の振る舞いを見せることができる.また,可変不可や変更リスクが存在するインスタンスについては,カプセル化によって保護が可能である.

2. オブジェクト指向の機能的利点

これまで 60 年もの歴史を持つオブジェクト指向であるが,それが形態や言語を変化させ,分散しつつも,ここまで続いてきた理由の一つとして,その機能的利点があるだろう,ここからはオブジェクト指向,およびその言語が持つ機能的利点をまとめていく,それは主に集約,量産,保護を可能としたところである.

2-1.集約

オブジェクト指向の特徴として挙げられる大きなものの一つ目がクラスによる集約を容易としたことである.これは,前章で取り上げたオブジェクト指向の基本概念のクラスと対応する.クラスの役割は,何らかの法則にしたがってオブジェクトをより具体化,分割化することであったが,このような同じような情報,機能を一つのクラスにまとめてやる集約が容易であることによって,二つの機能的利点が発生する.

一つ目の利点としては,名前づけが楽になるということである.オブジェクト指向においては,基本的にクラス内で名前が重複していなければ,インスタンスの名前は同じものを使

用しても良い.例えば,桜クラスと梅クラスがあったとして,それらは両方名前インスタンスを持つことができ,一方で名前のインスタンスを使えば,一方では使ってはならないということはない.このような,別クラスでの名前の重複が許されることは,機能的には同じだけれど,名前の重複を防ぐために別の名前を考案しなくてはならないといった手間や,それをもとに生じるプログラマー間の齟齬,また規約の複雑化を軽減することが可能になると考えられる.

二つ目の利点としてはコードの可読性の向上が挙げられる,クラスによって,機能や情報ごとに適切に分類してやり,それに名前をつけてやれば,それらの機能や情報が散らばっている時よりも,コードは直感的理解が可能で,そしてオブジェクト同士のつながりも見えやすい.そして,再利用する時も呼び出しやすい.これはまさに整理された図書館の本棚や整理された道具箱のようでどこに何があるのか,一貫性のある規則で分類されていれば,使うときに直線的に使うことができるし,探すのに労力を使うことも少なくなる.逆に言えば,この分類規則や名前の付け方が直感的でなく,独善的である場合,個人的な利用をする場合は効率的かもしれないが,他者が介在する場合,齟齬や効率性が落ちてしまう.このような点から,オブジェクト指向における一つのスキルとしていかにクラスの集約を整合的に一貫性のあるものにするのかということが重要視されている.

2-2. 量産

オブジェクト指向の特徴として挙げられる大きなものの二つ目としてインスタンスの量産を容易にしたことがある.まず,クラスは基本的には再利用可能であるため,そこからいくつものインスタンスを生成することができる.そして,そのクラスの機能は前章であげた,サブクラスへの継承が可能であり,同じインスタンスを持つクラスやその機能の量産が容易となっている.また,多相性によって継承先のサブクラスでインスタンスを書き換えることも可能である.このことによって発生する機能的利点は二つある.

一つ目はコード量の減少である.クラスが何回も利用でき,そこからいくつものインスタンスを生成することが可能となっていることによって,またインスタンスが継承できることによって,一つのクラスから多くの機能や情報を実装することができるようになる.一つのクラスによって多くの機能を包含することができれば,その機能を作るたびにその機能の中身や処理となるコードを書かなくても良くなる.したがって,プログラムを書く手間が機能を実装するたびに書いているよりも省け,コードが効率化される.

これに関連して二つ目として挙げられるのがコード伝達の容易性で,オブジェクトの機能が量産できることによって,多くの言語でいうパッケージやクラスライブラリ、フレームワークなど他人が作ったクラスを利用することが容易になり,アルゴリズムを考える手間が省けたり,より幅広いユーザーがプログラムを書くことが容易になったりする.

つまりクラスやインスタンスの量産が容易になることによって、コード量の減少,そしてコード伝達の易化,開発に必要な時間が短縮,プログラミングやコードが書ける人数が増加など開発の効率性の向上や、技術を応用しやすくなるといった恩恵がもたらされたと考えられる.

2-3.保護

オブジェクト指向の特徴として挙げられる大きなものの最後がクラスやカプセル化により保護が容易になったことである.クラスの中にインスタンスとして組み込まれている変数や関数は,クラスの外部から直接的にその関数を呼び出すことによるアクセスや書き換えは不可能である.またクラスの外部から一切のアクセス可能を不可とするカプセル化も可能である.

この保護を容易にすることによってプログラムの保守性が守られると考えられる.関数や変数をクラスで保護し,外部からの書き換えを不可能にしてやることによって,変数や関数の思わぬ書き換えやそれを原因とするエラーが発生しにくくなるのだ.これは大規模な開発によって関わるエンジニアの人数やコード数が増えた時など,一つの間違った書き換えなどによる連鎖的なトラブルが発生しやすい環境で特に恩恵をもたらす.先にあげた,集約や量産はクラスを用いなくても,関数や変数を用いた場合である程度は可能となるが,オブジェクト指向はそれに加えて保護を可能としたことによって,コードの思わぬ書き換えを防止するといった保守性を高め,集約や量産の効果をより安定的なものとしたと思われる.

2-4.小括

以上,本章では,オブジェクト指向が持つ機能的利点をまとめてきた.まとめると,オブジェクト指向が持つ機能的利点とは,クラスによる集約,そしてその量産を可能としたことによる,名前付作業の手間の削減,コードの可読性の向上,コードの再利用の可能性,コードの伝達の容易性をもたらし,開発作業の効率性を向上させたこと,そしてその集約,量産に保護が加わることによってコードを書き換えてしまうリスクが減り,集約や量産が持つ効率性の向上

効果をより安定的なものとしたということであると言える

3.オブジェクト指向の発展型

3-1.クラスライブラリー,フレームワークの発展

　オブジェクト指向はやがてクラスライブラリーやフレームワークといったものに発展した.

　まず,クラスライブラリとは他者や他の組織によって作られたクラスを保存しておく機能のことをいう.ライブラリというだけあってクラスが本棚に並べて置かれていることをイメージすると良い.オブジェクト指向以前も関数が集約されたライブラリである関数ライブラリなど類似した機能は存在したが,クラスライブラリはオブジェクト指向の利点を生かすことによって関数ライブラリよりも使い勝手が良くなったと考えられている.関数ライブラリにおいては,基本的には関数を呼び出して,利用するだけであった.しかし,それがオブジェクト指向のクラスライブラリとなることにより主に三つのことが可能となった.まず,ライブラリの中からクラスのインスタンスを作成してメソッドと変数定義をまとめて利用することが可能になった.関数ライブラリでは,関数を一つ一つ呼び出していたのを,オブジェクト指向が持つ集約機能によって,関数の集合としての一つの呼クラスの呼び出しによって,そこからいくつもの機能を実装できるようになり,ライブラリの利用がより効率的になったのである.そして次に多相性によってライブラリから呼び出される側のクラスの機能を呼び出す側の都合で置き換えることが可能となる.つまり何のクラスに呼び出されるのかによって関数の実装を変化させることができたり,すでに呼び出す側で作られた同一名の関数とも共存が可能となったりする.最後にライブラリ内のクラスにメゾットや変数を追加定義して新しいクラスを作成することが可能となる.継承の機能によってライブラリの中のクラスに新しいクラスやライブラリ内の別のクラスを継承し、使い勝手の良いクラスをカスタマイズすることが可能となる.つまるところ,クラスライブラリはオブジェクト指向の持つ機能的利点が体系化されたしたものとなっており,プログラミングを行う上での効率性の観点でなくてはならないものとなっている.

　クラスライブラリと並んでオブジェクト指向の発展型として捉えられているのがフレームワークである.フレームワークも基本的には他者や他の組織によって作られたクラスを保存しておく機能であることはクラスライブラリと同じであり,その利点も先ほどあげた３つのものがあげられる.ただクラスライブラリと異なるのは,クラスライブラリが特定の目的

を持たなクラスの集合であるのに対して,フレームワークは特定の目的を持つクラスの集合を指す.フレームワークは,アプリケーションなどの特定のものを作成することを想定して,それに必要と考えられるクラスの集合があらかじめ容易されているものを指す.プログラマーはフレームワークを利用し,自らが必要とするものをライブラリから呼び出すことによって,アプリケーションなどの機能をそれぞれ最初の状態から書く必要がなく,目的となるプログラムを作成することができ,また必要に応じてそれをカスタマイズすることも可能となる.そのため,フレームワークを利用することによって,コード量の短縮やそれに伴う開発時間の短縮などの開発の効率性がもたらされたと考えられる.

3-2.反復型開発プロセスの発展

オブジェクト指向の直接的な成果ではなく,またコンピューターなどのデバイスの性能の発展や小型化などの影響も絡んでくるのであるが,オブジェクト指向が持つ,集約,量産,保護といった機能は現在 web アプリケーションやソフトウェアの開発のプロセスである反復型開発プロセスの発展に寄与していることが言われている.

そもそも反復型開発プロセスは,要件定義,設計,プログラミング.テストの流れ（インテレーション）を複数回実施することによってソフトウェアの開発を行うことを指す。これを行うことによって,新たな要件の追加や完成品に対する環境の変化に合わせた対応が行いやすくなったとされている.反復型開発プロセスのテイストが濃く反映されているアジャイルソフトウェア開発,エクストリームプログラミング（XP）,テスト駆動開発やリファクタリングといった手法は現在の web やソフトウェアの開発において大きな役割と地位を占めている.

ではなぜオブジェクト指向が反復型開発プロセスを可能としたのであろうか。前述にもあるようにオブジェクト指向によって,クラスによる集約,保護,量産の機能が高まり,コードの可読性が向上したり,プログラムの保守性を保つことが容易になったり,継承の機能やクラスライブラリーやフレームワークの登場などプログラムの伝達が容易になった.そのことにより,一つのコードの改変が他のコードに与える影響が減少し,また変数名の変更によって同一名の違う機能を持つ変数の上書きなどの破壊的な改変のリスクも減少した.加えて,プログラムを部分的に分けて同時編集を行うことが容易になった.もちろん,これらが可能となったのは,コンピューターデバイスなどの機器の性能の向上や,git 管理などの発展の影響もある.ただ,オブジェクト指向が現在の主流な開発手法となっている反復型開発プロセ

スの発展に寄与していることは先にあげたオブジェクト指向の機能的利点や,それと反復型開発プロセスの相性のよさから窺い知ることができる.

つまり、現在の主流となっている開発のプロセスである,反復型開発プロセスはオブジェクト指向があることを前提においてそれぞれの工程や作業がなされているといっても過言ではないのである.

終わりに

本稿ではオブジェクト指向プログラミング（OOP）の基本的概念,機能的利点,そしてそれを生かして発展した機能や技術について触れた.オブジェクト指向とは,オブジェクト,そしてクラスが持つ振る舞い（インスタンス）,その間のやりとり,保護の範囲の決定.それらは継承,多相性,カプセル化というかたちで説明された.そして,オブジェクト指向は集約,量産,保護という特徴をもたらし開発をより効率的に,安全に,シンプルにした.さらに,最終的にそれらはクラスライブラリやフレームワーク,また反復型指向プロセスといった,より効率的なツール,そして開発プロセスを生んだのであった.

ここまで見るとわかる通り,オブジェクト指向は効率的な開発を行うための手段として優れているのである.ただ,まだその手段としてのオブジェクト指向の使い方は確定的なものではなく,例えば Metz(2016)など多くの専門書でオブジェクト指向の有効な使い方やコードの書き方が議論されているようである.今後の筆者など新卒エンジニアなどの課題としては,オブジェクト指向の基本的原則はおさえつつも,自らなりにそれをどのように業務に持ち込むのかを考察し,そしてそれを実践してみることだろう.また,オブジェクト指向そのものが目的であってはならず,非効率なこだわりや時と場合に応じてその程度を使い分ける柔軟性も求められるだろう.

参考文献

三谷純(2021) “プログラミング学習シリーズ：JAVA 第 3 版 入門編”,日経 BP.

平澤章(2021) “オブジェクト指向でなぜ作るのか：知っておきたい OOP、設計、開発の基礎知識”,日経 BP.

Bibliothek Suhrkamp(1918)『論理哲学論考』,野矢茂樹訳,岩波文庫.

Sandi Metz(2016) “オブジェクト指向設計実践ガイド~Ruby でわかる進化しつづける柔軟なアプリケーションの育て方”,高山泰基訳,技術評論社.

Sridhar Nerur and VenuGopal Balijepally. 2007. Theoretical reflections on agile development methodologies. Commun. ACM 50, 3 (March 2007), 79–83. https://doi.org/10.1145/1226736.1226739.

Stefik, M., & Bobrow, D. G. (1985). Object-Oriented Programming: Themes and Variations. *AI Magazine*, *6*(4), 40. https://doi.org/10.1609/aimag.v6i4.508.

Oxford learner'sdictionaries"Object" https://www.oxfordlearnersdictionaries.com/definition/american_english/object_1: May 28, 2022 final accessed.