Ⅰ オブジェクト指向の考え方

学習目標
① オブジェクト指向の意味を理解させる。
② オブジェクト指向の考え方が必要とされる理由を理解させる。
③ オブジェクト指向の長所と短所を理解させる。
④ オブジェクト指向の将来について知らせる。

1 オブジェクト指向を理解する心構え

コンピュータ上で、ソフトウェアを開発したことのある技術者であれば、「オブジェクト指向」、あるいは「オブジェクト指向プログラミング（Object Oriented Programming、ＯＯＰと略すこともある。）」という単語を耳にする機会は実に多いと思う。しかし、それがいったい何なのかについては、深く知らないし、知ろうとしない技術者が非常に多い。それは、オブジェクト指向がよくわからないし、そのことを新しく学んでプログラミングするよりは、いままで身につけているＦＯＲＴＲＡＮ、ＣＯＢＯＬ、Ｃという言語を使ってプログラミングした方が楽だと考えるからである。

しかし、ソフトウェアが複雑になればなるほど、オブジェクト指向の必要性は増す一方である。更に、最近はソフトウェアをプログラムするための道具（開発ツール）自身まで、オブジェクト指向のものに置き換えられてきている。そのため、「オブジェクト指向はよくわからない。」では通用しなくなっている。

そこで、今後もソフトウェアを開発していくつもりがあれば、オブジェクト指向という新しい考え方に正面から取り組もうという心構えが必要である。

確かに、オブジェクト指向という考え方は、取っつきにくく敷居が高く感じられるかもしれない。特に、オブジェクト指向を学び始めると次々と登場するクラス、オブジェクト、継承、抽象化、多様性、メッセージといった抽象的な専門用語は、学ぶ者に拒絶反応を起こさせるのに十分な役者揃（ぞろ）いである。

そこで、本書では、オブジェクト指向をなるべく容易に理解できるよう、段階的に、しかも専門用語の羅列にならないように説明していくことにする。

2 オブジェクト指向の意味

ここでは、オブジェクト指向の簡単な歴史と、その意味について説明する。オブジェクト指向の起源は意外と古く、１９６７年に開発されたＳｉｍｕｌａという言語が、最初のオブジェクト指向プログラミング言語といわれている。Ｓｉｍｕｌａは、ノルウェーで開発されたシミュレーションのための言語である。

その後、１９８０年代に、オブジェクト指向設計という考え方がGrady Booch達によって提案され、脚光を浴びるようになった。そして、オブジェクト指向プログラミング言語も多数誕生した。

それでは、オブジェクト指向の意味をなるべく専門用語を使わないで説明する。オブジェクト指向は、大変複雑な作業であるプログラミングを、なるべく簡単に行うための仕掛けで

ある。

これまで、より大きな規模のプログラミングを効率よく行うために、さまざまな技術が開発されてきた。アセンブラが考案され、高級言語が開発され、構造化プログラミング手法が提案された。そして、一人で行っていたプログラミングも、プロジェクトチームを結成し、多人数で分担して行うようになった。それらの技術等により、例えば、数千行といった規模のソフトウェアを開発するような場合には、かなり高い生産性を得ることができるようになった。

ところが、開発すべきソフトウェアの規模は、更に大きくなり続けた。そして、それにつれて、開発するプロジェクトチームもどんどん多人数になっていった。ところが、ある程度大きな規模のソフトウェア開発になると、いくら人数を増やしてもちっとも生産性が上がらなくなってしまったのである。

そこで提案されたのが、オブジェクト指向という考え方である。それは、大規模なソフトウェアを、多人数が分担してプログラミングするときに、特に効果を発揮する。ただし、その考え方を身につけようとすれば、これまでのプログラミングの考え方を捨て、新たな発想をもつ必要がある。これはかなり大変なことである。むしろ、C言語やFORTRAN、COBOLといった手続き型言語の知識や経験が全くなく、生まれて初めてオブジェクト指向の考え方に取り組む人のほうが、抵抗なくオブジェクト指向の考え方を受け入れられるかもしれない。

しかし、発想の転換ができるか否かが、とにかくオブジェクト指向という考え方の導入に成功するか、失敗するかの分岐点であるので、なんとか新しい考え方に慣れてほしい。

まず最初に、オブジェクト指向が持っている基本的な仕掛けを説明する。それは、以下のようなものである。

第1は、ある人が作った部分を他人に使わせる場合、具体的なプログラミングの詳細は教えず、使い方だけを教えるという仕掛けである。他の人は、ある人が作った部分の詳細を知らずに済むので、プログラミングが簡単になるわけである。

第2は、ある人が作った部分はなるべく流用し、もう一度作り直すことはしないという仕掛けである。いままで似たような処理を何度もプログラミングすることが行われてきたが、このような必要がなくなりプログラミングが簡単になるわけである。

第3は、自分で何でもやってしまうのではなく、他の人にお願いをして処理をしてもらうという仕掛けである。自分で何でもやっていた従来のプログラミングよりも、すっきりした処理ができる。

第4は、同じ名前をつけた部分がプログラムの中にいくつもあってもよく、しかも、それぞれ違う動作をするという仕掛けである。これにより、例えば、よく似ているが実は異なる動作をする部分にいちいち違う名前を割り当てて区別する必要がなくなり、プログラミングが簡単になるわけである。

そして、オブジェクト指向では、これらの仕掛けを容易に実現するために、新しい考え方を取り入れた。それは、「処理」中心のプログラミングから脱却し、プログラミングの対象となる世界を「もの」中心に分類し見直してみるという考え方である。オブジェクト指向の「オブジェクト」という用語は、この「もの」という単語の英語に由来する。

このような仕掛けを反映して、さまざまなオブジェクト指向プログラミング言語が開発された。

そして、もう一つ忘れてならないのが、必ず先に設計を行ってからプログラミングを行うという手順である。小さな規模のソフトウェアを開発する場合、いきなりパソコンの前に座り、プログラミングを始めるというスタイルで開発をしている人が多いと思うが、これは、オブジェクト指向の世界では通用しない。そこでは、まず問題を分析し設計を行った後に、初めてプログラミングを始めるというスタイルが要求される。このような手順は、最初のうちはまどろっこしく感じられるかもしれない。しかし、オブジェクト指向は、これまでのソフトウェア開発の考え方に比べ、より完成度の高いソフトウェアの開発に適している。これは、これまでの家内工業的なソフトウェア開発から脱皮する絶好の機会となることであろう。

3 オブジェクト指向の長所と短所

ここでは、オブジェクト指向の長所と短所について説明する。これまで説明してきたとおり、オブジェクト指向は、大規模なソフトウェアを開発するときに、そのプログラミングの複雑さを軽減してくれる。そして、それは大規模なソフトウェアにおける生産性と保守性を向上させることを意味している。これが、オブジェクト指向の考え方が持つ長所である。

一方、オブジェクト指向によるソフトウェア開発においては、複雑さを軽減するためのさまざまな仕掛けのために、コード効率の低下が発生したり、実行速度が多少遅くなる傾向がある。これが、オブジェクト指向の考え方が持つ短所である。

もちろん、数千行の規模のソフトウェアを一人で開発する場合には、何もオブジェクト指向の考え方を採用する必要はないかもしれない。勝手知ったるＣ言語で開発しても、それほど問題は発生しない。それは、規模が小さいので、一人の人間が全体を完全に把握することができるからである。

しかし、大規模なソフトウェアで、コード効率よりも保守性が重んじられるような場合、Ｃ言語では限界がある。もちろん、Ｃ言語で開発できないことはない。しかし、生産性と保守性の向上は見込めない。

また、複雑さを軽減してくれるという意味では、オブジェクト指向は、小さな規模のソフトウェアの開発をするときにも効果がある場合がある。それは、ＰＣ９８とＤＯＳ／Ｖといった機種の異なるパソコンや、ＷｉｎｄｏｗｓとＵＮＩＸといった異なるＯＳの上で動作するソフトウェアを作成する必要がある場合である。オブジェクト指向の考え方を適用し、機種に依存する部分や、ＯＳに依存する部分を外側から隠してしまうことができる。これによって、移植性の高いソフトウェアを実現できるという長所がある。

4 オブジェクト指向の将来

ここでは、オブジェクト指向の将来について説明する。日本において、１９９６年は、Ｗｉｎｄｏｗｓ９５というＯＳが急速に普及した年であった。Ｗｉｎｄｏｗｓ９５にははじめからネットワークを構築するための機能が含まれている。そのため、それまで単体で使用されることの多かったパソコンをＬＡＮ（Local Area Network）で接続して使用する機会が急激に増加した。また、インターネットの普及により、世界中のコンピュータが接続され、私たちはネットワーク環境の便利さを実感できるようになった。

そこで注目されているのが、分散オブジェクトと呼ばれる技術である。例えば、ドキュメント（文書）のやりとりを考えてみる。従来、複数のソフトウェアの間でドキュメントをや

りとりするには、「カット・アンド・ペースト」流の複合ドキュメント技術が使用されてきた。しかし、ドキュメントのやりとりができるのは、ほとんどの場合、単体のパソコン内のソフトウェア間に限定されていた。

ところが、ネットワーク環境の普及により、ドキュメントのやりとりは、ネットワーク経由で接続された、複数のパソコン上のソフトウェアの間でも行えるようになった。これを実現するために開発されたのが、ＯｐｅｎＤＯＣやＡｃｔｉｖｅＸといった、複合ドキュメントのための規格である。そして、それらを支えているのが、ＣＯＲＢＡやＤＣＯＭといった分散オブジェクトと呼ばれるインフラストラクチャ（基盤技術）である。

将来、ネットワークの重要性はますます増加すると考えられる。無論、分散オブジェクトは、複合ドキュメントのためだけでなく、さまざまな技術の基盤にもなっている。このような点から、分散オブジェクトについての知識を深めておくことは、将来必ず役に立つものと考えられる。

本書では、Ｖ章において分散オブジェクトを説明している。