オブジェクト指向

絶版に伴い、校正前の原稿テキストを公開したものです。基本的に原稿そのままをHTML形式に変換したものですので、誤字/脱字、説明不足の箇所もあるかも知れませんがご了承ください。初出:「PHPによるデザインパターン入門」(下岡秀幸/畑勝也/道端良 著, 秀和システム, ISBN4-7980-1516-4, 2006年11月23日発売)

「オブジェクト指向」と聞くと、最初はどうしても「何やら難しそう」といったイメージがありますね。

オブジェクト指向は、大雑把に言ってしまうと「操作手順よりも操作対象に注目し、オブジェクトという単位ですべてをとらえる」という「考え方」です。つまり、「どの様に操作するか」や「どうやって動作させるか」ということではなく「何を操作するか」に注目しているのです。

たとえば、テレビやビデオデッキなどの電化製品を考えてみましょう。電源を入れると映像が映ったり、用意されたスイッチを押すことで様々な動作をしますね。この場合、「テレビ」や「ビデオデッキ」といったオブジェクトがあり、「電源を入れる」「再生スイッチを押す」「チャンネルを変える」という操作をおこなっている、と捉えることができます。もちろん、操作をおこなっている「あなた」もオブジェクトとして捉えられます。

つまり、あなたがテレビを見たい場合は次のようになります。

> 「あなた」というオブジェクトが「テレビ」というオブジェクトに対して「電源を入れる」という操作をおこなう

非常に簡単ですね。もし、違う番組を見たいと思えば、「チャンネルを変える」という操作をおこなうだけで、期待する結果を得ることができます。このように、「オブジェクト指向」という考え方は、「対象のモノを操作する」という私たちが普段ごく自然に考えている考え方と非常に似ています。そして、「オブジェクトどうしがやりとりをおこなう」

では、プログラミングの世界にオブジェクト指向を適用した場合はどうでしょうか?

現実世界ではオブジェクトに相当する「モノ」がすでに存在していますが、プログラミングをおこなう場合、まずはどのような「モノ」があるかを特定する必要があります。そして、期待する動作をするよう、特定されたオブジェクトどうしをどうやりとりさせるかを定義します。この一連の流れはオブジェクト指向設計と呼ばれます。プログラム開発には再利用性、拡張性、保守性といった求められるものが多くありますが、オブジェクト指向設計の大きな目標はこれらを実現することにあります。

しかし、これらを実現しようとするには、「オブジェクト指向」という考え方を理解する必要があります。オブジェクト指向には小難しいキーワードがたくさん出てきますが、いきなり丸ごと理解しようとするのはやはり無理があります。

しかし、オブジェクト指向の中でも必ずおさえておくべきポイントとなるキーワードがあります。次に挙げる「カプセル化」「継承」「ポリモフィズム」の3つです。

カプセル化(encapsulation:情報の隠蔽)

先ほど「オブジェクト指向設計ではオブジェクトどうしをどうやりとりさせるかを定義する」と説明しました。それでは、この「オブジェクト」とはどの様なものでしょうか?

先ほどのテレビの例では、テレビに対して様々な操作をおこなうことができました。しかし、それぞれの操作では「内部でどの様に動いているのか」は意識していなかったはずです。「テレビ」というオブジェクトの外から見ると、テレビに対して何か操作をおこなうと内部で何やら動作がおこなわれ、期待した結果が返された、というだけです。

オブジェクト指向では「操作対象」に注目しており、「操作手順」やその操作に伴うデータはオブジェクトの中に隠されます。これを「カプセル化」といいます。

テレビの場合、操作に伴うデータとは、電源が入っているかどうかや選択されているチャンネルの情報になります。つまり、それらの情報は「テレビが知っている」、または「テレビ以外は知らなくて良い」とも言えるでしょう。また、そのデータにアクセスできるのは、用意された「操作」だけというところがポイントです。

オブジェクト指向設計では、「どのオブジェクトが何を知っているか」が重要になります。適切にカプセル化されたオブジェクトは、他のオブジェクトに影響することなく修正したり、他のシステムで再利用可能になります。

継承(inheritance)

継承とは、「他のクラスの特徴を引き継ぎ、新しい特徴を加えたり変更したりすること」です。

ここで「クラス」という単語が出てきましたので、簡単に説明しましょう。クラスとは、操作と操作に関連するデータをまとめたオブジェクトの雛形となるものです。裏返すと、この雛形(クラス)に具体的なデータを入れたものがオブジェクトとなります。

再度ここでテレビの場合を考えてみましょう。「受信した映像を映す」「チャンネルを変更できる」といった共通の機能を持ってはいますが、すべてのテレビで同じとは限りません。衛星放送や地上波デジタル放送を受信できたり、色々な付加機能を持ったものがあります。つまり、様々な雛形があり得るわけです。しかし、これら1つ1つについて雛形を作っていては、非常に非効率ですね。

そこで、共通の機能をもつ雛形を定義し、その機能を引き継いで色々な付加機能を持つテレビの雛形を作れるとなると、雛形の数をかなり抑えることができます。これが継承の考え方です。

この時、元になるクラスを「スーパークラス」「基底クラス」「親クラス」などと呼び、継承して新しく定義されたクラスを「サブクラス」「派生クラス」「子クラス」と呼びます。また、「親クラス」「子クラス」の表現のように、雛形どうしが非常に強い結びつきを持つようになります。

ポリモーフィズム（**polymorphism**：多態性）

ポリモーフィズム（多態性）とはちょっと聞き慣れない言葉ですね。しかし、オブジェクト指向設計や本書で見ていくデザインパターンでは、このポリモーフィズムは多用されていますので、どうしても理解しておきたいキーワードです。

またもやテレビの例を考えてみましょう。

テレビは電化製品です。当然ですね。電化製品には電源を入れるためのスイッチがあります*{ない製品もありますが、ここでは考えないことにします}。このスイッチを入れるとどうなるでしょうか？大雑把に言うと、テレビの場合映像が映るでしょうし、ビデオデッキの場合予約の受付画面が表示されるかも知れません。パソコンの場合、BIOSの初期化が始まりやがてOSが起動するでしょう。

このように「電源を入れる」という同じ操作にもかかわらず、実際の動作はそれぞれの電化製品で異なります。つまり、「電化製品の電源を入れる」という操作の具体的な内容は、「実際にどういう電化製品なのか」によって決まるということです。逆に、具体的に何かを知らなくとも操作することができる、とも言えます。

ポリモーフィズムとは、「あるオブジェクトへの操作が、呼び出し側ではなく受け手のオブジェクトによって定まる特性」と言えます。現実世界の人間や物でも、同じ操作をおこなっても具体的な人や物よって違った反応をすることが多いと思いますが、これをうまくソフトウェアに当てはめたものとなっています。

デザインパターンは、このオブジェクト指向の3大要素を効果的に活用したものとなっています。