オブジェクト指向と
ゲームプログラミング

○　Java 基礎編 － 第１回　オブジェクト指向プログラミング　○

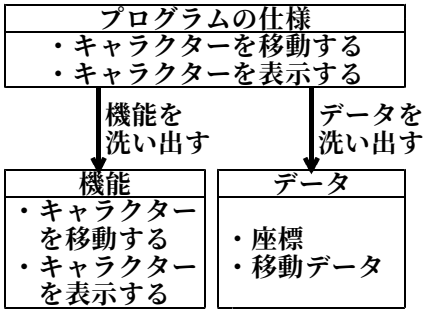
構造化プログラミングと問題点

構造化プログラミングとは、プログラムの構造を機能単位に細分化するというものです。構造化プログラミングのための設計のことを構造化設計と呼びます。構造化設計では、プログラムの仕様の中に、どのような機能があるかを洗い出します。すなわち、プログラム全体を機能の集まりであると考えるのです。

たとえば、以下の仕様の「キャラクター移動プログラム」を作成するとしましょう。構造化設計では、プログラムの仕様の中から「キャラクターを移動する」「キャラクターを表示する」という機能を洗い出します。

『キャラクター移動プログラム』
・キャラクターを移動する
・キャラクターを表示する

どのようなプログラムであっても、その構成要素は、機能と機能の対象となるデータに大別されます。これは、構造化プログラミングでも後述のオブジェクト指向プログラミングでも同じです。構造化設計では、機能の洗い出しが終わったら、引き続きデータの洗い出しを行います。キャラクター移動プログラムでは、「キャラクターの座標」および「移動データ」というデータが洗い出されます。



構造化設計は、一見して何ら問題がないように思えます。それは、プログラムの規模が小さいからです。大規模なプログラムでは、プログラムの仕様から洗い出した機能とデータが、それぞれ数百個～数千個にもなります。その機能とデータは、別々に記述されています。機能とデータが別々に管理され、結果的にどのデータがプログラムのどの部分と関係しているかということが見えにくくなってしまいます。これは、後からプログラムを修正したり改良することが困難になるということを意味しています。これが、構造化プログラミングの問題点です。

たとえば、プログラム全体で参照できるグローバルなデータを宣言すると、そのデータがどの部分で利用され、いつ変更されるかということがわかりづらくなります。これは、プログラムを再利用したり修正したりする上で大きな問題となります。グローバル変数を経由してデータをやりとりするような関数は、引数の受け渡しがなく、再利用しづらいものとなってしまいます。

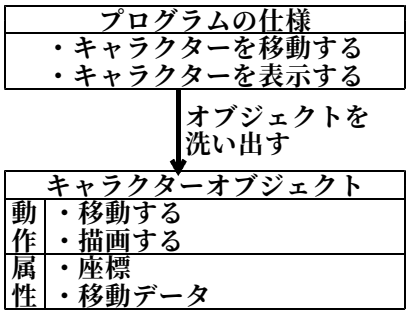
オブジェクト指向プログラミング

構造化プログラミングの問題点を解決する技法として、オブジェクト指向プログラミング(OOP:Object Oriented Programing)が生まれました。オブジェクト指向プログラミングの目的は、「プログラムの再利用性を高め、効率よく作成すること」にあります。オブジェクト指向プログラミングの手段は、「プログラムの構造を単純化すること」と「既存のプログラムを再利用すること」です。これにより、大規模なプログラムであっても、生産性が向上し、後から修正や改良することも容易になります。

オブジェクト指向プログラミングのための設計のことをオブジェクト指向設計と呼びます。オブジェクト指向設計では、プログラム全体を「もの」の集合体であると考えます。プログラムの中に、どのような「もの」すなわちオブジェクトがあるかを洗い出します。設計の第一歩で注目するものがオブジェクトであるところが、機能に注目していた構造化設計との大きな違いです。

構造化設計の例で示したのと同じ仕様の「キャラクター移動プログラム」をオブジェクト指向プログラミングで作成するとしましょう。オブジェクト指向設計では、プログラムの中から「キャラクター」というオブジェクトを洗い出します。これは、簡単な作業です。仕様を示した文章の中にある名詞を洗い出せば、それがオブジェクトとなるからです。

オブジェクトを洗い出したら、それぞれのオブジェクトの持つ動作と属性を洗い出します。構造化設計における機能とデータは、オブジェクト指向設計における動作と属性に相当します。オブジェクトが持つ機能とデータなので「動作」と「属性」と呼びます。
キャラクターには、「移動する」「表示する」という動作があり、「座標」「移動データ」という属性があります。



最終的には、プログラムの仕様の中から機能(動作)とデータ(属性)を洗い出しているという点では、構造化設計とオブジェクト指向設計に違いはありません。これは、あらゆるプログラムが機能とデータから構成されていることから当然のことです。ただし、オブジェクト指向設計では、まずはじめにオブジェクトを洗い出すことで、プログラムの構成要素を分類しています。オブジェクトを洗い出した後で、個々のオブジェクトが持つ動作と属性を洗い出すので、必然的にプログラムの持つ機能とデータが整理されます。このような設計に基づいてプログラミングを行えば、とても整理されたものになるというわけです。関係する機能とデータが、オブジェクトによって１つにまとめられているので、構造化プログラミングの問題点の１つである「データがどこでどのように処理されているかわかりづらい」ということを解決しています。そのため、後からプログラムを修正や改良することも容易です。これが、オブジェクト指向のメリットです。

オブジェクト指向プログラミングの概念

オブジェクト指向プログラミングには、以下の重要な概念があります。

- 抽象化
- カプセル化
- 継承
- ポリモーフィズム

これらは相互に関連し、全体でオブジェクト指向を構成しています。

・抽象化

抽象化とは、そのオブジェクトが具体的に何であるかや、どういう内部構造になっているかを気にしなくてもいいということです。オブジェクトを利用する側からは、どのようなデータを保持しているかと、どのような動作ができるのかさえわかればいいのです。これは通常、オブジェクトをクラス(class：分類、部類)にまとめることによって実現されます。詳細を考えることを省略でき、複雑さの削減をもたらします。

・カプセル化

カプセル化とは、オブジェクトを利用する側からオブジェクトの属性(変数)を直接読み書きすることを禁止し、オブジェクトの動作(メソッド)で間接的に属性を読み書きするように制限することです。オブジェクトの内部を隠すことにより、外部からの勝手な使用や不正な変更を防ぐことができます。これにより、不適切な値の代入によるオブジェクトの破壊を防ぐだけでなく、独立性を高くすることができます。

・継承

継承とは、すでにあるクラスには直接さわらずに、そのクラスを元に新しい拡張クラスを作成できる仕組みです。これにより、オブジェクトを階層的に組織することができます。たとえば、「犬」と「猫」というクラスがあったとします。これらはどちらも動物であり、「食べる」や「寝る」など、どんな動物にも共通する動作を持っています。そこで、これらの上位のクラスとして「動物」クラスを作り、犬や猫クラスはそこから「継承」させることにします。こうすることによって、共通する動作を１つにでき、あとは「顔を洗う」「吠える」などの、下位のクラスで特有のことだけを記述すればよいのです。これによりコード記述は大幅に節約できます。

・ポリモーフィズム
　ポリモーフィズム(polymorphism)とは、同じ名前を持った動作(同じ名前の関数)が状況に合わせて異なる動作をすることです。「犬」と「猫」の「鳴く」という動作を考えてみましょう。犬は「ワン」と鳴き、猫は「ニャー」と鳴きます。「鳴く」という動作は同じでも、実際の行動は異なります。このことをポリモーフィズムといいます。

　オブジェクト指向プログラミングに用いるのがオブジェクト指向型言語です。オブジェクト指向型言語は、４つの概念をすべてサポートしているプログラミング言語です。JavaやC++などが該当します。抽象化、カプセル化によって独立性が向上し、継承、ポリモーフィズムによって拡張しやすくなるので、プログラムの再利用を促進します。その結果として高い生産性をもたらします。

練習問題

１　以下の文章の内容が正しい場合には○、間違っている場合には×を付けましょう。

(１)構造化プログラミングは、古く、デメリットが多いので採用してはいけない。
(２)オブジェクト指向プログラミングの目的は、プログラムの再利用である。
(３)オブジェクト指向プログラミングは、小規模なプログラムの開発に使っても意味がない。
(４)オブジェクトの持つ機能を動作と呼び、データを属性と呼ぶ。
(５)Javaは、オブジェクト指向プログラミング言語である。
(６)構造化プログラミングを採用した場合とオブジェクト指向プログラミングを採用した場合では、同じ仕様のプログラムでも、完成したプログラムの機能や実際の動作が異なる。

２　以下の用語について簡潔に説明しましょう。

(１)構造化設計

(２)オブジェクト指向設計

(３)継承

(４)ポリモーフィズム

３　カプセル化のメリットを２つ挙げましょう。