Ⅵ 命の長いソフトウェアのために

学習目標
① クラスライブラリの役割を理解させる。
② 命名規則の役割を理解させる。
③ 有力なオブジェクト指向の将来を概観する。

1 クラスライブラリ

クラスライブラリは、既に説明したようにコードを共有化し、再利用可能とすることによって、ソフトウェアの生産性、保守性を向上させるものである。複数のクラスで共通のコード部分があれば、その部分を親クラスにすることにより、その部分が再利用可能なソフトウェア部品となる。ソフトウェア部品が集まりクラスライブラリの形で提供されるようになると、利用者の負担が大きく減ることになる。すなわち、システムでまったく新たに開発する必要があるところだけを開発すれば済むようになる。

また、ライブラリは、プログラムが複雑になればなるほどその重要性は増加する。それはプログラムが複雑になると、最初からプログラムのコードを作り直していたのでは手間がかかりすぎて実際的ではなくなるからである。

オブジェクト指向は、情報の管理をきっちりと行えるような仕掛けが多く用意されている。そのため、特にライブラリを効率的に構築できる。

しかし、喜ばしいことばかりではない。実は、複数あるクラスライブラリの一部しか、標準として残らないということである。例えば、マイクロソフト社のＣ＋＋言語コンパイラに付属してくるクラスライブラリが標準になったとしたら、他のクラスライブラリを使用していた人は、最後には標準であるクラスライブラリに乗り換えなければならない。クラスライブラリは、ソフトウェアの開発に枠をはめてしまうほど強力であり、乗り換えは大変困難である。これらのことを考えると、クラスライブラリというものは、実はわれわれは利用するだけの立場であり、提供する立場にはないことがわかる。われわれは、従来のソフトウェア開発技法で経験してきたような、自身が開発者という立場ではなく、利用者の立場になるということを深く認識せねばなるまい。そして、どのクラスライブラリが最後に標準になるかということを注意深く見守っていく必要がある。われわれが書くオブジェクト指向のプログラムが将来生き残れるか否かは、依存するクラスライブラリにかかっているのである。寿命の長いソフトウェアを作成しようとすれば、最後に標準になるクラスライブラリを選択しなければならない。

2 命名規則

クラスや変数の命名規則は、特に厳密な規定があるわけではない。しかし、大人数でソフトウェアを開発する場合は、「コーディング規約」を作成するので、そこで明確に命名規則についても記述しておかなければならない。

例えば、Ｗｉｎｄｏｗｓシステムをマイクロソフト社で開発するときには、図Ⅵ－１のような変数命名規則が採用された。

b	ブール（１バイト）
c	文字（１バイト）
dw	unsigned long の整数
f	１６ビット
h	ハンドル
l	long の整数
sz	NULLで終わる文字列へのポインタ
	：

図Ⅵ－１　Ｗｉｎｄｏｗｓシステム開発時の変数命名規則

ハンガリアン記法と呼ばれるこの記法は、変数の前にデータ型を示す小文字をつけるという単純なルールである。例えば、cTextという変数は、その変数名を見ただけで、文字（１バイト）型の変数であるということが一目で分かる。また、大文字と小文字を混在させ意味のある変数名をつけることによって、プログラムの読みやすさを向上させている。例えば、AXBUYという変数名では、いったい何を意味する変数か理解することが難しい。しかし、cFamilyNameならば、ファミリーネームを意味する文字（１バイト）型の変数ではないかと想像がつく。

このように、命名規則をあらかじめ決めておくことは大変重要である。そして、将来、ソフトウェアの部品化を考えているのであれば、なお一層クラスや変数の命名規則には注意を払うべきである。

3 オブジェクト指向言語

これまで、多くのオブジェクト指向言語が登場したが、大きな支持を得た言語は数えるほどしかない。それが、Ｃ＋＋言語であり、Ｊａｖａ言語である。従って、ソフトウェアの部品化を考えているのであれば、現在有力な言語か否か、将来も生き残れそうな言語か否かという問いかけを絶えず行う必要がある。

ここで、現在有力とされている、Ｃ＋＋とＪａｖａを比較してみる。

まず気がつくことは、開発されてからの年月の違いである。１９８６年に発表されたＣ＋＋に対して、Ｊａｖａが発表されたのは１９９５年のことであり、その差は１０年もある。進歩の異常に速いコンピュータの世界において、１０年という年月は極めて大きい。

そして、文法の違いである。ＪａｖａはＣ＋＋などの先輩格のオブジェクト指向言語を参考に開発された。そのため、Ｊａｖａは、ＧＯＴＯ文の排除、不要メモリの自動回収機能、単一継承のみのサポート、例外処理の強化、ポインタ機能の削除など、機能面ではＣ＋＋に比べかなり簡素化されている。

また、Ｊａｖａで書かれたソースプログラムは、バイトコードと呼ばれる仮想ＣＰＵ（Java Virtual Machine）用のコードに変換され、それをさまざまなプラットホーム用仮想ＣＰＵが解釈（インタープリット）して実行する。そのため、コードレベルのプラットホーム独立性を確保している。すなわち、Ｗｉｎｄｏｗｓ環境でもＭａｃｉｎｔｏｓｈ環境でもＵＮＩ

X環境でも、同じプログラムが実行できるのである。ただし、Ｊａｖａをそのまま実行した場合、インタープリタ方式の宿命として、Ｃ＋＋に比べて格段に遅い。そのため、ＪＩＴ（Just In Time）コンパイラを搭載した環境も増加している。ＪＩＴは、Ｊａｖａプログラム実行時に、その仮想ＣＰＵ用の機械語に翻訳して実行するというものである。これによって、インタープリタ方式のときに比べ、実行速度がかなり向上する。

しかし、最も違うのは、Ｃ＋＋よりもＪａｖａの方が、より純粋なオブジェクト指向言語であるという点である。それでは、純粋なオブジェクト指向言語の方が純粋でない言語より優れているかというと、そうは断言できないのである。オブジェクト指向言語のこれまでの歴史の中で、ＥｉｆｆｅｌやＳｍａｌｌｔａｌｋに代表される純粋なオブジェクト指向言語も数多く誕生したが、結局、純粋とはいえないＣ＋＋がオブジェクト指向言語として生き残ってきている。

日本では、Ｊａｖａブームが到来し、Ｊａｖａを使えば、初心者でも簡単にプログラムが作成できるといったばら色の考えが広まった。しかし、それは誤解である。オブジェクト指向という考え方は、本書で説明したように、それほど容易に理解できるものではない。しかも、純粋なオブジェクト指向言語であればあるほど、オブジェクト指向の枠内で思考しなければならず、学習曲線は緩やかにならざるをえない。

筆者は、Ｃ＋＋もＪａｖａも将来ある程度生き残るが、再度技術的なブレークスルーが実現されないと、それ以上の発展は望めないのではないかと考える。そして、現在よりもプログラミング環境が進化し、オブジェクト指向を応用したソフトウェアの部品化が更に進んだ場合に、はじめて初心者でも簡単にプログラムが作れるという世界がやってくると考える。

もちろん、将来何が起こるか誰にもわからない。しかし、Ｃ＋＋とＪａｖａを比較して、どちらがより長く生き残るかを予想するとしたら、Ｃ＋＋を挙げておきたい。もちろん、将来プログラミング環境が現在よりもずっと充実すると仮定しての話である。