特　集

3 VB.NETでラクをする頭の使い方

オブジェクト指向なんて知らないよ

初音 玲
HATSUNE,Akira

ラクを追求したらオブジェクト指向してた!?

Level

1 2 **3** 4 5

Technology Tools

- ☑ Visual Basic
- ☐ Visual C#
- ☐ Visual C++
- ☐ SQL Server
- ☐ Oracle
- ☐ Access 2002
- ☐ ASP.NET
- ☐ Other:

Samples

・この記事で取り上げたソースコードおよびサンプルプログラムは、http://www.shoeisha.com/mag/windev/からダウンロード可能です。

Windowsアプリケーションにおける開発ツールのデファクトスタンダードが、Visual BasicからVisual Basic.NETやVisual C#.NETに移行しつつある。もちろん、先進的な企業やVS.NETに対応したツールなどを作っているところは移行も完了しているのだろうが、一般的な企業が自社業務アプリケーションを構築する場合の選択としては、やっと始まったばかりというのが現状なのではないだろうか。

主力開発ツールの変更は、それなりに労力が必要なため、様子見の会社もあるだろう。また、開発者も「VB.NETはオブジェクト指向言語だ」というメッセージによって、「いまさらオブジェクト指向言語の勉強は……」と躊躇していたり、いざ、勉強を始めても挫折してしまったりする場合も多いのだろう。

しかし、本当にVB.NETはオブジェクト指向を知らなくては習得できない言語なのだろうか。また、オブジェクト指向を知らないとチーム開発が効率的にできないのだろうか。

実際、ある程度の規模のシステム開発にVB.NETを使ってみると、Visual Basicとの差異が気になったり、その差異に戸惑ったりすることはあるものの、“オブジェクト指向”をあまり意識しなくても十分効率的な開発ができる。

そこで、今回は簡単なサンプルを通して、オブジェクト指向を意識せずに効率的な開発を行なう過程を紹介しよう。

なにはなくとも構造化プログラミング

“構造化プログラミング”は、1960年代中ごろから1970年ごろにエドガー・ダイクストラ（Edsger Wybe Dijkstra）たちにより提唱された方法論で、大規模なプログラムを設計ミスなく効率よく記述することを目的としている。少しでもプログラミングに関連した知識を身につけていれば、一度は聞いたことのある用語だろう。

まずは、この30年以上も前に提唱された方法論のキーポイントをおさらいしておこう。

“設計にミスがなく効率よく記述さ

れたプログラム”とは、どのようなものだろうか。そこが明確になれば、そこに近づくための方法論も見えてくる。構造化プログラミングでは、このようなプログラムは「プログラム全体が理解しやすい」プログラムだと位置づけている。

“理解しやすかったら設計ミスもない”というのは、なんだか当たり前のようだが、そのことを証明するために、ダイクストラは、

①入口がひとつ、出口がひとつの構造を持った部品を組み合わせてプログラムを記述
②部品自体は、正しく動作していることが前提
③組み合わせる方法は、「順構造」「選択構造」「反復構造」の3つのみ

という「構造化定理」を考えつき、この定理に沿ったプログラムであれば「プログラム全体が理解しやすい」構造であることを証明した[注1]。

部品を組み合わせる

“部品を組み合わせて”などと書くと何か高度なことをやるように思えるが、たとえば、

```
intA = 1
```

のようなコードの「＝」も部品となる。この「＝」は、右側の値（入口）を左側の変数（出口）に設定する機能を持った部品なのだ。

つまり、部品といっても何も特別なものではなく、プログラム言語に用意されている関数や、.NET Frameworkのクラスライブラリなどもあわせて部品と考える。もちろん自分で作った関数やプロシージャも部品だ。

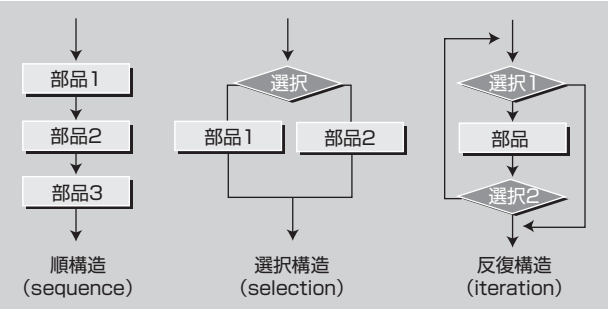
正しく動作していることが前提

先ほどの例のように

```
intA = 1
```

により変数intAに正しく「1」が設定されるかを確認していたらプログラムの理解は進まないだろう。よって、部品が正しく動くかどうかは「使うときではなく作るとき」に確認しなさいというのが構造化プログラミングだ。

図1：処理構造



部品の組み合せは3種類

「部品」というもののイメージは掴めただろうか。

最後のポイントは、「この部品をどのように配置してプログラムを記述するか」なのだが、構造化プログラミングでは、図1にあるように3つの構造だけで、すべての論理が記述できるとしている。

順構造

順構造は、最も基本的なプログラム構造だ。図1にあるように、上から下へプログラムが記述され、実行時も上から下へ処理が行なわれてゆく。

よって、順構造で記述されたプログラムは、先頭の部品[注2]から順番に確認していくことで理解できる構造になっている。

選択構造

選択構造は、実行する部品を選択する構造だ。

たとえば、変数intAの値が「1」の時には部品1を実行し、「1以外」のときは部品2を実行したい場合、

```
If intA = 1 Then
  (部品1)
Else
  (部品2)
End If
```

のようにプログラムを記述する。

注1）③の構造とともに「Goto文は有害である（Go to statement considered harmful）」という有名な言葉も残している。
注2）部品も、「順構造」「選択構造」「反復構造」のいずれかの構造になっている点を覚えておいて欲しい。