Mac OS X上でのJava言語の教育について

千葉商科大学政策情報学部
箕原辰夫
minohara@cuc.ac.jp

慶應義塾大学湘南藤沢キャンパス（SFC）において、Mac OS X上のJava開発環境を用いて、プログラミングの授業を行なった。Mac OS X 10.2上のProject Builderを主に用いた。ここでは、この授業で分かった問題点と、成果、およびMac OS X上でプログラミング教育をすることの優位性について報告する。

１．プログラミング教育の目的

そもそもプログラミング教育は、問題解決の手段としてコンピュータを用いるときの強力な記述手段あるいは解決手段を学ぶためにあるのであり、ウィンドウプログラミングや開発環境の使い方について覚えるためにあるのではない。そのような意味から、プログラミングで用いる変数や制御構文や、サブルーチン（Javaではメソッド呼出し）をうまく使うことができるか、あるいはデータを配列などの形にまとめて、効率良く検索することができるかなどがその教育の具体的な目的として挙げられるべきである。Mac OS XのJavaの開発環境は、そのような側面において、開発環境そのものが、多機能でありながらもプログラミングの教育目的をあまり邪魔することがないように設計されている。

２．開発環境について

長年プログラミングの教育をしていて感じることは、開発環境の善し悪しで、学生の習熟度が変わってくるということである。開発環境は、Simple is best.という言葉で表わされるように、簡単な方が良い。これは、コマンドラインツールが良いということではまったくない。エディタで記述してから、コンパイル、リンク、実行までの流れが簡単であるということ意味している。また、隠れた要因としては、メニューやメッセージが英語であることはデメリットになる。言語を英語で書くのが、プログラミングしている感触を促す反面、そのサポートをする開発環境は母国語でエラーを出したり、注意を促して欲しいというのが実感のようだ。そこまで英語だと、学生はメッセージを読まなくなる。以下にプログラミングの教育観点からMac OS X上で動くいくつかのJavaの開発環境に若干の批判を加える。

・JDK…これは、統合的開発環境（IDE）から起動されるべきもので前面に出てくる必要はない。コマンドラインベースで何かをやらないという否定からMacOSの歴史が始まっている。学生には、簡単な紹介しかしていない。

・Project Builder…Mac OS X 10.2までの主力開発環境であったが、10.3でも稼働する。プログラミング教育という面からすれば、１つのウィンドウの中で収まる（学生にそのように初期設定させる必要があるが）ので、非常に使い勝手が良かった。Javaの開発に関してもほとんど問題がなかった。Xcodeでは、なくなったが、JDKのコンパイラからの出力も見ることができて、Project Builderのメッセージに信頼が持てないときに、コンパイラの方を見てバグを探すことも可能であった。簡単なプログラムでも、１つのプロジェクトが1MBの大きさにもなり、クラスのインデックス作成に時間が取られるのは堪え難い部分もある。

・Xcode…日本語環境ではまだ安定していない。プロジェクトを新規に作ったときに、プログラム中のプロジェクト名の作成に問題がある。これからのバージョンに期待したいところだが、その期待には、Project Builderのように、なるべく多くのウィンドウを開かずに、プログラムが実行できるような機能も含まれる。

・CodeWarrior…MacOSでの標準的な開発環境であったが、商用で購入費用が掛かる以外は、最高の統合型開発環境である。実際に、どのように組み合わせて、コンパイラを起動させたり、プログラムを実行させているのかが直感的にわかり、扱いやすい。プロジェクトファイルの大きさもそれほど大きくならず、使い始めの学生でも、すぐに慣れることができる。

・JJEdit…Jake MacMullin氏が作ったJavaアプリケーション開発用の簡易統合環境なのだが、AppleScriptのスクリプトエディタのように、保存→コンパイル→実行のボタンが並び、それを順に押していくだけでアプリケーションが実行できてしまうのが良い。プロジェクト形式ではないので、ProjectBuilderやXcodeのように、大きなサイズのプロジェクトファイルを作らなくても済む。惜しむらくは、アプレット用の環境がないことである。HTMLも自動的に作ってくれて同じように動くのであれば、授業ではProject Builderは利用しなかっただろう。

・Eclispe…インストールも面倒で最悪に近い。また英語のメニューやメッセージなので、学生には取りつきにくい。フリーソフトであること以外に、メリットは感じられ

ない。UMLをJavaベースで学ぶツールと考えた方が良いであろう。特に、昔のemacsの拡張用にLispを記述するのと似ており、EclispeのUMLベースのプラグイン拡張のためにJavaで記述するのか、Javaを開発することに主眼があるのかわからない節がある。UMLなどの部分が災いして実行速度も遅い。これはWindowsベースで、商用のCodeWarriorが買えないときに使う代替ソフトの価値しか考えられない。Mac OS Xでは必要ない。

・Netbeans…Eclispeよりはましであるが、Javaベースなのでまだまだ速度が遅い感じがする。英語のメニューのままというのも、学生向きではない。

授業では、UNIXのファイルサーバに個人環境が置かれていたが、このファイルサーバには泣かされた。Project Builderで、保存しても、新たに変更保存されたことが認識されず、何回か、「ビルド＆実行」ボタンを押しても再コンパイルされず、いらいらさせられた。これは、湘南藤沢キャンパス特有の現象であると思われる。他のどのマシンでも、ローカルに動かしているProject Builderでこのような症状は出なかった。

３．ライブラリや言語仕様の選定

ここでは、実際に行なった教育内容から、いくつかの点を取り上げて、どのような考えで取捨選択したのかを報告していく。

・Swing vs AWT
この授業では、プログラミングの初学者ということもあり、AWTのライブラリを使うことにしている。ただし、Mac OS XのJDKの1.3.1および1.4.1上では、イベントハンドリングをしたときに、repaint( )が効かない不具合が発見されたので、しょうがなく、今回は途中から最低限のSwingのライブラリを導入せざるを得なかった。この不具合は、1.4.2updateでは、解決されているので、次回の授業では、教室の開発環境が、1.4.2updateにすることができるのであれば、再びAWTに戻す予定である。

Swingを何故嫌うのかは、余計な記述をいろいろしなければならないからだ。プログラミングの論理を学ぶことを目的とするのに、ライブラリの不必要な記述、しかもいきなり上位クラスのコンストラクタを呼び出すということを教えなければならないのはいかがなものかと思う。Swing嗜好の教員は、その部分は、そういうものを書かなければならないという教え方をせよと主張するのであろうが、Java言語で一番問題なのは、何故この部分の記述をしなければならないかを、初学者にわかりやすく教えることにある。クラスの定義や、継承などについても、もちろん、テンプレートを使い、それは開発環境が自動的に記述してきてくれるので、それほど学生自身は気にしていない部分もあるが、自分たちが記述しているプログラムのすべての箇所について、このような記述方法をしているのかの理由説明を、学生に伝える必要がある。

初学者のプログラミングクラスでSwingを学生に教えるよりも、AWTの必要な部分だけ教えておき、上級のクラスにいって、ある程度オブジェクト指向におけるクラス設計やクラス継承を学んだ後にはじめてSwingを利用する方が教育効果は上がるであろう。いきなりSwingの記述で混乱させるのはよくない。

・内部クラス・無名クラス
これらは、Java2より拡張された言語仕様なのであるが、あまりプログラミング的に美しいとはいえない。また、メソッドとオブジェクトの分離ができない言語仕様のところに、記述の量をすくなくするために、無理やり導入した感があるのは否めない。これらが導入される部分は、イベントハンドリングで、インタフェースをクラスに添付した際に、多くのメソッドを定義しなければならないという局面である。特に使う必要のないメソッドを定義するよりも、Adapterなどを用いて、必要なメソッドだけを定義するという目的で使われている。しかし、プログラム上でメソッド呼出しの実パラメータに、新たなクラス定義を持ってくるのはいかがなものか。初学者が混乱しない筈がない。無名クラスは、プログラミング教育では忌み嫌うべきであり、言語設計的にも忌み嫌われるべきものである。初期の言語仕様設計のあらをついて導入されたものを教育目的に使うべきではない。

内部クラスは、無名クラスに比べれば、導入された理由もわからないではないが、クラスの中にクラスを定義するのは、オブジェクトのモデルの観点からは、疑問を拭えない。内部クラスは、主に名前空間の問題を解決するために導入されたのであって、それであれば、名前空間を委譲するような仕組みを考えるべきであり、内部クラスでお茶を濁すのは言語設計的にあまり推奨されるべきものでない。それは、単一的なオブジェクトのモデルを崩すからだ。もし、内部クラスを導入するのであれば、複合オブジェクト（Composite Object）を記述する仕組みを更に用意するべきである。オブジェクト間の明示的な関連性を記述することができなくて、暗示的な名前空間の継承だけが入った内部クラスは、プログラミングの記述量を少なくするための無名クラスを正当化するために、単に導入されているような気がしてならない。Java2の設計者は初期のJavaの言語設計者と異なると聞いている。初期の設計者の統一的な枠組みを崩してしまった感が拭えない。

教育の現場で、イベントハンドリングのためのインタフェースを導入した際に、不必要なメソッドの記述量を少なくさせるためには、Adapterクラスと無名クラスを使うよりも、次のように定義されたSimpleAppletを継承する方が良いと思われる。

```
public class  SimpleApplet  extends Applet
        implements ActionListener,
        AdujstmentListener, ItemListener,
        KeyListener, MouseListener,
        MouseMotionListener, WindowListener {

        public void actionPerformed( ActionEvent e )
        {}
        // 以下、それぞれのリスナーに必要な
        // メソッドを定義しておく
}
```

学生が参照できるライブラリに、予め使用するすべてのリスナーが定義された、このようなSimpleAppletを定義しておき、このSimpleAppletを継承するというやり方で、学生のアプレットを記述させるのである。この方が、教育的には効果がある。いくつかのライブラリでは、実際にこのような方法を採っているものも存在する。このやり方は、イベントハンドリングをJava2で導入された委譲（delegation）ベースから、初期の継承ベースの記述に戻せるので、必要なメソッドだけ上書きすれば良いのである。

授業では、イベントハンドリングに関しては、不必要なメソッドも定義させるという形を採った。その方が、何をしているかということがわかりやすいからである。ただ、時間数の限られている場合には、Adapterと無名クラスを導入するよりは、上記のような方法を採るべきではないだろうか。最後に付記しておくが、たまに自分がプログラミングができることを自慢するために、わざと無名クラスを導入させて、学生に混乱をまき散らしているのではないかと感じさせる教員もいる。このような教員はプログラミングという科目に割り当てるべきではない。

・ファイルや画像の取扱い
アプレットベースでは、ファイルや画像などについて、サーバのクラスファイルが置かれているフォルダからローカルにアクセスし、しかも読むことしかできないように制限されている。アプリケーションでは、このような制限が課せられていないので、アプリケーションベースでのウィンドウプログラミングを導入する場合もあるが、これも初学者のプログラミング教育の目的から外れている。アプリケーションでのウィンドウプログラミングは、コンストラクタやクラスメソッドについて充分な理解と講義がないと進めることができない。そのため、今回の授業では、ファイルや画像はサーバローカルに読むだけという制限された中での授業を行った。

グラフィックスについても、Java2Dまで導入しない形で行なった。オブジェクト指向のクラス設計やオブジェクトの振る舞いを充分に系統だって学んだ後は、却って、オブジェクトモデルで画像描画できるJava2Dの方が面白いと思われる。その部分をやるのは、初学者以降のクラスとして科目を用意すべきであると思われる。

４．授業の評価

今回、湘南藤沢キャンパス（SFC）の割り当てられたクラスでは、授業参加者17名、SAなどの補助者はなしで行なった。人数が限られており、補助者もいなかったこともあり、各学生に教員自らが対応しなければならないので、授業の進行速度は遅くなり、結果的に、あまり速くならないで、学生の充分な理解を得たということが後でのアンケートで分かった。また、このクラスは、２コマ連続で４単位の必修のクラスであるが、Macintoshに興味ある学生だけに選択して貰ったので、クラスの個々の学生のモチベーションはかなり高いと感じられた。開発環境は、Mac OS X 10.2（Jaguar）上のProject Builderを主な開発環境として用い、補助的にJJEditをダウンロードさせ、短いアプリケーションを記述するときに用いた。

初学年における授業であるということで、プログラミングにおいて以下のことができることを目的とした。この目的については、最後の定期試験でもかなり割合で学生に習熟しているのが確認された。

・繰返しや条件分岐が使えるか
・配列を使ってデータを整理しているか
・メソッドを自分で定義できるか
・パラメータの受け渡しがわかっているか
・ファイルや画像などの取扱いはわかっているか
・文字列の取扱いはわかっているか
・イベントリスナーを用意し、ユーザからの入力に対処できるか

２コマ連続で、１４回〜１５回の授業で初めて一通りのことをこなせるようになった。自分でクラスを定義して、継承などを使いこなせるようになるのは、この次の段階である。

５．カリキュラム上の問題点

一番の問題点は、湘南藤沢キャンパス（SFC）では、次に続く授業がないということに尽きる。Javaを利用した高度なプログラミングの授業がないために、学生の興味が少し削がれている部分が感じられた。折角Javaを教えて、他の授業に出るためにはC/C++言語などを学び直さなければらないという実感があるようだ。もちろん、JavaはC言語系なので、一から学び直す必要はないだろうが、ポインタや構造体などについては、学ばなければならないだろう。また、言語的にも改良される前の版がC/C++言語なのだから、その不自然さや不条理さが残る部分に疑問を持ちながら学ばなければならない。非常勤なので、カリキュラムの提案については権限がないし、一人では担当もできないだろうが、もしJavaにまつわる科目を設置する契機があるの

であれば、以下のような授業があると実用にも耐えられるのではないだろうかと提案することを考えている。

・クラス設計や継承、およびSwingによるGUI設計
・Java2D,Advanced Imagingによる画像描画・画像処理
・Java3DやJOGLを利用した3DCG
・ServletとJSPなどのサーバサイドプログラミング
・XMLも含めたテキスト操作
・シミュレーションを行なうもの

また、プログラミングを受ける前の先修科目として用意されている情報関係の基礎的な実習科目の中で、画像も含めたマルチメディアの部分がどうも弱い気がしてならない。画像における標本化・量子化・圧縮の他に、カラーや透明度に関する基本的な知識、フォントやベクトル画像表現に関する知識が欠けており、デザイン的な側面も含めて、それらの実習をする科目が前にあると良いのではないかと思われる。

６．他の環境に対する優位性

Mac OS Xの環境では、ただで開発環境を手に入れることができる。JDKの上にXcodeやProjectBuilderなどの統合的な開発環境が、標準のものとして揃っているのは、非常にプログラミング教育には最適なものと言える。Windowsの場合は、英語のJBuilderやNetbeansもあるが、実行速度もまだまだ遅いのが嫌な場合は、JDKを直接使うことになるが、これは最早、前近代的な開発環境と言うしかないだろう。Windowsでまともな開発を行おうと思ったり、開発環境自体に教育の足を引っ張られないようなもの想定する場合は、商用ソフトであるCodeWarriorに頼らざるを得ない。本務校の千葉商科大学の方では、PC環境はWindowsしかないので、CodeWarriorを必然的に導入している。導入してから５年以上になるが、他の教員からも一度もクレームが出たことがない。しかし、CodeWarriorは、アカデミック版でも3万円ぐらいの出費が強いられるのが学生が個人的に自分の環境でやりたいと思った場合問題になってくるだろう。

SFCは、Windowsの中のそれもよくぞ選んだ劣悪な環境に初年度に慣らさせておき、良い環境に見向きもさせないという教育の仕方？があり、学生は使い勝手の悪い環境を、これしかないと思って使っている節がある。もちろん、現代の統合的な開発環境で教育を行なってしまうと、コンパイラの仕組みなどが感覚的に身に付かないのかも知れないが、ほとんど統合されていない開発環境をいつまでも非効率に使い続けるのには無理がある。実際にWindowsのJDKのコマンドベース（およびemacsもどきのエディタ）の開発環境で作業をしている学生をたまに見かけると、泣けるような効率の悪さでプログラミングを開発している。これは、どう考えても、教育効果が上がっているとは言えない。プログラミング教育の目的は、Unix至上主義者にありがちな開発環境の基本ソフトウェアを仕組みがわかるツールを使い続けることに主眼があるのではなく、いかに問題をプログラミングの枠組みで考えていくことにある。開発環境が、水のように抵抗なく使えることの方がその目的に添っているように思える。

プロジェクタの視認性の問題であるが、やはりプログラミング教育では、ほとんどの高校の実習室がそうなっているようにセンターモニター（２台ぐらいに１つの割合で教材提示用のモニターが設けられているもの）を使っての教材提示の方が良い。SFCのこの教室では、プロジェクタが前と後ろの両方につけられているが、それでも、センターモニターが欲しいと思う場合がある。また、教師のコンピュータは、サンプルとなるプログラムを提示するものと、その後ろで必要な作業をやるものと２つあった方が便利である。たとえば、プログラムを提示しつつ、もう一方のコンピュータで、APIの仕様などを探したり、切り替えてそれを学生に見せたりすることもできる。ただし、２つのコンピュータを置かなくても、Macintoshの場合は、複数のモニターを標準でサポートしているので、１台のMacintoshで対応できるのが優れている。この場合、複数のモニターのそれぞれをプロジェクタなり、センターモニターに表示する切替えスイッチを必要とする。これがSFCの利用している教室ではなく、しょうがなくPowerBookを持参して、２つのMacintoshを切り替えて表示するような形態を採るしかなかった。

プログラムを提示するフォントの問題であるが、これは標準のOsakaフォントで特に問題はなかった（もちろん等幅にする必要もない）。できれば必要に応じて、Mac OS Xで標準で用意されている人文的サンセリフフォント（Lucida Grande、Gill Sans, Optimaなど）と組み合わせて用いても良いだろう。大文字のI（アイ）や、小文字のl（エル）、あるいは数字の1（壱）を区別するために、Courierなどのフォントを使っている本もあるが、基本的には人文的サンセリフが一番見やすいし、効率的にも良い。もちろん、字の識別性は大事であるが、それよりも、プログラム全体の見やすさ、俯瞰の方が重要である。Windowsには、字の識別や俯瞰の両面で考慮されたプログラム提示用のMS Referential Gothicという非常に良いフォントがあるが、これは日本語には対応していない（組み合わせられない）のが残念である。なお、どの統合型開発環境を使うにせよ、学生の開発環境のフォントも最初に設定させる必要がある。ある程度、サイズの大きいものや見やすいタイプフェースを選んでおかないと後からエラーを発見するのも直すのも大変（教員だけでなく本人も）であるからである。

参考資料

Javaではじめるプログラミング、箕原辰夫、秀和システム、2001年（絶版）