計算機プログラミング I (1)

プログラミングの手順

2003 年 10 月 10 日　増原 英彦

以下のことを説明する。

- 教科書で用いるライブラリプログラムをホームディレクトリにコピーする
- コピーしたプログラムをコンパイルし実行する
- コピーしたプログラムをもとに別のプログラム編集、コンパイル、実行する

なお説明は教育用計算機システムの**Unix 環境**を仮定する。教育用計算機システムの NC 環境や、いわゆるパソコン上では、その環境に応じたコマンド等を使う必要がある。

1　プログラムのコンパイルと実行 (教科書 1.3, 2.4)

1.1　ライブラリプログラムのコピー (準備)

教科書では、説明のために主にタートルグラフィクスを表示するプログラムを用いている。このようなプログラムを簡単に作成するために、絵の表示など頻雑な処理を担当するライブラリプログラムをコピーした上で演習を行う。

手順 1 *(ライブラリプログラムのコピー)*
Unix 環境のターミナルウィンドウ *(kterm)* に「*/home/masuhara/getcp1* Return」と入力せよ。自分のホームディレクトリの下の *javabook* というディレクトリに、ライブラリプログラムがコピーされる。
注意*:* この操作は *javabook* の下にあるファイルを**上書き**する。

手順 2 *(手順 1 の確認)*
手順 *1* に続けて、ターミナルウィンドウ *(kterm)* に「*cd ; ls javabook* Return」と入力せよ。以下のように表示されればライブラリプログラムがコピーされていることが分かる。

```
masuhara@as304> cd ; ls javabook
README    event     gui       net
applet    graphics  io        turtle
masuhara@as304>
```

1.2　ソースプログラムのコンパイル

プログラムのコンパイルは`javac` というコマンドを用いる。コンパイルしたいソースプログラムは、`javac` コマンドの引数として指定する。ここでは、ライブラリプログラムに含まれる `T21.java` というソースプログラムをコンパイルする手順を説明する。

手順 3 *(コンパイル) Unix* 環境の *kterm* で以下の操作をせよ。ただし、 *()* で囲まれた操作は確認のためのものであり、コンパイルのために必須のものではない。

1. `cd ~/javabook/turtle` Return と入力し、ライブラリプログラムがコピーされたディレクトリへ移動する。*(*何もメッセージ表示されなければ成功*)*

2. *(pwd* Return と入力し、現在のディレクトリを表示する。*/home/g123456/javabook/turtle* と表示されれば成功*)*

3. *(cat T21.java* Return と入力し、プログラムの内容を表示する*)*

4. *javac T21.java* Return と入力し、コンパイルする。*(*何もメッセージ表示されなければ成功*)*

5. *(ls -l T21.class* Return と入力し、コンパイルされたクラスファイルが作られていることを確認する*)*

1.3 オブジェクトコードの実行

上でコンパイルしたプログラムのオブジェクトコード (クラスファイル) を、Java 仮想機械を使って実行する。Java 仮想機械を動かすには `java` というコマンドを用いる (「c」がないことに注意。) 実行したいオブジェクトコードは、`java` コマンドの引数として**クラス名**(クラスファイルの名前でないことに注意!) を書くことで指定する。

手順 4 *(*実行*) Unix* 環境の *kterm* で *java T21* Return と入力する。数十秒持つと画面上にウィンドウが開き、表示が変化する。

ここで、コマンドを入力したウィンドウに

```
Warning: Cannot convert string "-monotype-arial-regular-r-normal--*-140-
*-*-p-*-iso8859-1" to type FontStruct
...
```

や

```
java.awt.AWTException: cannot open XIM
        at sun.awt.motif.X11InputMethod.<init>(X11InputMethod.java:147)
        ...
```

といったメッセージが出力されることがあるが、これらは害のない警告なので無視ししてよい。(下線部を手掛りにせよ。)

多くのプログラムは処理が完了すると勝手に終了する。しかし教科書で用いている例題プログラムは、一連の表示をした後も指示を待ち続けるように作られているので、終了を指示する必要がある。

手順 5 *(*終了*)* 次の方法のどれかによって手順 *4* で実行したプログラムを終了させる*:*

- 開いたウィンドウの「*File*」メニューの「*Quit*」を選ぶ
- 開いたウィンドウの「閉じる」ボタンをクリックする
- *java* コマンドを実行したターミナルウィンドウ *(kterm)* で、「Control c」とタイプする

2 プログラムの編集

コピーしたプログラムをもとに、新たなプログラムを作ってみる。ここでは `T21.java` をもとに `T20.java` を作ってみる。

手順 6 *(*プログラムの編集*)*

1. ターミナルウィンドウ *(kterm)* に「*cp T21.java T20.java* Return」と入力する。*(*何もメッセージが表示されなければ成功*)*

2. エディタ *(emacs)* で *T20.java* を開く。

3. ファイルの中身を変更する。

   - 区別のため、最初の行のコメントの内容を変えておく
   - クラス名を *T21* から *T20* に変える。

     (2 行目の「***public class T21*** {」中の下線部がクラス名である。*)*

     Java 言語コンパイラには「*(*公開された*)* クラスは、クラス名に*.java* を付けたファイルにしまわれていること」という規則がある。ここでは、ファイル名 *T20.java* とクラス名 *T20* を対応させなければいけないことに注意。
   - *main* メソッドの内容を適当に変える[1]。例えば「*m1.rt(90);*」と「*m1.fd(90);*」と書かれた *2* 行を、その直後にコピーしてみる。

     (3 行目の「***public static void main(String[] args)***{」**以降**から対応する閉じ括弧「}」までがメソッドの内容である。*)*
   - 変更したファイルを保存する。

4. 手順 *3* と同様にして、編集したファイルををコンパイルする。ファイル名を *T20.java* に変えたことに注意。

5. 手順 *4* と同様にして実行する。クラス名が *T20* に変わっていることに注意。

3 デバグ

プログラムの誤りのことを俗に虫 (bug) という。プログラムの誤りを修正することをデバグ (debug) という。

プログラムの誤りには、実行前に分かる誤りと、実行中に判明する誤りがある。前者はコンパイルの際にコンパイラが誤りを報告してくれる。後者は、仮想機械が誤りを報告してくれることもあるし、何の報告もなく誤った結果を得たり、プログラムが終了しなかったりする。

コンパイラが報告する誤りは、誤りの起きた場所と誤りの種類を教えてくれるが、**常に適切とは限らない**。例えば、あるプログラムをコンパイルすると、次のように誤りの報告をする:

```
T21.java:9: ';' がありません。
a    f.add(m1);                  //f に m1 を追加
      ^
(以下略)
エラー 3 個
```

このようなメッセージからは、次のようにして誤りを修正する。

- まず、「`T21.java:9`」は、誤りの起きたファイル名と行番号であるので、エディタでそのファイル[2]を開き、示された行を探す。
- メッセージの 2 行目は誤りを見つけた行の内容であり、3 行目の「`^`」によって誤りの起きた場所が示される。どの場所で誤りが見つかったかを確認する。
- 1 行目の「`';'` がありません。」というのが (コンパイラにとっての) 誤りの内容である。
- これらから「『`a f;`』と書いてあれば (コンパイラにとっては) 理解できるが、『`;`』ではなく『`.`』だったので理解不能になった」ことが読みとれる。(プログラムの 4 行目に「`TurtleFrame f;`」という類似の文型がある。)
- しかし実際には、「`a`」が混入したことが原因であり、エラーメッセージは完全に適切ではない。結局、前後のプログラムの意味を考えて原因を推測して修正することになる。

[1] 編集の際、行の先頭で Tab キーをタイプすると、適切な桁にポイント (カーソル) が移動する。また、*M*-`x show-paren-mode` Return としておくと、対応する括弧を色分けして表示してくれる。

[2] javac は別のファイルを自動的にコンパイルする機能があるので、引数として与えたものとは違うファイルの誤りが報告されることもある。

4 練習

練習 1-1: (クラス名とファイル名) クラス名とファイル名が対応しない場合、どのような誤りが報告されるかを調べよ。

手順 6 に書いたように、Java コンパイラはクラス名とファイル名の `.java` の前が同じでなければいけない、という規則がある。この規則をやぶった場合にどのような誤りが報告されるだろうか。

練習 1-2: (改行・空白文字) プログラム `T21.java` の適当な行の適当な場所に改行や空白を入れてコンパイル・実行してみよ。単語の途中やコメントの途中だとどうか? どのような場所であれば改行や空白を入れてもコンパイルエラーにならないかをまとめよ。

練習 1-3: (行の複製) プログラム `T21.java` の適当な行を複製して、コンパイル・実行してみよ。コンパイルエラーになる場合、その理由を推測して説明せよ。

練習 1-4: (括弧) 数式で $1+2+3$ を $(1+2)+3$ と書いても結果が変わらないように、プログラム中の括弧 `( )` `{ }` も余計に書いても結果が変わらない場合がある。プログラム `T21.java` に括弧を追加しても結果が変わらないのはどのような場所か? また、`T21.java` には余計な括弧はあるか?

練習 1-5: (名前) プログラムでは、処理や値に名前をつける。名前の中にはプログラムを書く人が自由に決められるものもあるし、そうでないものもある。`T21.java` の中には沢山の名前があるが、それらの中で名前を変えてもプログラムの動きが変わらないものはどれか? プログラム中に現われる単語を変更してコンパイルしてみよ。

(ヒント: 同じ名前は一勢に変更しなければいけないこともある。)

練習 1-6: (名前) プログラム中で使われる名前にはどのようなものが許されるのだろうか? `m` を `myturtle` のような長い名前にしてもよいのだろうか? 名前に数字を使ってもよいのだろうか? `2hikimenokame` のようなものは良いのだろうか? 名前の中に記号が入ってもいいのだろうか?