Akira 虎の巻

中村貴英

平成 23 年 5 月 20 日

はじめに

この文章は可視化ソフト Akira を開発，改良する人向けのマニュアルです．また中村貴英が書いているので，一切の文責は私にあります．

目 次

第1章　方針

開発者の心得を記述します.

1.1　開発方針

皆で利用・開発するものだ！

既に原子の可視化ソフトは世にたくさんあります．にも関わらず独自の可視化ソフトを開発することの意義は,「こういう風に見せたい」と思った時に自分で少しコードを改変することで思った事を実現できることにあります．そのためには：

- 利用者がコードの中身を知っていること，または利用者が容易にコードの中身を理解できることが大事である.
- 各自が機能の追加を行うのことを出来る限り容易にするため，オブジェクト指向の Java を利用し，カプセル化されたコードとするよう心掛ける.
- 高機能よりは，容易かつ高い拡張性を求める.

オブジェクト指向らしく，なるべくカプセル化する

オブジェクト指向の概念に関しては，ウェブ上にも多くの資料があるので各自で勉強してください．Java は典型的なオブジェクト指向言語なので，可能な限り，オブジェクト（物）にはその物の持っている性質（property）と，その物が提供する機能（function）を過不足なく実装することを心掛ける．そうすることで，カプセル化されたオブジェクトを作ることができ，使い回しの効くプログラムとなる.

可能ならば，MVC モデルに準拠する

Model-View-Controller モデルに準拠して，この 3 つが独立した形に書いておくとカプセル化の度合いが強くなり，再利用性が増す．Akira においては Model とは原子の集まりであり，理想としては原子の集まりが View からは独立しており，他のオブジェクトにより OpenGL を用いて描かれるようになっていて欲しい.

プラットホームに依存しないコードを書く

Java を用いる利点の一つは，一度コードを書いたらどのプラットホーム（Unix, Linux, Mac, Windows）でも動作することである．特定のプラットホームでしか動作しないコードを含めるのは，この思想に反するので極力避けるべきである.

設計で迷ったら「Simplicity」を重視する

Java 言語の設計原理は KISS (Keep It Small and Simple) らしい．メンテナンスの際にも Simple な方がやり易く，バグを減らすことに繋がる．

使い方の大幅変更を要する場合は developer 全員に相談すべし

一般に，マイナー・バージョン・チェンジで使い方の変更があるようなソフトはユーザーが使いたがらない．しかし，変更の種類に依っては入力ファイルの変更が必要だったり，config ファイルの変更が必要であったりと，使い方そのものに影響を及ぼすことも考えられる．そのような場合には，自分のブランチを用意して自分専用として変更する．その変更が皆のためになるものであると思ったら，developer 全員（もしくは mailing list）に相談してから変更すべし．さらに，使い方の変更は update したユーザに的確に伝わるように努力すべし．

GUI の要素に focus が合わないようにする

ボタンなどの GUI 要素に focus が合っていると，キーボード入力がそこに取られてしまい，キー操作が効かなくなってしまうため，focus は常に frame に合わせるようにしたい．

```
hoge.setFocusable( false );
```

とすることで，hoge という GUI 要素に focus を合わせないように設定できる．

基本的にクラスの変数には外部からアクセスしない

インスタンス変数を private もしくは protected にする，method 経由で変数をゲット・セットするのが Java の流儀らしい（getter，setter または accessor?と呼ばれる）．インスタンス変数を public にする基準として，以下の条件を設ける：

- 他のインスタンス変数から独立していて，単独で変更されても内部の整合性を乱さない．
- どちらにしろ，get/set の method を書く必要がある．
- そのインスタンス変数の実装が将来も変更されないと予想される．

この条件が満たされない場合でも，速度を気にする変数の場合には public にした方が良い場合があるらしい．

高速なコードとするために...

パフォーマンスは大事だが，初めからパフォーマンスを気にしてコーディングをするのは避ける．視認性，保守の容易さを優先する．パフォーマンス向上はアプリケーション全体の測定を行い，ホットスポットに対して行なう．(頑張って高速化したつもりでも，つまらない事をしていた，なんてことのないように.）ちなみに Java の場合，

- new は時間が掛かるので，ループの中では使うのを避けた方が良いらしい．
- オブジェクトの解放は garbage collection が行なってくれるが，それを助けるために，使わなくなったオブジェクトに null を代入するとよいらしい．

1.2 コーディングルール

個人で開発する訳ではないので，Akira 開発に際してのコーディングのルールを定めておく．このルールに則ってコーディングを行うことを強く推奨する．しかし，100%準拠するする必要はない．十分な理由があってルールから外れることはままあることなので．

public クラスはそのクラス名の１ファイルにする

これは Java の規則かな？public でないクラスは，そのクラスを最も多く利用する public クラスのファイルに含めてよい．

インデントは空白２文字

一般には空白４文字の場合が多いが，４文字は空けすぎと思う．(by R.K.) エディタとして emacs を利用している人は，以下を.emacs に書くとインデントを２文字にしてくれる．

emacs の設定

```
(autoload 'c++-mode  "cc-mode" "C++ Editing Mode" t)
(autoload 'c-mode    "cc-mode" "C Editing Mode" t)
(autoload 'java-mode "cc-mode" "Java Editing Mode" t)
(add-hook 'c-mode-common-hook
'(lambda ()
(setq c-indent-level 2)
(setq c-brace-imaginary-offset 0)
(setq c-argdecl-indent 2)
(setq c-continued-statement-offset 2)
(setq c-label-offset -2)
(setq c-brace-offset 0)
(setq tab-width 2)
(setq c-basic-offset 2)
(setq comment-style 'extra-line)
(setq comment-start "/*")
(setq comment-end "*/")
))
```

１行は８０文字程度とする

Java の場合は変数名が長いため右に伸びる傾向があるが，ソースコードをプリントアウトする時などを考えると８０文字程度で折り返す方が良いだろう．

コメントを書く

このプログラムは複数人で開発しているので，他の人があなたが書いたコードを読んだ際に理解に時間が掛かると思ったら，理解し易いようにコメントを書くこと．コメントはコードの理解を助け，バグを減らすと期待できるので，面倒でも書くこと．また Java には，ある書式に従って書いたコメントをそのままドキュメント化する Javadoc という機能があるので，

上手くコメントを書けばそれがそのソースコードの仕様書 (API) となる．日本語でも良い．ただし，文字コードは UTF-8 no BOM とする．no BOM で書かないと，コンパイルができない (OSX Tiger/Leopard).

クラスの名前は物とする

オブジェクト指向なので，各クラスはオブジェクト（物）である．故にクラスの名前は物であるべきであろう．名詞であれば良いのでなく物の名前とすべき（事とすべきではない）．例えば，貨幣の単位換算をするクラスを作りたいとしたら，ConvertingMoney というクラスではなく，MoneyConverter とすべき．

クラス名は大文字から始め，変数名とメソッド名は小文字から始める

クラス名は大文字から始めるのが Java の流儀．かつ，その名前がいくつかの word からなる場合は，それぞれの word の先頭を大文字にする．(アンダースコアは使わない)（例えば，SuperGreatGoalKeeper.）変数名とメソッド名は最初の文字は小文字とする．いくつかの word から成る場合は２つめの word からは先頭を大文字とする．(例えば，doSomethingWithSomebody().）このコーディング規則は，大文字が「らくだのこぶ」の様に見える事から， CamelCase と呼ばれる．

定数は全て大文字とする

この場合単語の区切りがわかりにくいので，アンダースコアを特例として使用する．(例えば，MAX_NUM_OF_ATOMS.）

変数名は長くても良いから意味が分かるようにする

Java 言語では変数名は説明がなくても意味が通じるようなものが推奨される．例えば，ある原子が表示されるかされないかの判定に使う boolean 型の変数に名前を付けるとしたら，isVisible のようになるだろう．

import の際の*を避ける

import の*は読む人には，何の import を必要としているのか分かり難い．ただし，同一パッケージから３つ以上 import する場合は冗長性を避けるために*を使う．

1.3　リビジョン管理

Akira のリビジョン（バージョン）管理には Mercurial を利用している．使い方はウェブ上にたくさん資料があるので，各自勉強すること．おそらく，これまでリビジョン管理などはしたことがないと思うので，いくつかのルールを決めておく．

コミット・コメントは簡潔かつ意味のあるものを書く

コミットの際には必ずコメントを書くこと．リポジトリ・ウェブを見ると分かるが，コミット・ログにはコメントは一行しか表示されない．そのため，コメントは簡潔に何を変更したのかを他人に伝えるように書くこと．

バグがない事を確認してからコミットする

リポジトリへのコミットは変更のセーブではないので，頻繁に行う必要はない．自分が行おうとした変更が完成し，変更したことによるバグがないことを確認した後にコミットすること．あまり意味のないコミットが増えないように心掛けよう．

タグは自由に付けて良い（ただしルールあり）

タグを付けておくと，そのリビジョンを取り出したり，そこに戻ったりが容易なので，自由にタグを付けてよい．ただし，各自で自由に付けるタグの名前はその人の決まった prefix から始まるものでなければならない（例えば，kobayashi-1）．他に，version-??や release-2010.??.?? のようなタグが付けられるものと思われる．

第2章　開発方法

自分で Akira をコンパイルする方法を述べます.

2.1　Mercurial の使い方

2.1.1　事始め

ソースコード一式は，Mercurial で

```
hg clone https://project-akira.googlecode.com/hg/ project-akira
```

として，手に入れます.

2.1.2　最新版に

```
hg pull
hg update
```

反映

```
ant clean
cp Akira.jar ~/myLocal/AKira
```

2.1.3　状況を確認する

```
hg status
```

M は modified，R は removed，!は追跡不能，?は未追跡，を意味する.

2.1.4　コミット

セーブ点を作る.

```
hg commit
```

新しくファイルを追加した場合は，そのファイルを追跡しているかを確認すること．コミットを行う前に，hg status を行って，追跡漏れしているファイルが無いか調べましょう．

2.1.5 パッチの作り方

Mercurial では，以下のようにしてパッチを作ります．

```
hg export -o a.patch tip
```

出来たパッチを中村宛に送ってください．

コミット権限は，オーナーである小林と中村しかもっていません．多くの貢献をしてくれる方には commit 権限を差し上げます．

2.1.6 変更をサーバーへ上げる (commiter only)

```
hg push
```

この際のユーザーは gmail アドレス，パスワードは web ページに記載されているものか gmail のアドレスです．

2.2 コンパイル方法

2.2.1 OSX/Linux でのコンパイル

設定

JOGL_LIB という環境変数に jogl のライブラリを置いた場所を指定して下さい．例えば，

```
export JOGL_LIB=~/myLocal/javalib/jogl
export DYLD_LIBRARY_PATH=$JOGL_LIB:$DYLD_LIBRARY_PATH
```

を.bashrc か.bash_profile に追記するということです．

コンパイルテスト

build.xml のあるディレクトリで

```
ant
```

とすると，Akira がコンパイルされ，実行されます．JOGL_LIB の設定が失敗している場合，実行されません．もう一度 JOGL_LIB の設定を見直してください．[1]

実行形式の生成

```
ant jar
```

として jar ファイルを作ります．

2.2.2　windows でのコンパイル

JDK のインストール

JDK をインストールします．(JRE ではありません) コントロールパネル-システム-詳細設定-環境変数にて JAVA_HOME の値を

```
C:Program Files\Java\jdk******** (←インストールされた場所)
```

として自分で設定しますさらに Path に JAVA_HOME\bin を追加 (なぜこのくらいの設定をインストーラーでやってくれないのか...)

ant

ant をダウンロードして，内容物をすべて

```
C:Program Files\ant (どこでも良いが私はこうした)
```

に入れます．更にコントロールパネル- システム-詳細設定-環境変数にて ANT_HOME の値を

```
C:Program Files\ant
```

として自分で設定しますさらに Path に%ANT_HOME% \bin を追加

build.xml の書き換え

環境変数 JOGL_LIB を設定しても，ant でうまく処理してくれないので，build.xml を直接変更します．build.xml の先頭にある

[1] ant とは java 用の make です

```
<property name="jogl_dir"  value="\${env.JOGL_LIB}" />
```

を

```
<property name="jogl\_dir"  value="\jogl2" />
```

に書き換えます.

第3章　Akiraの構造

Akira は AkiraConverter と AkiraViewer で構成されます．主要な構成ファイルの特徴を述べます．これにより，Akira の全体像を把握しやすくなることを期待します．AkiraConverter.java と AkiraViewer.java がメインファイルです．

さて，以下のコマンドで Akira のソースコード一式がコピーされます．

```
hg clone https://project-akira.googlecode.com/hg/ project-akira
```

以下では，project-akira 以下のファイルについて説明します．

3.1　project-akira/の構成

clone して出来たディレクトリの内容は以下です．

AkiraConverter.conf
: 開発用の AkiraConverter 設定ファイル

build.xml
: ant 用のファイル．makefile みたいなもの．

converter/
: AkiraConverter 用のファイルが格納されている

data/
: 基本データのクラスが格納されている

img/
: Akira で使う image ファイルが格納されている

input-samples/
: 開発用のサンプルファイル

jar-libs/
: Akira で用いるライブラリファイルが格納されている

make-package.sh
: 公開用のファイルを作るスクリプト

manual/
このマニュアルを生成する tex ファイルが格納されている

rotation.AkiraCmb
コンボモードのサンプルファイル

tools/
AkiraConverter と AkiraViewer で共通して使うクラスが格納されている

utils/
プラットホーム別のスクリプト

viewer/
AkiraViewer 用のクラスが格納されている

AkiraConverter で使うクラスは converter/に，AkiraViewer で使うクラスは viewer/に格納されています．共通して使うクラスは data/や tools/に格納されています．

表 3.1: data/の構成

ファイル名	内容
Atoms.java	原子のクラスファイル
Bond.java	ボンドのクラスファイル
Bonds.java	ボンドをまとめるためのクラスファイル
Const.java	定数を定義してあるファイル

表 3.2: converter/の内容

ファイル名	内容
AkiraConverter.conf	AkiraConverter 用の設定ファイル
AkiraConverter.java	AkiraConverter のメインクラス
ConfCreater.java	AkiraConverter 用の設定ファイルを生成するクラス
ConvConfig.java	AkiraConverter.conf を読み込んで，値を保存するクラス
Tool.java	AkiraConverter で使う関数
reader/	各種データフォーマット別の読み込み関数

表 3.3: viewer/の内容

ファイル名	内容
AkiraViewer.java	main 関数があるクラス．Look&Feel も設定する．
BackRenderingWindow.java	後ろからの描画を行うクラス．enjoy モードで使用．

table continued on next page

continued from previous page

ファイル名	内容
Controller.java	実質的な main クラス．このクラスが全クラスを保持している．
KeyController.java	キーボード操作を統括する
LF/	Look&Feel 関係のクラスが置いてある．
RenderingWindow.java	描画が行われるメインウインドウ．
RenderingWindowMenuController.java	RenderingWindow のメニューを管理．
SplashWindow.java	AkiraViewer が起動する時のスプラッシュウインドウ．
UpdateManager.java	Update を管理する．
ViewConfig.java	AkiraViewer の設定が保持されるクラス．
ViewConfigWindow.java	AkiraViewer の設定を変更するためのウインドウ
ViewConfigWindowMenuController.java	ViewConfigWindow のメニュー
informationPanel/	InformationWindow の各パネル．
keys.html	key のヘルプファイル
renderer/	機能別の rendering クラスが格納されている
viewConfigPanel/	設定を変更する為の各種パネル

表 3.4: tools/の構成

ファイル名	内容
BondCreator.java	ボンドを生成するクラス
Coordinate.java	xyz 座標から曲座標系への返還
Exponent.java	
InvMat.java	逆行列計算
LUDecomposition.java	逆行列計算
MyColorEditor.java	
MyColorRenderer.java	
MyFileIO.java	Akira ファイル用の File IO
MyFilter.java	Open ダイアログ用のフィルタ
MyOpen.java	AkiraViewer での Open ダイアログ
PairList.java	リストを生成する
SlideInNotification.java	スライドインする警告を実装している
Tokens.java	文字列分解用のクラス
colorpicker/	colorpicker 用のクラス

表 3.5: jar-lib/の構成

ファイル名	内容
BareBonesBrowserLaunch.jar	web ブラウザを立ち上げる為に必要なライブラリ
TableLayout.jar	レイアウト用のライブラリ
ant.jar	アーカイブされたファイルを読み込む為に必要なもの
forms-1.2.1.jar	RenderingWindow で使うステータスバー用のライブラリ
jlibeps.jar	eps ファイルを出力する為のファイル

3.2 AkiraConverter

main クラスは AkiraConverter.java です．ここに書いてある流れをよく理解して欲しい．この AkiraConverter クラスでは，データを読み込み，Akira ファイルに書き出します．新たなデータフォーマットに対応するには，converter/reader 以下に追加します．

Atoms クラスに値をセットする事です．具体的には，h-matirx，原子種，座標，補助データをセットします．**座標は実座標なので注意してください．**

3.2.1 AkiraConverter.java

設定ファイルを応じて，読み込み関数を切り替えます．

3.2.2 converter/reader

各種データフォーマットに対応した読み込みクラスが集まっています．AkiraAscii.java がスタンダードな読み込みクラスなので，新たに追加する場合はこれを参考にしてください．

3.3 AkiraViewer

これもメインクラスは AkiraViewer.java です．起動順は

1. AkiraViewer
2. Controler
3. ViewConfigWindow
4. RenderingWindow

です．この順に読んでください．これらの昨日は

AkiraViewer
: Look&Feel や，コマンドラインオプションを解析

Controler
　実質的なメイン

RenderingWindow
　描画

ViewConfigWindow
　設定ファイルを編集

以下のセクションで，重要なファイルに対してコメントします．

3.3.1 Controler.java

各クラスはこの Controller クラスを経由して他のクラスの変数やメソッドを利用します．

3.3.2 ViewConfig.java

Viewer の設定変数が格納されているクラス．これは丸ごと保存されます．この値は ViewerConfigWindow で編集されます．

3.3.3 RenderingWindow.java

OpenGL のセットアップを始め，実際に描画するメソッド (display) を持ちます．新たに描画メソッドを追加したら，このクラスも修正しなければ反映されません．各種描画メソッドの詳細は JavaDoc を参照してください．

1. RenderingWindow() (コンストラクタ)
2. initialize()
3. init()
4. display()

の順で読むとよいでしょう．

付 録A　ant の使い方

java の make です．build.xml が makefile に対応します．

付 録B　Mercurialの使い方

mercurial は水銀なので，元素記号の hg がコマンド．なんて素敵なセンスなのでしょう．

B.1　設定ファイル

B.1.1　.hgrc

これが無ければ，コミット出来ないようです．

```
[ui]
username = YOUR_NAME <YOUR_EMAIL>
verbose = True
```

B.1.2　.hgignore

追跡を無視するファイルを書いておく．

```
syntax: glob
*.class
*.jar
```

B.2　基本操作

最もよく使われると思われるコマンドについて説明する．

hg pull　サーバーに変更を問い合わせる．

hg update　pull してきた変更を適用する．

hg status　現在の状況を確認する．

hg commit　コミット点を作る

hg push　変更をサーバーへ上げる．