計算機プログラミング I (13)

インターフェースとスレッド

2001 年 1 月 16 日 増原 英彦

アニメーションのように時間とともに変化する絵を描くには、(1) 定期的に状態を変化さる計算と (2) 現在の状態にあわせて絵を描いたり、ボタンが押されたときの処理を行ったりする計算、の 2 つを同時に行う必要がある。

プログラムを実行しているとき、CPU は「現在実行中の文」に書かれている命令を実行しては「次の文」に進む処理を繰り返している。このような CPU が行っている計算の流れをスレッド(thread)と呼ぶ。

Java 言語をはじめとするいくつかのプログラミング言語では、1 つのプログラムを複数のスレッドによって実行させることができる。これによって、1 つのスレッドは定期的に状態を変化させ、もう 1 つのスレッドに画面を描かせることでアニメーション等が実現できる。

Java 言語でスレッドを作るには、特別なインターフェース(interface) を持ったクラスのオブジェクトを与える必要ことで、スレッドが行う計算を指定する。そこで、まずインターフェースの概念を理解した上で、スレッドを使ったプログラムを見てみよう。

1 インターフェース

1.1 異なるクラス・共通の操作

前回の「積分」の例で見たように、異なるクラスに属するオブジェクトに対して共通の操作を使いたい場合は多い。

Java 言語のプログラムで、オブジェクトを操作する (つまり、インスタンスメソッドを実行したり、インスタンス変数の値を読み書きする) ためには、オブジェクトの型を決める必要がある。結果として、全く別のクラスのオブジェクトに対して共通して実行できるメソッドを定義することができなかった。(前回の「積分」では、様々な「関数」を持つクラスを`Constant` クラスの子クラスとして定義することで対処していた。)

例えば、「`draw(g)` というメソッドが実行されたらグラフィック画面 g に適当な絵を描く」クラスとして「点」`Point` と「関数」`Function` があったとする。

```
class Point {
   ...2 次元座標の点の定義...
   draw(java.awt.Graphics g) { g 上に小さな円を描く }
}
class Function {
   ... 適当な関数の定義...
   draw(java.awt.Graphics g) { g 上に関数のグラフを描く }
}
```

Point クラスか Function クラスのオブジェクトを使って画面を描くアプレットは下のようになるだろう:

```
public class ShowSomething extends java.applet.Applet {
  // 現在表示させたいオブジェクトを drawer に登録しておく
  ???? drawer; // Point か Function クラスのオブジェクト
  public void paint(java.awt.Graphics g) {
    drawer.draw(g); // 現在表示させたいオブジェクトを表示
  }
}
```

このアプレットのインスタンス変数 drawer には、状況によって Point クラスのオブジェクトかFunction クラスのオブジェクトが登録されるものとする。どちらのクラスにも draw というメソッドがあるので、paint の定義は共通して使えるはずである。

しかし、Point と Function は別の型なので、このままでは変数 drawer に適切な型をつけることができない。

1.2 インターフェース = 共通のメソッドを持ったクラス

Java 言語には、同じメソッドを持っている (異なる) クラスを統一的に扱うために、インターフェース(interface) という仕掛けがある。これを使えば、同じメソッドを持つ複数のクラスのオブジェクトを 1 つの型の値として扱うことができるため、共通した操作を定義することができる。

上の例をインターフェースを使って定義すると次のようになる。まず、「draw というメソッドを持っている」ことを示すインターフェースを下のように定義する:

```
interface Drawable { // 自分自身を描くことができるクラス
  void draw(java.awt.Graphics g);
}
```

2 行目は、「このインターフェースを実装している (後述) クラスは draw という名前のメソッドを持っている」ことを示している。クラスの定義と違って、メソッドの本文がなく「;」で終っていることに注意。

次にクラスPoint とFunction は、「draw というメソッドを持っている」、つまり「Drawable インターフェースを実装している」ことを宣言する。これには、クラス定義の際にimplements という「おまじない」を書く:

```
class Point implements Drawable {
   ...2 次元座標の点の定義...
   public void draw(java.awt.Graphics g) { g 上に小さな円を描く }
}
class Function implements Drawable {
   ... 適当な関数の定義...
   public void draw(java.awt.Graphics g) { g 上に関数のグラフを描く }
}
```

それぞれのクラスにはdrawというメソッドがあり、引数の型(java.awt.Graphics)と個数がDrawableの定義と一致している必要がある。

最後に、これらのクラスのオブジェクトを使って表示するアプレットは下のようになる:

```
public class ShowSomething extends java.applet.Applet {
  // 現在表示させたいオブジェクトを drawer に登録しておく
  Drawable drawer; // Point か Function クラスのオブジェクト
  public void paint(java.awt.Graphics g) {
    drawer.draw(g); // 現在表示させたいオブジェクトを表示
  }
}
```

ここで、インターフェースの名前 Drawable を型としてインスタンス変数 drawer を定義している。この変数に代入できるのは、クラス定義のときにimplements Drawable と書かれていたクラス、つまり、Point クラスか Function クラスのオブジェクトである。これらのクラスには draw というメソッドがあるので、メソッド paint のように共通した操作を書くことができる。

2 スレッドとアニメーション

時間とともに点が移動するアニメーションを例に、スレッドの使い方を見てみる。まず、アニメーションを表示するアプレットは図 1 のようになる。右端に「*」が付いている行を無視すれば、第 10 回で登場した点 (Point) クラスのオブジェクトを作り、それを表示するだけのアプレットである。このプログラムに「定期的に点 p の位置を変化させ、アプレットに再表示させる」計算を同時に動かすことでアニメーションができる。

```
 1 public class Animation extends java.applet.Applet {
 2   Point p;                      // 表示したい点
 3   Thread t;                     // 点を移動させるスレッド                *
 4   public void start() {         // アプレットが実行開始されたとき
 5     this.p = new Point(0,0);    // 点を作って覚える
 6     Point dp = new Point(2,3);  // 点の速度ベクトルを作る                *
 7     // スレッドの処理内容: 点 p を速度 dp で動かす                         *
 8     PointMover pm = new PointMover(p,dp,this); //                          *
 9     this.t = new Thread(pm);    // スレッドを作る                         *
10     this.t.start();             // スレッドの実行を開始                   *
11   }
12   public void paint(java.awt.Graphics g) { // アプレットが表示されたとき
13     this.p.draw(g);             // 点を表示する
14   }
15   public void stop() {          // アプレットが中止されたとき            *
16     this.t.stop();              // スレッドを止める (バージョン 1.1 まで)  *
17   }                             //                                        *
18 }
```

図 1: アニメーションを行うアプレット

8 行目では、PointMover というクラスのオブジェクトを作っている。このオブジェクトは定期的に点の位置を変化させる処理を定義しており、図 2 のようになっている。

このクラスのコンストラクタは点 p, 点の速度ベクトル dp, と点が表示されているアプレット a の 3 つを受け取り、それぞれをインスタンス変数に代入して覚えている。また、メソッド run は、(1) 点 p をベクトル dp だけ移動させ、(2) アプレット a の画面の再表示を要求し、(3) 0.1 秒間休む[1]、という処理をひたすら繰り返している。

```
 1 class PointMover implements Runnable { // 点を移動させる処理
 2   Point p, dp;                  // 移動させる点とその速度
 3   java.applet.Applet a;         // 点を表示しているアプレット
 4   PointMover(Point p, Point dp, java.applet.Applet a) { // コンストラクタ
 5     this.p = p; this.dp = dp; this.a = a;
 6   }
 7   public void run() {           // Runnable インターフェースで定義
 8     int interval = 100;         // 休みの長さ (ミリ秒)
 9     while (true) {
10       this.p.move(dp);          // 点を少し移動させる
11       this.a.repaint();         // 再描画を要求する
12       try {Thread.currentThread().sleep(interval);} // 一定時間休む
13       catch (Exception e) {}    // try ~ catch は「おまじない」
14     }
15   }
16 }
```

図 2: 点を動かす処理を定義したクラス

メソッド run を新しく作ったスレッドによって実行させているのが図 1 の 9,10 行目である。まず、9 行目の this.t = new Thread(pm); は新しくスレッドを作る操作である。詳しく言うと、

- new Thread(pm) は、Java 言語に付属している Thread というクラスのオブジェクトを 1 つ作っている。
- オブジェクトと同時に、停止状態の「スレッド」が 1 つ作られる。
- 作られたオブジェクトは、スレッドの実行を開始・停止などの制御のために用いられる。
- コンストラクタの引数は、スレッドがどのような計算をするかを与えるためのオブジェクトである (後述)。

9 行目の this.t.start(); では、スレッドの実行を開始している。このstart メソッドが実行されると、スレッドはコンストラクタの引数として与えられたオブジェクト (ここでは pm) の run というメソッドを実行を始める。この実行は start を実行したものとは別のスレッドによって行われるので、start メソッドの実行自体はすぐに終了する。

[1] 12 行目–13 行目がその処理である。Thread.currentThread() はThread クラスの static メソッドで、現在実行中のスレッドオブジェクトを得るためのものである。この返り値、つまり現在実行中のスレッドオブジェクトのsleep メソッドを実行すると、スレッドの実行が一定時間停止する。この文の周りにある try { ~ } catche (Exception e) {} は、中の文を実行しているときに生じた「例外」を処理するための文である。詳しくは参考書等を参照せよ。

結局、新しく作られたスレッドは、オブジェクト pm の run メソッドを実行する。このメソッドは(すでに述べたように)、点の位置を少し動かしてはアプレットを再描画させるものなので、点がゆっくりと移動しているように見える。

図 1 の 15–17 行目では、アプレットが停止されたときに、スレッドも同時に停止させている。インスタンス変数 t はPointMover の run を実行しているスレッドであり、これの stop メソッドを実行すると[2]、スレッドの実行が強制的に停止される。

練習 13-1: (アニメーション) クラス Animation, Point, PointMover の定義を完成させよ。さらに HTML ファイルを作り、アニメーションを行うアプレットを完成させよ。(クラス Point の定義は、第 10 回の資料 + 練習 10-3 で追加した draw メソッドがあればよい。)

練習 13-2: (跳ねる点) クラス Animation, PointMover の定義を変更して点が画面の端に到達したときに跳ね返るプログラムを作れ。具体的には、PointMover の run メソッドの繰り返しの中で、移動後の点 p の x 座標が負またはアプレットの表示部分の横幅より大きい場合、速度ベクトル dp の x 成分の値を -1 倍する。同様に点 p の y 座標が負またはアプレットの表示部分の高さより大きい場合、速度ベクトル dp の x 成分の値を -1 倍する。アプレット a の横幅・高さを調べるには、

a.getSize().width またはa.getSize().height

のようにする。

練習 13-3: (大きくなる円) アプレットの中心に円を表示するクラス Circle と、時間とともに円の半径を増やしてゆくクラス Pump を定義せよ。これらのクラスを使って、円が大きくなるアニメーションを作れ。余力があれば、周期的に円が大きくなったり小さくなるようにせよ。

練習 13-4: (2 つの点) クラス Animation の定義を拡張して、2 つの点が同時に動くようなプログラムを作れ。

3 スレッドのまとめ

- スレッドを作るには、new Thread(x) のように、Thread クラスのオブジェクトを作る。
- Thread クラスのコンストラクタの引数 (上のx) は、Runnable インターフェースを持ったクラスのオブジェクトでなければいけない。
- Runnable インターフェースは、

  ```
  interface Runnable { void run(); }
  ```

  というものである。つまり、run という名前のメソッドを持っているクラスを示している。
- スレッドオブジェクトのstart メソッドを実行されると、スレッドを作ったときに与えたオブジェクトのrun メソッドが実行される。

[2]ただし、このメソッドは Java 言語のバージョン 1.1 までのものであり、それ以降のものを使う場合には別の方法をとらなくてはいけない。

- スレッドオブジェクトの`stop`メソッドを実行すると、スレッドの実行が停止する。

4 おわりに

この授業で学んだことは、以下のとおりである:

- プログラムの作り方、コンパイル方法、実行方法
- プログラムの基本概念: 変数・条件分岐・繰り返し・メソッド・配列
- オブジェクト指向プログラムの基本概念: クラス・メソッド・継承
- Java 言語の諸機能: アプレット・スレッド・インターフェース

一方、重要ではあるが、この授業で扱わなかった内容としては次のものが挙げられる:

- 様々なアルゴリズムの考え方 (整列・探索・etc.)
- 様々なデータ構造 (リスト構造・木構造・etc.)
- 計算時間の理論的な扱い (「計算量」の理論)
- 高度なオブジェクト指向言語の機能 (抽象クラス・etc.)
- Java 言語特有の諸機能 (パッケージ・アクセス制限・etc.)
- 「良い」プログラムを書くためのノウハウ (デザインパターン・etc.)

4.1 お知らせ等

- 質問や相談はこれまで同様、電子メール・ニュースなどで受け付けます。メールでは難しい質問・相談を直接行いたい場合は、電子メールで訪問時間を予約して下さい。
- 履習者全員へのお知らせは web ページに掲示します。
- 提出したレポート等について個別に連絡をとる場合は電子メールで行います。

4.2 さらに学ぶために

- 計算機プログラミング II — 高度なアルゴリズム・データ構造を用いたプログラミング
- 計算機科学概論 — 計算機の仕組やプログラムの理論など
- ACM プログラミングコンテスト —`http://icpc.score.is.tsukuba.ac.jp/index-j.html`

5 おまけ:スレッドを使ったプログラムの例

```
1  // 分子: 位置と速度を持ち、定期的に位置が変化するオブジェクト
2  class Molecular implements Runnable {// スレッドに渡すため Runnable を持つ
3    Point position, velocity;     // 位置と速度
4    java.applet.Applet a;         // 表示される場所
5    Thread t;                     // 位置を変化させるスレッド
6    // コンストラクタ
7    Molecular(Point position, Point velocity,  java.applet.Applet a) {
8      this.position = position; this.velocity = velocity; this.a = a;
9      this.t = new Thread(this);
10   }
11   public void run() {           // スレッドが実行するメソッド
12     while (true) {
13       position.move(velocity);  // 点の位置を更新して
14       // 点が画面からはみ出した場合には、速度の x,y 成分を反転させる
15       if (position.getX() < 0 || a.getSize().width < position.getX())
16         velocity.x = -velocity.getX();
17       if (position.getY() < 0 || a.getSize().height < position.getY())
18         velocity.y = -velocity.getY();
19       a.repaint();              // アプレットに表示の更新を指示
20       // 適当な間隔だけ休む
21       try {Thread.currentThread().sleep(Mole.interval);}
22       catch (Exception e) {}
23     }
24   }
25   void draw(java.awt.Graphics g) { position.draw(g); } // 画面に描く
26   void start() { this.t.start(); } // スレッドの開始
27   void stop() { this.t.stop(); } // スレッドの停止
28 }
29
30 // マウスの動きから点の位置と速度を計算するオブジェクトの定義
31 // マウスボタンに反応するため java.awt.event.MouseAdapter の子クラスとして定義
32 class Launcher extends java.awt.event.MouseAdapter {
33   Point pressPosition;          // ボタンが押された場所
34   java.util.Date pressTime;     // ボタンが押された時刻
35   Mole m;                       // 位置と速度を受け取るオブジェクト
36   Launcher(Mole m) { this.m = m; } // コンストラクタ
37   // マウスボタンが押されたときに実行されるメソッド
38   public void mousePressed(java.awt.event.MouseEvent e) {
39     // 押された場所と時刻を覚えておく
40     this.pressPosition = new Point(e.getX(), e.getY());// 場所
41     this.pressTime = new java.util.Date(); // 時刻
42   }
43   // マウスボタンが離されたときに実行されるメソッド
44   public void mouseReleased(java.awt.event.MouseEvent e) {
45     // 離された場所と時刻を調べる
46     Point releasePosition = new Point(e.getX(), e.getY());// 場所
47     java.util.Date releaseTime = new java.util.Date(); // 時刻
```

図 3: スレッドを使ったプログラムの例 (その 1)

```
    //(続き)
    // ボタンが押されていた時間 (ミリ秒) を計算
    long interval = releaseTime.getTime() - pressTime.getTime();
    // 点の速度を計算: 単位は (Mole.interval) ミリ秒あたりのピクセル数として
    // (releasePosition-pressPosition)*Mole.interval/interval で計算
    pressPosition.scale(-1);
    pressPosition.move(releasePosition);
    pressPosition.scale(Mole.interval*1.0/interval);
    // 点の位置と速度をアプレットに渡す
    m.addMole(releasePosition, pressPosition);
  }
}

public class Mole extends java.applet.Applet {
  static int interval = 100;    // 表示間隔 (ミリ秒)
  // 表示されている全ての「分子」オブジェクトを覚える「配列」オブジェクト
  java.util.Vector moles = new java.util.Vector();
  public void init() {
    // マウスの動きを読むオブジェクトを作る
    Launcher launcher = new Launcher(this);
    // アプレットに対するマウス操作が launcher に伝わるように登録
    this.addMouseListener(launcher);
    // アプレット上でマウスボタンが押されたときと離されたときに
    // launcher のメソッドが実行されるように指示
    this.enableEvents(java.awt.event.MouseEvent.MOUSE_RELEASED |
                      java.awt.event.MouseEvent.MOUSE_PRESSED);
  }
  // マウスを離したときに launcher から実行されるメソッド
  void addMole(Point position, Point velocity) {
    // 読みとった場所と速度を持つ点を作り
    Molecular m = new Molecular(position, velocity, this);
    moles.addElement(m);        // 配列に覚えておく
    m.start();                  // スレッドの実行を開始
  }
  public void paint(java.awt.Graphics g) { // アプレットが表示されたとき
    for (java.util.Enumeration m = moles.elements();
         m.hasMoreElements(); ) { // 全ての「分子」オブジェクトについて
      Molecular mole = (Molecular)m.nextElement();
      mole.draw(g);             // 画面に表示する
    }
  }
  public void stop() {          // アプレットが中止されたとき
    for (java.util.Enumeration m = moles.elements();
         m.hasMoreElements(); ) { // 全ての「分子」オブジェクトについて
      Molecular mole = (Molecular)m.nextElement();
      mole.stop();              // スレッドを止める
    }
  }
}
```

図 4: スレッドを使ったプログラムの例 (その 2)