計算機プログラミング I (12)

穴埋め的プログラミングとアプレット

2001 年 1 月 9 日 増原 英彦

1 穴埋め的プログラム

1.1 積分のアルゴリズム

関数を数値積分するアルゴリズムを考える。被積分関数を f として、N の個の区間に分割することにする。$a + N\delta = b$ とおけば下のように近似できる。

$$\int_a^b f(x)\mathrm{d}x \simeq \delta \sum_{i=0,\ldots,N-1} f(a + i\delta)$$

これをプログラムにすると下のようになるだろう。プログラム中の f を好きな関数に置きかえることで様々な関数の積分を計算することができる。しかし Java 言語では、メソッドをパラメータとして受け渡しすることはできない[1]ので、積分のアルゴリズムを任意の関数について再利用することが簡単ではない。

```
1 int intervals = 1000;          // 微少区間の数 (N)
2 double width=b-a;              // 区間の幅 (Nδ)
3 double sum = 0;                // = f(a)+f(a+δ)+...+f(x)
4 for (int i=0; i<intervals; i=i+1) {
5   double x = a+i*width/intervals; // a+iδ=a+iNδ/N より
6   sum = sum + f(x);            // f は被積分関数
7 }
8 double integral = sum*width/intervals;   // Nδ/NΣ f=δΣ f ≒∫ fdx より
```

1.2 積分を行うクラス

継承を使うと子クラスでインスタンスメソッドを上書きできることに注目しよう。まず、

- 被積分関数 f
- f を数値積分するアルゴリズム integral

をそれぞれインスタンスメソッドとして持つクラスを定義しておく。このクラスを継承して、好みの f の定義を持つクラスを作っておけば、積分のアルゴリズム integral はそのままで異なる被積分関数の積分ができるようになる。

プログラムとしては下のようになる:

[1] 整数・実数・配列などのデータは、変数に代入したしり、関数の引数や返り値として受け渡しをすることができる。このようなデータを、第 **1** 級データ(first class data) という。Lisp や ML のような関数型言語と呼ばれる言語では関数も第 1 級データになっているため、被積分関数をパラメータ化することも容易に行える。

```
class Constant {                // 被積分関数と数値積分アルゴリズム
  // 被積分関数（ここでは定数関数。子クラスはこれを上書きする。）
  double f(double x) { return 1; } // f(x)=1
  // 関数 f の [a,b] 区間の積分を計算する
  double integral(double a, double b) {
    int intervals = 1000;       // 微少区間の数
    double width=b-a;           // 区間の幅
    double sum = 0;             // = f(a)+f(a+δ )+...+f(x)
    for (int i=0; i<intervals; i=i+1) {
      double x = a+i*width/intervals; // a+i δ より
      sum = sum + this.f(x);    // (関数はインスタンスメソッド)
    }
    return sum*width/intervals; // 和をδ したものが積分値
  }
  // g の (x,y) 上に色 c の点を打つ
  void dot(java.awt.Graphics g, java.awt.Color c, double x, double y) {
    g.setColor(c); int size = 3; g.fillOval((int)x,(int)y,size,size);
  }
  // 関数 f(x) とその積分∫ fxdx をグラフ表示する
  // size x size の画面に x=xmin..xmax, y=-ymax..ymax の範囲を表示
  void plot(int size, double xmin, double xmax, double ymax) {
    // グラフィック画面の作成
    java.awt.Graphics g = Drawer2.createFrame(size,size);
    // x=xmin..xmax について点をプロットする
    for (int i = 0; i < size; i=i+2) {
      double x = (xmax-xmin)*i/size; // (x-xmin)/(xmax-xmin)=i/size より
      double fx = this.f(x);    // f(x) の計算 (インスタンスメソッド)
      double intFx = this.integral(xmin,x); // 積分値の計算
      // 画面に点を打つ (ymax-y)/2ymax=j/size より
      dot(g,java.awt.Color.red,i,(ymax-fx)*size/2/ymax);// 被積分関数 (赤)
      dot(g,java.awt.Color.blue,i,(ymax-intFx)*size/2/ymax); // 積分関数 (青)
    }
  }
}
```

このクラス定義を使えば、Constant o = new Constant(); のようにして「定数関数」を表わすオブジェクトを作ることができ、o.f(100) のようにして関数の値を計算したり、o.integral(0,100) のようにしてその積分を計算させることができる。さらに、グラフィック画面に表示させるメソッドもあるので、以下のようにして関数とその積分の型を表示させることもできる (以前に示したクラスDrawer2 を使用している)。

```
Constant o = new Constant();      // 定数関数のオブジェクトを作り、
o.plot(200, 0, 10, 10);           // 200x200 の画面に x=0..10,y=-10..10 を表示
```

1.3 被積分関数の穴埋め

クラスConstant の子クラスを作り、メソッドf を上書きするだけで、その関数の積分を求めることができるようになる。別の見方をすれば、クラスConstant は積分 (とグラフ表示) を行うメソッドを提供しており、「積分をとる対象となる関数」のところに穴をあけているとも言える。

```
1  class Sin extends Constant {    // 正弦関数の積分
2    //メソッド f を上書きして被積分関数を指定する
3    double f(double x) { return Math.sin(x); }
4    public static void main(String[] args) {
5      Sin o = new Sin();    // オブジェクトを作り
6      o.plot(200, 0, 2*Math.PI, 2); // x=0..2π ,y=-2..2 を表示
7    }
8  }
```

練習 **12-1:** (積分)　正弦関数にならって、他の関数 (例えば二次関数、cos, tan, log) などの積分を行うプログラムを作り、積分アルゴリズムが再利用可能なことを確めよ。

練習 **12-2:** (積分関数の積分)　関数 $g(x)$ の積分の積分、つまり

$$\int_a^b \left(\int_0^y g(x) \mathrm{d}x \right) \mathrm{d}y$$

は下のようなクラスによって計算できる。この定義を完成させ、適当な関数の積分の積分を計算させよ。

```
class DoubleIntegral extends Constant {
  Constant g;// このオブジェクトは関数 g の積分値を関数の値とする
  // コンストラクタで対象となる関数のオブジェクトを受け取る
  DoubleIntegral(Constant g) { this.g = g; }
  // この関数の関数 g の 0 から x までの積分: this.f(x)=g.integral(0,x)
  double f(double x) { return g.integral(0,x); }
}
```

練習 **12-3:** (微分)　関数 $f(x)$ の微分は下のような式で近似できる。クラスConstant に微分を行うメソッドを追加せよ。さらに、メソッドplot が微分関数も表示するように改造せよ。

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \simeq \frac{f(x(1+\varepsilon))}{1+\varepsilon} \qquad (\varepsilon\text{は小さな定数})$$

2　アプレット

アプレットとは web ブラウザを通して実行できる小さなプログラムのことである。アプレットが実行されるときには、web ブラウザ上にアプレットを表示する画面領域を用意したり、ページの切り替えとともにアプレットを起動・停止するといった処理が必要となる。

Java 言語の java.applet.Applet クラスは、全てのアプレットで共通する画面の準備など提供する。また、初期化や画面描画などのアプレットごとに異なる処理を子クラスが穴埋めする形でプログラムを作る。

2.1　簡単なアプレット

簡単なアプレットの例として、画面に三角形を表示するものを示す。

2.1.1 プログラム

プログラムは下のようなものになる。

- 1 行目のクラス定義が public class Triangle ... のようになっている。この public は、web ブラウザが、ここで定義するクラスTriangle のオブジェクトを作ることができるようにするための指示[2]である。
- Triangle クラスは java.applet.Applet クラスの子クラスとして定義される。この java.applet.Applet クラスがアプレットに共通する機能を提供している。
- ?–?行目のメソッド paint は、アプレットが画面に表示されたときに実行されるメソッドである。このメソッドを子クラスで上書きすることで、様々な表示をさせることができる。
  - このメソッドは、web ブラウザが自動的に実行するため、public という指示をつけなければならない。
  - このメソッドの引数 g は、表示する画面を表わすオブジェクトである。
  - ?行目の this.getSize() は java.applet.Applet クラス (の親クラス) に定義されているメソッドで、アプレットの表示部分の大きさを調べるメソッドである。返り値は幅 (width) と高さ (height) の 2 つのインスタンス変数を持つ java.awt.Dimension クラスのオブジェクトである。
  - ?–?行目では、三角形の頂点を計算し、オブジェクト g のメソッド drawLine を使って三角形の辺を描いている。

```
 1  public class Triangle extends java.applet.Applet { // 三角形を表示
 2    int numEdges = 3;              // 辺の数
 3    // アプレットが画面に表示されたとき、以下のメソッドが実行される
 4    public void paint(java.awt.Graphics g) {
 5      java.awt.Dimension size = this.getSize(); // 表示部分の大きさを得る
 6      int ox = size.width/2, oy = size.height/2; // 中心の位置を計算
 7      double dA = 2*Math.PI/numEdges; // 頂点の中心角を計算
 8      // 0 番目の座標を計算
 9      int x0 = ox+(int)(ox*Math.cos(0)), y0 = oy+(int)(oy*Math.sin(0));
10      for (int i = 1; i <= numEdges; i=i+1) { // i=1,..,numEdges 番目の点について
11        // i 番目の座標を計算
12        int x1 = ox+(int)(ox*Math.cos(i*dA)), y1 = oy+(int)(oy*Math.sin(i*dA));
13        g.drawLine(x0,y0,x1,y1);  // i-1 から i 番目へ線を引く
14        x0 = x1; y0 = y1;         // 次の繰り返しのために i 番目の座標を覚える
15      }
16    }
17  }
```

[2]Java 言語では、クラスをパッケージという単位でまとめられており、クラスの定義は同じパッケージの中からしか「見えない」ようになっている。ユーザが作ったアプレットは web ブラウザとは別パッケージに所属しているため、ブラウザからアプレットオブジェクトを作ることができるようにするためには、public という指示が必要になる。

2.1.2 HTML ファイル

アプレットは下の例のように HTML ページに埋め込む形で実行される。ファイル名は好きなものでよいが、ここでは `triangle.html` としておく。

- `<applet ...>` ~ `</applet>`がアプレットを表示させるタグである。
- `code=` には、アプレットが入っているクラスのファイル名を書く。(ファイル名なので `.class` まで書くことに注意)。
- `width=` と `height=` には、アプレットの表示部分の幅と高さを書く。

```
1 <html><body>
2 これは三角形を描く applet です。
3 <applet code=Triangle.class
4         width=200
5         height=200>
6 </applet>
7 </body></html>
```

2.1.3 実行

アプレットを実行させるには、web ブラウザの他に `appletviewer` というプログラムがある。ターミナルウィンドウ (kterm) から `appletviewer triangle.html` Return と入力すると、`triangle.html` 中に書かれているアプレットが実行される。

また、web ブラウザを使って `triangle.html` というファイルを表示させても、同様にアプレットが実行される。このとき、エラーメッセージなどはブラウザの「Java コンソール」などのウィンドウに出力されることに注意。

自分が用意したアプレットを、web サーバを使って公開する場合には、通常の web ページを公開する場合と同様の手続を踏むことになる。この際、**HTML** ファイルだけでなく関係するクラスファイル[3]も全て **web** サーバ上に置く必要があることに注意[4]。

練習 **12-4:** (簡単なアプレット)　三角形を描くアプレットを参考に、好みの絵を描くアプレットを作れ。(絵を描くためにどのようなメソッドがあるかは、`java.awt.Graphics` クラスのメソッドを調べればよい。)

練習 **12-5:** (タートルグラフィクス)　第 10 回に紹介した「タートル・グラフィクス」では、`Drawer2` クラスを使って作ったウィンドウに絵を描くようになっていた。これを、アプレット画面に表示するように改造せよ。

---

[3]例えばクラス`Turtle` がアプレットで、その中で`Point` クラスを使っていた場合、`Turtle.java` と`Point.java` の両方を置く必要がある。ただし、`java.` で始まるクラスは web ブラウザに用意されていることになっているので置く必要はない。

[4]別のディレクトリにあるクラスファイルを使う場合は`<applet code=`c `codebase=`d`...>`のようにしてディレクトリ d を指定する。

2.2 対話的なアプレット

Java 言語には、ボタンやメニューなどを画面に表示するクラスが用意されている[5]。これらを使えば、単に絵を表示するだけでなく、対話をしながら処理を進めるようなアプレットも作ることができる。ここでは、ボタンを使うプログラムの例を紹介する。

下に示すプログラムでは、EdgeIncrementer, Polygon という 2 つのクラスを定義している。さらに Java 言語に用意されている java.awt.Button クラスを使用している。これらのクラスの関係は次のとおりである。

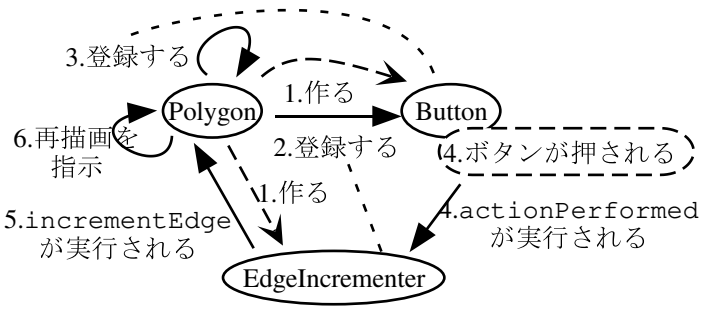

図 1: 対話的なアプレットのクラス間の関係

1. クラス Polygon はTriangle の子クラスである。Web ブラウザがこのクラスのオブジェクトを作ると、まず init というメソッドを実行する (?行目)。このメソッド中では、java.awt.Button クラスのオブジェクトと EdgeIncrementer というクラスのオブジェクトを作る。

2. Button オブジェクトに EdgeIncrementer オブジェクトを登録する (?行目)。

3. Polygon オブジェクト自身に Button オブジェクトを登録する。これによって、Polygon オブジェクトが表示されるときに、自動的にボタンも表示されるようになる。

4. ボタンがクリックされると、ボタンに登録されたオブジェクトのactionPerformed メソッドが実行される。従って、2. で登録された EdgeIncrementer オブジェクトのactionPerformed メソッドが実行される。

5. そのメソッド中には、Polygon オブジェクトのincrementEdge メソッドを実行するように書いてある (?行目) ので、それが実行される。

6. incrementEdge メソッドは、Polygon オブジェクトのインスタンス変数numEdges を 1 増やした後に、自分自身のrepaint メソッドを実行する。このメソッドが実行されると、ブラウザがアプレットのpaint メソッドを実行して画面の描き直しをする。親クラスのTriangle のpaint メソッドは、インスタンス変数numEdges の数だけの辺を持った多角形を描くようになっているので、結果として辺の数が増えた多角形が描かれる。

クラス EdgeIncrementer の定義には

[5]これらは**Abstract Window Toolkit** (AWT) と呼ばれるクラスライブラリの一部で、アプレットだけでなく、画面を使って対話するアプリケーションからも使うことができる。

```
class EdgeIncrementer implements java.awt.event.ActionListener
```

のように書いてある。この下線部は、ボタンが押されたときの処理を定義していて、ボタンに登録できるオブジェクトであることを表わすためのオマジナイである。(actionPerformed というメソッドを持っていることに対応する。)

```
1  // ボタンが押されたときに「辺を増やす」処理をオブジェクトとして定義
2  class EdgeIncrementer implements java.awt.event.ActionListener {
3    Polygon p;                    // 多角形を表示するアプレット
4    EdgeIncrementer(Polygon p) { this.p = p; } // アプレットを覚えるコンストラクタ
5    // ボタンが押されたときに呼ばれるメソッド
6    public void actionPerformed(java.awt.event.ActionEvent e) {
7      p.incrementEdge();          // p の辺を増やす (だけ)
8    }
9  }
10
11 // 多角形を表示するアプレット (三角形アプレットの拡張)
12 public class Polygon extends Triangle {
13   public void init() {  // このアプレットが最初に使われるときに呼ばれるメソッド
14     // 「辺を増やす」ボタンを作る
15     java.awt.Button incrementButton = new java.awt.Button("one more edge");
16     // ボタンが押されたときの処理を定義したオブジェクトを作る
17     EdgeIncrementer inc = new EdgeIncrementer(this);
18     // ボタンに登録をする
19     incrementButton.addActionListener(inc);
20     // ボタンをアプレットの部品として登録する (自動的に表示される)
21     this.add(incrementButton);
22   }
23   void incrementEdge() {  // 辺を増やす処理
24     numEdges = numEdges+1;      // 辺の数 (インスタンス変数) を増やして
25     this.repaint();             // 再描画を要求する
26   }
27 }
```

練習 **12-6:** 辺を減らす　上にならって辺の数を減らすボタンを作れ。

練習 **12-7:** 入力領域　java.awt.TextField というクラスを使えば、キーボードからの文字列を入力させるオブジェクトを作ることができる。多角形の辺の数をキーボードから入力させるプログラムを次のようにして作れ:

- Polygon クラスの子クラスを作る
- java.awt.TextField 型のインスタンス変数numEdgeInput を追加する
- init メソッドの中で、
  - java.awt.TextField クラスのオブジェクトを作り、インスタンス変数 numEdgeInput に覚えておく。
  - Polygon クラス同様、EdgeIncrementer オブジェクトを作る。
  - EdgeIncrementer オブジェクトを numEdgeInput に登録する。

- incrementEdge メソッドの中では、numEdgeInput に入力された文字を読みとり (getText メソッドを使う)、Integer.parseInt メソッドを使って整数に変換してインスタンス変数numEdges を更新し、repaint メソッドを実行する。

3　第3回レポート課題

以下の2問両方についてプログラムを作成し、説明・実行結果とともに提出せよ。ただし、「(optional)」と書かれた小問は余力があるもののみ解けばよい。

提出方法:　レポートは教育用計算機システムの電子メールによって以下の提出用アドレスに送ること。

提出用: masuhara-cp1-report3@lecture.ecc.u-tokyo.ac.jp
相談用: masuhara-cp1@lecture.ecc.u-tokyo.ac.jp

アドレスの書き間違いに注意せよ。

期限:　2001年1月23日(火) — 特別な事情がある場合を除いて、遅れて提出することは認めない。

配点:　全体の成績の25点分に相当する。

注意:

- レポート提出の際、cc:フィールドに自分のアドレスも書いて、自分にも同じメッセージを送っておくこと。(紛失した際の確認のため。)
- レポートは1つの電子メールメッセージにまとめること。添付ファイルはプレーンテキストとして読める形式であれば使ってもよいが、MIME base64 encoding 等の形式のものは無視するので注意せよ。(この意味が分からなければ、添付ファイルを使わない方がよいだろう。)
- 期限内にレポートを出し直したい場合は、レポート全体を再度送ればよい。逆に言うと、最後に送られたメールだけを見て採点するので、追加分だけを送らないこと。
- 説明や実行結果のないレポートは採点しない。プログラムおよびレポートは、必ず自分で作成すること。他人のレポート(プログラムを含む)を写したものがあった場合は、写した者、写された者の両者ともに0点とすることがある。
- 提出されたメールが正しく受け付けられた場合、数分以内に提出内容の全文を引用した確認のメッセージが折り返し送られるのでそれを必ず確認せよ。もし上手く受け付けられない/確認メッセージ来ない場合は相談用のアドレスに〆切前に相談すること。
- レポートに絵や画像を含めたい場合は、webページ上に画像を用意し、レポートにはそのURLのみを書くこと。

課題 3-1 (銀行口座)

「銀行口座」が持つべき情報と、「銀行口座」に対する操作を決め、それをクラス・メソッドとして定義せよ。

1. (必須) まず、1つの「口座」は少くとも「残高」を表わす整数を持つものとする。「口座」に対する操作として「作成」「残高照会」「入金」「引出」の内容を決めて(日本語で)説明せよ。

   次に、上の説明に従って「口座」のクラスを定義せよ。「口座」に対する操作はコンストラクタやインスタンスメソッドとして定義せよ。

定義したクラスを用いて、1000 円, 50 円がそれぞれ預金されている 2 つの銀行口座 a, b を作り、a から b へ 600 円移す操作を 2 回行うプログラムを作れ。その際、2 つの口座の残高の推移を表示させよ。

注意: 説明の形式は自由。「口座」の持つ情報に「所有者の名前」などを必要に応じて追加してよい。また「口座」に対する操作も必要なものを追加してよい。いずれの場合も、追加したものの説明を (日本語で) 書くこと。操作の際の例外的な事態 (負の金額を預金する等) の扱いは各自で適当に定め、どのようにするかを説明中に書くこと。

2. (optional) 暗証番号付き口座

「銀行口座」は普通、暗証番号によって確認できない限り預金の引出しができない。そこで、暗証番号付きの口座を上で作ったクラスの子クラスとして定義せよ。この「口座」に対する「引出」「暗証番号付きの引出」などの操作の内容を決めて説明し、クラスを定義せよ。

定義したクラスを用いて、暗証番号のない場合や、暗証番号が合っている場合、暗証番号が間違っている場合に残高が正しく変化することを確認するプログラムを作れ。

注意: 暗証番号の変更などの操作も自由に追加してよい。文字列を比較する場合は==ではなく、`equals` というメソッドを使う必要がある。下のプログラムを参考にせよ:

```
String s = "hello", t = "hello", u = "world";
if (s == s) System.out.println("s と s は同じオブジェクトです。");
if (s == t) System.out.println("s と t は同じオブジェクトです。");
if (s.equals(t)) System.out.println("s と t は同じ文字列です。");
if (s.equals(u)) System.out.println("s と u は同じ文字列です。");
```

課題 2-2 (分数)

「分数」は「分母」と「分子」を表わす 2 つの整数からなるデータであり、分母や分子を調べたり、加減乗除算などの演算がある。

1. (必須) 分数を表わすクラスを定義し、それを使って分数の計算を行うプログラムを作成せよ。

   - 分数どうしの演算 (足し算・掛け算など) は、演算の結果を新しくオブジェクトを作って返すメソッドとして定義せよ。
   - 約分は行わないでよい。
   - その他、必要なメソッド (文字列や実数への変換など) を定義してよい。
   - 分数の計算としては、すくなくとも以下のものを行うこと。

   (計算)

   $$f(g,x) = \frac{g + x/g}{2}$$

   としたとき、

   $$f(1,x), \quad f(f(1,x),x), \quad f(f(f(1,x),x),x), \tag{1}$$

   をそれぞれ分数で計算するプログラムを作り、$x = 3$ の場合について結果を分数と実数 (近似値) で表示させよ。(ニュートン法で x の平方根を計算している。)

   - 以下は分数を表わすクラスの定義例である。定義方法が全く分からない場合は参考にせよ。

```
class Rational { // 分数を表わすクラス
  // インスタンス変数:  分子と分母
  long numer, denominator;
  // コンストラクタ:  分数オブジェクトを与えられた分子と分母で初期化
  Rational(long numer, long denominator) { [      ] }
  // 与えられた分子と分母を表す新しい分数を作るメソッド
  Rational make(long numer, long denominator) {
    return new Rational(numer, denominator);
  }
  long numer() { [      ] } // 分子を返す
  long denominator() { [      ] } // 分母を返す
  // このオブジェクトと与えられた分数 x との和を返す
  Rational add(Rational x) {
    [分母分子を計算]
    return this.make([和の分子], [和の分母]); // 新しくオブジェクトを作って返す
  }
  // 文字列に変換
  public String toString() { // "分子/分母" のような文字列に変換する
    return this.numer() + "/" + this.denominator();
  }
  [以下、他のメソッドが続く]
}
```

2. (optional) 上で定義した分数の子クラスとして、分数を作る際に約分 (分母と分子を最大公約数で割ったものにしておく) するようなものを定義して、式 (1) を計算せよ。

   (新しく分数を作るメソッドmake をインスタンスメソッドとして定義することで、ほとんどのメソッドの定義を変更する必要がないことが分かるだろう。)