

# Java プログラミング入門

神戸大学工学部情報知能工学科  
田村 直之 (tamura@kobe-u.ac.jp)

1999 年 9 月

## 1 Java とは何?

- 1995 年にサン・マイクロシステムズ社が開発し、現在最も注目を集めているプログラミング言語
  - <http://www.sun.co.jp/java/>
- Netscape などの WWW ブラウザ上で動作するプログラム (Applet と呼ばれる) を書くことができ、実際に非常に多くのプログラムが WWW 上で利用できる。
  - <http://www.javacats.com/>
- アーキテクチャ独立. 計算機の CPU や OS の異なった様々なプラットフォーム上で、同一プログラムを利用できる。
  - 抽象機械 JVM の機械語にコンパイルされ、JVM インタプリタで実行される.
  - ユニコードの採用
- セキュリティー

### 1.1 Java 言語の特徴: C との違いを中心に

- オブジェクト指向言語
- C に似た構文
- ポインタがない
- 配列範囲などの動的なチェック
- ガーベジ・コレクション
- マルチスレッドのサポート
- 例外処理

### 1.2 簡単なプログラム

#### HelloKobe.java プログラム

- (1) HelloKobe.java を作成する.

(2) コンパイル (HelloKobe.class が作成される)

```
javac HelloKobe.java
```

(3) 実行 (HelloKobe.class が実行される)

```
java HelloKobe
```

#### HelloKobe.java

```
1 public class HelloKobe {
2     public static void main(String args[]) {
3         System.out.println("Hello Kobe!");
4     }
5 }
```

### AccApplet.java プログラム

(1) AccApplet.java を作成する.

(2) AccApplet.html を作成する.

(3) コンパイル

```
javac AccApplet.java
```

(4) 実行

appletviewer AccApplet.html あるいは

Netscape などのブラウザで, AccApplet.html ファイルを開く

#### AccApplet.html

```
1 <html>
2 <head>
3 <title>AccApplet</title>
4 </head>
5 <body>
6 <center>
7 <applet code="AccApplet.class" width=400 height=200>
8 </applet>
9 </center>
10 </body>
11 </html>
```

```

AccApplet.java
1  import java.applet.Applet;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  public class AccApplet extends Applet {
6      Accumulator acc;
7      TextField accField;
8      TextField inputField;
9
10     public void init() {
11         acc = new Accumulator();
12         accField = new TextField(10);
13         accField.setEditable(false);
14         inputField = new TextField(10);
15         inputField.addActionListener(new ActionListener() {
16             public void actionPerformed(ActionEvent e) {
17                 inputAction();
18             }
19         });
20         add(accField);
21         add(inputField);
22         validate();
23     }
24
25     void inputAction() {
26         try {
27             int x = Integer.parseInt(inputField.getText());
28             acc.add(x);
29         } catch (NumberFormatException e) {}
30         inputField.selectAll();
31         repaint();
32     }
33
34     public void paint(Graphics g) {
35         accField.setText(Integer.toString(acc.value()));
36     }
37 }
38
39 class Accumulator {
40     int accumulator = 0;
41
42     void add(int x) {
43         accumulator += x;
44     }
45
46     int value() {
47         return accumulator;
48     }
49 }

```

## 練習問題

問 1 上記の Java プログラムを実際に行動せよ。

## 2 オブジェクト指向とは何?

D. Gelernter と S. Jagannathan は、「プログラムとは機械である」と述べている。すなわちプログラムは、物質的な実体は存在しないが、ある機能を実現するための機械の一種(ソフトウェア機械)と考えることができる。

すると、プログラムを設計しコーディングすることは、機械を設計し作成することに相当する。機械の設計・作成において、我々は普通、すでにある部品を利用し、それらの機能を組み合わせることによって新しい機能を持った機械あるいは部品を作り上げる。

**オブジェクト指向プログラミング**は、まさにそれと同じことをプログラムの作成において行うための考え方である。すなわち、なんらかの機能を持った**オブジェクト**を組み合わせることで新しい機能を持ったオブジェクトを構築し、全体として一つのソフトウェア機械を実現するのである。

たとえば、前述の `AccApplet.java` は、以下のオブジェクトを組み合わせることで実現されている。

- `acc`: 累計を数える累算器 (アキュムレータ)
- `accField`: 累算器の値の表示場所
- `inputField`: 数の入力場所

最後の二つは Java システムにあらかじめ用意されているオブジェクト (正確には用意されているクラスのオブジェクト) であり、最初の累算器はこのプログラム中で機能を定義しているオブジェクトである。

Java には数多くのクラスが用意されており (JDK1.1.7 で 477)、プログラマはこれらおよび自分で定義したオブジェクトを自由に組み合わせることで自分のプログラムを作成できる。

Java 以外のオブジェクト指向プログラミング言語としては、C++, Objective C, Smalltalk などがある。

### 2.1 オブジェクトとクラス

では、Java でのオブジェクトとはどんなものなのだろうか? 累算器オブジェクトを見ると、以下のようなものから構成されている。

- 累計を覚えておく `int` 型の変数 `accumulator`
- 累計に加算する機能を果たす `add`
- 累計を答える機能を果たす `value`

一般には、Java のオブジェクトは、以下のものから構成される。

- オブジェクトの状態を覚えておくための**インスタンス変数**  
累算器の場合は変数 `accumulator`
- オブジェクトの機能を果たすための**メソッド**  
累算器の場合は `add` と `value`

したがって、オブジェクトを定義するには、インスタンス変数とメソッドを定義すればよい。しかし、プログラム中では、同様の振る舞いをする複数のオブジェクトを必要とすることが多い。たとえば、複数の累算器が必要な場合、それぞれの累算器を別々に定義するのは無駄である。

そこで Java では、オブジェクトの型紙となる**クラス**を定義する。クラスの定義を型紙として、型紙から生成された具体例 (**インスタンス**) が個々のオブジェクトとなる。オブジェクトの生成は `new` 命令で行う。次のプログラム例を見てみよう。

### AccTest.java

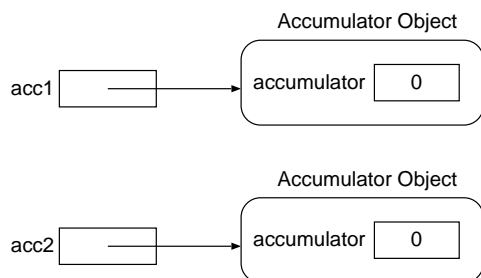
```
1 public class AccTest {
2     public static void main(String args[]) {
3         Accumulator acc1;
4         Accumulator acc2;
5         acc1 = new Accumulator();
6         acc2 = new Accumulator();
7         acc1.add(10);
8         acc2.add(20);
9         acc1.add(30);
10        acc2.add(40);
11        System.out.println(acc1.value());
12        System.out.println(acc2.value());
13    }
14 }
15
16 class Accumulator {
17     int accumulator = 0;
18
19     void add(int x) {
20         accumulator += x;
21     }
22
23     int value() {
24         return accumulator;
25     }
26 }
```

class Accumulator 以降の行が、Accumulator クラスの定義である。このプログラムでは、二つの Accumulator オブジェクトを生成している。

```
Accumulator acc1;
Accumulator acc2;
acc1 = new Accumulator();
acc2 = new Accumulator();
```

変数 acc1, acc2 は、Accumulator オブジェクトを参照する変数として宣言されている。文 acc1 = new Accumulator() により、新しい Accumulator オブジェクトが生成され、そのオブジェクトへの参照が acc1 に代入される。次の文 acc2 = new Accumulator() も同様である。

それぞれの Accumulator オブジェクトは、別々のインスタンス変数を持っている。



オブジェクトに対するメソッドの呼び出しは、以下のようにして行う。

```
acc1.add(10);
acc2.add(20);
```

まず、acc1の参照しているAccumulatorオブジェクトに対して、add(10)というメソッドを実行し、次に、acc2の参照しているAccumulatorオブジェクトに対して、add(20)というメソッドを実行している。それぞれのオブジェクトのインスタンス変数は、別々だから結局40と60が表示される。

変数acc1, acc2は、Accumulatorオブジェクトを参照する変数であるという点に注意する。たとえば、以下の代入を行った後は、acc1とacc2は全く同じオブジェクトを参照することになる。

```
acc1 = acc2;
```

## 練習問題

問2 以下の場合、何が表示されるか。

```
acc1 = new Accumulator();
acc2 = new Accumulator();
acc1.add(10);
acc2.add(20);
acc1 = acc2;
acc1.add(30);
acc2.add(40);
System.out.println(acc1.value());
System.out.println(acc2.value());
```

問3 Accumulatorクラスの定義に、累算器の値を0にリセットするメソッドvoid reset()の定義を加えよ。

## 2.2 クラス階層と継承

前節のAccumulatorクラスを拡張し、これまでの平均値を返すメソッドを追加することを考えよう。

Accumulatorクラス自体の定義を書き換えるのも良いが、ここではそうせずに、Accという新しいクラスを定義したいとする(たとえば、Accumulatorクラスの定義は他人が書いたものなので変更したくない、などの理由で)。

Accクラスの定義を書いてみると、例えば次のようになるだろう。

```
class Acc {
    int accumulator = 0;
    int count = 0;

    void add(int x) {
        accumulator += x;
        count++;
    }

    int value() {
        return accumulator;
    }

    double average() {
        return (double)accumulator / (double)count;
    }
}
```

この程度ならば、新たにAccクラスを定義するのは簡単である。しかし、元のAccumulatorクラスが多くの機能を持っており、プログラムの規模も大きい場合、ほぼ同じ機能を持った別のAccクラスを新たに定義するのは無駄だろう。

Java では、Acc クラスを Accumulator クラスの**拡張クラス**として定義することで、インスタンス変数やメソッドの定義を**継承**することができる。

拡張クラスは、**サブクラス**あるいは**子クラス**とも呼ばれる。逆に、元のクラス (今の場合 Accumulator クラス) は、**スーパークラス**あるいは**親クラス**と呼ばれる。すなわち、Acc クラスは Accumulator クラスのサブクラスであり、Accumulator クラスは Acc クラスのスーパークラスである。

サブクラスでは、スーパークラスにあるインスタンス変数やメソッドは、再度定義する必要はなく、そのまま利用できる。

サブクラスおよびスーパークラスというクラス間の関係を**クラス階層**と呼ぶ。クラス階層は、生物学での生物の分類によく似ている。人間のスーパークラスは哺乳類であり、哺乳類のスーパークラスは脊椎動物、脊椎動物のスーパークラスは動物、動物のスーパークラスは生物と続く。

Java ではクラス階層は木構造になっている。すなわち、どのクラスも複数のスーパークラスを持つことはない。また、Java では、クラス階層の頂点にあるスーパークラスは Object クラスと呼ばれる特別なクラスである。先ほどの Accumulator クラスは Object クラスのサブクラスになる。

Java でサブクラスを定義する時は、extends キーワードを用いる。次のプログラムは、Acc クラスを Accumulator クラスのサブクラスとして定義した例である。

#### AccTest2.java

```
1 public class AccTest2 {
2     public static void main(String args[]) {
3         Acc acc;
4         acc = new Acc();
5         acc.add(10);
6         acc.add(20);
7         System.out.println(acc.average());
8     }
9 }
10
11 class Accumulator {
12     int accumulator = 0;
13
14     void add(int x) {
15         accumulator += x;
16     }
17
18     int value() {
19         return accumulator;
20     }
21 }
22
23 class Acc extends Accumulator {
24     int count = 0;
25
26     void add(int x) {
27         accumulator += x;
28         count++;
29     }
30
31     double average() {
32         return (double)accumulator / (double)count;
33     }
34 }
```

スーパークラスの Accumulator クラスに定義されているインスタンス変数 accumulator とメソッド value

は、サブクラスの Acc でも同一なので、記述が省かれている。すなわち、これらは Accumulator クラスから継承されている。

ただし add メソッドは、Accumulator クラスとは異なっているので、Acc クラスで定義し直している。このようなメソッドの再定義を、メソッドの**オーバーライド**と呼ぶ。

まとめると以下のようにになっている。

- インスタンス変数 accumulator : 継承
- インスタンス変数 count : 追加
- メソッド add : オーバーライド
- メソッド value : 継承
- メソッド average : 追加

このようにスーパークラスからの違いだけを記述する方法を**差分プログラミング**という。Java の継承のメカニズムは、差分プログラミングを可能にしている。

複数のスーパークラスを持つことを許す(多重継承と呼ぶ)オブジェクト指向言語も存在するが、プログラムの動作の理解が困難になる。Java では分かりやすさの点から単一継承である。

Acc クラスのメソッド add をよく見ると、一部は Accumulator クラスの add と同一である。この場合は一行だけだから、再度記述しても問題ないが、もっと多い場合にはどのようにすれば良いだろう。これは、Acc クラスの add メソッド内から Accumulator クラスの add メソッドを呼び出せば解決する。Java でスーパークラスのメソッドを呼び出すには、**super** キーワードを用いる。

```
void add(int x) {
    super.add(x);
    count++;
}
```

これを super.add(x) ではなく、単に add(x) としたとすると、Acc クラスの add メソッドが呼び出されるので無限ループになる。

また、super とよく似たキーワードに **this** がある。this は、現在操作されようとしているオブジェクト自体を参照する。

## 練習問題

問 4 Acc クラスの average メソッドの定義を、value メソッドを利用するように書き直せ。

問 5 Acc クラスのサブクラスとして、分散を返すメソッド double variance() を持つクラス AccV を定義せよ。  $x_1, x_2, \dots, x_n$  の分散  $v$  は以下の式で求めることができる (ここで  $\mu$  は平均値)。

$$v = \frac{1}{n} \left( \sum x_i^2 - n\mu^2 \right)$$

## 3 Java 言語

### 3.1 基本データ型

型	値	サイズ
byte	符号付き整数 (-128..127)	8 ビット
short	符号付き整数 (-32768..32767)	16 ビット
int	符号付き整数 ( $-2^{31}..2^{31} - 1$ )	32 ビット
long	符号付き整数 ( $-2^{63}..2^{63} - 1$ )	64 ビット
float	浮動小数点数	32 ビット
double	浮動小数点数	64 ビット
char	ユニコード文字	16 ビット
boolean	論理値 (true または false)	1 ビット

### 3.2 配列

Java では、配列はオブジェクトと同様に取り扱われている。

```
int a[];
a = new int[100];
for (int i=0; i<a.length; i++) {
    a[i] = i;
}
```

上のプログラムで、変数 `a` は整数配列を参照する変数として宣言されている。 `a` の初期値は `null` である。次の文で、サイズ 100 の整数配列が生成され、 `a` はその配列を参照する。 `a.length` は、 `a` の参照している配列のサイズを値とする。

多次元配列は以下のように記述する。

```
double m[][];
m = new double[100][200];
```

`m[i]` は、それぞれサイズ 200 の `double` 配列への参照となる。したがって、 `m[i][j]` で各要素の値を取り出せる。

### 3.3 文字列

Java では、 `String` オブジェクトを文字列として使用する。

```
String s;
s = "abracadabra";
System.out.println(s.length());
if (s.equals("sesami")) {
    System.out.println("open " + s);
}
```

`length` は、文字列の長さを返すメソッドである。 `equals` は、文字列の比較を行うメソッドである。 `+` 演算子で連結した文字列を求めることができる。

### 3.4 文

構文は、C と良く似ている。

- if-else 文
- switch 文
- for 文
- while 文
- do-while 文
- break 文, continue 文

goto 文はない。

- import 文
- try-catch 文

## 4 プログラム例

### 4.1 複素数クラス

Complex クラスは他のプログラムからも利用できるように、Complex.java ファイル中で public なクラスとして宣言する。

ComplexTest.java

```
1 public class ComplexTest {
2     public static void main(String args[]) {
3         Complex z1, z2, z;
4         z1 = new Complex(1.0, 2.0);
5         z2 = new Complex(3.0, 4.0);
6         z = z1.plus(z2).plus(5.0);
7         System.out.println(z);
8     }
9 }
```

```

Complex.java
1 public class Complex {
2     public double re = 0.0;
3     public double im = 0.0;
4
5     public Complex() {
6     }
7
8     public Complex(double x) {
9         re = x;
10    }
11
12    public Complex(double x, double y) {
13        re = x;
14        im = y;
15    }
16
17    public Complex plus(double x) {
18        return new Complex(re + x, im);
19    }
20
21    public Complex plus(Complex z) {
22        return new Complex(re + z.re, im + z.im);
23    }
24
25    public Complex minus(Complex z) {
26        return new Complex(re - z.re, im - z.im);
27    }
28
29    public String toString() {
30        return re + "+" + im + "i";
31    }
32 }

```

Complex クラス中の `Complex()`, `Complex(double x)`, `Complex(double x, double y)` は、**コンストラクタ**と呼ばれる。コンストラクタは、オブジェクトを `new` 命令で生成する時に呼び出される。どのコンストラクタが呼び出されるかは、`new` 命令で生成する時の引数によって異なる。

`plus` メソッドも、`double` が引数で与えられた場合と `Complex` オブジェクトが引数で与えられた場合の二通りが定義されている。

`toString` メソッドは、`Complex` オブジェクトを文字列に変換するメソッドである。`toString` メソッドは、文字列に変換する必要がある場合 (例えば `System.out.println` で印刷する場合) に、暗黙的に呼び出される。

## 練習問題

問 6 `Complex` クラスに、積を計算するメソッド `times` を追加せよ。

## 4.2 グラフ表示アプレット

```
Graph0.java
1  import java.applet.Applet;
2  import java.awt.*;
3
4  public class Graph0 extends Applet {
5      int f(int i, int w, int h) {
6          double x, y;
7          x = 4.0 * Math.PI * i / w;
8          y = Math.sin(x);
9          return (int)(h * (3.0 - y) / 5.0);
10     }
11
12     public void paint(Graphics g) {
13         int w = getSize().width;
14         int h = getSize().height;
15         int i0 = 0;
16         int j0 = f(i0, w, h);
17         for (int i = 1; i < w ; ++i) {
18             int j = f(i, w, h);
19             g.drawLine(i0, j0, i, j);
20             i0 = i;
21             j0 = j;
22         }
23     }
24 }
```

Applet の表示には `paint` メソッドが呼び出される。Applet の幅のドット数は `getSize().width` で、高さのドット数は `getSize().height` で得られる。

`drawLine` は、Applet の描画部分である Graphics オブジェクトに線を描く。他には、以下のメソッドが Graphics オブジェクトに対して利用できる。

- `draw3DRect(int x, int y, int width, int height, boolean raised)`
- `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
- `drawLine(int x1, int y1, int x2, int y2)`
- `drawOval(int x, int y, int width, int height)`
- `drawPolygon(int xPoints[], int yPoints[], int nPoints)`
- `drawRect(int x, int y, int width, int height)`
- `drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`
- `drawString(String str, int x, int y)`
- `fill3DRect(int x, int y, int width, int height, boolean raised)`
- `fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
- `fillOval(int x, int y, int width, int height)`
- `fillPolygon(int xPoints[], int yPoints[], int nPoints)`
- `fillRect(int x, int y, int width, int height)`
- `fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`

- setColor(Color c)
- setFont(Font font)

#### Graph1.java

```

1  import java.applet.Applet;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  public class Graph1 extends Applet {
6      Button incButton;
7      int n = 0;
8
9      public void init() {
10         incButton = new Button("Increment Frequency");
11         incButton.addActionListener(new ActionListener() {
12             public void actionPerformed(ActionEvent e) {
13                 ++n;
14                 repaint();
15             }
16         });
17         add(incButton);
18     }
19
20     int f(int i, int w, int h) {
21         double x, y;
22         x = 4.0 * Math.PI * i / w;
23         y = Math.sin(n * x);
24         return (int)(h * (3.0 - y) / 5.0);
25     }
26
27     public void paint(Graphics g) {
28         int w = getSize().width;
29         int h = getSize().height;
30         int i0 = 0;
31         int j0 = f(i0, w, h);
32         for (int i = 1; i < w ; ++i) {
33             int j = f(i, w, h);
34             g.drawLine(i0, j0, i, j);
35             i0 = i;
36             j0 = j;
37         }
38     }
39 }

```

Applet の init メソッドは、Applet が生成された後に呼び出される。Graph1.java プログラムでは、Button オブジェクトを生成し、add メソッドで Applet に加えている。Button オブジェクトに対する addActionListener メソッドは、Button が押された時に実行する手続きを指定している。Applet の repaint メソッドは、表示が変化する時に呼び出しておく必要がある。

#### 練習問題

問 7 Graph1 クラスに、周波数を減らすボタンを追加せよ。

### 4.3 移動するボールのアプレット

```
BallApplet.java
1  import java.applet.Applet;
2  import java.awt.*;
3
4  public class BallApplet extends Applet implements Runnable {
5      Ball b1;
6      Thread animator;
7
8      public void init() {
9          b1 = new Ball(20, 30, 15, 10, 5);
10     }
11
12     public void start() {
13         if (animator == null) {
14             animator = new Thread(this);
15             animator.start();
16         }
17     }
18
19     public void stop() {
20         if (animator != null) {
21             animator.stop();
22             animator = null;
23         }
24     }
25
26     public void run() {
27         while (animator != null) {
28             repaint();
29             try {
30                 Thread.sleep(20);
31             } catch (InterruptedException e) {}
32             int w = getSize().width;
33             int h = getSize().height;
34             b1.move(w, h);
35         }
36     }
37
38     public void paint(Graphics g) {
39         b1.paint(g);
40     }
41 }
42
43 class Ball {
44     int x, y, r, dx, dy;
45
46     Ball(int x0, int y0, int r0, int dx0, int dy0) {
47         x = x0; y = y0; r = r0; dx = dx0; dy = dy0;
48     }
49
50     void move(int w, int h) {
51         x += dx;
52         y += dy;
53         x = (x + w) % w;
54         y = (y + h) % h;
55     }
56
57     void paint(Graphics g) {
58         g.setColor(Color.red);
59         g.fillArc(x-r, y-r, 2*r, 2*r, 0, 360);
60     }
61 }
```

#### 練習問題

- 問 8 複数のボールを移動してみよ。  
問 9 壁で反射するようにせよ。

#### 4.4 立方体表示アプレット

CubeApplet.java

```
1 import java.applet.Applet;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class CubeApplet extends Applet {
6     Scrollbar sliderX;
7     Scrollbar sliderY;
8     View3D v;
9     Cube c;
10
11     public void init() {
12         int a = 15;
13         int b = 15;
14         v = new View3D(a, b, 200, 200);
15         c = new Cube(0.0, 0.0, 0.0, 50.0);
16         c.project(v);
17         setLayout(new BorderLayout());
18         sliderX = new Scrollbar(Scrollbar.VERTICAL, a, 1, -45, 45);
19         sliderX.addAdjustmentListener(new AdjustmentListener() {
20             public void adjustmentValueChanged(AdjustmentEvent e) {
21                 v.setA(sliderX.getValue());
22                 c.project(v);
23                 repaint();
24             }
25         });
26         add("East", sliderX);
27         sliderY = new Scrollbar(Scrollbar.HORIZONTAL, -b, 1, -45, 45);
28         sliderY.addAdjustmentListener(new AdjustmentListener() {
29             public void adjustmentValueChanged(AdjustmentEvent e) {
30                 v.setB(-sliderY.getValue());
31                 c.project(v);
32                 repaint();
33             }
34         });
35         add("South", sliderY);
36     }
37
38     public void paint(Graphics g) {
39         c.paint(g);
40     }
41 }
42
```

```

View3D.java
1  import java.awt.*;
2
3  public class View3D {
4      int a, b;
5      int dx, dy;
6
7      public View3D(int a0, int b0, int dx0, int dy0) {
8          a = a0; b = b0;
9          dx = dx0; dy = dy0;
10     }
11
12     public void setA(int a0) {
13         a = a0;
14     }
15
16     public void setB(int b0) {
17         b = b0;
18     }
19
20     public Point projectPoint(double x, double y, double z) {
21         double sa = Math.sin(a*Math.PI/180.0);
22         double ca = Math.cos(a*Math.PI/180.0);
23         double sb = Math.sin(b*Math.PI/180.0);
24         double cb = Math.cos(b*Math.PI/180.0);
25         double x1 = cb*x + sa*sb*y + ca*sb*z;
26         double y1 = ca*y - sa*z;
27         // double z1 = - sb*x + sa*cb*y + ca*cb*z;
28         return new Point((int)x1+dx, (int)y1+dy);
29     }
30 }

```

```

Cube.java
1  import java.awt.*;
2
3  public class Cube {
4      double x, y, z;
5      double w;
6      Point p000, p001, p010, p011, p100, p101, p110, p111;
7
8      public Cube(double x0, double y0, double z0, double w0) {
9          x = x0; y = y0; z = z0; w = w0;
10     }
11
12     public void project(View3D v) {
13         p000 = v.projectPoint(x-w, y-w, z-w);
14         p001 = v.projectPoint(x-w, y-w, z+w);
15         p010 = v.projectPoint(x-w, y+w, z-w);
16         p011 = v.projectPoint(x-w, y+w, z+w);
17         p100 = v.projectPoint(x+w, y-w, z-w);
18         p101 = v.projectPoint(x+w, y-w, z+w);
19         p110 = v.projectPoint(x+w, y+w, z-w);
20         p111 = v.projectPoint(x+w, y+w, z+w);
21     }
22
23     public void drawLine(Point p0, Point p1, Graphics g) {
24         g.drawLine(p0.x, p0.y, p1.x, p1.y);
25     }
26
27     public void paint(Graphics g) {
28         drawLine(p000, p001, g);
29         drawLine(p010, p011, g);
30         drawLine(p100, p101, g);
31         drawLine(p110, p111, g);
32         drawLine(p000, p010, g);
33         drawLine(p001, p011, g);
34         drawLine(p100, p110, g);
35         drawLine(p101, p111, g);
36         drawLine(p000, p100, g);
37         drawLine(p001, p101, g);
38         drawLine(p010, p110, g);
39         drawLine(p011, p111, g);
40     }
41 }

```

## 練習問題

問 10 立方体以外の図形の表示を試してみよ。

- プログラム例
  - 複素数
  - 行列
  - 多角形
  - ロケット
  - 時計
  
- 動的メソッド検索, 多相型
- コンストラクタ
- クラス変数, クラスメソッド
- 情報隠蔽, カプセル化, モジュール化, 抽象データ型
- ソフトウェアの部品化, 再利用, 実装でなく概念のモデル化
- abstract クラス
- インターフェイス
- Applet, init, start, stop, run, paint
- 並行計算, マルチスレッド, 同期
- 委譲イベントモデル
- ガーベジコレクション
- セキュリティー
- javac, JVM, JIT
- 例外処理
- グラフィックス, 時計
- ネットワーク