JavaによるGUIの記述

200911434 青木大祐
メールアドレス： s0911434@coins.tsukuba.ac.jp

提出期限 2012年5月18日
提出日 2012年5月18日

第1章　課題1

1.1　クラスとインスタンスの意味

1.1.1　クラス

クラスとは、あるデータについての属性と手続きを纏めたものである。これによりデータとそれに関する操作が集約され、データ同士の関連性が明確になる。クラスはデータの設計図とも言える。

1.1.2　インスタンス

設計図であるクラスから実際に生成したデータ。

1.2　図形の描画

1.2.1　楕円、七角形、独自の図形（十角形）

以下のように MyDrawing 抽象クラスを実装し、これを元に各図形の実装を行った。インスタンスの生成は Builder パターンを用いることでコードの見通しの良いコードを記述できるように努めた。細かいメソッドなどについてはソースコード中の javadoc を参照のこと。

```
1  /**
2   * Abstract class for Shapes
3   */
4  public abstract class MyDrawing {
5      protected int x, y, h, w;
6      protected Color lineColor, fillColor;
7      protected int lineWidth;
8      protected boolean shadow = false;
9
10     /**
11      * Builder for SomeShape extends MyDrawing
12      * usage:
13      *   SomeShape s = new SomeShape.Builder(x, y).someSettingFoo(20).someSettingBar(Color.RED).build();
14      */
15     public static abstract class Builder {
16         protected final int x;
17         protected final int y;
18
19         protected int w, h;
20         protected Color line, fill;
21         protected int lw;
22         protected boolean shadow;
23
24         /**
25          * Make Builder object for building some instance extends MyDrawing
26          * @param x horizontal position of the upper-left of this layer
27          * @param y vertical position of the upper-left of this layer
28          */
29         public Builder(int x, int y) {
30             this.x = x;
31             this.y = y;
32         }
33
34         /**
35          * Set layer size with this Builder object
36          * @param w width of this layer
37          * @param h height of this layer
38          * @return this own Builder object which size was set
39          */
40         public Builder size(int w, int h) {
41             this.w = w;
42             this.h = h;
43             return this;
```

```
44          }
45
46          /**
47           * Set the edge color of this shape
48           * @param c edge color of this shape
49           * @return this own Builder object which color was set
50           */
51          public Builder lineColor(Color c) {
52              line = c;
53              return this;
54          }
55
56          /**
57           * Set the area color of this shape
58           * @param c area color of this shape
59           * @return this own Builder object which color was set
60           */
61          public Builder fillColor(Color c) {
62              fill = c;
63              return this;
64          }
65
66          /**
67           * Set the width of edge of this shape
68           * @param i width of edge
69           * @return this own Builder object which width of edge set
70           */
71          public Builder lineWidth(int i) {
72              lw = i;
73              return this;
74          }
75
76          public Builder shadow(boolean shadow) {
77              this.shadow = shadow;
78              return this;
79          }
80
81          public abstract MyDrawing build();
82
83
84      }
85
86      /**
87       * Make some instance extends MyDrawing with Builder pattern
88       * @param b Builder object with settings of shape
89       */
90      protected MyDrawing(Builder b) {
91
92          this.x = b.x;
93          this.y = b.y;
94          this.w = b.w;
95          this.h = b.h;
96
97          if (b.line == null) {
98              this.lineColor = Color.BLACK;
99          } else {
100             this.lineColor = b.line;
101         }
102
103         if (b.fill == null) {
104             this.fillColor = Color.WHITE;
105         } else {
106             this.fillColor = b.fill;
107         }
108
109         if (b.lw == 0) {
110             this.lineWidth = 2;
111         } else {
112             this.lineWidth = b.lw;
113         }
114
115         this.shadow = b.shadow;
116     }
117
118     protected MyDrawing() { /* do nothing */ }
119
120     /**
121      * Draw this shape on the Display
122      * @param g target Graphic object to draw
123      **/
124     public void draw(Graphics g) {
125         if (shadow) {
126             drawWithShadow(g);
127         } else {
128             drawShape(g);
129         }
130     }
131
132     /* 省略 */
133 }
```

これを継承して任意の正多角形を描画するクラスを実装した。ソースコードは以下の通り。Builder の vertex メソッドに頂点の数を指定して生成する。

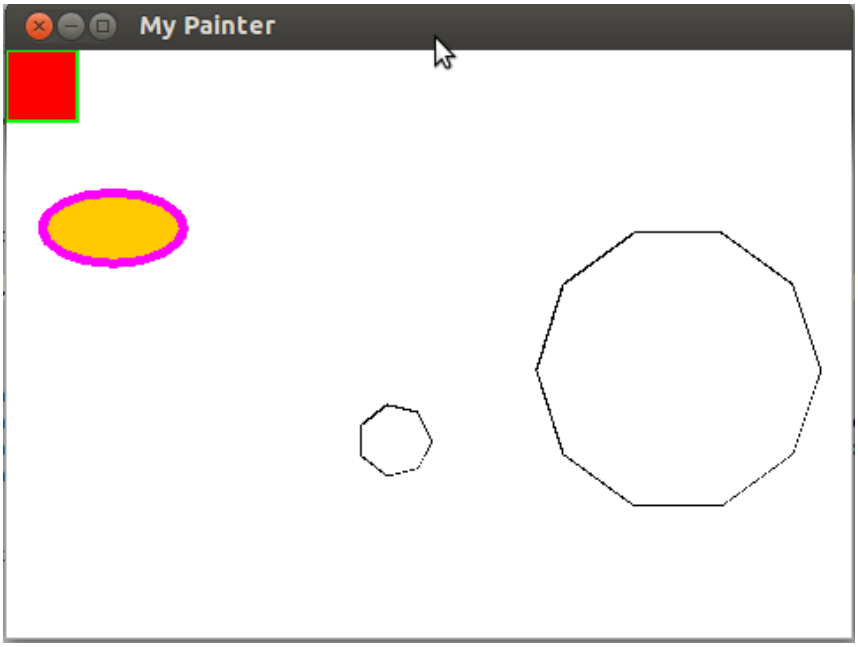
```
public class MyRegPolygon extends MyDrawing {

    /**
     * number of vertex
     */
    private int vertex;

    public static class Builder extends MyDrawing.Builder {
        private int vertex;
        public Builder(int x, int y) {
            super(x, y);
        }

        public MyRegPolygon build() {
            return new MyRegPolygon(this);
        }

        public Builder vertex(int v) {
            this.vertex = v;
            return this;
        }
    }

    public MyRegPolygon(Builder b) {
        super(b);
        this.vertex = b.vertex;
    }

    /**
     * regular constructor
     * @param vertex number of vertex
     */
    public MyRegPolygon(int vertex) {
        this.vertex = vertex;
    }

    /**
     * Draw this shape
     * @param g Graphics object
     */
    @Override
    public void drawShape(Graphics g) {
        int[] vert_x = new int[vertex];
        int[] vert_y = new int[vertex];

        for (int i = 0; i < vertex; ++i) {
            vert_x[i] = (int) ((x + w/2) + ((double)(h/2) * Math.cos(((double)2 * Math.PI) * i / (double)(
                vertex))));
            vert_y[i] = (int) ((y + h/2) + ((double)(h/2) * Math.sin(((double)2 * Math.PI) * i / (double)(
                vertex))));
        }

        Graphics2D g2 = (Graphics2D) g;
        g2.setStroke(new BasicStroke(this.getLineWidth()));
        g2.setColor(lineColor);
        g2.drawPolygon(vert_x, vert_y, vertex);
        g2.setColor(fillColor);
        g2.fillPolygon(vert_x, vert_y, vertex);

    }

    /**
     * clone this object
     * @return new instance cloned from this
     */
    @Override
    public MyRegPolygon clone() {
        MyRegPolygon c = (MyRegPolygon) new Builder(x,y).vertex(vertex).size(w,h).fillColor(fillColor).
            lineColor(lineColor).lineWidth(lineWidth).build();
        return c;
    }
}
```

実行部分のソースコードは以下の通り。

```
public class MainForm extends JFrame {

    public MyCanvas canvas = new MyCanvas(this);

    public MainForm() {
        super("My Painter");

```

```
8           JPanel jp = new JPanel(new BorderLayout());
9
10          getContentPane().add(jp);
11
12          jp.add(BorderLayout.CENTER, canvas);
13          setSize(new Dimension(480, 360));
14          setVisible(true);
15          this.repaint();
16          canvas.draws.add(new MyRegPolygon.Builder(200, 200).vertex(7).size(40, 40).build());
17          canvas.draws.add(new MyRectangle.Builder(0, 0).size(40,40).fillColor(Color.RED).lineColor(Color.GREEN).
                build());
18          canvas.draws.add(new MyOval.Builder(20, 80).size(80,40).fillColor(Color.ORANGE).lineColor(Color.MAGENTA
                ).lineWidth(5).build());
19          canvas.draws.add(new MyRegPolygon.Builder(300, 100).vertex(10).size(160,160).build());
20      }
21
22      public static void main (String[] args) {
23          MainForm mf = new MainForm();
24      }
25
26  }
27
28  class MyCanvas extends JPanel {
29
30      /**
31       * ArrayList which contains all shapes
32       */
33      public ArrayList<MyDrawing> draws = new ArrayList<MyDrawing>();
34      private final MainForm mf;
35
36      public MyCanvas(MainForm mf) {
37          setBackground(Color.white);
38          this.mf = mf;
39      }
40
41      @Override public void paint(Graphics g) {
42          super.paint(g);
43          for(MyDrawing d : draws) {
44              d.draw(g);
45              repaint();
46          }
47      }
48  }
```

実行結果のスクリーンショットを以下に示す。



第2章　課題2

2.1　クリックした位置に画像を描画

2.2　マウスドラッグによる描画

MouseListener と MouseMotionListener を用いてマウスの状態を取得し、ポインタの位置に追従するように伸縮する描画の実装を行った。具体的な各図形は、MotionListener クラスを継承して実装されている。MoitonListener クラスの実装を以下に示す。

```
public class MotionListener extends MouseAdapter implements MouseMotionListener, ActionListener{

    protected final MainForm mf;
    protected MyDrawing current;
    protected Boolean shadow;

    public MotionListener(MainForm mf) {
        this.mf = mf;

    }

    @Override
    public void mousePressed(MouseEvent e) {
        mf.canvas.repaint();
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        try {
            current.setSize(-current.getX() + e.getX(),  -current.getY() + e.getY());

        } catch(Exception ex) {
            System.out.println(current.toString() + ex + " at Dragged");
        }
        mf.canvas.repaint();
    }

    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        mf.canvas.removeMouseListener(mf.canvas.getMouseListeners()[0]);
        mf.canvas.removeMouseMotionListener(mf.canvas.getMouseMotionListeners()[0]);
        mf.canvas.addMouseListener(this);
        mf.canvas.addMouseMotionListener(this);
    }

}
```

このクラスは Button に対する ActionListener の役割も備えており、設定されたボタンが押下されると canvas の MouseListener、MouseMotionListener を適切に設定する。これを用いて実装した PolyListener クラスのソースコードを以下に示す。

```
public class PolyListener extends MotionListener implements ActionListener {
    private final int vertex;
    public PolyListener(MainForm mf, int vertex) {
        super(mf);
        this.vertex = vertex;
    }


    @Override
    public void mousePressed(MouseEvent e) {
        current = new MyRegPolygon.Builder(e.getX(), e.getY()).vertex(7).shadow(false).size(1, 1).build();
        mf.canvas.draws.add(current);
    }
}
```

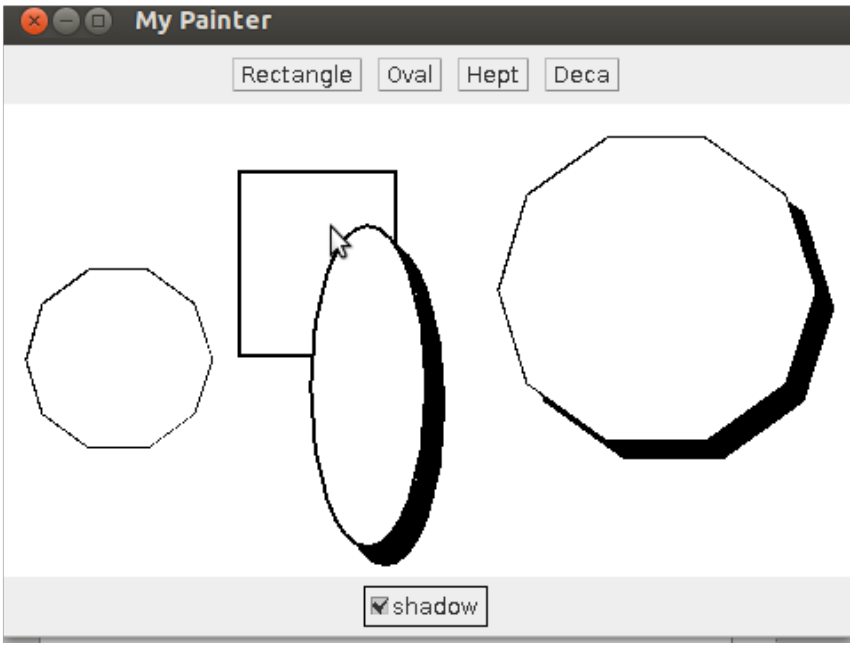
2.3 影の描画

CheckBox の状態によって影を描画する機能を実装した。マウスのボタンが押下され、Builder がインスタンスを生成する際に shadow を設定する。CheckBox にチェックが入っている場合は drawWithShadow が、入っていない場合は drawShape が draw メソッド内で実行され、それぞれの図形が描画される。ソースコードは以下の通り。

```
1  public void draw(Graphics g) {
2          if (shadow) {
3              drawWithShadow(g);
4          } else {
5              drawShape(g);
6          }
7      }
8
9      public abstract void drawShape(Graphics g);
10
11     public void drawWithShadow(Graphics g) {
12         MyDrawing shadow = this.clone();
13         shadow.setFillColor(Color.BLACK);
14         shadow.setLineColor(Color.BLACK);
15         shadow.move(10, 10);
16         shadow.drawShape(g);
17         this.drawShape(g);
18     }
```

drawWithShadow メソッドは MyDrawing クラス側に実装されているため、継承したクラスであれば独自に再実装することなく影の機能が利用できる。

実行例のスクリーンショットを以下に示す。ウィンドウ上部のボタンで描画する図形を選択し、下部のチェックボックスで影の有無を指定する。



第3章　感想

それなりに大きなプログラムを Java で記述するのは初めてだったが、Java の良さに触れられたと思う。今まで Java は保守的で（かつ悪い意味で）固い言語だと思っていたが、実際に触ってみると柔軟な記述が可能であり、今までの思い込みは間違っていたのだと感じた。特にジェネリクスの型パラメータやリフレクションを用いたメタプログラミングの技法など、（今回は使わなかったが）いろいろ勉強になったと思う。