

第1回レポート

1. 目的

ペイントプログラムへの機能追加を行い、クラス・インスタンス・メソッドなどのJavaおよびオブジェクト指向言語の特徴を理解する。

2. 課題

本レポートでは、配布されたプログラムを改変し機能拡充を図る。

実装する課題は以下の通りである。

1. 画面上に描画される円の数を30個に制限する。
2. 円を描画するたびに、円の直径を10pixel, 50pixelと交互に変化させる。
3. 円に適当な色を付ける。
4. 2個目以降の円描画時に、直前に描画した円との中心間に線を引く。
5. 四角形を描画できるようにする。
6. 現在描画されている円および四角形の数をGUI上に表示する。
7. 現在描画されている図形をすべて削除するボタンを追加する。
8. 2.で実装した機能を改変し、GUI上で任意の描画数を指定できるようにする。
9. 3.で実装した機能を改変し、GUI上で任意の描画サイズを指定できるようにする。
10. 4.で実装した機能を改変し、GUI上で任意の色を指定できるようにする。

3. 理論

1. `java.awt.button`クラス. ボタンの生成などを行う.
2. `java.awt.Label`クラス.GUI上にテキストを表示させるために用いる.
3. `java.awt.TextField`クラス. テキスト入力欄の生成や受け取りに用いる.
4. `java.awt.Checkbox`クラス. チェックボックスの生成などを行う.
5. `java.awt.CheckboxGroup`クラス. 生成したチェックボックスを追加することで, ラジオボタンとして扱える.
6. `java.awt.event.ActionListener`インターフェース. ボタンの入力を受け取る.
7. `java.awt.event.ItemListener`インターフェース. チェックボックスの入力を受け取る.
8. `java.awt.Color`クラス. 色の情報を格納する際などに用いる.

4. プログラム

パッケージ内の構成, および実行に係るファイルの階層は以下の通りである.

```
shell : tree
1 report
2   └─ Circle.java
3   └─ Coord.java
4   └─ Figure.java
5   └─ Line.java
6   └─ PaintReport.java
7     └─ Rectangle.java
```

以下はウィンドウへの描画を行うメインのクラス`PaintReport`である. 提示した課題の実装箇所はソースコード内にコメント形式で記述する. なお, 特にコメントの記述がない場合は, 配布プログラムから変更を加えていない.

```
java : paintreport.java
1 package report;
```

```

2 import java.awt.*;
3 import java.awt.event.*;
4 import java.util.ArrayList;
5
6
7 public class PaintReport extends Frame implements MouseListener,
8 MouseMotionListener {
9     int x, y, objSize = 30; //図形の初期サイズを変数objSizeにて定義する。
10    boolean sizeChange = false; //課題2.で実装した機能と任意のサイズを切り替えるため、GUIの指定状況を格納する変数。trueは課題2.を表す。
11    boolean sizeBig = false; //falseを10px, trueを50pxとし、サイズを交互に変更する際に利用する変数。
12    boolean objRect = false; //図形を切り替える変数。
13
14    ArrayList<Figure> objList;
15    ArrayList<Figure> lineList; //図形間に引いた線の情報を格納する。
16    Figure obj;
17
18    /* GUI上に表示するボタン・テキスト・フォームなどを宣言する. */
19    static Button clearButton = new Button("Clear"); //クリアボタン
20    static Button sizeincButton = new Button("+"); //プラスボタン
21    static Button sizedecButton = new Button("-"); //マイナスボタン
22    static Button applyButton = new Button("Apply"); //変更適応ボタン
23    static Label objcountLabel = new Label(); //図形数表示ラベル
24    static Label sizeLabel = new Label(); //図形サイズラベル
25    static Label countLabel = new Label("MaxObjectCount"); //最大表示数のラベル
26    static Label redLabel = new Label("R :"); //赤色入力フォームのラベル
27    static Label greenLabel = new Label("G :"); //緑色入力フォームのラベル
28    static Label blueLabel = new Label("B :"); //青色入力フォームのラベル
29    static TextField objcountField = new TextField(); //最大表示数入力
30    static TextField redField = new TextField(); //赤色入力フォーム
31    static TextField greenField = new TextField(); //緑色入力フォーム
32    static TextField blueField = new TextField(); //青色入力フォーム
33    static CheckboxGroup cbg = new CheckboxGroup(); //サイズ変更モードを切り替えるチェックボックスのグループ
34    static CheckboxGroup cbg2 = new CheckboxGroup(); //図形を切り替えるチェックボックスのグループ
35    static Checkbox staticsizeCheckbox = new Checkbox("Static size",cbg,
36    true); //サイズ指定モードを有効にするチェックボックス
37    static Checkbox changesizeCheckbox = new Checkbox("Change size",cbg,
38    false); //課題2.モードを有効にするチェックボックス

```

```

36     static Checkbox circleCheckbox = new Checkbox("Circle", cbg2, true); //円モードを有効にするチェックボックス
37     static Checkbox rectCheckbox = new Checkbox("Rectangle", cbg2, false); //四角形モードを有効にするチェックボックス
38
39     static int drawCnt = 30; //最大図形表示数を指定する変数。
40     static Color color = new Color(0,0,0); //図形色を指定するColorインスタンス変数。
41
42     public static void main(String[] args) {
43         PaintReport f = new PaintReport();
44         f.setSize(1280,720);
45         f.setLayout(null);
46         f.setTitle("Report Sample");
47         f.addWindowListener(new WindowAdapter() {
48             @Override public void windowClosing(WindowEvent e) {
49                 System.exit(0);
50             }
51         });
52     };
53
54     /* ボタンなどのインターフェースを配置する座標を設定する。 */
55     clearColorButton.setBounds(20, 40, 80, 30); //クリアボタン
56     objcountLabel.setBounds(20, 70, 120, 30); //図形数表示ラベル
57
58     sizeincButton.setBounds(20, 120, 50, 20); //プラスボタン
59     sizeLabel.setBounds(20, 145, 120, 30); //図形サイズラベル
60     sizedecButton.setBounds(20, 180, 50, 20); //マイナスボタン
61
62     staticsizeCheckbox.setBounds(20, 210, 100, 20); //サイズ指定モード
63     changesizeCheckbox.setBounds(20, 240, 100, 20); //課題2.モード
64
65     countLabel.setBounds(20, 280, 100, 20); //最大表示数ラベル
66     objcountField.setBounds(20, 310, 50, 20); //最大表示数フォーム
67     redLabel.setBounds(20, 340, 20, 20); //Redラベル
68     redField.setBounds(40, 340, 30, 20); //Red入力フォーム
69     greenLabel.setBounds(80, 340, 20, 20); //Greenラベル
70     greenField.setBounds(100, 340, 30, 20); //Greenフォーム
71     blueLabel.setBounds(140, 340, 20, 20); //Blueラベル
72     blueField.setBounds(160, 340, 30, 20); //Blueフォーム
73     applyButton.setBounds(20, 370, 70, 20); //変更適応ボタン
74
75     circleCheckbox.setBounds(20, 400, 100, 20); //円モード

```

```

76     rectCheckbox.setBounds(20, 430, 100, 20); //四角形モード
77
78     /* ボタンなどのインターフェースをウィンドウに追加する */
79     f.add(clearButton);
80     f.add(objcountLabel);
81     f.add(sizeincButton);
82     f.add(sizeLabel);
83     f.add(sizedecButton);
84     f.add(countLabel);
85     f.add(objcountField);
86     f.add(applyButton);
87     f.add(staticsizeCheckbox);
88     f.add(changesizeCheckbox);
89     f.add(redLabel);
90     f.add(redField);
91     f.add(greenLabel);
92     f.add(greenField);
93     f.add(blueLabel);
94     f.add(blueField);
95     f.add(circleCheckbox);
96     f.add(rectCheckbox);

97
98     f.btnExit(); //ボタン入力に応じた処理を行うメソッドを呼び出す。
99     f.checkEvent(); //チェックボックスに応じた処理を行うメソッドを呼び出
す
100    f.labelUpdate(); //ラベルの内容を変更するメソッドを呼び出す。
101
102    objcountField.setText(Integer.valueOf(drawCnt).toString()); //最大表
示数を変数の初期値に設定する。入力フォームはString型のみ対応なので,
String->int変換も行う。
103
104    /* RGBそれぞれの入力フォームを0で初期化する。 */
105    redField.setText("0");
106    greenField.setText("0");
107    blueField.setText("0");
108
109    f.setVisible(true);
110 }
111
112 /* ボタンの入力に応じた処理を定義するメソッド。 */
113 void btnEvent() {
114     clearButton.addActionListener(new ActionListener() { //Clearボタンの
処理。

```

```

115     @Override public void actionPerformed(ActionEvent e) {
116         objList.clear(); //objList内の図形を消す。
117         lineList.clear(); //lineList内の線を消す。
118         labelUpdate(); //ラベルの更新を行う。
119         repaint(); //図形を再描画する（何も表示しない）。
120     }
121 });
122 sizeincButton.addActionListener(new ActionListener() { //プラスボタン
の処理。
123     @Override public void actionPerformed(ActionEvent e) {
124         objSize += 10; //objSize変数に10を足す。
125         labelUpdate(); //ラベルの更新を行う。
126     }
127 });
128 sizedecButton.addActionListener(new ActionListener() { //マイナスボタ
ンの処理。
129     @Override public void actionPerformed(ActionEvent e) {
130         if(objSize > 0) { //objSizeがマイナスになった場合は処理を行わな
い。
131             objSize -= 10; //objSize変数を10減らす。
132         }
133         labelUpdate(); //ラベルの更新を行う。
134     }
135 });
136 applyButton.addActionListener(new ActionListener() { //適応ボタンの処
理。
137     @Override public void actionPerformed(ActionEvent e) {
138         objList.clear(); //最大描画数が変わった場合、一度描画された図形は
表示され続けてしまうため、一度すべての図形を消す。
139         lineList.clear(); //上と同じ
140         fieldUpdate(); //入力フォームの変更を取得する。
141         labelUpdate(); //ラベルの更新を行う。
142         repaint();
143     }
144 });
145 }
146
147 void checkEvent() {
148     staticsizeCheckbox.addItemListener(new ItemListener() { //サイズ指定
モードの処理，
149         public void itemStateChanged(ItemEvent e) {
150             sizeChange = false; //課題2.モードではない事を表すため、falseを代
入する。

```

```

151     }
152   });
153   changesizeCheckbox.addItemListener(new ItemListener() { //課題2.モードの処理。
154     public void itemStateChanged(ItemEvent e) {
155       sizeChange = true; //課題2.モードを表すtrueを代入する。
156     }
157   });
158   circleCheckbox.addItemListener(new ItemListener() { //円モードの処理,
159     public void itemStateChanged(ItemEvent e) {
160       objRect = false; //円を表すfalseを代入する。
161     }
162   });
163   rectCheckbox.addItemListener(new ItemListener() { //四角形モードの処理。
164     public void itemStateChanged(ItemEvent e) {
165       objRect = true; //四角形を表すtrueを代入する。
166     }
167   });
168 }
169 }
170
171 /* ラベルの内容を変更するメソッド */
172 void labelUpdate() {
173   objcountLabel.setText("ObjectCount is " +objList.size()); //objListの長さを取得し、図形数としてラベルに表示する。
174   sizeLabel.setText("Size " +objSize+ "px"); //現在指定されているサイズをobjSize変数で取得し、ラベルに表示する。
175 }
176
177 /* 入力フォームの内容を取得するメソッド */
178 void fieldUpdate() {
179   int in = Integer.parseInt(objcountField.getText()); //最大表示数フォームを取得し、int型に変換する。
180   if(0 < in)drawCnt = in; //入力された数が0より大きければ、最大表示数として設定する。
181
182   int Red = Integer.parseInt(redField.getText()); //赤色フォームを取得し、int型に変換する。
183   int Green = Integer.parseInt(greenField.getText()); //上と同じ。
184   int Blue = Integer.parseInt(blueField.getText()); //上と同じ。

```

```

185     if(!(Red < 0 || 255 < Red || Green < 0 || 255 < Green || Blue < 0 ||  

186     255 < Blue)) {  

187         color = new Color(Red, Green, Blue); //RGBに入力された数が 0 <= RGB  

188         <= 255 を満たしていれば、colorに代入する。  

189     }  

190 }  

191  

192     PaintReport() {  

193         objList = new ArrayList<Figure>();  

194         lineList = new ArrayList<Figure>();  

195         addMouseListener(this);  

196         addMouseMotionListener(this);  

197     }  

198  

199     @Override public void paint(Graphics g) {  

200         Figure f;  

201         for(int i = 0; i < objList.size(); i++) {  

202             f = objList.get(i);  

203             f.paint(g);  

204         }  

205  

206         for(int i = 0; i < lineList.size(); i++) { //lineListに保存された線を  

207             f = lineList.get(i);  

208             f.paint(g);  

209         }  

210         @Override public void mousePressed(MouseEvent e) {  

211             x = e.getX();  

212             y = e.getY();  

213  

214             /* GUIで選択されたサイズ変更モードに応じた処理を行う */  

215             if(sizeChange == true) { //サイズ指定モードが選択されている時の処理。  

216                 if (sizeBig == false) { //sizeBigがfalseのとき、大きさを10pxにして  

217                     sizeBigを反転する。  

218                     objSize = 10;  

219                     sizeBig = !sizeBig;  

220                 } else { //sizeBigがtrueのとき、大きさを50pxにしてsizeBigを反転す  

221                     objSize = 50;  

222                     sizeBig = !sizeBig;  

223                 }  

224             }

```

```

223
224     if(objRect == true) { //objRectがtrueのとき，四角形で図形を描画する。
225         obj = new Rectangle(objSize,color); //描画の際，GUIで指定された
226         objSizeとcolorを用いる。
227         } else { //objRectがfalseのとき，円で図形を描画する
228             obj = new Circle(objSize,color); //描画の際，GUIで指定されたobjSize
229             objSizeとcolorを用いる。
230             }
231             obj.moveTo(x, y);
232             repaint();
233         }
234         @Override public void mouseReleased(MouseEvent e) {
235             x = e.getX();
236             y = e.getY();
237             obj.moveTo(x,y);
238             objList.add(obj);
239             obj = null;
240             if(objList.size() > drawCnt) { //描画されている図形が指定された数を超
241             えた際の処理。
242                 objList.remove(0); //リストの先頭の図形を削除。
243                 lineList.remove(0); //リストの先頭の線を削除。
244             }
245             if(objList.size() >= 2) { //図形が二つ以上ある時の処理
246                 obj = new
247                 Line(objList.get(objList.size()-2),objList.get(objList.size()-1));// 前に
248                追加された図と今追加された図の座標の間に線を引く。
249                 lineList.add(obj);
250                 obj = null;
251             }
252             System.out.println(objList.size()); //コンソールに描画されている図形
253             の数を表示。
254             labelUpdate(); //ラベルをアップデート。
255             repaint();
256         }
257         @Override public void mouseClicked(MouseEvent e) {}
258         @Override public void mouseEntered(MouseEvent e) {}
259         @Override public void mouseExited(MouseEvent e) {}
260         @Override public void mouseDragged(MouseEvent e) {
261             x = e.getX();
262             y = e.getY();

```

```
260     if(obj != null) obj.moveTo(x,y);
261     repaint();
262 }
263 @Override public void mouseMoved(MouseEvent e) {}
264
265 }
```

以下は円を生成する**Circle**クラスである。

```
java : circle.java
1 package report;
2
3 import java.awt.*;
4
5 public class Circle extends Figure {
6     int size;
7     Color color;
8
9     Circle(int size) { //第二引数が指定されなかった場合、黒で円を生成する。
10        this.size = size;
11        this.color = new Color(0,0,0);
12    }
13    Circle(int size,Color color) { //第二引数が指定された場合、指定された色
14        this.size = size;
15        this.color = color;
16    }
17    @Override public void paint(Graphics g) {
18        g.setColor(color);
19        g.drawOval(x - size/2, y - size/2, size, size);
20    }
21 }
```

以下は四角形を生成する**Rectangle**クラスである。**Circle**クラスを参考に記述した。

```
java : rectangle.java
1 package report;
```

```
2
3 import java.awt.*;
4
5 public class Rectangle extends Figure {
6     int size;
7     Color color;
8
9     Rectangle(int size) { //第二引数が指定されていない場合、黒で四角形を生成
10    this.size = size;
11    this.color = new Color(0,0,0);
12 }
13 Rectangle(int size,Color color) { //第二引数が指定されている場合、指定さ
14    this.size = size;
15    this.color = color;
16 }
17 @Override public void paint(Graphics g) {
18    g.setColor(color);
19    g.drawRect(x - size/2, y - size/2, size, size); //四角形を生成する。
20
21 }
22 }
```

以下は線を生成するLineクラスである。Circleクラスを参考に記述した。

```
java : line.java
1 package report;
2
3 import java.awt.*;
4
5 public class Line extends Figure { //Figureクラスを継承し、Lineクラスを作
成する。
6     Color color;
7     Coord startPoint, endPoint; //線の開始・終了位置を入れるCoord型のインス
タנסを作成。
8
9     Line(Coord startPoint,Coord endPoint) { //第一引数で線の開始位置、第二引
数で線の終了位置を受け取る。
10    this.color = new Color(0,0,0); //色を黒に設定。
```

```
11     this.startPoint = startPoint; //受け取った開始位置を8行目で作成したインスタンスに代入。
12     this.endPoint = endPoint; //受け取った終了位置を8行目で作成したインスタンスに代入。
13 }
14 @Override public void paint(Graphics g) {
15     g.setColor(color);
16     g.drawLine(startPoint.x,startPoint.y,endPoint.x,endPoint.y); //開始位置から終了位置まで線を引く
17 }
18 }
```

以下はポジションに関するクラス**Coord**である。

```
java : coord.java
1 package report;
2
3 public class Coord {
4     int x, y;
5     Coord(){
6         x = y = 0;
7     }
8     Coord(int x , int y){ //引数で座標x,yを受け取り，ローカル変数に代入する
9         this.x = x;
10        this.y = y;
11    }
12    public void move(int dx, int dy) {
13        x += dx;
14        y += dy;
15    }
16    public void moveto(int x, int y) {
17        this.x = x;
18        this.y = y;
19    }
20 }
```

以下は**Figure**クラスである。配布プログラムから変更は加えていない。

java : figure.java

```
1 package report;  
2  
3 import java.awt.Graphics;  
4  
5 public class Figure extends Coord {  
6     int color;  
7     Figure(){  
8         color = 0;  
9     }  
10    public void paint(Graphics g) {  
11    }  
12 }
```

5. 動作検証, および実行環境

動作の検証にあたり, 下記条件の通り動作する場合にのみ正常動作とみなす.

1. アプリケーションを起動し, 何も変更しない状態で図形が30個以上描画されない. また, 図形を2個以上描画し, 図形の間に線が描かれる.
2. [Change size] チェックボックスを選択し, 図形が10px, 50pxと交互に切り替わる.
3. [R][G][B]に0~255の適当な数値を入力し, [apply]ボタンを押した後に描画した図形の色が変わる. なお, 3つの値が0に近いほど黒に近くなるため, 入力する値が小さいと変化がわかりにくい場合があるが, 正常である.
4. 線を除く, 現在描画されている図形の数が **ObjectCount is** 図形の数の形式で GUI上に表示される.
5. [Clear]ボタンを押し, 描画されていた全ての図形が削除される.
6. [MaxObjectCount]に1以上の適当な数値を入力し\[Apply]ボタンを押すと, 描かれる図形の最大数が指定した数以上描画されない.
7. [Static size] チェックボックを選択し[+][-]ボタンを押した際, 図形の大きさが10pxずつ増減する. また, 現在のサイズが **Size** 図形の大きさの形式で表示される. なお, 図形の大きさは常に0pxを下回らない.
8. [Rectangle] チェックボックスを選択し, 四角形が描画される. また, 上記全ての条件で円と同様に描画される.

本プログラムの制作, および動作の検証は以下の環境で行なった.

Device Macbook Air

OS MacOS Ventura Version 13.0.1

CPU Apple M1

Memory 16GB

Java version Java17

eclipse version 2022-09 (4.25.0)

6. 実行結果

検証1.

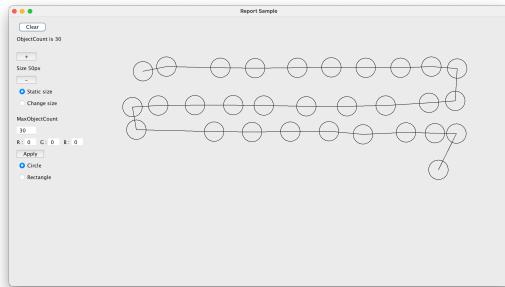


図1：図形を30個描画.

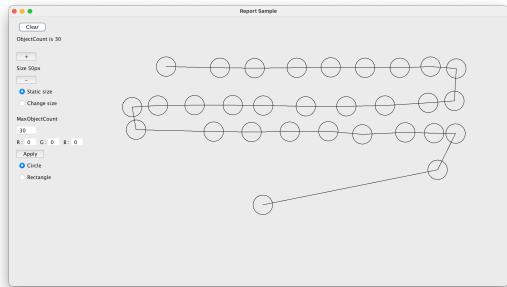


図2：図形を31個描画.

図1、図2のように、値に変更を加えず実行した場合には30個を超えて図形が描画されることはない。

検証2.

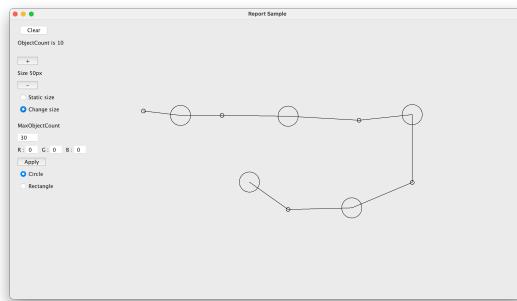


図3；[Change Size]を選択し、実行.

図3のように、[Change Size]を選択し実行すると、 $10\text{px} \times 50\text{px}$ と交互にサイズを変えて描画された。

検証3.

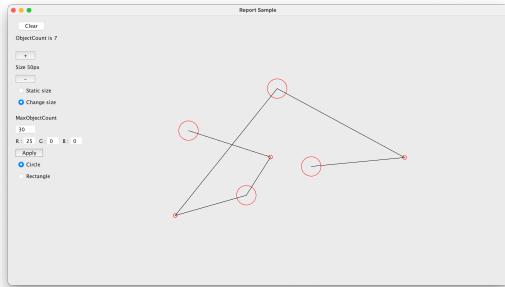


図4 : Rを255で実行。

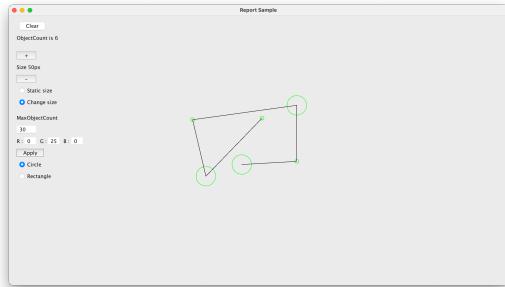


図5 : Gを255で実行。

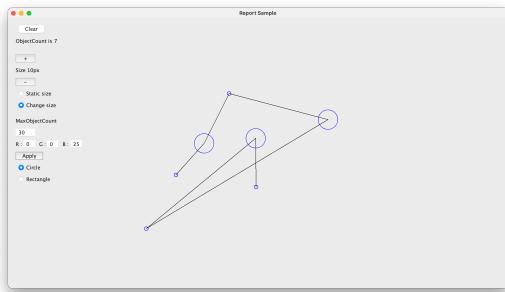


図6 : Bを255で実行。

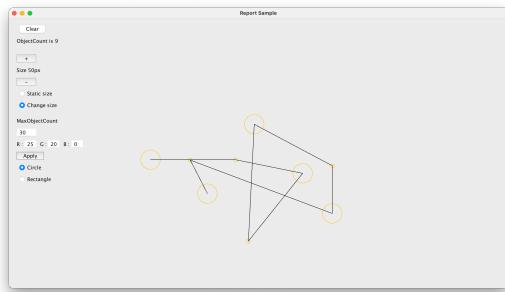


図7 : Rを255, Gを200, Bを0で実行

図4, 図5, 図6, 図7ののように、赤, 緑, 青をそれぞれ256段階で調整できた。

検証4.

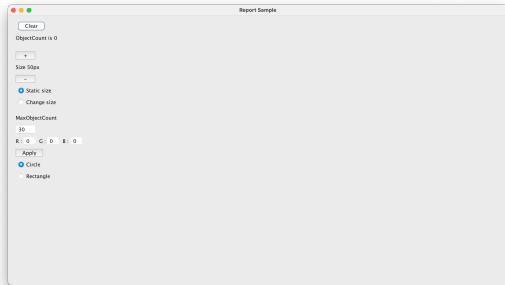


図8：図形を何も描画しない状態。

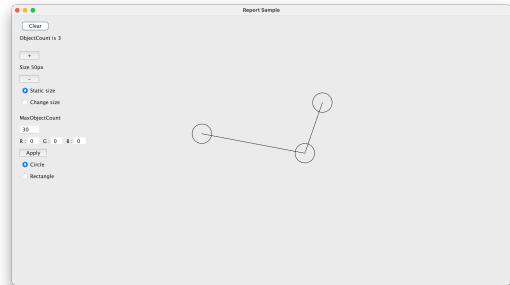


図9：図形を3つ描画。

図8、図9のように、**ObjectCount**の数値が描画されている図形に応じて変わった。

検証5.

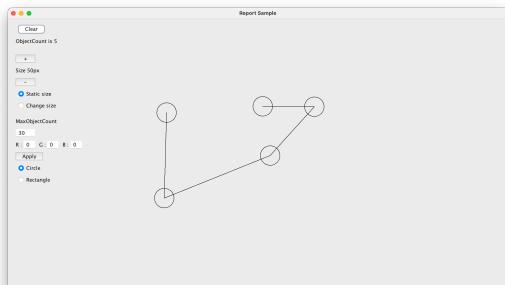


図10：図形を複数個描画した。

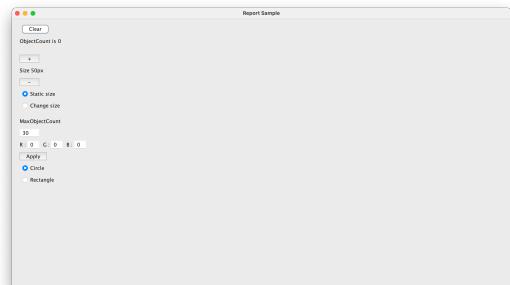


図11：[Clear]を押した。

図10、図11のように、[Clear]を押すことで描画されている図形を削除できた。

検証6.

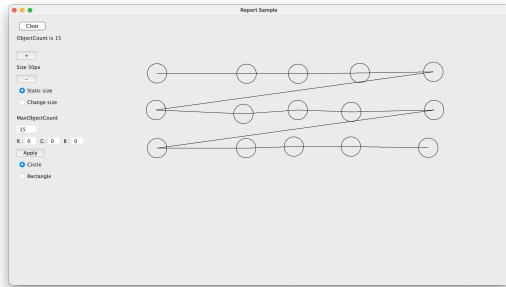


図12：最大描画数を15に設定し、図形を15個描画。

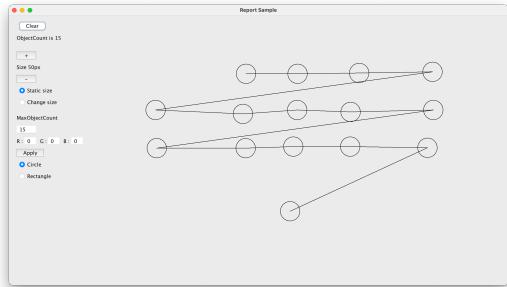


図13：同設定で図形を16個描画。

図12、図13のように、`MaxObjectCount`で指定した描画数を超えて描画されることはない。

検証7.

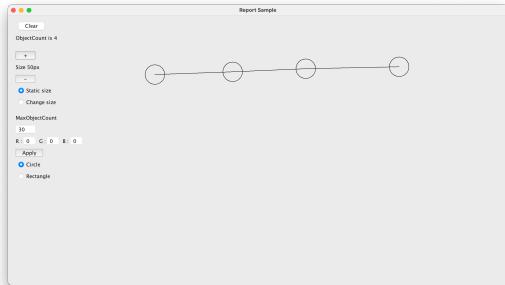


図14：大きさ50pxで実行。

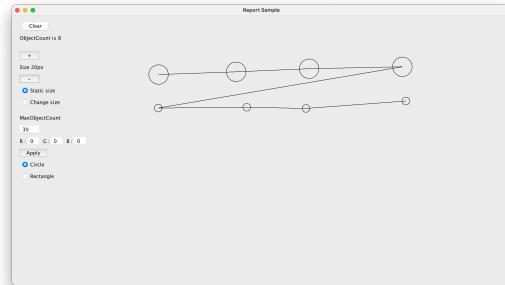


図15：大きさ20pxで実行。

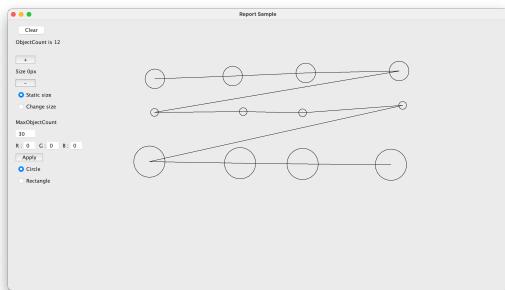


図16：大きさ80pxで実行。

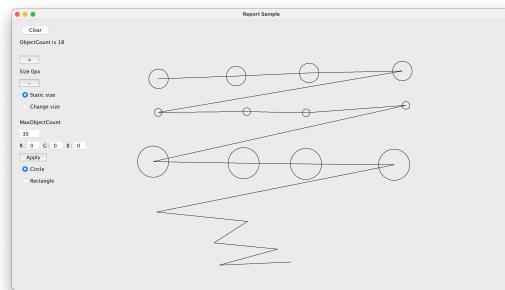


図17：大きさ0pxで実行。

図14, 図15, 図16, 図17のように、指定した大きさで図形が描画される。また、0pxでは図形は何も描画されない。

検証8.

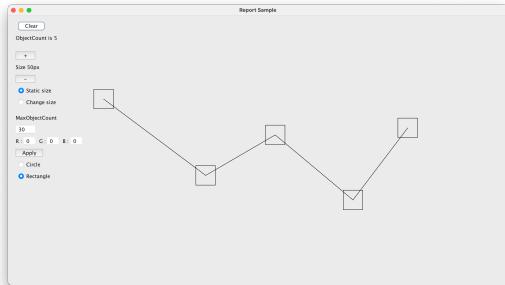


図18:[Rectangle]を選択し图形を描画.

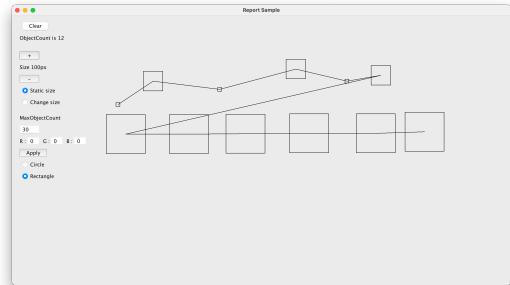


図19:[Change Size]で图形を6個描画→10pxで6個描画

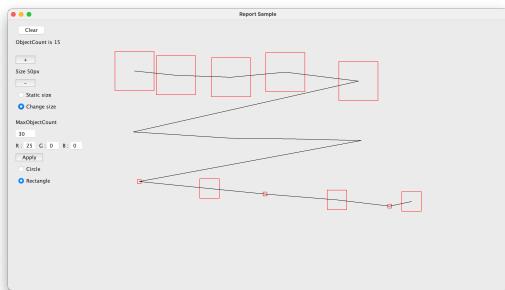


図20:Rを255に設定し、大きさを50px→0px→Change Sizeと変更し描画.

図18のように、[Rectangle]を選択することで四角形を描画できた。

図19、図20のように、[Circle]選択時と[Rectangle]選択時とでは图形の形以外に動作の違いはなかった。

7.まとめ

インスタンス・クラス・メソッドなどのオブジェクト指向言語特有の記述を用いてプログラムの制作を行い、その特徴を学んだ。

目標とするアプリケーションの制作には成功したものの、オブジェクト指向言語の特徴を活かした最も効率的なプログラムを制作することは今後の課題であると感じる。

8.感想

ボタン等を実装するにあたり調査したところ、awt・swing・JavaFXといったGUIに関するフレームワークがあることが分かった。awt以外のフレームワークでは今回のプログラムとの相性が悪いようだったため、swingとJavaFXの使用は見送ったが、実際にJavaでアプリケーションを構築する場合にはawtでは不十分であると実感した。アプリケーションを構築する際、フレームワーク同士の相性なども考慮し、事前に目標の動作とそれに必要なライブラリを検討しておかなければ、開発途中でフレームワークを選びなおさなくてはいけなくなる場合があると分かった。