

第2回レポート

1. 目的

ペイントアプリケーションの開発を通し、前回の課題に引き続きオブジェクト指向言語の特性を利用したプログラムを作成し、より効率的なプログラム作成を学ぶ。

とくに、前回の課題では理解が完全ではなかった継承などを積極的に取り入れたプログラム制作し理解を深める。

また、前回使用できなかったSwingなども取り入れ、より実践的なプログラム制作を行う。

2. 課題

1. 各図形の線に個別に色をつけられる様にする。
2. どの方向へマウスを動かしても四角形が描画されるようにする。
3. 楕円の描画を可能にする。
4. 折れ線の描画を可能にする。
5. Undo機能を実装する。
6. Redo機能を実装する。
7. 塗りつぶし図形の描画を可能にする。
8. ファイル名を指定した保存・読み込みを可能にする。
9. 2.を拡張し、Shiftキーを押すことで正方形の描画を可能にする。
10. 4.を拡張し、Shiftキーを押すことで正円の描画を可能にする。
11. 描画するオブジェクトに合わせて最適なカーソルに切り替える。
12. プログラム内にステータスバーを配置し、アプリケーションの状態を表示する。
13. プログラムを終了する際、保存を促すダイアログを表示するプログラムを実装する。
14. 描画中の図形を全て削除するボタンを実装する。

15. ツールバーを追加し以下の機能を実装する。

1. ボタンにツールチップを追加する。
2. ボタンにアイコンを追加する。
3. ボタンホバー時のカーソルを変更する。

3. 理論

上記課題を実装するにあたり、必要となるクラス・およびインターフェースを調査した。以下はその一覧である。

- **swing** クラス。awtに代わりGUIを実装するために使用。
- **filechooser** クラス。ファイルの読み込み・書き込みの際のファイル選択ウィンドウを実装するために使用。
- **JOptionPane** クラス。終了時のダイアログを実装するために使用。

4. プログラム

本プログラムの実行に必要なファイルは、以下の通りである。

```
bash : tree
1 report2
2   └── Circle.java
3   └── Coord.java
4   └── Dot.java
5   └── Figure.java
6   └── Line.java
7   └── MenuBtn.java
8   └── MenuItemBtn.java
9   └── IconButton.java
10  └── PaintReport.java
11  └── Rect.java
12  └── toolbar.java
```

```
java : paintreport.java
```

```
1 package report2;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.*;
6 import java.io.*;
7 import java.util.ArrayList;
8 import java.util.Objs;
9
10 public class PaintReport extends Frame implements MouseListener,
11 MouseMotionListener, ComponentListener, KeyListener {
12
13     // 変数、配列などの宣言 //
14     static Dimension screenSize =
15         Toolkit.getDefaultToolkit().getScreenSize();
16     static int width = screenSize.width, height = screenSize.height;
17     static toolbar toolbar = null;
18     static PaintReport f = new PaintReport();
19     int x, y;
20     int mode = 0, undo = 0;
21     boolean isShift = false, isEnter = false, isDrawing = false;
22     ArrayList<report2.Figure> objList;
23     report2.Figure obj;
24     Label statusLabel = new Label("Wait...");
25
26     // コンストラクタ //
27     PaintReport() {
28
29         objList = new ArrayList<>();
30         addMouseListener(this);
31         addMouseMotionListener(this);
32         addComponentListener(this); // ウィンドウのサイズ変更を取得する
33         addKeyListener(this);
34         setCursor(new Cursor(Cursor.WAIT_CURSOR)); // 起動直後、カーソルを
35         読み込み状態に変更する
36         setLayout(new BorderLayout());
37         Panel statusPanel = new Panel();
38         statusPanel.add(statusLabel);
39         statusPanel.setLayout(new GridLayout());
40         statusPanel.setBackground(Color.LIGHT_GRAY);
41         add(statusPanel, BorderLayout.SOUTH);
42     }
43
44     // メインメソッド //
45 }
```

```
41  public static void main(String[] args) {
42      f.setBounds(width / 4, height / 4, 640, 480);
43      f.setTitle("Main Window");
44      f.addWindowListener(new WindowAdapter() {
45          @Override
46          public void windowClosing(WindowEvent e) {
47              quit();
48          }
49      });
50      f.setVisible(true);
51
52      toolbar = new toolbar(f);
53
54      toolbar.setBounds(width / 4 + 645, height / 4, 200, 480);
55  }
56
57 // アプリケーションの終了を行うメソッド //
58 public static void quit() {
59     switch (JOptionPane.showConfirmDialog(f, "Save before exiting?"))
60     {
61         case JOptionPane.YES_OPTION:
62             toolbar.saveDialog();
63             break;
64         case JOptionPane.NO_OPTION:
65             System.exit(0);
66             break;
67         case JOptionPane.CANCEL_OPTION:
68             default:
69                 break;
70     }
71
72 // Undoを行うメソッド //
73 void undo() {
74     if (undo < objList.size()) {// <--undo
75         undo += 1;
76         setStatus("Undo : " + undo); //status
77         repaint();
78     }
79 }
80
81 // Redoを行うメソッド //
82 void redo() {
```

```
83     if ( undo > 0 ) {
84         undo -= 1;
85         setStatus("Undo : " + undo); //status
86         repaint();
87     }
88 }
89
90 // 描画されている図形を削除するメソッド //
91 public void clear() {
92     objList.clear();
93     undo = 0;
94     obj = null;
95     mode = 0;
96     isDrawing = false;
97     repaint();
98 }
99
100 // ファイルを保存するメソッド //
101 public void save(String fname) {
102     try {
103         FileOutputStream fos = new FileOutputStream(fname);
104         ObjectOutputStream oos = new ObjectOutputStream(fos);
105         oos.writeObject(objList);
106         oos.close();
107         fos.close();
108         setStatus("file saved."); //status
109         fos.close();
110     } catch (IOException e) {
111         setStatus("ERROR : failed to save."); //status
112     }
113     repaint();
114 }
115
116 // ファイルの読み込みを行うメソッド //
117 @SuppressWarnings("unchecked")
118 public void load(String fname) {
119     try {
120         FileInputStream fis = new FileInputStream(fname);
121         ObjectInputStream ois = new ObjectInputStream(fis);
122         objList = (ArrayList<report2.Figure>) ois.readObject();
123         ois.close();
124         fis.close();
125         setStatus("loaded"); //status
```

```

126     } catch (IOException e) {
127         setStatus("ERROR : failed to load."); //status
128     } catch (ClassNotFoundException e) {
129         setStatus("ERROR : class not found."); //status
130     }
131     repaint();
132 }
133
134 // 画面に図形を描画するメソッド //
135 @Override public void paint(Graphics g) {
136     report2.Figure f;
137     for (int i = 0; i < (objList.size() - undo); i++) {
138         f = objList.get(i);
139         f.paint(g);
140     }
141     if (toolbar != null) { //ツールバーが表示されるまで処理を行わない
142         toolbar.setUndo(0 < objList.size() && undo < objList.size());
143         toolbar.setRedo(undo > 0);
144         if (mode >= 1) obj.paint(g);
145     }
146 }
147
148 // メインウィンドウでクリックイベントが発生した際の処理 //
149 @Override public void mousePressed(MouseEvent e) {
150     x = e.getX();
151     y = e.getY();
152
153     // Undoを押した後に画面がクリックされた場合、Undoした分のオブジェ
154     // クトを削除する //
155     if (undo != 0) { // <---undo delete
156         for (; undo > 0; undo--) {
157             objList.remove(objList.size() - 1);
158         }
159     }
160
161     // ツールバーで選択されている図形モードを確認し、選択に応じた図形
162     // を生成する //
163     switch (toolbar.getObjMode()) {
164         case "dott":
165             isDrawing = false;
166             mode = 1;

```

```
165         obj = new Dot(toolbar.getColor(),
166 toolbar.setFillStatus());
167         break;
168
169     case "circle":
170         isDrawing = false;
171         mode = 2;
172         obj = new report2.Circle(toolbar.getColor(),
173 toolbar.setFillStatus());
174         break;
175
176     case "rect":
177         isDrawing = false;
178         mode = 2;
179         obj = new Rect(toolbar.getColor(),
180 toolbar.setFillStatus());
181         break;
182
183     case "line":
184         mode = 2;
185         if (!isDrawing) {
186             obj = new Line(new Coord(x, y), toolbar.getColor());
187             isDrawing = true;
188         }
189         break;
190     }
191     if (obj != null) {
192         obj.moveto(x, y);
193         obj.setPerfect(isShift);
194         repaint();
195     }
196 }
197
198 // マウスのクリックが解除された際の処理 //
199 @Override public void mouseReleased(MouseEvent e) {
200     x = e.getX();
201     y = e.getY();
202     if (mode == 1) obj.moveto(x, y);
203     else if (mode == 2) obj.setWH(x - obj.x, y - obj.y);
204     if (mode >= 1) {
```

```

205         objList.add(obj);
206         obj = null;
207     }
208     mode = 0;
209     repaint();
210 }
211
212 // override //
213 @Override public void mouseClicked(MouseEvent e) {}
214 @Override public void mouseEntered(MouseEvent e) {}
215 @Override public void mouseExited(MouseEvent e) {}
216
217 // マウスがクリック状態で移動している際の処理 //
218 @Override public void mouseDragged(MouseEvent e) {
219     x = e.getX();
220     y = e.getY();
221
222     if (mode == 1) {
223         obj.moveTo(x, y);
224     } else if (mode == 2) {
225         obj.setWH(x - obj.x, y - obj.y);
226     }
227     obj.setPerfect(isShift);
228     repaint();
229 }
230
231 // マウスカーソルを動かしている際の処理 //
232 @Override public void mouseMoved(MouseEvent e) {
233     x = e.getX();
234     y = e.getY();
235
236     // 線描画モードで、線とカーソルとの間にプレビュー線を引くための処理 //
237     if (isDrawing && Objects.equals(toolbar.getObjMode(), "line")) {
238         obj = new Line(objList.get(objList.size() - 1 - undo),
239                     toolbar.getColor());
240         obj.moveTo(x, y);
241         mode = 1;
242         repaint();
243     }
244
245     // メインウィンドウがリサイズされた際の処理 //

```

```
246     @Override public void componentResized(ComponentEvent e) {
247         if (toolbar != null) {
248             toolbar.setLocation(getX() + getWidth() + 10, getY());
249         }
250     }
251
252     // メインウィンドウが移動された際の処理 //
253     @Override public void componentMoved(ComponentEvent e) {
254         if (toolbar != null) {
255             toolbar.setLocation(getX() + getWidth() + 10, getY());
256         }
257     }
258     @Override public void componentShown(ComponentEvent e) {}
259     @Override public void componentHidden(ComponentEvent e) {}
260     @Override public void keyTyped(KeyEvent e) {}

261
262     // キーボードのキーが押された際の処理 //
263     @Override public void keyPressed(KeyEvent e) {
264
265         // 押されたキーに応じて処理を行う //
266         switch (e.getKeyCode()) {
267             case KeyEvent.VK_SHIFT:
268                 isShift = true;
269                 setStatus("shift: true");
270                 break;
271             case KeyEvent.VK_ENTER:
272                 isEnter = true;
273                 isDrawing = false;
274                 setStatus("enter: true");
275                 obj = null;
276                 mode = 0;
277                 repaint();
278                 break;
279         }
280     }
281
282     // キーボードのキーが離された際の処理 //
283     @Override public void keyReleased(KeyEvent e) {
284
285         // 離されたキーに応じて処理を行う //
286         switch (e.getKeyCode()) {
287             case KeyEvent.VK_SHIFT:
288                 isShift = false;
```

```

289         setStatus("shift: false");
290         break;
291     case KeyEvent.VK_ENTER:
292         isEnter = false;
293         setStatus("enter: false");
294         break;
295     }
296 }
297
298 // <--- getter & setter ---> //
299
300 // ステータスバーの内容を設定するセッター //
301 public void setStatus(String status) {
302     statusLabel.setText(status);
303 }
304
305 // メインウィンドウ内のカーソルを設定するセッター //
306 public void setCursor_this(int Cursor) {
307     f.setCursor(new Cursor(Cursor));
308 }
309 }
```

PaintReportクラスは主にメインウィンドウ内のイベントに応じた処理を行うクラスである。

13-14：PCの画面の大きさを取得し、格納する変数。

15-16：インスタンスを格納しておくための変数。

17：マウスの座標を取得し、格納するための変数。

18：modeは描画モードを格納するための変数、undoは描画履歴をいくつ遡るかを格納するための変数。主に課題5を実装するために用いる。

19：isShift、isEnterはキーが押されているかどうかを格納するための変数。isDrawingは折れ線を描画中かどうかを格納する変数。

22：ステータスバーに表示させるテキスト。課題12を実現させるために用いる。

29：ウィンドウのサイズ変更イベントを受け取るための宣言。

3 0 : キー入力イベントを受け取るための宣言.

3 1 : 起動直後には描画が行えないため、準備中を示すためにカーソルを変更する.

3 3 - 3 7 : 課題12. ステータスバーを生成し、ウィンドウの下部に追加する処理.

4 2 : 画面の大きさから中心を計算し、画面の中央に640×480のメインウィンドウを配置する.

4 4 - 4 9 : ウィンドウの閉じるボタンを押した際の処理。実際にプログラムを終了する処理は、`quit`メソッド内で行う。

5 2 : ツールバーをインスタン化し、変数に格納する。

5 4 : 画面の大きさからメインウィンドウの位置を計算し、メインウィンドウの右側に配置する。

5 9 : 課題13. `quit`メソッドが実行されると、ダイアログを表示する。ダイアログ内のどのボタンが押されたかは、戻り値で判断する。

6 0 - 6 2 : Yesが押された際の処理。ファイルに保存する処理を行うため、`saveDialog`メソッドを呼び出す。

6 3 - 6 5 : Noが押された際の処理。アプリケーションを正常終了する。

6 6 : Cancelが押された際の処理。何も行わない。

7 3 - 7 9 : 課題5. `undo`変数を加算する処理。過去に描画した図形の数より少ない場合にのみ、`undo`変数を加算する。

8 2 - 8 8 : 課題6. `undo`変数を減算する処理。`undo`が押された数よりも少ない場合に減算する。

9 1 - 9 8 : 描画されている図形などを削除し、各パラメーターをリセットする。おもに課題14.

1 0 1 - 1 1 5 : 課題8. ファイルへの保存を行うメソッド。引数で指定された絶対パスで保存する。例外処理として、何らかのエラーが発生した場合にはステータスバーに表示する。

117-130：課題8. 保存したファイルの読み込みを行うメソッド。引数で指定された絶対パスから読み込む。例外処理として、エラーが発生した場合にはステータスバーに表示する。

137：課題5. 6. **undo**変数に応じて、**objList**内の描画される図形の数を減らすことでのUndo, Redo機能を実装している。

141：ツールバーが表示されるまで**setUndo**などのメソッドを実行できないため、表示されていない時には実行されない様にする処理。

142-143：課題5. 6. Undo, Redoの限界になった時、ボタンを無効化する処理。

154-158：課題5. 新たな図形を描画しようとした場合、Undoが行われていれば、**undo**変数に応じて**objList**から図形を削除する。

161-196：**getObjMode**メソッドでツールバーにて選択されている図形モードを取得し、モードに応じた図形を生成する。

180-186：課題4. 線の生成を行うメソッド。最初のクリックは開始点を決めるための動作のため、1回目のクリックでは点を描画する。**drawStatus**が**false**の場合は1回目のクリック、**true**は折れ線の描画中を表す。

193, 227：課題9. 10. Shiftキーが押されている場合円と四角形を1:1にするため、**setPerfect**メソッドにShiftキーの状況を代入する。

232-243：課題4. 線描画モードで、直前のクリック位置からカーソルまでの線（プレビュー線）を引くための処理。**objList**に保存されている一つ前の図形の位置を開始位置、現在のカーソルの位置を終了位置として線を引く。

246-250：課題15. メインウィンドウのサイズが変更された際、ツールバーがウィンドウの右横に来る様にツールバーを移動させる処理。

253-257：課題15. メインウィンドウの移動に合わせて、ツールバーも移動させるための処理。

263-280：課題4. 9. 10. キーが押された際に、押されたキーに応じて処理を行うメソッド、

267-270 : Shiftキーが押されている際の処理。`isShift`を`true`に変更し、ステータスバーに状態を表示する。

271-278 : Enterが押されている際の処理。`isEnter`を`true`にし、`isDrawing`を`false`にする。描画されているプレビュー線を削除するなどを行う。ステータスバーに状態を表示する。

283-296 : キーが離された際の処理。ステータスバーに状態を表示し、それぞれの押下状態を表す変数に`false`を代入する。

301-303 : 課題12。ステータスバーに表示するテキストを設定するためのセッターメソッド。

306-309 : 課題11。メインウィンドウ内のカーソルを設定するためのセッターメソッド。

```
java : toolbar.java
1 package report2;
2
3 import javax.swing.*;
4 import javax.swing.border.TitledBorder;
5 import javax.swing.filechooser.FileNameExtensionFilter;
6 import java.awt.*;
7 import java.awt.event.ActionEvent;
8 import java.awt.event.ActionListener;
9 import java.awt.event.WindowAdapter;
10 import java.awt.event.WindowEvent;
11
12 class toolbar extends JFrame implements ActionListener {
13
14     // 変数・配列などの宣言 //
15     PaintReport paintReport;
16     MenuIconBtn fillButton = new MenuIconBtn("\uf5c7", "fill", "Object
fill ON/OFF");
17     MenuIconBtn undoButton = new MenuIconBtn("\uf3e5", "undo", "Return to
previous step");
18     MenuIconBtn redoButton = new MenuIconBtn("\uf064", "redo", "Return to
next step");
19     JLabel nowColor = new JLabel("\uf0c8");
20     private final ButtonGroup objModeGroup = new ButtonGroup();
21     private final JFileChooser file = new JFileChooser(".");
```

```
22  private Color selectColor = Color.black;
23  private boolean fillStatus = false;
24
25  // コンストラクタ //
26  public toolbar(PaintReport paintReport) {
27      this.paintReport = paintReport;
28      getContentPane().setLayout(new FlowLayout());
29      setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
30      setTitle("Toolbar");
31      addWindowListener(new WindowAdapter() {
32          @Override
33          public void windowClosing(WindowEvent e) {
34              PaintReport.quit();
35          }
36      });
37
38      file.setFileFilter(new
39          FileNameExtensionFilter("PaintData(*.dat)", "dat"));
40
41      ui();
42      setVisible(true);
43  }
44
45  // ボタンのレイアウト・配置を行うメソッド //
46  void ui() {
47      GridLayout col2Layout = new GridLayout(0, 2, 5, 5);
48
49      // Undo button //
50      undoButton.addActionListener(this);
51      undoButton.setEnabled(false);
52      add(undoButton);
53
54      // Redo button //
55      redoButton.addActionListener(this);
56      redoButton.setEnabled(false);
57      add(redoButton);
58
59      // Object style selector group //
60      TitledBorder styleBorder = new TitledBorder("ObjectStyle");
61      JPanel panel1 = new JPanel();
62      panel1.setLayout(col2Layout);
63      panel1.setBorder(styleBorder);
64      add(panel1);
```

```
64          // Menu group //
65          TitledBorder toolBorder = new TitledBorder("Menu");
66          JPanel panel2 = new JPanel();
67          panel2.setLayout(col2Layout);
68          panel2.setBorder(toolBorder);
69          add(panel2);
70
71
72          // Color setting group //
73          TitledBorder colorBorder = new TitledBorder("Color");
74          JPanel panel3 = new JPanel();
75          panel3.setPreferredSize(new Dimension(180, 110));
76          panel3.setBorder(colorBorder);
77          add(panel3);
78
79
80          // Dott button //
81          MenuTButton dotToggleButton = new MenuTButton("Dot", "dott",
82          true);
83          dotToggleButton.addActionListener(this);
84          objModeGroup.add(dotToggleButton);
85          panel1.add(dotToggleButton);
86
87
88          // Circle button //
89          MenuTButton circleButton = new MenuTButton("Circle", "circle",
90          false);
91          circleButton.addActionListener(this);
92          objModeGroup.add(circleButton);
93          panel1.add(circleButton);
94
95
96          // Rectangle button //
97          MenuTButton rectToggleButton = new MenuTButton("Rect", "rect",
98          false);
99          rectToggleButton.addActionListener(this);
100         objModeGroup.add(rectToggleButton);
101         panel1.add(rectToggleButton);
102
```

```

103     // Clear button //
104     MenuItemBtn clearButton = new MenuItemBtn("\uf1f8", "clear",
105     "Clear all objects");
106     clearButton.addActionListener(this);
107     panel2.add(clearButton);
108
109     // File import button //
110     MenuItemBtn loadButton = new MenuItemBtn("\uf07c", "load", "Load
111     the drawn object from file");
112     loadButton.addActionListener(this);
113     panel2.add(loadButton);
114
115     // File export button //
116     MenuItemBtn saveButton = new MenuItemBtn("\uf0c7", "save", "Save
117     the drawn object to file");
118     saveButton.addActionListener(this);
119     panel2.add(saveButton);
120
121     // Close button //
122     MenuItemBtn closeButton = new MenuItemBtn("\uf011", "close",
123     "Quit this program");
124     closeButton.addActionListener(this);
125     panel2.add(closeButton);
126
127     // Color picker button //
128     MenuItemBtn colorButton = new MenuItemBtn("\uf5c3", "color",
129     "Select object color");
130     colorButton.addActionListener(this);
131     colorButton.setPreferredSize(new Dimension(50, 50));
132     panel3.add(colorButton);
133
134     // Fill object switch //
135     fillButton.addActionListener(this);
136     fillButton.setPreferredSize(new Dimension(50, 50));
137     panel3.add(fillButton);
138
139     // Color prev //
140     JLabel nowColorTitle = new JLabel("Selected color : ");
141     panel3.add(nowColorTitle);
142
143     nowColor.setForeground(getColor());
144     nowColor.setFont(new Font("Font Awesome 6 Free", Font.PLAIN,
145     15));

```

```
140     panel3.add(nowColor);
141
142     // Change-to Toolbar ready //
143     paintReport.setCursor(this(Cursor.DEFAULT_CURSOR));
144     paintReport.setStatus("Ready.");
145 }
146
147 // ボタンのイベント処理を行うメソッド //
148 @Override
149 public void actionPerformed(ActionEvent e) {
150     switch (e.getActionCommand()) {
151         case "undo":
152             paintReport.undo();
153             break;
154
155         case "redo":
156             paintReport.redo();
157             break;
158
159         case "color":
160             selectColor = JColorChooser.showDialog(this, "Color
Selector", Color.black);
161             if(selectColor != null) {
162                 paintReport.setStatus("R:" + getColor().getRed() + "
G:" + getColor().getGreen() + " B:" + getColor().getBlue());
163                 nowColor.setForeground(getColor());
164             }
165             break;
166
167         case "fill":
168             if (fillStatus) {
169                 fillButton.setText("\uf5c7");
170                 fillStatus = !fillStatus;
171                 paintReport.setStatus("False");
172             } else {
173                 fillButton.setText("\uf043");
174                 fillStatus = !fillStatus;
175                 paintReport.setStatus("True");
176             }
177             break;
178
179         case "clear":
180             paintReport.clear();
```

```
181         paintReport.setStatus("Cleared all objects");
182         break;
183
184     case "load":
185         if (file.showOpenDialog(paintReport) ==
186             JFileChooser.APPROVE_OPTION) {
187             paintReport.clear();
188             paintReport.load(file.getSelectedFile().getPath());
189
190             paintReport.setStatus(file.getSelectedFile().getPath());
191         } else {
192             paintReport.setStatus("canceled or error");
193         }
194         break;
195
196     case "save":
197         saveDialog();
198         break;
199
200     case "close":
201         PaintReport.quit();
202         break;
203
204     case "dott":
205         paintReport.setCursor(this(Cursor.DEFAULT_CURSOR));
206         break;
207
208     case "circle":
209         paintReport.setCursor(this(Cursor.CROSSHAIR_CURSOR));
210         break;
211
212     default:
213         paintReport.setStatus("not-set ActionCommand");
214         break;
215     }
216 }
217
218 // 保存用のファイル選択画面を呼び出すメソッド //
219 public void saveDialog() {
220     if (file.showSaveDialog(paintReport) ==
JFileChooser.APPROVE_OPTION) {
```

```

221         paintReport.save(file.getSelectedFile().getPath());
222     } else {
223         paintReport.setStatus("canceled or error");
224     }
225 }
226
227 // <--- getter & setter ---> //
228
229 // 選択されている図形モードを受け取れるゲッター //
230 public String getObjMode() {
231     return objModeGroup.getSelection().getActionCommand();
232 }
233
234 // 塗りつぶしがオンになっているか受け取れるゲッター //
235 public Boolean getFillStatus() {
236     return this.fillStatus;
237 }
238
239 // 指定された色を受け取れるゲッター //
240 public Color getColor() {
241     return this.selectColor;
242 }
243
244 // Undoボタンの無効・有効を設定するセッター //
245 public void setUndo(boolean undo) {
246     undoButton.setEnabled(undo);
247 }
248
249 // Redoボタンの無効・有効を設定するセッター //
250 public void setRedo(boolean redo) {
251     redoButton.setEnabled(redo);
252 }
253 }

```

課題15. Toolbarクラスは、ツールバーの生成、およびツールバー内のイベントを処理するクラスである。

15：PaintReportクラスをインスタンス化し、格納しておくための変数。

16-19：グローバルで操作が必要なUIの部品を宣言する。

20：図形モードを選択するトグルボタンのグループ。

21：課題8. 読み込み・保存の際に保存先とファイル名を指定するためのファイルチューザーを格納した変数.

22：課題1. 選択された色を格納する変数. 黒で初期化している.

23：課題7. 塗りつぶし状態を格納する変数. `false`で初期化しているので、起動時は塗りつぶしされない.

26-42：主にウィンドウの生成に関する処理を行うコンストラクタ.

28：UIの要素をウィンドウ内で整列させるためのレイアウトを設定している.

29：課題13. アプリケーションの終了の際、ダイアログを表示させる必要があるので、ウィンドウのバツボタンが押された際にアプリケーションが終了してしまうよう、デフォルトの終了処理を無効化する.

38：ファイルの選択の際、デフォルトで`.dat`のみが選択されるよう、フィルターを設定.

40：ツールバーのボタンを配置するメソッドを呼び出す.

45-145：ツールバーにボタンなどのUI要素を追加するメソッド.

46：ボタンのレイアウトを変数に格納する.

48-56：Undoボタン、Redoボタンをツールバーに追加する。起動時は何も図形が描画されていないので、初期状態は無効化している。アイコンはFontAwesomeのアイコンを文字コードを使い指定している。クリックイベントが走るよう、アクションリスナーを定義している。

59-77：各ボタンを行う処理ごとにグループ化し見やすくするため、タイトルを設定した囲み線を生成し、ツールバーに追加している。レイアウトは46で定義したレイアウトを用いる。

80-101：図形を切り替えるトグルボタンを生成し、囲み線内に追加する。選択状態はドットボタンに`true`を与え、それ以外は`false`にしているので、起動時はドットボタンがアクティブになる。

104-121：プログラムの操作を行うボタンを生成し、囲み線内に追加する。アイコンはFontAwesomeのアイコンを文字コードを使い指定している。クリックイベントを取得できるよう、アクションリスナーを定義している。

124-127：課題1. カラーピッカーを表示するボタンを生成し、囲み線内に追加している。アクションリスナー、アイコンは他のボタンと同様に設定している。

130-132：課題7. 塗りつぶしの切り替えを行うボタンを生成し、囲み線内に追加している。今回あえて **JToggleButton** を用いず、アイコンを変更することで状態を表すこととした。

135-140：課題1. 選択した色を確認するためのラベルを生成し、囲み線内に追加している。FontAwesomeの四角形アイコンを文字コードを用いて指定し、ラベルの前景色（フォントの色）を **getColor** で取得した色に変更することでプレビュー作成している。

148-216：ボタンが押された際のイベント処理を行うメソッド。

150：**e.getActionCommand()** でクリックされたボタンに設定されたアクションコマンドを取得し、**switch**文でボタンごとの処理を条件分岐している。

160-165：課題1. 色選択ボタンが押された際にカラーピッカーを表示させる処理。選択された色を **selectColor** 変数に格納し、**selectColor** が **null** でない時、ステータスバーに選択された色を表示し、選択色のプレビューラベルにも色を反映させる。

168-177：課題7. 塗りつぶしボタンの処理。**fillStatus** が **true** の場合、塗りつぶしでないことを表すアイコンに変更し **fillStatus** を反転、ステータスバーに塗りつぶしが無効であることを表示する。**fillStatus** が **true** の場合は、全く逆の処理を行う。

184-192：課題8. ファイル読み込みボタンを押した際、ファイルチューザーを表示する処理。正常終了した場合にのみ、現在描画されている図形を全て削除、ステータスバーに読み込んだファイルのパスを表示し、**PaintReport** クラス内の **load** メソッドをファイルパスを引数に代入して呼び出す。キャンセルやエラーによってファイルチューザーが正常終了しなかった場合には、ステータスバーにエラーメッセージを表示する。

195-196：課題8. 保存ボタンが押された際の処理. ファイルチューザーを表示させる**saveDialog**メソッドを呼び出す.

199-200：課題13. 閉じるボタンが押された際の処理. 終了の際にダイアログを表示させるメソッドを呼び出す.

203-210：課題11. 選択された図形モードに適したカーソルを設定する処理.

213-214：まだ定義されていないボタンのイベントが発生した場合, そのことをステータスバーに表示する.

219-225：課題8. 保存の際にファイルチューザーを表示させるメソッド. ファイルを開く際の処理とほとんど同じであるが, ファイルパスを代入するのは**save**メソッドである.

231-233：図形モードの選択されているボタンに設定されているアクションコマンドを返すゲッターメソッド.

236-238：課題7. 塗りつぶしを行うかを返すゲッターメソッド.

241-243：課題1. カラーピッカーで選択された色を返すゲッター

246-253：課題5. Undo, Redoボタンの有効・無効を切り替えるセッタ一.

```
java : rect.java
1 package report2;
2
3 import java.awt.*;
4
5 public class Rect extends report2.Figure {
6
7     // 変数の宣言 //
8     int X, Y, W, H, Width, Height;
9
10    // コンストラクタ //
11    Rect(Color color, Boolean fs) {
12        this.color = color;
13        this.fillStatus = fs;
```

```

14    }
15
16    // 描画を行うメソッド //
17    @Override public void paint(Graphics g) {
18        g.setColor(color);
19        if (isPerfect) { //正方形へ切り替え
20
21            // 幅と高さの大きい方の値を採用する //
22            if (Math.abs(w) > Math.abs(h)) {
23                W = w;
24                H = w;
25            } else {
26                W = h;
27                H = h;
28            }
29        } else { //長方形へ切り替え
30            W = w;
31            H = h;
32        }
33
34        // <--- 四方向どこに動かしても描画されるようにする処理 ---> //
35
36        // マウスを上側に動かした際の処理 //
37        if (W < 0) {
38            X = x + W;
39            Width = -W;
40        } else {
41            X = x;
42            Width = W;
43        }
44
45        // マウスを左側に動かした際の処理 //
46        if (H < 0) {
47            Y = y + H;
48            Height = -H;
49        } else {
50            Y = y;
51            Height = H;
52        }
53        // 塗りつぶしの処理 //
54        if (fillStatus) {//塗りつぶし図形を描画する
55            g.fillRect(X, Y, Width, Height);
56        } else {//線のみの図形を描画する。

```

```

57         g.drawRect(X, Y, Width, Height);
58     }
59 }
60 }
```

Rectクラスは**Figure**クラスを継承した四角形を描画するクラスである。

8：このクラスで使う変数の宣言。

11-14：課題1. 7. コンストラクタ。引数で色と塗りつぶしか否かを受け取る。

17-52：四角形の描画を行うメソッド。

19-32：課題9。変数**isPerfect**は正方形を表す変数であり、**true**の場合には幅と高さの大きい方の値を幅と高さ両方に代入することで正方形を描く。**false**の場合には値に変更を加えていない。

37-43：課題2. クリックした点よりカーソルを上側に動かした際の処理。高さ変数の符号を反転し、描画の開始点をカーソルの上方向への移動分ずらす。下側へカーソルを移動する場合には値を変更しない。

46-52：課題2. クリックした点よりカーソルを左側に動かした際の処理。幅変数の符号を反転し、描画の開始点をカーソルの左方向への移動分ずらす。右側へカーソルを移動する場合には値を変更しない。

54-59：課題7. **fillStatus**変数が**true**の場合、塗りつぶしの図形を描画する。**false**の場合には線の図形を描画する。

```

java : dot.java
1 package report2;
2
3 import java.awt.*;
4
5 public class Dot extends Figure {
6
7     // 変数の宣言 //
8     int size;
9 }
```

```

10 // コンストラクタ //
11 Dot(Color color, Boolean fs) {
12     this.color = color;
13     this.fillStatus = fs;
14     size = 10;
15 }
16
17 // 描画するメソッド //
18 @Override public void paint(Graphics g) {
19     g.setColor(color);
20     if (fillStatus) {
21         g.fillOval(x - size / 2, y - size / 2, size, size);
22     } else {
23         g.drawOval(x - size / 2, y - size / 2, size, size);
24     }
25 }
26 }
```

Dotクラスは、**Figure**クラスを継承したドットを描画するクラスである。

11-15：課題7. 引数で色と塗りつぶしの可否を受け取り、変数に代入する。

19：課題1. 描画色を引数で指定された色に設定する。

20-25：課題7. **fillStatus**が**true**の場合塗りつぶしで描画し、**false**の場合は線で描画する。

```

java : line.java
1 package report2;
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
7     // 変数の宣言 //
8     Coord sPoint;
9
10    // コンストラクタ //
11    Line(Coord sPoint, Color color) {
12        this.color = color;
```

```

13     this.sPoint = sPoint;
14 }
15
16 // 描画するメソッド //
17 @Override public void paint(Graphics g) {
18     g.setColor(color);
19     g.drawLine(sPoint.x, sPoint.y, x, y); //開始位置から終了位置まで線
20     を引く
21 }

```

Lineクラスは、**Figure**クラスを継承した折れ線を描画するクラスである。

8：線の開始点を格納する変数。

11-14：線の開始点と色を引数で受けとり、変数に代入する。

18：課題1. 引数で受け取った色を設定する。

19：開始点から終了点までの線を描く。

```

java :circle.java
1 package report2;
2
3 import java.awt.*;
4
5 public class Circle extends Figure {
6
7     // 変数の宣言 //
8     int width, height;
9
10    // コンストラクタ //
11    Circle(Color color, Boolean fs) {
12        this.color = color;
13        this.fillStatus = fs;
14    }
15
16    // 描画を行うメソッド //
17    @Override public void paint(Graphics g) {
18        g.setColor(color);
19

```

```

20     // 楕円・正円を切り替える //
21     if (isPerfect) {
22         width = (int) Math.sqrt(w * w + h * h);
23         height = (int) Math.sqrt(w * w + h * h);
24     } else {
25         width = (int) (Math.sqrt(2.0) * Math.abs(w));
26         height = (int) (Math.sqrt(2.0) * Math.abs(h));
27     }
28
29     // 塗りつぶしを切り替える //
30     if (fillStatus) {
31         g.fillOval(x - width, y - height, width * 2, height * 2);
32     } else {
33         g.drawOval(x - width, y - height, width * 2, height * 2);
34     }
35 }
36 }
```

Circleクラスは、**Figure**クラスを継承した円を描くクラスである。

8：幅、高さを格納する変数を宣言する。

11-14：課題7。引数で色と塗りつぶしの可否を受け取り、変数に代入する。

21-27：**isPerfect**が**true**の場合、クリックした位置からの移動距離（半径）を求め、幅と高さの変数に同一の値を代入する。**false**の場合、楕円を描く。

30-34：**fillStatus**が**true**の場合、塗りつぶしの図形を描画する。**false**の場合、線のみの図形を描画する。

```

java : figure.java
1 package report2;
2
3 import java.awt.*;
4
5 public class Figure extends Coord {
6
7     // 変数wの宣言 //
8     Color color;
9     boolean fillStatus = false, isPerfect = false;
```

```

10     int w, h;
11
12     // コンストラクタ //
13     Figure() {
14         w = h = 0;
15     }
16
17     public void paint(Graphics g) {
18     }
19
20     public void setWH(int w, int h) {
21         this.w = w;
22         this.h = h;
23     }
24
25     // 正方形・正円を設定するセッター //
26     public void setPerfect(boolean isPerfect) {
27         this.isPerfect = isPerfect;
28     }
29 }
```

Figureクラスは、**Coord**メソッドを継承したメソッドである。

8：課題1．引数で受け取った色を格納するための、Color型の変数。

9：課題7．9．10．**fillStatus**は引数で受け取った塗りつぶしの可否を格納する変数、**isPerfect**は正円・正方形を描く際に**true**にする変数。

26-28：正方形・正円を描画するかどうか、**isPerfect**に代入するためのセッター。

java : coord.java

```

1 package report2;
2
3 import java.io.Serializable;
4
5 public class Coord implements Serializable {
6
7     // 変数の宣言 //
8     int x, y;
```

```

9   Coord() {x = y = 0;}
10
11 // コンストラクタ //
12 Coord(int x, int y) {
13     this.x = x;
14     this.y = y;
15 }
16
17 public void move(int dx, int dy) {
18     x += dx;
19     y += dy;
20 }
21
22 public void moveto(int x, int y) {
23     this.x = x;
24     this.y = y;
25 }
26 }
```

Coordクラスは、配布のプログラムから変更を加えていない。

```

java : menuiconbtn.java
1 package report2;
2
3 import java.awt.*;
4
5 class MenuIconBtn extends MenuBtn {
6
7     // コンストラクタ //
8     public MenuIconBtn(String text, String command, String tooltip) {
9         super(text, command, tooltip); // 継承元に渡す
10        setFont(new Font("Font Awesome 6 Free", Font.PLAIN, 15));
11    }
12 }
```

MenuIconBtnクラスは、ツールバーにてアイコン付きのボタンを生成する際に用いるクラスである。 **MenuBtn**クラスを継承する。

8：引数で表示するテキスト， アクションコマンド， ツールチップに表示する内容をString型で受け取る。

9：継承元のコンストラクタに引数で受け取ったテキスト， アクションコマンド， ツールチップの内容を渡す。

10：課題15-2. ボタンのフォントをFont Awesome 6 Freeに設定し， アイコンの表示を可能にする。

```
java : menutbutton.java
1 package report2;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 class MenuTButton extends JToggleButton {
7
8     // コンストラクタ //
9     public MenuTButton(String text, String command, Boolean selected) {
10         super(text, selected); //継承元に渡す
11         setActionCommand(command);
12         setPreferredSize(new Dimension(80, 40));
13         setCursor(new Cursor(Cursor.HAND_CURSOR));
14     }
15 }
```

MenuTButtonクラスは， ツールバーに表示するトグルボタンを生成する際に用いるクラスである。 JToggleButtonクラスを継承する。

9：引数で表示するテキスト， アクションコマンド， 選択状態を受け取る。

10：受け取ったボタンに表示するテキストと選択状態を継承元に渡す。

11：引数で指定されたアクションコマンドをボタンに追加する。

12：ボタンの大きさを設定する。

13：課題15-1. ボタンにホバーした際のカーソルを手のカーソルに変える。

```
java : menubtn.java
```

```

1 package report2;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 class MenuBtn extends JButton {
7
8     // コンストラクタ //
9     public MenuBtn(String text, String command, String tooltip) {
10        super(text);
11        setCursor(new Cursor(Cursor.HAND_CURSOR));
12        setActionCommand(command);
13        setPreferredSize(new Dimension(80, 40));
14        setToolTipText(tooltip);
15    }
16}
```

MenuBtnクラスは、ツールバーに表示するボタンを生成する際に用いるクラスである。**JToggleButton**クラスを継承する。

9：引数で表示するテキスト、アクションコマンド、ツールチップの内容を受け取る。

10：受け取ったボタンに表示するテキストを継承元に渡す。

11：課題15-3. ボタンにホバーした際のカーソルを手のカーソルに変える。

12：引数で指定されたアクションコマンドをボタンに追加する。

13：ボタンの大きさを設定する。

13：課題15-3. ボタンにホバーした際のツールチップを設定する。

5. 動作検証、および実行環境

1. 色選択ボタンを押し、カラーピッカーが起動することを確認する。図形が選択した色で描画されることを確認する。
2. どの方向へ動かしても四角形が描画されることを確認する。

3. 橋円が描画されるか確認する.
4. 折れ線が描画されることを確認する. Enterを押して折れ線の描画がリセットされることを確認する.
5. Undoボタンを押して、直前に描画された図形が消えることを確認する。図形が何も表示されていない場合、Undoボタンが無効化されていることを確認する。
6. Redoボタンを押して、Undoされていた図形が再度表示されることを確認する。最新の図形が描画されている場合、Redoボタンが無効化されていることを確認する。
7. 塗りつぶしボタンを押すことで、描画する図形の塗りつぶし切り替えが可能であることを確認する。
8. ファイル保存ボタンを押し、描画されている図形を保存した状態で全てのオブジェクトを削除し保存したファイルを読み込んだ際、先ほど描画した図形が表示されることを確認する。
9. 四角形描画モードにて、Shiftキーを押している間だけ正方形が描画されることを確認する。
10. 円描画モードにて、Shiftキーを押している間だけ正円が描画されることを確認する。
11. 描画する図形モードを切り替えた際、カーソルが切り替わることを確認する。
カーソルは以下の様に切り替わる
 - ドット描画モード → 矢印
 - 円描画モード → クロスヘア
 - 四角形描画モード → クロスヘア
 - 線描画モード → クロスヘア
12. アプリケーションの操作に応じて、アプリケーション下部に表示されているステータスバーに現在のアプリケーションの状態が表示されることを確認する。
13. いずれかの方法でアプリケーションを終了した場合、保存を促すダイアログが表示され、選択したボタンに応じて以下の様に処理されることを確認する。
 - Yes → 保存ダイアログが開く。

- No → 何も表示されずに終了する.
 - Cancel → プログラムは終了されず、ダイアログが閉じる.
14. 削除ボタンを押し、描画されていた図形が削除されることを確認する。
15. メインウィンドウの右側にツールバーが表示されることを確認し、以下の操作が行えることも併せて確認する。
1. ボタンにカーソルがホバーした際、機能説明のツールチップが表示される。
 2. ボタンにアイコンが表示されている。
 3. ボタンにカーソルがホバーした際、手のカーソルに変わる。

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。

なお、アイコンの表示にはフォントを使用しているため、下記のフォントが実行環境にインストールされていることが動作の必須要件となる。

"フォントの入手ページ：<https://fontawesome.com/download>
上記リンクから、Free for Desktop を選択してダウンロードが可能"

また、Solid-900,.otf以外の同シリーズフォントがインストールされている場合、アイコンが表示されない場合がある。

Device Macbook Air

CPU Apple M1

OS macOS Ventura 13.0.1

Memory 16GB

Java version Java17

IDE IntelliJ IDEA 2022.1.4

Icon Font Font Awesome 6 Free-Solid-900.otf (Version 6.2.1)

6. 実行結果

検証1.

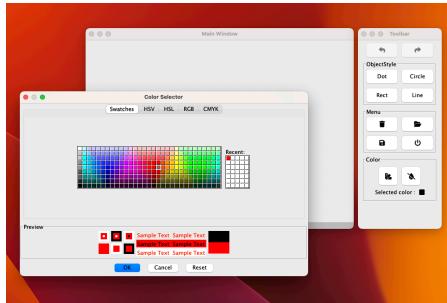


図1：色選択ボタンを押してカラーピッカーが表示された。

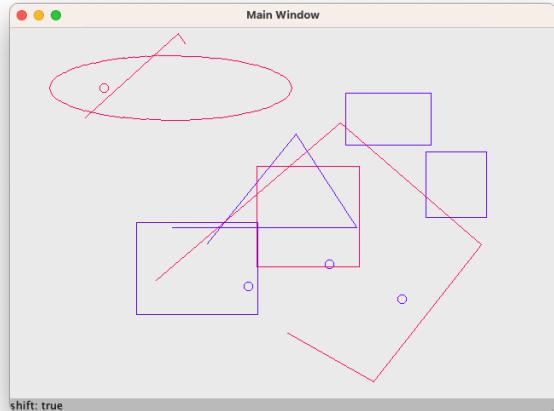


図2：指定した色で描画された。.

図1のように、カラーパレットアイコンの色選択ボタンを押すとカラーピッカーが表示されるた。

また、図2のようにカラーピッカーで指定した色を使い、描画することができた。

検証2.

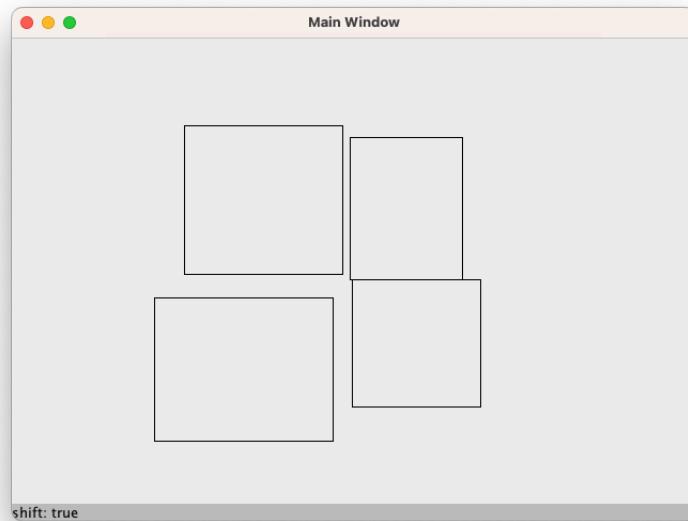


図3：中心から四方向へマウスをドラッグした。

図3のようにウィンドウの中心から四方向にマウスをドラッグし、四角形を描画することができた。

検証3.

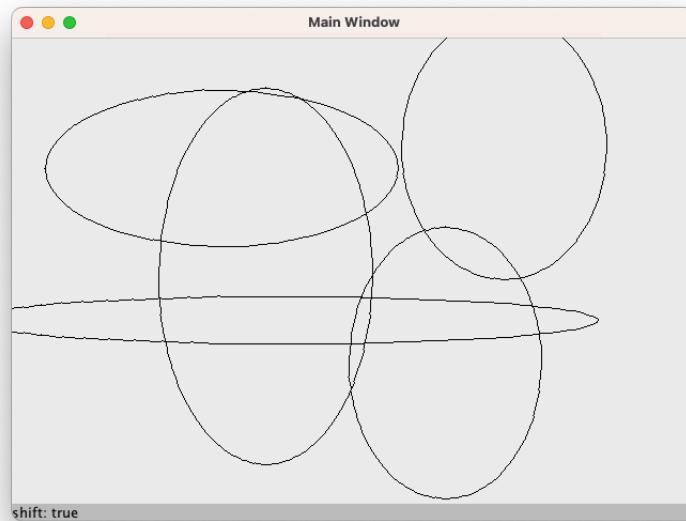


図4：橙円を描画.

図4のように、円描画モードでマウスをドラッグすることにより、橙円の描画が行えた。

検証4.

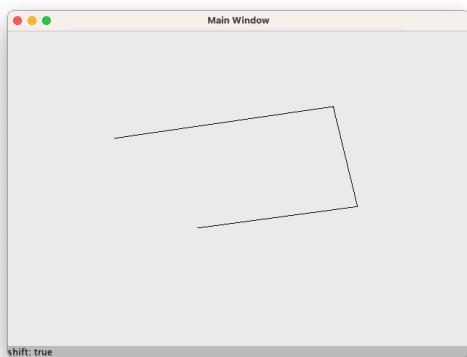


図5：折れ線の描画.

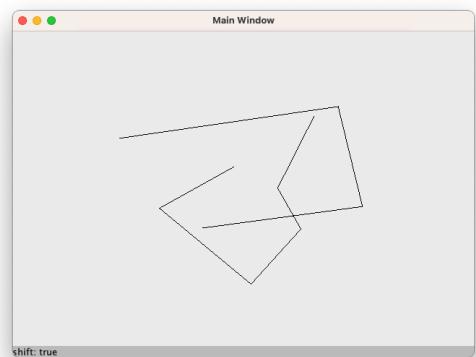


図6：Enterで一度描画をリセットし、再度描画した.

図5のように連続した折れ線をびようができた。

また、図6のように折れ線の描画、Enterを押して描画を完了することで複数の折れ線の描画が行えた。

検証5.

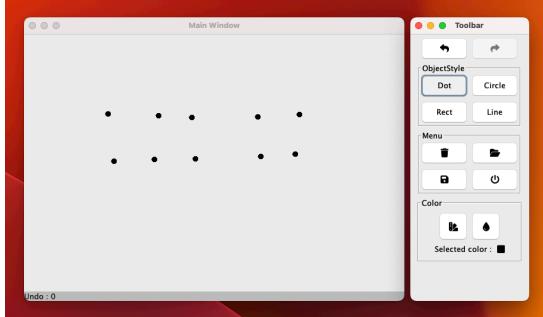


図7：ドットを10個描画。

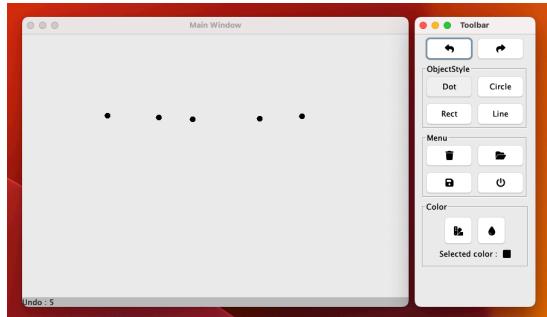


図8：Undoボタンを5回押した。

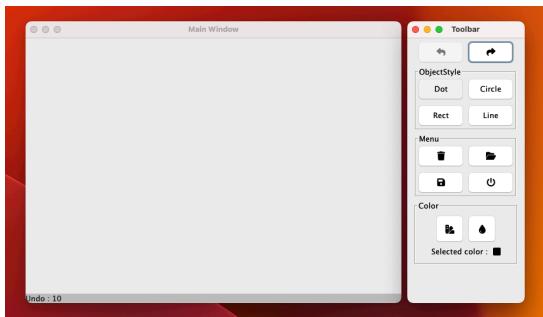


図9：Undoを10回押した。

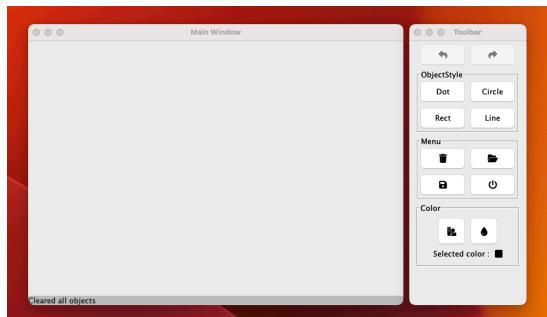


図10：ドットを10個描画したのち、削除ボタンを押した。

図7の様に、図形を描画するとUndoボタンが押せるようになった。

図8、図9のように、Undoボタンを押すことで描画されていた図形が消えていった。

また、図9、図10のように、ウィンドウ上に図形が何も描画されていない状況でUndoボタンを押すことはできない。

検証6.

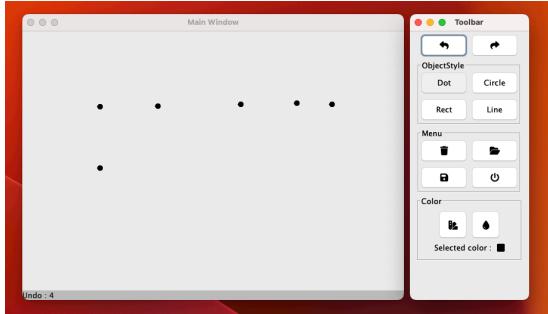


図11：ドットを10個描画したのち、 Undo を4回押した。

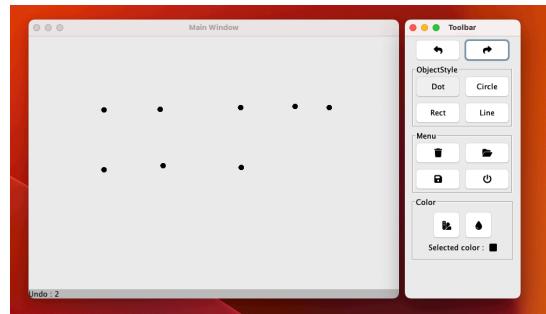


図12：Redo を2回押した。

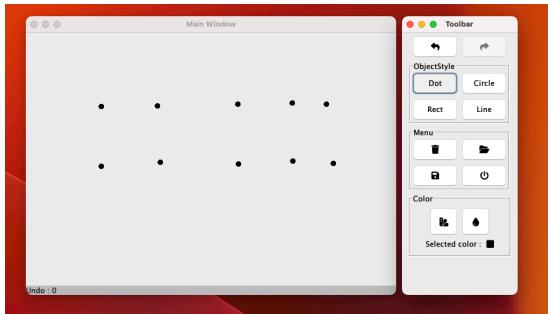


図13：Redo を4回押した。

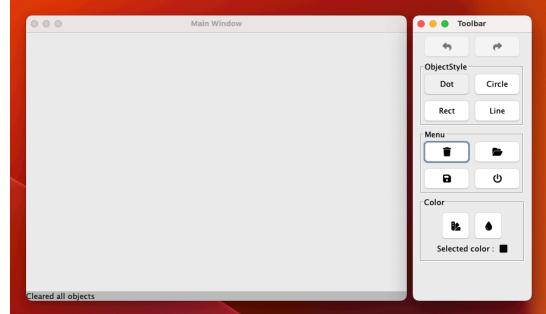


図14：削除ボタンを押した。

図11のように、 Undo を押して遡ると Redo ボタンが押せるようになった。

図12、図13のように、 Redo を押すことで Undo によって表示されていなかった図形が再び描画された。

図13、図14のように、 ウィンドウの表示が最新の状態では Redo ボタンは押せない。

検証7.

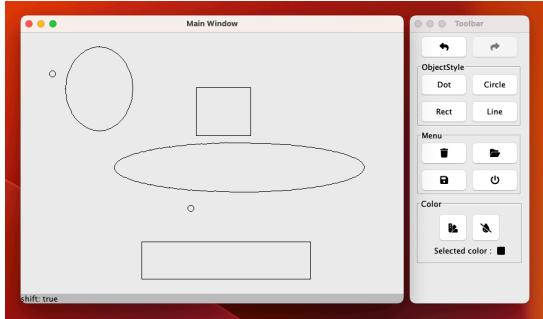


図15：塗りつぶしをオフにして描画。

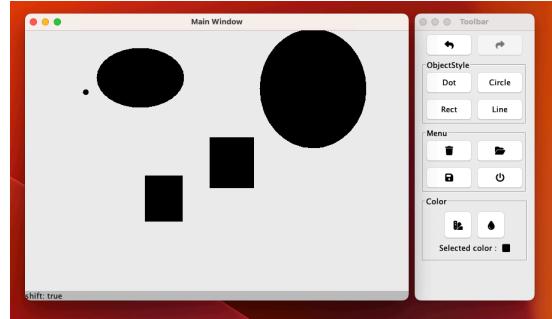


図16：塗りつぶしをオンにして描画。

図15、図16のように、塗りつぶしのオン・オフを切り替えて図形を描画可能。

検証8.

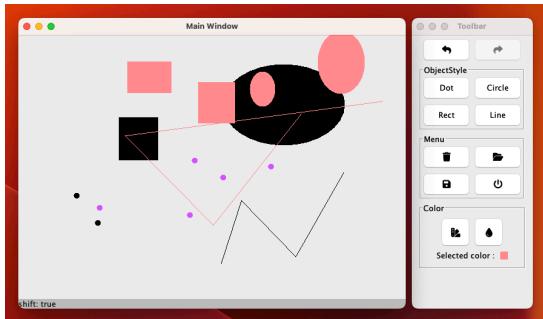


図17：図を描画し、保存。



図18：一度プログラムを終了後、被再度開いた。

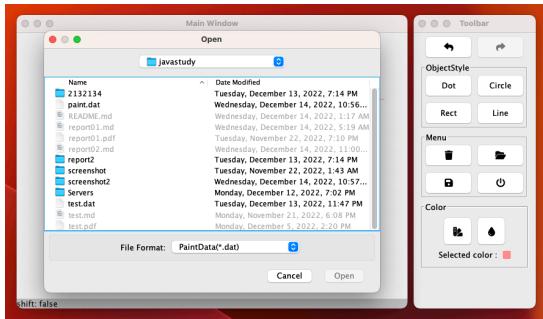


図19：ファイルを開くダイアログ。

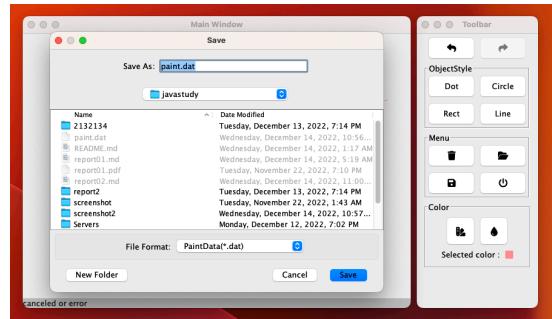


図20：ファイルを保存するダイアログ。

図17、図18のように、描画した図の位置や色なども保存され、読み込むと再度描画された。

図19、図20のように、開くボタン、保存ボタンを押すとダイアログが表示された。

検証9. 検証10.

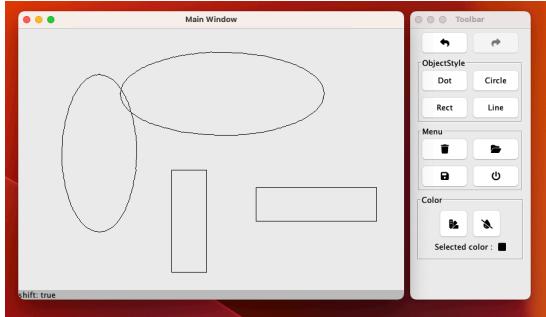


図21：Shiftを押さずに描画。

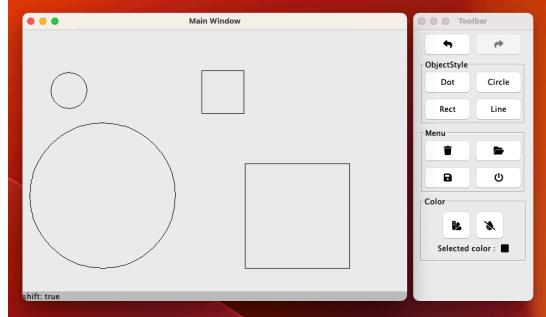


図22：Shiftを押して描画。

図21のように、Shiftを押さずに描画すると楕円と長方形が描ける。

図22のように、Shiftを押して描画すると正円と正方形を描画できる。

検証11.

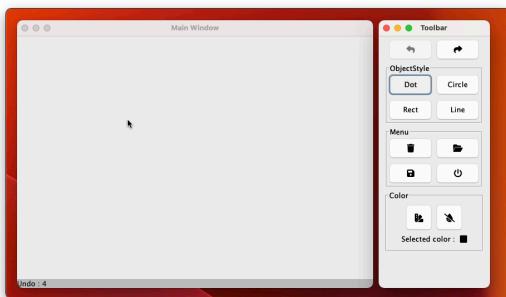


図23：ドットを選択。

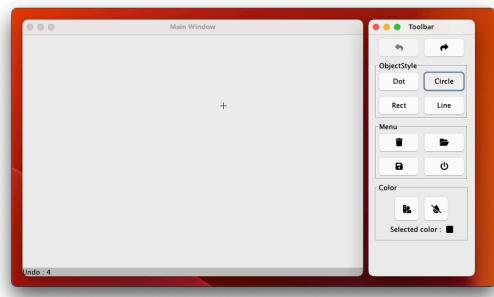


図24：円を選択。

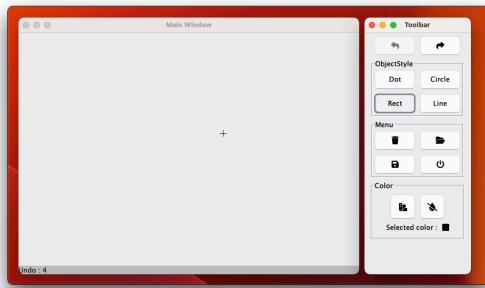


図25：四角形を選択。

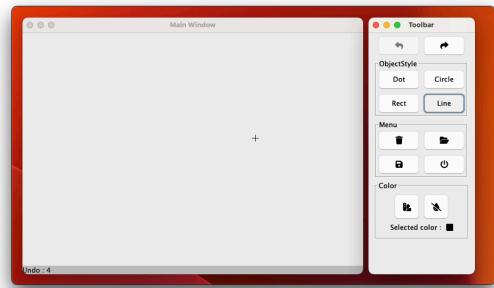


図26：線を選択。

図23～図26からわかるように、図の種類に適したカーソルに変わっている。

検証12.

上記1から11の検証を確認すると、ウィンドウ下部に表示されているステータスバーが、行った操作を表示していることがわかる。

検証13.

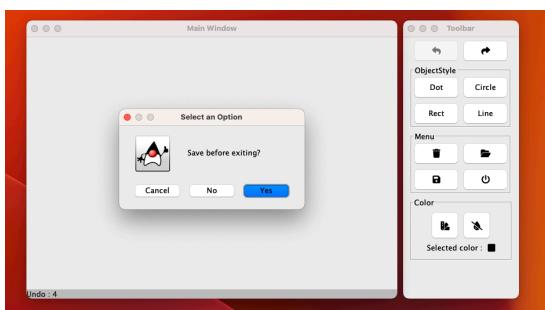


図27：閉じる操作を行った。

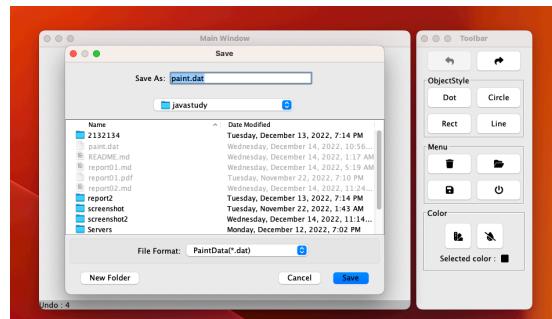


図28：表示されたダイアログのYesを押した。

図27のように、閉じる操作を行うと、確認メッセージとともにダイアログが表示された。

図28のように、Yesを押すことで保存を行える。

また、写真に収められないため図はないが、Noを押した場合にはプログラムが終了し、Cancelを押した場合には再びアプリケーションの使用を続行できた。

なお、ウィンドウ上部のバツボタンにてプログラムを終了した場合も同様である。

検証14.

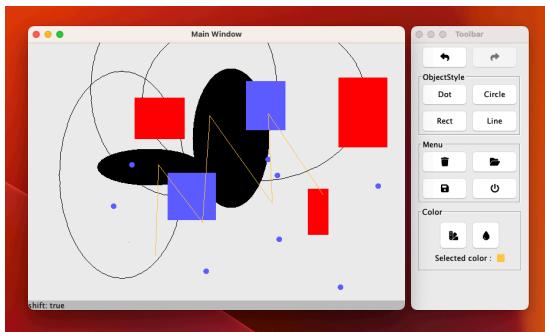


図29：図を描画。

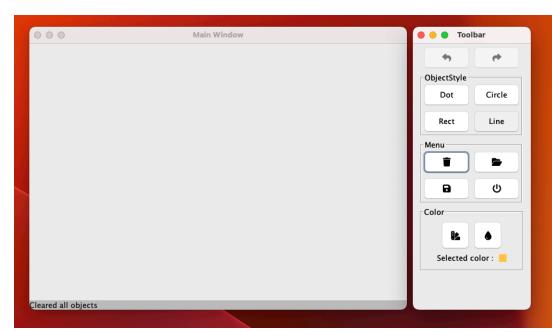


図30：削除ボタンを押した。

図29、図30からわかるように、削除ボタンを押すことで描画されていた図形を全て削除できる。

検証15.

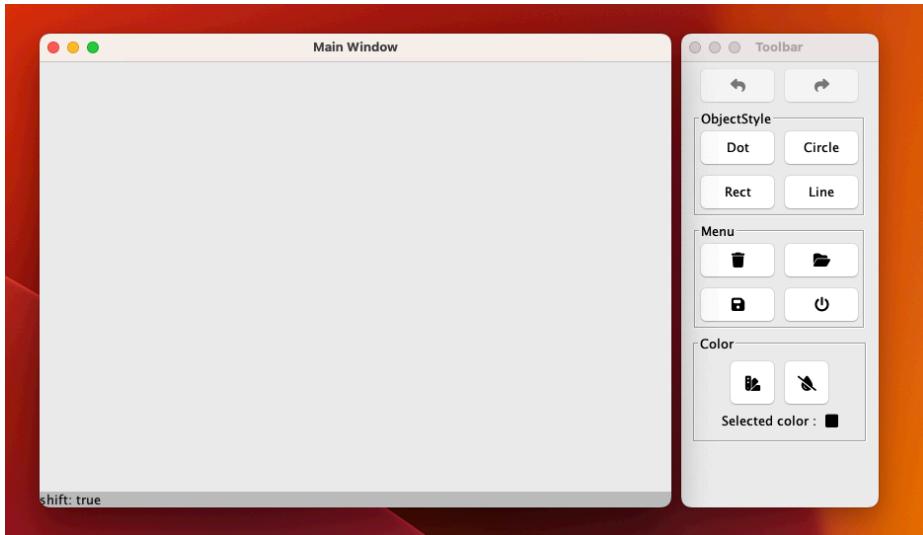


図3 1：起動した直後の画面。



図3 2：ツールバー



図3 4：塗りつぶしボタンにホバーした。

図3 1のように、起動するとメインウィンドウの右側にツールバーが表示された。

図3 2～図3 4のように、ツールバーのボタンにはアイコンが表示され、ボタンにホバーすると、そのボタンの簡単な動作説明がツールチップにて表示される。

7. まとめ

継承などを積極的に利用し、Javaのオブジェクト指向言語としての特性を学んだ。また、awtでは実現できなかったUIの細かな動作をSwingを使い構築することで、より実践的なアプリケーションを開発するスキルを身につけることができた。ボタンにアイコンを付けるなど、視覚的にわかりやすいUIを制作することで、ユーザビリティが格段に向上了ることがわかった。図形の移動や変形、回転など、実装できなかった機能に関しては今後の課題である。