

avr 02, 17 16:17	Makefile	Page 1/3
1	# Executables	
2	OSTYPE = \$(shell uname -s)	
3	JAVAC = javac	
4	JAVA = java	
5	A2PS = a2ps-utf8	
6	GHOSTVIEW = gv	
7	DOCP = javadoc	
8	ARCH = zip	
9	PS2PDF = ps2pdf -sPAPERSIZE=a4	
10	DATE = \$(shell date +%Y-%m-%d-%H-%M-%S)	
11	# Options de compilation	
12	#CFLAGS = -verbose	
13	CFLAGS =	
14	CLASSPATH=.	
15		
16	JAVAOPTIONS = --verbose	
17		
18	PROJECT=TP FiguresEditor	
19	# nom du fichier d'impression	
20	OUTPUT = \$(PROJECT)	
21	# nom du répertoire ou se situera la documentation	
22	DOC = doc	
23	# lien vers la doc en ligne du JDK	
24	WEBLINK = "http://docs.oracle.com/javase/8/docs/api/"	
25	# lien vers la doc locale du JDK	
26	LOCALLINK = "file:///Users/davidroussel/Documents/docs/java/api/"	
27	# nom de l'archive	
28	ARCHIVE = \$(PROJECT)	
29	# format de l'archive pour la sauvegarde	
30	ARCHFMT = zip	
31	# Répertoire source	
32	SRC = src	
33	# Répertoire bin	
34	BIN = bin	
35	# Répertoire Listings	
36	LISTDIR = listings	
37	# Répertoire Archives	
38	ARCHDIR = archives	
39	# Répertoire Figures	
40	FIGDIR = graphics	
41	# noms des fichiers sources	
42	MAIN = Editor ShapesDemo2D	
43	SOURCES = \$(foreach name, \$(MAIN), \$(SRC)/\$(name).java) \	
44	\$(SRC)/figures/package-info.java \	
45	\$(SRC)/figures/Figure.java \	
46	\$(SRC)/figures/NGon.java \	
47	\$(SRC)/figures/Polygon.java \	
48	\$(SRC)/figures/Rectangle.java \	
49	\$(SRC)/figures/RoundedRectangle.java \	
50	\$(SRC)/figures/Circle.java \	
51	\$(SRC)/figures/Drawing.java \	
52	\$(SRC)/figures/Ellipse.java \	
53	\$(SRC)/figures/enums/package-info.java \	
54	\$(SRC)/figures/enums/FigureType.java \	
55	\$(SRC)/figures/enums/LineType.java \	
56	\$(SRC)/figures/enums/PaintToType.java \	
57	\$(SRC)/figures/listeners/AbstractFigureListener.java \	
58	\$(SRC)/figures/listeners/creation/AbstractCreationListener.java \	
59	\$(SRC)/figures/listeners/creation/NGonCreationListener.java \	
60	\$(SRC)/figures/listeners/creation/package-info.java \	
61	\$(SRC)/figures/listeners/creation/PolygonCreationListener.java \	
62	\$(SRC)/figures/listeners/creation/RectangularShapeCreationListener.java \	
63	\$(SRC)/figures/listeners/creation/RoundedRectangleCreationListener.java \	
64	\$(SRC)/figures/listeners/package-info.java \	
65	\$(SRC)/figures/listeners/SelectionFigureListener.java \	
66	\$(SRC)/figures/listeners/transform/AbstractTransformShapeListener.java \	
67	\$(SRC)/figures/listeners/transform/MoveShapeListener.java \	
68	\$(SRC)/figures/listeners/transform/package-info.java \	
69	\$(SRC)/figures/listeners/transform/RotateShapeListener.java \	
70	\$(SRC)/figures/listeners/transform/ScaleShapeListener.java \	
71	\$(SRC)/figures/treemodels/AbstractFigureTreeModel.java \	
72	\$(SRC)/figures/treemodels/FigureTreeModel.java \	
73	\$(SRC)/figures/treemodels/FigureTypeTreeModel.java \	
74	\$(SRC)/figures/treemodels/package-info.java \	
75	\$(SRC)/filters/EdgeColorFilter.java \	
76	\$(SRC)/filters/FigureFilter.java \	
77	\$(SRC)/filters/FigureFilters.java \	
78	\$(SRC)/filters/FillColorFilter.java \	
79	\$(SRC)/filters/LineFilter.java \	
80	\$(SRC)/filters/ShapeFilter.java \	
81	\$(SRC)/utils/FlyweightFactory.java \	
82	\$(SRC)/utils/IconFactory.java \	
83	\$(SRC)/utils/IconItem.java \	
84	\$(SRC)/utils/package-info.java \	
85	\$(SRC)/utils/PaintFactory.java \	
86	\$(SRC)/utils/StrokeFactory.java \	
87	\$(SRC)/utils/Vector2D.java \	
88	\$(SRC)/widgets/DrawingPanel.java \	
89	\$(SRC)/widgets/EditorFrame.java \	
90	\$(SRC)/widgets/EditorFrame2.java \	

avr 02, 17 16:17	Makefile	Page 2/3
91	\$(SRC)/widgets/enums/OperationMode.java \	
92	\$(SRC)/widgets/enums/package-info.java \	
93	\$(SRC)/widgets/enums/TreeType.java \	
94	\$(SRC)/widgets/InfoPanel.java \	
95	\$(SRC)/widgets/JLabeledComboBox.java \	
96	\$(SRC)/widgets/package-info.java \	
97	\$(SRC)/widgets/TreesPanel.java	
98		
99	OTHER = \$(SRC)/images/About.png \	
100	\$(SRC)/images/About_small.png \	
101	\$(SRC)/images/Black.png \	
102	\$(SRC)/images/Blue.png \	
103	\$(SRC)/images/Circle.png \	
104	\$(SRC)/images/Circle_small.png \	
105	\$(SRC)/images/Clear.png \	
106	\$(SRC)/images/Clear_small.png \	
107	\$(SRC)/images/ClearFilter.png \	
108	\$(SRC)/images/ClearFilter_small.png \	
109	\$(SRC)/images/Creation.png \	
110	\$(SRC)/images/Creation_small.png \	
111	\$(SRC)/images/Cyan.png \	
112	\$(SRC)/images/Dashed.png \	
113	\$(SRC)/images/Dashed_small.png \	
114	\$(SRC)/images/Delete.png \	
115	\$(SRC)/images/Delete_small.png \	
116	\$(SRC)/images/Details.png \	
117	\$(SRC)/images/Details_small.png \	
118	\$(SRC)/images/EdgeColor.png \	
119	\$(SRC)/images/EdgeColor_small.png \	
120	\$(SRC)/images/Edition.png \	
121	\$(SRC)/images/Edition_small.png \	
122	\$(SRC)/images/Ellipse.png \	
123	\$(SRC)/images/Ellipse_small.png \	
124	\$(SRC)/images/FillColor.png \	
125	\$(SRC)/images/FillColor_small.png \	
126	\$(SRC)/images/Filter.png \	
127	\$(SRC)/images/Filter_small.png \	
128	\$(SRC)/images/Green.png \	
129	\$(SRC)/images/Logo.png \	
130	\$(SRC)/images/Magenta.png \	
131	\$(SRC)/images/Move.png \	
132	\$(SRC)/images/Move_small.png \	
133	\$(SRC)/images/MoveDown.png \	
134	\$(SRC)/images/MoveDown_small.png \	
135	\$(SRC)/images/MoveUp.png \	
136	\$(SRC)/images/MoveUp_small.png \	
137	\$(SRC)/images/Ngon.png \	
138	\$(SRC)/images/Ngon_small.png \	
139	\$(SRC)/images/None.png \	
140	\$(SRC)/images/None_small.png \	
141	\$(SRC)/images/Orange.png \	
142	\$(SRC)/images/Others.png \	
143	\$(SRC)/images/Polygon.png \	
144	\$(SRC)/images/Polygon_small.png \	
145	\$(SRC)/images/Quit.png \	
146	\$(SRC)/images/Quit_small.png \	
147	\$(SRC)/images/Rectangle.png \	
148	\$(SRC)/images/Rectangle_small.png \	
149	\$(SRC)/images/Red.png \	
150	\$(SRC)/images/Redo.png \	
151	\$(SRC)/images/Redo_small.png \	
152	"\$(SRC)/images/Rounded Rectangle.png" \	
153	"\$(SRC)/images/Rounded Rectangle_small.png" \	
154	\$(SRC)/images/RoundedRectangle.png \	
155	\$(SRC)/images/RoundedRectangle_small.png \	
156	\$(SRC)/images/Solid.png \	
157	\$(SRC)/images/Solid_small.png \	
158	\$(SRC)/images/Star.png \	
159	\$(SRC)/images/Star_small.png \	
160	\$(SRC)/images/Style.png \	
161	\$(SRC)/images/Style_small.png \	
162	\$(SRC)/images/Tree.png \	
163	\$(SRC)/images/Tree_small.png \	
164	\$(SRC)/images/Undo.png \	
165	\$(SRC)/images/Undo_small.png \	
166	\$(SRC)/images/White.png \	
167	\$(SRC)/images/Yellow.png \	
168	TP5.pdf	
169		
170	.PHONY : doc ps	
171		
172	# Les cibles de compilation	
173	# pour générer l'application	
174	all : \$(foreach name, \$(MAIN), \$(BIN)/\$(name).class)	
175		
176	#règle de compilation générique	
177	\$(BIN)/%.class : \$(SRC)/%.java	
178	\$(JAVAC) -sourcepath \$(SRC) -classpath \$(BIN):\$(CLASSPATH) -d \$(BIN) \$(CFLAGS) <	
179		
180	# Edition des sources \$(EDITOR) doit être une variable d'environnement	

avr 02, 17 16:17

## Makefile

Page 3/3

```

181 edit :
182     $(EDITOR) $(SOURCES) Makefile &
183
184 # nettoyer le répertoire
185 clean :
186     find bin/ -type f -name "*.class" -exec rm -f {} \;
187     rm -rf ~/. $(DOC)/* $(LISTDIR)/*
188
189 #realclean : clean
190 #     rm -f $(ARCHDIR)/*.$(ARCHFMT)
191
192 # générer le listing
193 $(LISTDIR) :
194     mkdir $(LISTDIR)
195
196 ps : $(LISTDIR)
197     $(A2PS) -2 --file-align=fill --line-numbers=1 --font-size=10 \
198     --chars-per-line=100 --tabsize=4 --pretty-print \
199     --highlight-level=heavy --prologue="gray" \
200     -o$(LISTDIR)/$(OUTPUT).ps Makefile $(SOURCES)
201
202 pdf : ps
203     $(PS2PDF) $(LISTDIR)/$(OUTPUT).ps $(LISTDIR)/$(OUTPUT).pdf
204
205 # générer le listing lisible pour Gérard
206 bigps :
207     $(A2PS) -1 --file-align=fill --line-numbers=1 --font-size=10 \
208     --chars-per-line=100 --tabsize=4 --pretty-print \
209     --highlight-level=heavy --prologue="gray" \
210     -o$(LISTDIR)/$(OUTPUT).ps Makefile $(SOURCES)
211
212 bigpdf : bigps
213     $(PS2PDF) $(LISTDIR)/$(OUTPUT).ps $(LISTDIR)/$(OUTPUT).pdf
214
215 # voir le listing
216 preview : ps
217     $(HOSTVIEW) $(LISTDIR)/$(OUTPUT); rm -f $(LISTDIR)/$(OUTPUT) $(LISTDIR)/$(OUTPUT)~
218
219 # générer la doc avec javadoc
220 doc : $(SOURCES)
221     $(DOCP) -private -d $(DOC) -author -link $(LOCALLINK) $(SOURCES)
222     # $(DOCP) -private -d $(DOC) -author -linkoffline $(WEBLINK) $(LOCALLINK) $(SOURCES)
223
224 # générer une archive de sauvegarde
225 $(ARCHDIR) :
226     mkdir $(ARCHDIR)
227
228 archive : pdf $(ARCHDIR)
229     $(ARCH) $(ARCHDIR)/$(ARCHIVE)-$(DATE).$(ARCHFMT) $(SOURCES) $(LISTDIR)/*.pdf $(OTHER) $(BIN) Mak
230     efile $(FIGDIR)/*.pdf
231
232 # exécution des programmes de test
233 run : all
234     $(foreach name, $(MAIN), $(JAVA) -classpath $(BIN):$(CLASSPATH) $(name) $(JAVAOPTIONS) )

```

avr 02, 17 16:17

## Editor.java

Page 1/2

```

1 import java.awt.EventQueue;
2
3 import javax.swing.UIManager;
4 import javax.swing.UIManager.LookAndFeelInfo;
5 import javax.swing.UnsupportedLookAndFeelException;
6
7 import widgets.EditorFrame;
8
9 /**
10  * Programme principal lançant la fenêtre {@link EditorFrame}
11  * @author davidroussel
12  */
13 public class Editor
14 {
15     /**
16      * Programme principal
17      * @param args arguments : le nom du look and feel à utiliser
18      */
19     public static void main(String[] args)
20     {
21         /**
22          * Mise en place du look and feel du système, ou celui fourni en
23          * argument du programme
24          */
25         try
26         {
27             if (args.length == 0)
28             {
29                 UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
30             }
31             else
32             {
33                 String lookAndFeelName = args[0];
34                 LookAndFeelInfo[] lafis = UIManager.getInstalledLookAndFeels();
35                 for (LookAndFeelInfo lafi : lafis)
36                 {
37                     if (lafi.getName().toLowerCase().equals(lookAndFeelName.toLowerCase()))
38                     {
39                         UIManager.setLookAndFeel(lafi.getClassName());
40                         break;
41                     }
42                 }
43             }
44         }
45         catch (ClassNotFoundException e)
46         {
47             System.err.println("Look and feel could not be found");
48             e.printStackTrace();
49         }
50         catch (InstantiationException e)
51         {
52             System.err.println("new instance of the class couldn't be created");
53             e.printStackTrace();
54         }
55         catch (IllegalAccessException e)
56         {
57             System.err.println("Look and feel class or initializer isn't accessible");
58             e.printStackTrace();
59         }
60         catch (UnsupportedLookAndFeelException e)
61         {
62             System.err.println("isSupportedLookAndFeel() is false");
63             e.printStackTrace();
64         }
65         catch (ClassCastException e)
66         {
67             System.err.println("className does not identify a class that extends LookAndFeel");
68             e.printStackTrace();
69         }
70     }
71
72     // Mise en place spécifique à Mac OS X
73     String osName = System.getProperty("os.name");
74     if (osName.startsWith("Mac OS"))
75     {
76         macOSSettings();
77     }
78
79     /**
80      * Création de la fenêtre
81      */
82     final EditorFrame frame = new EditorFrame();
83
84     /**
85      * Insertion de la fenêtre dans la file des événements GUI
86      */
87     EventQueue.invokeLater(new Runnable()
88     {
89         @Override
90         public void run()

```

avr 02, 17 16:17

Editor.java

Page 2/2

```

91     {
92         try
93         {
94             frame.pack();
95             frame.setVisible(true);
96         }
97         catch (Exception e)
98         {
99             e.printStackTrace();
100        }
101    }
102    });
103 }
104 }
105
106 /**
107  * Mise en place des options spécifiques à MacOS.
108  * A virer si votre système n'est pas MacOS car com.apple... risque
109  * de ne pas exister
110  */
111 private static void macOSSettings()
112 {
113     // Remettre les menus au bon endroit (dans la barre en haut)
114     System.setProperty("apple.laf.useScreenMenuBar", "true");
115
116     // ImageIcon imageIcon = new ImageIcon(
117     //     Editor.class.getResource("/images/Logo.png"));
118     // if (imageIcon.getImageLoadStatus() == MediaTracker.COMPLETE)
119     // {
120     //     // Titre de l'application
121     //     System.setProperty(
122     //         "com.apple.mrj.application.apple.menu.about.name",
123     //         "Figure Editor");
124     //     // Chargement d'une icône pour le dock
125     //     com.apple.eawt.Application.setApplication().setDockIconImage(
126     //         imageIcon.getImage());
127     // }
128 }
129 }
130

```

avr 02, 17 16:17

ShapesDemo2D.java

Page 1/4

```

1  import java.awt.BasicStroke;
2  import java.awt.Color;
3  import java.awt.Dimension;
4  import java.awt.Font;
5  import java.awt.FontMetrics;
6  import java.awt.GradientPaint;
7  import java.awt.Graphics;
8  import java.awt.Graphics2D;
9  import java.awt.RenderingHints;
10 import java.awt.event.WindowAdapter;
11 import java.awt.event.WindowEvent;
12 import java.awt.geom.Arc2D;
13 import java.awt.geom.Ellipse2D;
14 import java.awt.geom.GeneralPath;
15 import java.awt.geom.Line2D;
16 import java.awt.geom.Path2D;
17 import java.awt.geom.Rectangle2D;
18 import java.awt.geom.RoundRectangle2D;
19
20 import javax.swing.JApplet;
21 import javax.swing.JFrame;
22
23 /*
24  * Copyright (c) 1995, 2008, Oracle and/or its affiliates. All rights reserved.
25  * Redistribution and use in source and binary forms, with or without
26  * modification, are permitted provided that the following conditions are met:
27  * - Redistributions of source code must retain the above copyright notice, this
28  *   list of conditions and the following disclaimer.
29  * - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and
30  *   the following disclaimer in the documentation and/or other materials provided
31  *   with the distribution.
32  * - Neither the name of Oracle nor the names of its
33  *   contributors may be used to endorse or promote products derived from this
34  *   software without specific prior written permission.
35  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
36  * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
37  * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
38  * EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
39  * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
40  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
41  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
42  * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
43  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
44  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
45  */
46
47 /*
48  * This is like the FontDemo applet in volume 1, except that it uses the Java 2D
49  * APIs to define and render the graphics and text.
50  */
51 @SuppressWarnings("serial")
52 public class ShapesDemo2D extends JApplet
53 {
54     protected final static int maxCharHeight = 15;
55     protected final static int minFontSize = 6;
56
57     protected final static Color bg = Color.white;
58     protected final static Color fg = Color.black;
59     protected final static Color red = Color.red;
60     protected final static Color white = Color.white;
61
62     protected final static BasicStroke stroke = new BasicStroke(2.0f);
63     protected final static BasicStroke wideStroke = new BasicStroke(8.0f,
64         BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND);
65
66     protected final static float lastWidth = 20.0f;
67     protected final static float dash1[] = { 2*lastWidth };
68     protected final static BasicStroke dashed = new BasicStroke(1.0f,
69         BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND, 10.0f, dash1, 0.0f);
70     protected final static BasicStroke fatDashed = new BasicStroke(lastWidth,
71         BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND, lastWidth, dash1, 0.0f);
72     protected Dimension totalSize;
73     protected FontMetrics fontMetrics;
74
75     @Override
76     public void init()
77     {
78         // Initialize drawing colors
79         setBackground(bg);
80         setForeground(fg);
81     }
82
83     FontMetrics pickFont(Graphics2D g2, String longString, int xSpace)
84     {
85         boolean fontFits = false;
86         Font font = g2.getFont();
87         FontMetrics fontMetrics = g2.getFontMetrics();
88         int size = font.getSize();
89         String name = font.getName();
90         int style = font.getStyle();

```

avr 02, 17 16:17

## ShapesDemo2D.java

Page 2/4

```

91     while (!fontFits)
92     {
93         if ((fontMetrics.getHeight() ≤ maxCharHeight)
94             ^ (fontMetrics.stringWidth(longString) ≤ xSpace))
95         {
96             fontFits = true;
97         }
98         else
99         {
100             if (size ≤ minFontSize)
101             {
102                 fontFits = true;
103             }
104             else
105             {
106                 g2.setFont(font = new Font(name, style, --size));
107                 fontMetrics = g2.getFontMetrics();
108             }
109         }
110     }
111 }
112
113 return fontMetrics;
114 }
115
116 @Override
117 public void paint(Graphics g)
118 {
119     Graphics2D g2 = (Graphics2D) g;
120     g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
121         RenderingHints.VALUE_ANTIALIAS_ON);
122     Dimension d = getSize();
123     int gridWidth = d.width / 6;
124     int gridHeight = d.height / 2;
125
126     fontMetrics = pickFont(g2, "Filled and Stroked GeneralPath", gridWidth);
127
128     Color fg3D = Color.lightGray;
129
130     // on commence par effacer le fond
131     g2.setColor(getBackground());
132     g2.fillRect(0, 0, d.width, d.height);
133
134     g2.setPaint(fg3D);
135     g2.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
136     g2.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
137     g2.setPaint(fg);
138
139     int x = 5;
140     int y = 7;
141     int rectWidth = gridWidth - (2 * x);
142     int stringY = gridHeight - 3 - fontMetrics.getDescent();
143     int rectHeight = stringY - fontMetrics.getMaxAscent() - y - 2;
144
145     // draw Line2D.Double
146     g2.draw(new Line2D.Double(x, (y + rectHeight) - 1, x + rectWidth, y));
147     g2.drawString("Line2D", x, stringY);
148     x += gridWidth;
149
150     // draw Rectangle2D.Double
151     g2.setStroke(stroke);
152     g2.draw(new Rectangle2D.Double(x, y, rectWidth, rectHeight));
153     g2.drawString("Rectangle2D", x, stringY);
154     x += gridWidth;
155
156     // draw RoundRectangle2D.Double
157     g2.setStroke(dashed);
158     g2.draw(new RoundRectangle2D.Double(x, y, rectWidth, rectHeight, 10, 10));
159     g2.drawString("RoundRectangle2D", x, stringY);
160     x += gridWidth;
161
162     // draw Arc2D.Double
163     g2.setStroke(wideStroke);
164     g2.draw(new Arc2D.Double(x, y, rectWidth, rectHeight, 90, 135,
165         Arc2D.OPEN));
166     g2.drawString("Arc2D", x, stringY);
167     x += gridWidth;
168
169     // draw Ellipse2D.Double
170     g2.setStroke(stroke);
171     g2.draw(new Ellipse2D.Double(x, y, rectWidth, rectHeight));
172     g2.drawString("Ellipse2D", x, stringY);
173     x += gridWidth;
174
175     // draw GeneralPath (polygon)
176     int x1Points[] = { x, x + rectWidth, x, x + rectWidth };
177     int y1Points[] = { y, y + rectHeight, y + rectHeight, y };
178     GeneralPath polygon = new GeneralPath(Path2D.WIND_EVEN_ODD,
179         x1Points.length);
180     polygon.moveTo(x1Points[0], y1Points[0]);

```

avr 02, 17 16:17

## ShapesDemo2D.java

Page 3/4

```

181     for (int index = 1; index < x1Points.length; index++)
182     {
183         polygon.lineTo(x1Points[index], y1Points[index]);
184     }
185
186     polygon.closePath();
187
188     g2.draw(polygon);
189     g2.drawString("GeneralPath", x, stringY);
190
191     // NEW ROW
192     x = 5;
193     y += gridHeight;
194     stringY += gridHeight;
195
196     // draw GeneralPath (polyline)
197
198     int x2Points[] = { x, x + rectWidth, x, x + rectWidth };
199     int y2Points[] = { y, y + rectHeight, y + rectHeight, y };
200     GeneralPath polyline = new GeneralPath(Path2D.WIND_EVEN_ODD,
201         x2Points.length);
202     polyline.moveTo(x2Points[0], y2Points[0]);
203     for (int index = 1; index < x2Points.length; index++)
204     {
205         polyline.lineTo(x2Points[index], y2Points[index]);
206     }
207
208     g2.draw(polyline);
209     g2.drawString("GeneralPath (open)", x, stringY);
210     x += gridWidth;
211
212     // fill Rectangle2D.Double (red)
213     g2.setPaint(red);
214     g2.fill(new Rectangle2D.Double(x, y, rectWidth, rectHeight));
215     g2.setPaint(fg);
216     g2.drawString("Filled Rectangle2D", x, stringY);
217     x += gridWidth;
218
219     // fill RoundRectangle2D.Double
220     GradientPaint redtoWhite = new GradientPaint(x, y, red, x + rectWidth,
221         y, white);
222     g2.setPaint(redtoWhite);
223     g2.fill(new RoundRectangle2D.Double(x, y, rectWidth, rectHeight, 10, 10));
224     g2.setPaint(fg);
225     g2.drawString("Filled RoundRectangle2D", x, stringY);
226     x += gridWidth;
227
228     // fill Arc2D
229     g2.setPaint(red);
230     g2.fill(new Arc2D.Double(x, y, rectWidth, rectHeight, 90, 135,
231         Arc2D.OPEN));
232     g2.setPaint(fg);
233     g2.drawString("Filled Arc2D", x, stringY);
234     x += gridWidth;
235
236     // fill Ellipse2D.Double
237     redtoWhite = new GradientPaint(x, y, red, x + rectWidth, y, white);
238     g2.setPaint(redtoWhite);
239     g2.fill(new Ellipse2D.Double(x, y, rectWidth, rectHeight));
240     g2.setPaint(fg);
241     g2.drawString("Filled Ellipse2D", x, stringY);
242     x += gridWidth;
243
244     // fill and stroke GeneralPath
245     int x3Points[] = { x, x + rectWidth, x, x + rectWidth };
246     int y3Points[] = { y, y + rectHeight, y + rectHeight, y };
247     GeneralPath filledPolygon = new GeneralPath(Path2D.WIND_EVEN_ODD,
248         x3Points.length);
249     filledPolygon.moveTo(x3Points[0], y3Points[0]);
250     for (int index = 1; index < x3Points.length; index++)
251     {
252         filledPolygon.lineTo(x3Points[index], y3Points[index]);
253     }
254     filledPolygon.closePath();
255
256     g2.setPaint(red);
257     g2.fill(filledPolygon);
258
259     g2.setStroke(fatDashed);
260     g2.setPaint(fg);
261     g2.draw(filledPolygon);
262
263     g2.drawString("Filled and Stroked GeneralPath", x, stringY);
264
265 }
266
267 public static void main(String s[])
268 {
269     JFrame f = new JFrame("ShapesDemo2D");
270     f.addWindowListener(new WindowAdapter()
271     {

```

avr 02, 17 16:17

**ShapesDemo2D.java**

Page 4/4

```
271         @Override
272         public void windowClosing(WindowEvent e)
273         {
274             System.exit(0);
275         }
276     });
277     JApplet applet = new ShapesDemo2D();
278     f.getContentPane().add("Center", applet);
279     applet.init();
280     f.pack();
281     f.setSize(new Dimension(550, 100));
282     f.setVisible(true);
283 }
284
285 }
```

avr 02, 17 16:17

**package-info.java**

Page 1/1

```
1  /**
2   * Package contenant les différents widgets (éléments graphiques)
3   */
4  package widgets;
```

avr 02, 17 16:17

## Figure.java

Page 1/6

```

1 package figures;
2
3 import java.awt.BasicStroke;
4 import java.awt.Color;
5 import java.awt.Graphics2D;
6 import java.awt.Paint;
7 import java.awt.Shape;
8 import java.awt.geom.AffineTransform;
9 import java.awt.geom.Point2D;
10 import java.awt.geom.Rectangle2D;
11
12 import figures.enums.FigureType;
13 import figures.enums.LineType;
14 import utils.PaintFactory;
15 import utils.StrokeFactory;
16
17 /**
18  * Classe commune à toutes les sortes de figures
19  *
20  * @author davidroussel
21  */
22 public abstract class Figure
23 {
24     /**
25      * La forme à dessiner
26      */
27     protected Shape shape;
28
29     /**
30      * Couleur du bord de la figure
31      */
32     protected Paint edge;
33
34     /**
35      * Couleur du bord de la sélection
36      */
37     protected static final Paint selectedEdge =
38         PaintFactory.getPaint(Color.LIGHT_GRAY);
39
40     /**
41      * Couleur de remplissage de la figure
42      */
43     protected Paint fill;
44
45     /**
46      * Caractéristiques de la bordure des figure : épaisseur, forme des
47      * extrémités et [evt] forme des jointures
48      */
49     protected BasicStroke stroke;
50
51     /**
52      * Caractéristique de la bordure des figures sélectionnées
53      */
54     protected static final BasicStroke selectedStroke =
55         StrokeFactory.getStroke(LineType.DASHED, 2.0f);
56
57     /**
58      * La translation à appliquer à cet objet
59      * @note sert à déplacer cet objet. pour ce faire il faut
60      * avant de dessiner cet objet appliquer cette transformation ainsi que
61      * sa rotation et son facteur d'échelle puis les retirer après le dessin.
62      */
63     protected AffineTransform translation;
64
65     /**
66      * La rotation à appliquer à cet objet
67      * @note sert à tourner cet objet. pour ce faire il faut
68      * avant de dessiner cet objet appliquer cette transformation ainsi que
69      * sa translation et son facteur d'échelle puis les retirer après le dessin.
70      */
71     protected AffineTransform rotation;
72
73     /**
74      * Le facteur d'échelle à appliquer à cet objet
75      * @note sert à changer l'échelle cet objet. pour ce faire il faut
76      * avant de dessiner cet objet appliquer cette transformation ainsi que
77      * sa translation et sa rotation puis les retirer après le dessin.
78      */
79     protected AffineTransform scale;
80
81     /**
82      * Le numéro d'instance de cette figure
83      * 1 si c'est la première figure de ce type, etc.
84      */
85     protected int instanceNumber;
86
87     /**
88      * Indique si la figure fait partie des figures sélectionnées
89      */
90     protected boolean selected;

```

avr 02, 17 16:17

## Figure.java

Page 2/6

```

91
92 /**
93  * Constructeur d'une figure abstraite à partir d'un style de ligne d'une
94  * couleur de bordure et d'une couleur de remplissage. Les styles de lignes
95  * et les couleurs étant souvent les mêmes entre les différentes figures ils
96  * devront être fournis par un flyweight. Le stroke, le edge et le fill
97  * peuvent chacun être null.
98  *
99  * @param stroke caractéristiques de la ligne de bordure
100  * @param edge couleur de la ligne de bordure
101  * @param fill couleur (ou gradient de couleurs) de remplissage
102  */
103 protected Figure(BasicStroke stroke, Paint edge, Paint fill)
104 {
105     this.stroke = stroke;
106     this.edge = edge;
107     this.fill = fill;
108     shape = null;
109     translation = new AffineTransform();
110     translation.setToIdentity();
111     rotation = new AffineTransform();
112     rotation.setToIdentity();
113     scale = new AffineTransform();
114     scale.setToIdentity();
115     selected = false;
116 }
117
118 /**
119  * Constructeur de copie assurant une copie distincte de la figure
120  * @param f la figure à copier
121  */
122 protected Figure(Figure f)
123 {
124     shape = null; // Shapes must be copied in subclasses
125     edge = PaintFactory.getPaint(f.edge);
126     fill = PaintFactory.getPaint(f.fill);
127     stroke = StrokeFactory.getStroke(f.stroke);
128     translation = new AffineTransform(f.translation);
129     rotation = new AffineTransform(f.rotation);
130     scale = new AffineTransform(f.scale);
131     instanceNumber = f.instanceNumber;
132     selected = f.selected;
133 }
134
135 /**
136  * Création d'une copie distincte de la figure
137  */
138 @Override
139 public abstract Figure clone();
140
141 /**
142  * Déplacement du dernier point de la figure (utilisé lors du dessin d'une
143  * figure tant que l'on déplace le dernier point)
144  *
145  * @param p la nouvelle position du dernier point
146  */
147 public abstract void setLastPoint(Point2D p);
148
149 /**
150  * Dessin de la figure dans un contexte graphique fourni par le système.
151  * Met en place le stroke et les couleurs, puis dessine la forme géométrique
152  * correspondant à la figure (figure remplie d'abord si le fill est non
153  * null, puis bordure si le edge est non null)
154  *
155  * @param g2D le contexte graphique
156  */
157 public final void draw(Graphics2D g2D)
158 {
159     // Get the current transform
160     AffineTransform savedT = g2D.getTransform();
161
162     // Perform transformations
163     g2D.transform(getTransform());
164
165     // Render
166     if (fill != null)
167     {
168         g2D.setPaint(fill);
169         g2D.fill(shape);
170     }
171     if ((edge != null) ^ (stroke != null))
172     {
173         g2D.setStroke(stroke);
174         g2D.setPaint(edge);
175         g2D.draw(shape);
176     }
177
178     // Restore original transform
179     g2D.setTransform(savedT);
180 }

```

avr 02, 17 16:17

## Figure.java

Page 3/6

```

181
182 /**
183  * Dessin de la sélection de la figure (son soulignement) dans un contexte
184  * graphique fourni par la système.
185  * @note le dessin de la sélection doit être séparé du dessin des figures
186  * de manière à ce que les sélection apparaissent par dessus les figures
187  * @param g2D le contexte graphique
188  */
189 public final void drawSelection(Graphics2D g2D)
190 {
191     if (selected)
192     {
193         g2D.setPaint(selectedEdge);
194         g2D.setStroke(selectedStroke);
195         g2D.draw(getBounds2D()); // getBounds uses current transform
196     }
197 }
198
199 /**
200  * Normalise une figure de manière à exprimer tous ses points par rapport
201  * à son centre puis transfère la position réelle du centre dans l'attribut
202  * {@link #translation}
203  */
204 public abstract void normalize();
205
206 /**
207  * Accesseur en lecture de la translation courante
208  * @return la translation courante
209  */
210 public AffineTransform getTranslation()
211 {
212     return translation;
213 }
214
215 /**
216  * Accesseur en lecture de la rotation courante
217  * @return la rotation courante
218  */
219 public AffineTransform getRotation()
220 {
221     return rotation;
222 }
223
224 /**
225  * Accesseur en lecture de l'échelle courante
226  * @return l'échelle courante
227  */
228 public AffineTransform getScale()
229 {
230     return scale;
231 }
232
233 /**
234  * Produit la transformation complète de cet objet
235  * (facteur d'échelle)*(rotation)*(translation)
236  * @return la transformation combinant le facteur d'échelle, la rotation et
237  * la translation de cette figure.
238  */
239 public AffineTransform getTransform()
240 {
241     AffineTransform transform = (AffineTransform) translation.clone();
242     transform.concatenate(scale);
243     transform.concatenate(rotation);
244
245     return transform;
246 }
247
248 /**
249  * Mise en place d'une translation
250  * @param translation la translation à mettre en place
251  */
252 public void setTranslation(AffineTransform translation)
253 {
254     this.translation = translation;
255 }
256
257 /**
258  * Déplace la figure de dx et dy
259  * @param dx la variation d'abscisse de la figure
260  * @param dy la variation d'ordonnée de la figure
261  */
262 public void translate(double dx, double dy)
263 {
264     translation.translate(dx, dy);
265 }
266
267 /**
268  * Mise en place d'une rotation
269  * @param rotation la rotation à mettre en place
270  */

```

avr 02, 17 16:17

## Figure.java

Page 4/6

```

271 public void setRotation(AffineTransform rotation)
272 {
273     this.rotation = rotation;
274 }
275
276 /**
277  * Fait tourner la figure d'un certain angle autour de son barycentre
278  * @param deltaAngle l'angle de rotation de la figure
279  */
280 public void rotate(double deltaAngle)
281 {
282     rotation.rotate(deltaAngle);
283 }
284
285 /**
286  * Mise en place d'un facteur d'échelle
287  * @param scale le facteur d'échelle à mettre en place
288  */
289 public void setScale(AffineTransform scale)
290 {
291     this.scale = scale;
292 }
293
294 /**
295  * Change l'échelle de la figure
296  * @param deltaScale le facteur d'échelle à appliquer à la figure
297  */
298 public void scale(double deltaScale)
299 {
300     scale.scale(deltaScale, deltaScale);
301 }
302
303 /**
304  * Obtention du rectangle englobant de la figure.
305  * Obtenu grâce au {@link Shape#getBounds2D()}
306  * @return le rectangle englobant de la figure
307  */
308 public Rectangle2D getBounds2D()
309 {
310     /*
311      * Attention, il faut appliquer la transformation affine courante
312      * au Rectangle2D résultant de l'appel à shape.getBounds2D();
313      */
314     Rectangle2D bounds = shape.getBounds2D();
315     double minX = bounds.getMinX();
316     double minY = bounds.getMinY();
317     double maxX = bounds.getMaxX();
318     double maxY = bounds.getMaxY();
319     Point2D[] corners = new Point2D[] {
320         new Point2D.Double(minX, minY),
321         new Point2D.Double(maxX, minY),
322         new Point2D.Double(maxX, maxY),
323         new Point2D.Double(minX, maxY)
324     };
325     Point2D[] tCorners = new Point2D[4];
326     for (int i = 0; i < 4; i++)
327     {
328         tCorners[i] = new Point2D.Double();
329     }
330
331     getTransform().transform(corners, 0, tCorners, 0, corners.length);
332
333     double x = tCorners[0].getX();
334     double y = tCorners[0].getY();
335     double w = 0.0;
336     double h = 0.0;
337
338     for (int i = 0; i < 4; i++)
339     {
340         double tx = tCorners[i].getX();
341         x = (x < tx ? x : tx);
342
343         double ty = tCorners[i].getY();
344         y = (y < ty ? y : ty);
345     }
346
347     for (int i = 0; i < 4; i++)
348     {
349         double tw = tCorners[i].getX() - x;
350         w = (tw > w ? tw : w);
351
352         double th = tCorners[i].getY() - y;
353         h = (th > h ? th : h);
354     }
355
356     bounds.setFrame(x, y, w, h);
357
358     return bounds;
359 }
360

```

avr 02, 17 16:17

## Figure.java

Page 5/6

```

361 /**
362  * Obtention du barycentre de la figure.
363  * @return le point correspondant au barycentre de la figure
364  */
365 public abstract Point2D getCenter();
366
367 /**
368  * Teste si le point p est contenu dans cette figure.
369  * Utilise {@link Shape#contains(Point2D)}
370  * @param p le point dont on veut tester s'il est contenu dans la figure
371  * @return true si le point p est contenu dans la figure, false sinon
372  */
373 public boolean contains(Point2D p)
374 {
375     /*
376      * TODO Ce point p doit subir la transformation inverse
377      * de celle subie par la figure pour déterminer si le point p fait
378      * partie de la figure
379      */
380     return false;
381 }
382
383 /**
384  * Accesseur du type de figure selon {@link FigureType}
385  * @return le type de figure
386  */
387 public abstract FigureType getType();
388
389 /**
390  * Accesseur en lecture du {@link Paint} du contour
391  * @return le {@link Paint} du contour
392  */
393 public Paint getEdgePaint()
394 {
395     return edge;
396 }
397
398 /**
399  * Mutateur du {@link Paint} du contour
400  * @param edge le nouveau {@link Paint} à mettre dans {@link #edge}
401  */
402 public void setEdgePaint(Paint edge)
403 {
404     if (edge != null)
405     {
406         this.edge = edge;
407     }
408     else
409     {
410         System.err.println(getClass().getSimpleName() + "::setEdgePaint : null paint");
411     }
412 }
413
414 /**
415  * Accesseur en lecture du {@link Paint} du remplissage
416  * @return le {@link Paint} du remplissage
417  */
418 public Paint getFillPaint()
419 {
420     return fill;
421 }
422
423 /**
424  * Mutateur du {@link Paint} du contour
425  * @param edge le nouveau {@link Paint} à mettre dans {@link #fill}
426  */
427 public void setFillPaint(Paint fill)
428 {
429     if (fill != null)
430     {
431         this.fill = fill;
432     }
433     else
434     {
435         System.err.println(getClass().getSimpleName() + "::setFillPaint : null paint");
436     }
437 }
438
439 /**
440  * Accesseur en lecture du {@link BasicStroke} du contour
441  * @return le {@link BasicStroke} du contour
442  */
443 public BasicStroke getStroke()
444 {
445     return stroke;
446 }
447
448 /**
449  * Mutateur du {@link BasicStroke} du contour
450  * @param edge le nouveau {@link BasicStroke} à mettre dans {@link #stroke}

```

avr 02, 17 16:17

## Figure.java

Page 6/6

```

451 */
452 public void setStroke(BasicStroke stroke)
453 {
454     if (stroke != null)
455     {
456         this.stroke = stroke;
457     }
458     else
459     {
460         System.err.println(getClass().getSimpleName() + "::setStroke : null stroke");
461     }
462 }
463
464 /**
465  * Accesseur de la propriété {@link #selected}
466  * @return la valeur de {@link #selected}
467  */
468 public boolean isSelected()
469 {
470     return selected;
471 }
472
473 /**
474  * Mutateur de la propriété {@link #selected}
475  * @param selected la nouvelle valeur de selected
476  */
477 public void setSelected(boolean selected)
478 {
479     this.selected = selected;
480 }
481
482 @Override
483 public String toString()
484 {
485     return getClass().getSimpleName() + " " + String.valueOf(instanceNumber);
486 }
487 }

```



avr 02, 17 16:17

## Rectangle.java

Page 1/2

```

1 package figures;
2
3 import java.awt.BasicStroke;
4 import java.awt.Paint;
5 import java.awt.geom.Point2D;
6 import java.awt.geom.Rectangle2D;
7 import java.awt.geom.RectangularShape;
8
9 import figures.enums.FigureType;
10
11 /**
12  * Classe de Rectangle pour les {@link Figure}
13  *
14  * @author davidroussel
15  */
16 public class Rectangle extends Figure
17 {
18     /**
19      * Le compteur d'instance des cercles.
20      * Utilisé pour donner un numéro d'instance après l'avoir incrémenté
21      */
22     private static int counter = 0;
23
24     /**
25      * Création d'un rectangle avec les points en haut à gauche et en bas à
26      * droite
27      *
28      * @param stroke la type de trait
29      * @param edge la couleur du trait
30      * @param fill la couleur de remplissage
31      * @param topLeft le point en haut à gauche
32      * @param bottomRight le point en bas à droite
33      */
34     public Rectangle(BasicStroke stroke, Paint edge, Paint fill, Point2D topLeft,
35                     Point2D bottomRight)
36     {
37         super(stroke, edge, fill);
38         instanceNumber = ++counter;
39         double x = topLeft.getX();
40         double y = topLeft.getY();
41         double w = (bottomRight.getX() - x);
42         double h = (bottomRight.getY() - y);
43
44         shape = new Rectangle2D.Double(x, y, w, h);
45
46         // System.out.println("Rectangle created");
47     }
48
49     /**
50      * Constructeur de copie assurant une copie distincte du rectangle
51      * @param rect le rectangle à copier
52      */
53     public Rectangle(Rectangle rect)
54     {
55         super(rect);
56         Rectangle2D oldRectangle = (Rectangle2D) rect.shape;
57         shape = new Rectangle2D.Double(oldRectangle.getMinX(),
58                                       oldRectangle.getMinY(),
59                                       oldRectangle.getWidth(),
60                                       oldRectangle.getHeight());
61     }
62
63     /**
64      * Création d'une copie distincte de la figure
65      * @see figures.Figure#clone()
66      */
67     @Override
68     public Figure clone()
69     {
70         return new Rectangle(this);
71     }
72
73     /**
74      * Création d'un rectangle sans points (utilisé dans les classes filles
75      * pour initialiser seulement les couleur et le style de trait sans
76      * initialiser {@link #shape}).
77      *
78      * @param stroke la type de trait
79      * @param edge la couleur du trait
80      * @param fill la couleur de remplissage
81      */
82     protected Rectangle(BasicStroke stroke, Paint edge, Paint fill)
83     {
84         super(stroke, edge, fill);
85
86         shape = null;
87     }
88
89     /**
90      * Déplacement du point en bas à droite du rectangle à la position

```

avr 02, 17 16:17

## Rectangle.java

Page 2/2

```

91 * du point p
92 *
93 * @param p la nouvelle position du dernier point
94 * @see figures.Figure#setLastPoint(Point2D)
95 */
96 @Override
97 public void setLastPoint(Point2D p)
98 {
99     if (shape != null)
100     {
101         Rectangle2D.Double rect = (Rectangle2D.Double) shape;
102         double newWidth = p.getX() - rect.x;
103         double newHeight = p.getY() - rect.y;
104         rect.width = newWidth;
105         rect.height = newHeight;
106     }
107     else
108     {
109         System.err.println(getClass().getSimpleName() + "::setLastPoint: null shape");
110     }
111 }
112
113 /**
114 * Obtention du barycentre de la figure.
115 * @return le point correspondant au barycentre de la figure
116 */
117 @Override
118 public Point2D getCenter()
119 {
120     RectangularShape rect = (RectangularShape) shape;
121
122     Point2D center = new Point2D.Double(rect.getCenterX(), rect.getCenterY());
123     Point2D tCenter = new Point2D.Double();
124     getTransform().transform(center, tCenter);
125
126     return tCenter;
127 }
128
129 /**
130 * Normalise une figure de manière à exprimer tous ses points par rapport
131 * à son centre. puis transfère la position réelle du centre dans l'attribut
132 * {@link #translation}
133 */
134 @Override
135 public void normalize()
136 {
137     Point2D center = getCenter();
138     double cx = center.getX();
139     double cy = center.getY();
140     RectangularShape rectangle = (RectangularShape) shape;
141     translation.translate(cx, cy);
142     rectangle.setFrame(rectangle.getX() - cx,
143                       rectangle.getY() - cy,
144                       rectangle.getWidth(),
145                       rectangle.getHeight());
146 }
147
148 /**
149 * Accesseur du type de figure selon {@link FigureType}
150 * @return le type de figure
151 */
152 @Override
153 public FigureType getType()
154 {
155     return FigureType.RECTANGLE;
156 }
157 }

```

avr 02, 17 16:17

## Drawing.java

Page 1/8

```

1 package figures;
2
3 import java.awt.BasicStroke;
4 import java.awt.Paint;
5 import java.awt.geom.Point2D;
6 import java.util.Observable;
7 import java.util.Observer;
8 import java.util.SortedSet;
9 import java.util.TreeSet;
10 import java.util.Vector;
11 import java.util.stream.Stream;
12
13 import figures.enums.FigureType;
14 import figures.enums.LineType;
15 import filters.FigureFilters;
16 import utils.StrokeFactory;
17
18 /**
19  * Classe contenant l'ensemble des figures à dessiner (LE MODELE)
20  * @author davidroussel
21  */
22 public class Drawing extends Observable
23 {
24     /**
25      * Liste des figures à dessiner (protected pour que les classes du même
26      * package puissent y accéder)
27      */
28     protected Vector<Figure> figures;
29
30     /**
31      * Liste triée des indices (uniques) des figures sélectionnées
32      */
33     protected SortedSet<Integer> selectionIndex;
34
35     /**
36      * Figure située sous le curseur.
37      * Déterminé par {@link #getFigureAt(Point2D)}
38      */
39     private Figure selectedFigure;
40
41     /**
42      * Le type de figure à créer (pour la prochaine figure)
43      */
44     private FigureType type;
45
46     /**
47      * La couleur de remplissage courante (pour la prochaine figure)
48      */
49     private Paint fillPaint;
50
51     /**
52      * La couleur de trait courante (pour la prochaine figure)
53      */
54     private Paint edgePaint;
55
56     /**
57      * La largeur de trait courante (pour la prochaine figure)
58      */
59     private float edgeWidth;
60
61     /**
62      * Le type de trait courant (sans trait, trait plein, trait pointillé,
63      * pour la prochaine figure)
64      */
65     private LineType edgeType;
66
67     /**
68      * Les caractéristiques à appliquer au trait en fonction de {@link #type} et
69      * {@link #edgeWidth}
70      */
71     private BasicStroke stroke;
72
73     /**
74      * Etat de filtrage des figures dans le flux de figures fournit par
75      * {@link #stream()}
76      * Lorsque {@link #filtering} est true le dessin des figures est filtré
77      * par l'ensemble des filtres présents dans {@link #shapeFilters},
78      * {@link #fillColorFilter}, {@link #edgeColorFilter} et
79      * {@link #lineFilters}.
80      * Lorsque {@link #filtering} est false, toutes les figures sont fournies
81      * dans le flux des figures.
82      * @see #stream()
83      */
84     private boolean filtering;
85
86     /**
87      * Filtres à appliquer au flux des figures pour sélectionner les types
88      * de figures à afficher
89      * @see #stream()
90      */

```

avr 02, 17 16:17

## Drawing.java

Page 2/8

```

91 private FigureFilters<FigureType> shapeFilters;
92
93 /**
94  * Filtre à appliquer au flux des figures pour sélectionner les figures
95  * ayant une couleur particulière de remplissage
96  */
97 // private FillColorFilter fillColorFilter; // TODO décommenter lorsque prêt
98
99 /**
100  * Filtre à appliquer au flux des figures pour sélectionner les figures
101  * ayant une couleur particulière de trait
102  */
103 // private EdgeColorFilter edgeColorFilter; // TODO décommenter lorsque prêt
104
105 /**
106  * Filtres à appliquer au flux des figures pour sélectionner les figures
107  * ayant un type particulier de lignes
108  */
109 private FigureFilters<LineType> lineFilters;
110
111 /**
112  * Le status du modèle après une opération (et
113  * deuxièmement avant un {@link #update()}).
114  * Prends ses valeurs en tant que combinaison (bitwise OR)
115  * des valeurs de {@link Status}
116  */
117 private int status;
118
119 /**
120  * Constructeur de modèle de dessin
121  */
122 public Drawing()
123 {
124     figures = new Vector<Figure>();
125     selectionIndex = new TreeSet<Integer>(Integer::compare);
126     shapeFilters = new FigureFilters<FigureType>();
127
128     // fillColorFilter = null; // TODO décommenter lorsque prêt
129     // edgeColorFilter = null; // TODO décommenter lorsque prêt
130     lineFilters = new FigureFilters<LineType>();
131
132     fillPaint = null;
133     edgePaint = null;
134     edgeWidth = 1.0f;
135     edgeType = LineType.SOLID;
136     stroke = StrokeFactory.getStroke(edgeType, edgeWidth);
137     filtering = false;
138     selectedFigure = null;
139     status = Status.NORMAL.value;
140     System.out.println("Drawing model created");
141 }
142
143 /**
144  * Nettoyage avant destruction
145  */
146 @Override
147 protected void finalize()
148 {
149     // Aide au GC
150     figures.clear();
151     figures = null;
152     selectionIndex.clear();
153     selectionIndex = null;
154     fillPaint = null;
155     edgePaint = null;
156     edgeType = null;
157     stroke = null;
158     shapeFilters.clear();
159     shapeFilters = null;
160     // fillColorFilter = null; // TODO décommenter lorsque prêt
161     // edgeColorFilter = null; // TODO décommenter lorsque prêt
162     lineFilters.clear();
163     lineFilters = null;
164 }
165
166 /**
167  * Mise à jour du ou des {@link Observer} qui observent ce modèle. On place
168  * le modèle dans un état "changé" puis on notifie les observateurs.
169  */
170 public void update()
171 {
172     setChanged();
173     notifyObservers(status); // pour que les observateurs soient mis à jour
174     status = Status.NORMAL.value;
175 }
176
177 /**
178  * Création d'un état courant du modèle de dessin
179  * @return un état courant contenant une copie de l'état du modèle
180  */

```

avr 02, 17 16:17

## Drawing.java

Page 3/8

```

181 public State createState()
182 {
183     // TODO Remplacer par l'implémentation ...
184     return null;
185 }
186
187 /**
188  * Mise en place d'un état particulier remplaçant l'état courant du modèle
189  * de dessin
190  * @param state l'état à mettre en place
191  * @return true si l'état courant a été correctement remplacé, false sinon
192  */
193 public boolean setState(State state)
194 {
195     // TODO Remplacer par l'implémentation ...
196     return false;
197 }
198
199 /**
200  * Mise en place d'un nouveau type de figure à générer
201  * @param type le nouveau type de figure
202  */
203 public void setFigureType(FigureType type)
204 {
205     this.type = type;
206 }
207
208 /**
209  * Accesseur de la couleur de remplissage courante des figures
210  * @return la couleur de remplissage courante des figures
211  */
212 public Paint getFillPaint()
213 {
214     return fillPaint;
215 }
216
217 /**
218  * Mise en place d'une nouvelle couleur de remplissage
219  * @param fillPaint la nouvelle couleur de remplissage
220  */
221 public void setFillPaint(Paint fillPaint)
222 {
223     this.fillPaint = fillPaint;
224 }
225
226 /**
227  * Accesseur de la couleur de trait courante des figures
228  * @return la couleur de remplissage courante des figures
229  */
230 public Paint getEdgePaint()
231 {
232     return edgePaint;
233 }
234
235 /**
236  * Mise en place d'une nouvelle couleur de trait
237  * @param edgePaint la nouvelle couleur de trait
238  */
239 public void setEdgePaint(Paint edgePaint)
240 {
241     this.edgePaint = edgePaint;
242 }
243
244 /**
245  * Accesseur du trait courant des figures
246  * @return le trait courant des figures
247  */
248 public BasicStroke getStroke()
249 {
250     return stroke;
251 }
252
253 /**
254  * Mise en place d'un nouvelle épaisseur de trait
255  * @param width la nouvelle épaisseur de trait
256  */
257 public void setEdgeWidth(float width)
258 {
259     edgeWidth = width;
260     /*
261     * TODO Il faut régénérer le stroke
262     */
263 }
264
265 /**
266  * Mise en place d'un nouvel état de ligne pointillée
267  * @param type le nouveau type de ligne
268  */
269 public void setEdgeType(LineType type)
270 {

```

avr 02, 17 16:17

## Drawing.java

Page 4/8

```

271     edgeType = type;
272     /*
273     * TODO Il faut régénérer le stroke
274     */
275 }
276
277 /**
278  * Accesseur en lecture du {@link #status}
279  * @return le status courant
280  */
281 public int getStatus()
282 {
283     return status;
284 }
285
286 /**
287  * Vérifie si un status particulier fait partie
288  * du status courant
289  * @param status le status recherché
290  * @return true si le status recherché fait partie
291  * du status courant.
292  */
293 public boolean hasStatus(Status status)
294 {
295     return (this.status & status.value) != 0;
296 }
297
298 /**
299  * Mutateur du {@link #status}
300  * @param status la nouvelle valeur du {@link #status}
301  */
302 public void setStatus(Status status)
303 {
304     this.status = status.value;
305 }
306
307 /**
308  * Ajout d'un bit de flag de status
309  * @param status le status à combiner au status courant
310  */
311 public void addStatus(Status status)
312 {
313     this.status = (this.status | status.value);
314 }
315
316 /**
317  * Initialisation d'une figure de type {@link #type} au point p et ajout de
318  * cette figure à la liste des {@link #figures}
319  * @param p le point où initialiser la figure
320  * @return la nouvelle figure créée à x et y avec les paramètres courants
321  */
322 public Figure initiateFigure(Point2D p)
323 {
324     /*
325     * TODO Maintenant que l'on s'apprête effectivement à créer une figure on
326     * ajoute les Paints et le Stroke aux factories
327     */
328
329     /*
330     * TODO Obtention de la figure correspondant au type de figure choisi grâce à
331     * type.getFigure(...)
332     */
333     Figure newFigure = null; // TODO remplacer par type.getType(...)
334
335     /*
336     * TODO Ajout de la figure à #figures
337     */
338
339     /* TODO Notification des observers */
340
341     return newFigure;
342 }
343
344 /**
345  * Obtention de la dernière figure (implicitement celle qui est en cours de
346  * dessin)
347  * @return la dernière figure du dessin
348  */
349 public Figure getLastFigure()
350 {
351     // TODO Remplacer par l'implémentation ...
352     return null;
353 }
354
355 /**
356  * Obtention de la dernière figure contenant le point p.
357  * @param p le point sous lequel on cherche une figure
358  * @return une référence vers la dernière figure contenant le point p ou à
359  * défaut null.
360  */

```

avr 02, 17 16:17

## Drawing.java

Page 5/8

```

361 public Figure getFigureAt(Point2D p)
362 {
363     selectedFigure = null;
364
365     /*
366      * TODO Recherche dans le flux des figures de la DERNIERE figure
367      * contenant le point p.
368      */
369     return selectedFigure;
370 }
371
372 /**
373  * Retrait de la dernière figure (sera déclenché par une action undo)
374  * @post le modèle de dessin a été mis à jour
375  */
376 public void removeLastFigure()
377 {
378     // TODO Compléter ...
379 }
380
381 /**
382  * Effacement de toutes les figures (sera déclenché par une action clear)
383  * @post le modèle de dessin a été mis à jour
384  */
385 public void clear()
386 {
387     // TODO Compléter ...
388 }
389
390 /**
391  * Accesseur de l'état de filtrage
392  * @return l'état courant de filtrage
393  */
394 public boolean getFiltering()
395 {
396     return filtering;
397 }
398
399 /**
400  * Changement d'état du filtrage
401  * @param filtering le nouveau statut de filtrage
402  * @post le modèle de dessin a été mis à jour
403  */
404 public void setFiltering(boolean filtering)
405 {
406     // TODO ... filtering ...
407 }
408
409 /**
410  * Ajout d'un filtre pour filtrer les types de figures
411  * @param filter le filtre à ajouter
412  * @return true si le filtre n'était pas déjà présent dans l'ensemble des
413  *         filtres filtrant les types de figures. false sinon
414  * @post si le filtre a été ajouté, une mise à jour est déclenchée
415  */
416 // TODO décommenter lorsque prêt
417 public boolean addShapeFilter(ShapeFilter filter)
418 {
419     // TODO ... shapeFilters ...
420     return false;
421 }
422
423 /**
424  * Retrait d'un filtre filtrant les types de figures
425  * @param filter le filtre à retirer
426  * @return true si le filtre faisait partie des filtres filtrant les types
427  *         de figure et a été retiré. false sinon
428  * @post si le filtre a été retiré, une mise à jour est déclenchée
429  */
430 // TODO décommenter lorsque prêt
431 public boolean removeShapeFilter(ShapeFilter filter)
432 {
433     // TODO ... shapeFilters ...
434     return false;
435 }
436
437 /**
438  * Mise en place du filtre de couleur de remplissage
439  * @param filter le filtre de couleur de remplissage à appliquer
440  * @post le {@link #fillColorFilter} est mis en place et une mise à jour
441  *         est déclenchée
442  */
443 // TODO décommenter lorsque prêt
444 public void setFillColorFilter(FillColorFilter filter)
445 {
446     // TODO ... fillColorFilter ...
447 }
448
449 /**
450

```

avr 02, 17 16:17

## Drawing.java

Page 6/8

```

451 * Mise en place du filtre de couleur de trait
452 * @param filter le filtre de couleur de trait à appliquer
453 * @post le #edgeColorFilter est mis en place et une mise à jour
454 *         est déclenchée
455 */
456 // TODO décommenter lorsque prêt
457 public void setEdgeColorFilter(EdgeColorFilter filter)
458 {
459     // TODO ... edgeColorFilter ...
460 }
461
462 /**
463  * Ajout d'un filtre pour filtrer les types de ligne des figures
464  * @param filter le filtre à ajouter
465  * @return true si le filtre n'était pas déjà présent dans l'ensemble des
466  *         filtres filtrant les types de lignes. false sinon
467  * @post si le filtre a été ajouté, une mise à jour est déclenchée
468  */
469 // TODO décommenter lorsque prêt
470 public boolean addLineFilter(LineFilter filter)
471 {
472     // TODO ... lineFilters ...
473     return false;
474 }
475
476 /**
477  * Retrait d'un filtre filtrant les types de lignes
478  * @param filter le filtre à retirer
479  * @return true si le filtre faisait partie des filtres filtrant les types
480  *         de lignes et a été retiré. false sinon
481  * @post si le filtre a été retiré, une mise à jour est déclenchée
482  */
483 // TODO décommenter lorsque prêt
484 public boolean removeLineFilter(LineFilter filter)
485 {
486     // TODO ... lineFilters ...
487     return false;
488 }
489
490 /**
491  * Remise à l'état non sélectionné de toutes les figures
492  */
493 public void clearSelection()
494 {
495     // TODO Compléter ...
496 }
497
498 /**
499  * Mise à jour des indices des figures sélectionnées dans {@link #selectionIndex}
500  * d'après l'interrogation de l'ensemble des figures (après filtrage).
501  * @note {@link #status} doit être modifié avant l'appel d'
502  *        {@link #updateSelection()}
503  */
504 public void updateSelection()
505 {
506     // TODO Compléter ...
507 }
508
509 /**
510  * Indique s'il existe des figures sélectionnées
511  * @return true s'il y a des figures sélectionnées
512  */
513 public boolean hasSelection()
514 {
515     // TODO Remplacer par l'implémentation
516     return false;
517 }
518
519 /**
520  * Destruction des figures sélectionnées.
521  * Et incidemment nettoyage de {@link #selectionIndex}
522  */
523 public void deleteSelected()
524 {
525     // TODO Compléter ...
526 }
527
528 /**
529  * Applique un style particulier aux figures sélectionnées
530  * @param fill la couleur de remplissage à appliquer aux figures sélectionnées
531  * @param edge la couleur de trait à appliquer aux figures sélectionnées
532  * @param stroke le style de trait à appliquer aux figures sélectionnées
533  */
534 public void applyStyleToSelected(Paint fill, Paint edge, BasicStroke stroke)
535 {
536     // TODO Compléter ...
537 }
538
539 /**
540  * Déplacement des figures sélectionnées en haut de la liste des figures.

```

avr 02, 17 16:17

## Drawing.java

Page 7/8

```

541  * En conservant l'ordre des figures sélectionnées
542  */
543  public void moveSelectedUp()
544  {
545      // TODO Compléter ...
546
547
548
549      // Mise à jour des index des figures sélectionnées & notif observers
550      status = Status.REORDERED.value;
551      updateSelection();
552  }
553
554  /**
555   * Accès aux figures dans un stream afin que l'on puisse y appliquer
556   * de filtres
557   * @return le flux des figures éventuellement filtrés par les différents
558   * filtres
559   */
560  public Stream<Figure> stream()
561  {
562      Stream<Figure> figuresStream = figures.stream();
563      if (filtering)
564      {
565          // TODO Compléter avec ...
566          // if (filters.size() > 0)
567          // {
568          //     figuresStream = figuresStream.filter(filters);
569          // }
570
571
572      return figuresStream;
573  }
574
575  /**
576   * Enum contenant différents flags indiquant l'état dans lequel se trouve
577   * le modèle de dessin après une opération.
578   * Par exemple, une figure peut avoir été ajoutée, ou enlevée, les figures
579   * réordonnées etc, etc.
580   */
581  public enum Status
582  {
583      /**
584       * Aucun status particulier
585       */
586      NORMAL(0),
587      /**
588       * Une ou plusieurs figures ont été ajoutées
589       */
590      ADDED(1),
591      /**
592       * Une ou plusieurs figures ont été enlevées
593       */
594      REMOVED(2),
595      /**
596       * Les figures ont été réordonnées
597       */
598      REORDERED(4);
599
600      /**
601       * Valeur interne (puissance de 2 pour
602       * pouvoir l'utiliser dans un bitwise OR)
603       */
604      public final int value;
605
606      /**
607       * Constructeur valué
608       * @param value la valeur de l'enum
609       */
610      Status(int value)
611      {
612          this.value = value;
613      }
614
615      /**
616       * Vérifie si une valeur d'enum particulière
617       * fait partie de flags
618       * @param flags une combinaison (bitwise OR)
619       * de valeurs d'enum
620       * @return
621       */
622      public boolean isIn(int flags)
623      {
624          return (value & flags) != 0;
625      }
626
627
628      /**
629       * Classe permettant de sauvegarder l'état courant du modèle de dessin.
630       * (Quitte à déplacer des attributs de {@link Drawing} dans cette classe

```

avr 02, 17 16:17

## Drawing.java

Page 8/8

```

631  * si besoin).
632  * @note On considérera que l'état courant concerne uniquement les
633  * figures présentes dans {@link Drawing#figures}
634  */
635  public class State
636  {
637      // TODO Compléter ...
638  }
639

```

avr 02, 17 16:17

## package-info.java

Page 1/1

```

1  /**
2   * Package contenant les différents widgets (éléments graphiques)
3   */
4   package widgets;

```

avr 02, 17 16:17

## FigureType.java

Page 1/2

```

1  package figures.enums;
2
3  import java.awt.BasicStroke;
4  import java.awt.Paint;
5  import java.awt.Point;
6  import java.awt.geom.Point2D;
7
8  import javax.swing.JLabel;
9
10 import figures.Drawing;
11 import figures.Figure;
12 import figures.Rectangle;
13 import figures.listeners.creation.AbstractCreationListener;
14 import figures.listeners.creation.RectangularShapeCreationListener;
15
16 /**
17  * Enumeration des différentes figures possibles
18  * @author davidroussel
19  */
20 public enum FigureType
21 {
22     /**
23      * Les différents types de figures
24      */
25     CIRCLE, ELLIPSE, RECTANGLE, ROUNDED_RECTANGLE, POLYGON, NGON, STAR;
26
27     /**
28      * Nombre de figures référencées ici (à changer si on ajoute des types de
29      * figures)
30      */
31     public final static int NbFigureTypes = 7;
32
33     /**
34      * Obtention d'une instance de figure correspondant au type
35      * @param stroke la césure du trait (ou pas de trait si null)
36      * @param edge la couleur du trait (ou pas de trait si null)
37      * @param fill la couleur de remplissage (ou pas de remplissage si null)
38      * @param x l'abscisse du premier point de la figure
39      * @param y l'ordonnée du premier point de la figure
40      * @return une nouvelle instance correspondant à la valeur de cet enum
41      * @throws AssertionError si la valeur de cet enum n'est pas prévue
42      */
43     public Figure getFigure(BasicStroke stroke,
44                             Paint edge,
45                             Paint fill,
46                             Point2D p)
47     {
48         throws AssertionError
49
50         switch (this)
51         {
52             case CIRCLE:
53                 return null; // TODO new Circle(stroke, edge, fill, p, 0.0f);
54             case ELLIPSE:
55                 return null; // new Ellipse(stroke, edge, fill, p, p);
56             case RECTANGLE:
57                 return new Rectangle(stroke, edge, fill, p, p);
58             case ROUNDED_RECTANGLE:
59                 return null; // TODO new RoundedRectangle(stroke, edge, fill, p, p, 0);
60             case POLYGON:
61                 Point pp = new Point(Double.valueOf(p.getX()).intValue(),
62                                     Double.valueOf(p.getY()).intValue());
63                 return null; // TODO new Polygon(stroke, edge, fill, pp, pp);
64             case NGON:
65                 return null; // TODO new NGon(stroke, edge, fill, p);
66             case STAR:
67                 return null; // TODO new Star(stroke, edge, fill, p);
68         }
69
70         throw new AssertionError("FigureType unknown assertion: " + this);
71     }
72
73     /**
74      * Obtention d'un CreationListener adequat pour la valeur de cet enum
75      * @param model le modèle de dessin à modifier
76      * @param tipLabel le label dans lequel afficher les conseils utilisateur
77      * @return une nouvelle instance de CreationListener adéquate pour le type
78      * de figure de cet enum
79      * @throws AssertionError si la valeur de cet enum n'est pas prévue
80      */
81     public AbstractCreationListener getCreationListener(Drawing model,
82                                                         JLabel tipLabel)
83     {
84         throws AssertionError
85
86         switch (this)
87         {
88             case CIRCLE:
89             case ELLIPSE:
90             case RECTANGLE:
91                 return new RectangularShapeCreationListener(model, tipLabel);
92             case ROUNDED_RECTANGLE:

```

avr 02, 17 16:17

## FigureType.java

Page 2/2

```

91     return null; // TODO new RoundedRectangleCreationListener(model, tipLabel);
92     case POLYGON:
93         return null; // TODO new PolygonCreationListener(model, tipLabel);
94     case NGON:
95         return null; // TODO new NGonCreationListener(model, tipLabel);
96     case STAR:
97         return null; // TODO StarCreationListener(model, tipLabel);
98     }
99
100     throw new AssertionError("FigureType unknown assertion: " + this);
101 }
102
103 /**
104  * Représentation sous forme de chaîne de caractères
105  * @return une chaîne de caractère représentant la valeur de cet enum
106  * @throws AssertionError si la valeur de cet enum n'est pas prévue
107  */
108 @Override
109 public String toString() throws AssertionError
110 {
111     switch (this)
112     {
113         case CIRCLE:
114             return new String("Circle");
115         case ELLIPSE:
116             return new String("Ellipse");
117         case RECTANGLE:
118             return new String("Rectangle");
119         case ROUNDED_RECTANGLE:
120             return new String("Rounded Rectangle");
121         case POLYGON:
122             return new String("Polygon");
123         case NGON:
124             return new String("Ngon");
125         case STAR:
126             return new String("Star");
127     }
128
129     throw new AssertionError("FigureType unknown assertion: " + this);
130 }
131
132 /**
133  * Obtention d'un tableau de chaîne de caractères contenant l'ensemble des
134  * nom des figures
135  * @return un tableau de chaîne de caractères contenant l'ensemble des nom
136  * des figures
137  */
138 public static String[] stringValues()
139 {
140     FigureType[] values = FigureType.values();
141     String[] stringValues = new String[values.length];
142
143     for (int i = 0; i < stringValues.length; i++)
144     {
145         stringValues[i] = values[i].toString();
146     }
147
148     return stringValues;
149 }
150
151 /**
152  * Conversion d'un entier en FigureType
153  * @param i l'entier à convertir en FigureType
154  * @return le FigureType correspondant à l'entier
155  */
156 public static FigureType fromInteger(int i)
157 {
158     switch (i)
159     {
160         case 0:
161             return CIRCLE;
162         case 1:
163             return ELLIPSE;
164         case 2:
165             return RECTANGLE;
166         case 3:
167             return ROUNDED_RECTANGLE;
168         case 4:
169             return POLYGON;
170         case 5:
171             return NGON;
172         case 6:
173             return STAR;
174         default:
175             return POLYGON;
176     }
177 }
178 }

```

avr 02, 17 16:17

## LineType.java

Page 1/2

```

1 package figures.enums;
2 import java.awt.BasicStroke;
3
4 /**
5  * Le type de trait des lignes (continu, pointillé, ou sans trait)
6  * @author davidroussel
7  */
8 public enum LineType
9 {
10     /**
11      * Pas de trait
12      */
13     NONE,
14     /**
15      * Trait plein
16      */
17     SOLID,
18     /**
19      * Trait pointillé
20      */
21     DASHED;
22
23     /**
24      * Le nombre de type de lignes (à changer si l'on rajoute un type de ligne)
25      */
26     public static final int NbLineTypes = 3;
27
28     /**
29      * Conversion d'un entier vers un {@link LineType}
30      * A utiliser pour convertir l'index de l'élément sélectionné d'un combobox
31      * dans le type de ligne correspondant
32      * @param i l'entier à convertir
33      * @return le LineType correspondant
34      */
35     public static LineType fromInteger(int i)
36     {
37         switch (i)
38         {
39             case 0:
40                 return NONE;
41             case 1:
42                 return SOLID;
43             case 2:
44                 return DASHED;
45             default:
46                 return NONE;
47         }
48     }
49
50     /**
51      * Conversion d'un {@link BasicStroke} en type de ligne
52      * @param stroke le stroke à examiner
53      * @return le type de ligne correspondant (NONE si le stroke est nul,
54      * SOLID si le stroke ne contient pas de dash array, DASHED si le stroke
55      * contient un dash array.
56      */
57     public static LineType fromStroke(BasicStroke stroke)
58     {
59         if (stroke == null)
60         {
61             return LineType.NONE;
62         }
63         else
64         {
65             float[] dashArray = stroke.getDashArray();
66             if (dashArray == null)
67             {
68                 return LineType.SOLID;
69             }
70             else
71             {
72                 return LineType.DASHED;
73             }
74         }
75     }
76
77     /**
78      * Représentation sous forme de chaîne de caractères
79      * @return une chaîne de caractères représentant la valeur de cet enum
80      */
81     @Override
82     public String toString() throws AssertionError
83     {
84         switch (this)
85         {
86             case NONE:
87                 return new String("None");
88             case SOLID:
89                 return new String("Solid");
90

```

avr 02, 17 16:17

## LineType.java

Page 2/2

```

91         case DASHED:
92             return new String("Dashed");
93     }
94
95     throw new AssertionError("LineType Unknown assertion " + this);
96 }
97
98 /**
99  * Obtention d'un tableau de string contenant tous les noms des types.
100  * A utiliser lors de la création d'un combobox avec :
101  * LineType.stringValues()
102  * @return un tableau de string contenant tous les noms des types
103  */
104 public static String[] stringValues()
105 {
106     LineType[] values = LineType.values();
107     String[] stringValues = new String[values.length];
108     for (int i = 0; i < values.length; i++)
109     {
110         stringValues[i] = values[i].toString();
111     }
112
113     return stringValues;
114 }
115 }

```

avr 02, 17 16:17

## PaintToType.java

Page 1/1

```

1 package figures.enums;
2 import java.awt.Paint;
3 import figures.Drawing;
4
5 /**
6  * Enumeration de ce à quoi s'applique une couleur ({@link Paint}) à utiliser
7  * dans le {@link widgets.EditorFrame.ColoItemListener}
8  */
9
10 * @author davidroussel
11 */
12 public enum PaintToType
13 {
14     /**
15      * La couleur s'applique au remplissage
16      */
17     FILL,
18     /**
19      * La couleur s'applique au trait
20      */
21     EDGE;
22
23     /**
24      * Application d'une couleur au modèle de dessin en fonction de la valeur de
25      * l'enum
26      *
27      * @param paint la couleur à appliquer
28      * @param drawing le modèle de dessin sur lequel appliquer la couleur
29      * @throws AssertionError si le type de l'enum est inconnu
30      */
31     public void applyPaintTo(Paint paint, Drawing drawing)
32         throws AssertionError
33     {
34         switch (this)
35         {
36             case FILL:
37                 drawing.setFillPaint(paint);
38                 break;
39             case EDGE:
40                 drawing.setEdgePaint(paint);
41                 break;
42             default:
43                 throw new AssertionError(
44                     "PaintApplicationType unknown assertion " + this);
45         }
46     }
47
48     /**
49      * Représentation sous forme de chaîne de caractères
50      *
51      * @return une chaîne de caractères représentant la valeur de cet enum
52      */
53     @Override
54     public String toString() throws AssertionError
55     {
56         switch (this)
57         {
58             case FILL:
59                 return new String("Fill");
60             case EDGE:
61                 return new String("Edge");
62         }
63
64         throw new AssertionError("PaintApplicationType Unknown assertion "
65             + this);
66     }
67 }
68
69 }

```



avr 02, 17 16:17

## AbstractFigureListener.java

Page 1/3

```

1 package figures.listeners;
2
3 import java.awt.event.MouseEvent;
4 import java.awt.event.MouseListener;
5 import java.awt.event.MouseMotionListener;
6 import java.awt.event.MouseWheelListener;
7 import java.awt.geom.Point2D;
8
9 import javax.swing.JLabel;
10
11 import figures.Drawing;
12 import figures.Figure;
13
14 /**
15  * Listener (incomplet) des événements souris pour agir sur les figures.
16  * Chaque action sur les figures (création ou transformation) est graphiquement
17  * construite par une suite de pressed/drag/release ou de clicks qui peut être
18  * différente pour chaque type d'action. Aussi les classes filles devront
19  * implémenter leur propre xxxFigureListener assurant la gestion des événements
20  * souris.
21  * @author davidroussel
22  */
23 public abstract class AbstractFigureListener
24     implements MouseListener, MouseMotionListener, MouseWheelListener
25 {
26     /**
27      * Le drawing model à modifier par ce creationListener. Celui ci contient
28      * tous les éléments nécessaires à la modification du dessin par les
29      * événements souris.
30      */
31     protected Drawing drawingModel;
32
33     /**
34      * La figure en cours de dessin. Obtenue avec
35      * {@link Drawing#initiateFigure(java.awt.geom.Point2D)}. Evite d'avoir à
36      * appeler {@link Drawing#getLastFigure()} à chaque fois que la figure en
37      * cours de construction est modifiée.
38      */
39     protected Figure currentFigure;
40
41     /**
42      * Le label dans lequel afficher les instructions nécessaires à la
43      * complétion de la figure
44      */
45     protected JLabel tipLabel;
46
47     /**
48      * Le point de départ de la création de la figure. Utilisé pour comparer le
49      * point de départ et le point terminal pour éliminer les figures de taille
50      * 0;
51      */
52     protected Point2D startPoint;
53
54     /**
55      * Le point terminal de la création de la figure. Utilisé pour comparer le
56      * point de départ et le point terminal pour éliminer les figures de taille
57      * 0;
58      */
59     protected Point2D endPoint;
60
61     /**
62      * le conseil par défaut à afficher dans le {@link #tipLabel}
63      */
64     public static final String defaultTip =
65         new String("Cliquez pour initier une figure");
66
67     /**
68      * Le tableau de chaines de caractères contenant les conseils à
69      * l'utilisateur pour chacune des étapes de la création. Par exemple [0] :
70      * cliquez et maintenez enfoncé pour initier la figure [1] : relâchez pour
71      * terminer la figure
72      */
73     protected String[] tips;
74
75     /**
76      * Le nombre d'étapes (typiquement click->drag->release) nécessaires à la
77      * création de la figure
78      */
79     protected final int nbSteps;
80
81     /**
82      * L'étape actuelle de création de la figure
83      */
84     protected int currentStep;
85
86     /**
87      * Constructeur protégé (destiné à être utilisé par les classes filles)
88      * @param model le modèle de dessin à modifier par ce listener
89      * @param infoLabel le label dans lequel afficher les conseils d'utilisation
90      * @param nbSteps le nombre d'étapes de l'action à réaliser

```

avr 02, 17 16:17

## AbstractFigureListener.java

Page 2/3

```

91 */
92 protected AbstractFigureListener(Drawing model,
93     JLabel infoLabel,
94     int nbSteps)
95 {
96     drawingModel = model;
97     currentFigure = null;
98     tipLabel = infoLabel;
99     this.nbSteps = nbSteps;
100    currentStep = 0;
101
102    // Allocation du nombres de conseils utilisateurs nécessaires
103    tips = new String[(nbSteps > 0 ? nbSteps : 0)];
104
105    if (drawingModel == null)
106    {
107        System.err.println("AbstractCreationListener caution null "
108            + "drawing model");
109    }
110
111    if (tipLabel == null)
112    {
113        System.err.println("AbstractCreationListener caution null "
114            + "tip label");
115    }
116
117    /**
118     * Initialisation de l'action :
119     * Détermine le point de départ ({@link #startPoint})
120     * Les classes filles devront réutiliser cette méthode pour récupérer le
121     * point de départ de l'action. Puis elles devront initier l'action
122     * et enfin passer à l'étape suivante (éventuellement en mettant à jour
123     * le modèle dessin.
124     * Passe à l'étape suivante avec {@link #nextStep()} ce qui met à jour
125     * le {@link #tipLabel}.
126     * Met à jour le modèle de dessin avec {@link Drawing#update()}
127     * A utiliser dans {@link MouseListener#mousePressed(MouseEvent)} ou bien
128     * dans {@link MouseListener#mouseClicked(MouseEvent)} suivant l'action à
129     * réaliser.
130     * @param e l'événement souris à utiliser pour initier la création d'une
131     * nouvelle figure à la position de cet événement
132     */
133    public abstract void startAction(MouseEvent e);
134
135    /**
136     * Terminaison de l'action sur une figure:
137     * remet l'étape courante à 0 en passant à l'étape suivante (ce qui met à
138     * jour le {@link #tipLabel} avec {@link #updateTip()}.
139     * détermine la position du point de terminaison de la figure
140     * ({@link #endPoint}). puis met à jour le dessin ({@link Drawing#update()}).
141     * A utiliser dans un {@link MouseListener#mousePressed(MouseEvent)} ou bien
142     * dans un
143     * {@link MouseListener#mouseClicked(MouseEvent)} suivant la figure à créer.
144     * @param e l'événement souris à utiliser lors de la terminaison d'un figure
145     */
146    public abstract void endAction(MouseEvent e);
147
148    /**
149     * Récupération du point de départ de l'action
150     * @param e l'événement souris d'où l'on veut récupérer le point de départ
151     */
152    public void setStartPoint(MouseEvent e)
153    {
154        startPoint = e.getPoint();
155    }
156
157    /**
158     * Récupération du point de terminaison de l'action
159     * @param e l'événement souris d'où l'on veut récupérer le point de terminaison
160     */
161    public void setendPoint(MouseEvent e)
162    {
163        endPoint = e.getPoint();
164    }
165
166    /**
167     * Passage à l'étape suivante et mise à jours des conseils utilisateurs
168     * relatifs à l'étape suivante.
169     * Lorsque le passage à l'étape suivante dépasse le nombre d'étapes prévues
170     * l'étape courante est remise à 0.
171     * @see #currentStep
172     * @see #updateTip()
173     */
174    protected void nextStep()
175    {
176        if (currentStep < (nbSteps - 1))
177        {
178            currentStep++;
179        }
180    }

```

avr 02, 17 16:17

## AbstractFigureListener.java

Page 3/3

```

181     else
182     {
183         currentStep = 0;
184     }
185
186     //      System.out.println(getClass().getSimpleName() + " nextStep to step "
187     //      + currentStep);
188
189     updateTip();
190 }
191
192 /**
193  * Mise à jour du conseil dans le {@link #tipLabel} en fonction de l'étape
194  * courante
195  */
196 protected void updateTip()
197 {
198     if (tipLabel != null)
199     {
200         tipLabel.setText(tips[currentStep]);
201     }
202     else
203     {
204         System.err.println(getClass().getSimpleName() + "::updateTip: null tipLabel");
205     }
206 }
207 }

```

avr 02, 17 16:17

## AbstractCreationListener.java

Page 1/2

```

1  package figures.listeners.creation;
2
3  import java.awt.event.MouseEvent;
4  import java.awt.event.MouseListener;
5  import java.awt.event.MouseMotionListener;
6  import java.awt.geom.Point2D;
7
8  import javax.swing.JLabel;
9
10 import figures.Drawing;
11 import figures.Drawing.Status;
12 import figures.listeners.AbstractFigureListener;
13
14 /**
15  * Listener (incomplet) des événements souris pour créer une figure. Chaque
16  * figure (Cercle, Ellipse, Rectangle, etc) est graphiquement construite par une
17  * suite de pressed/drag/release ou de clicks qui peut être différente pour
18  * chaque type de figure. Aussi les classes filles devront implémenter leur
19  * propre xxxCreationListener assurant la gestion de la création d'une nouvelle
20  * figure.
21  * @author davidroussel
22  */
23 public abstract class AbstractCreationListener extends AbstractFigureListener
24 implements MouseListener, MouseMotionListener
25 {
26     /**
27      * Constructeur protégé (destiné à être utilisé par les classes filles)
28      * @param model le modèle de dessin à modifier par ce creationListener
29      * @param infoLabel le label dans lequel afficher les conseils d'utilisation
30      * @param nbSteps le nombre d'étapes de création de la figure
31      */
32     protected AbstractCreationListener(Drawing model,
33                                         JLabel infoLabel,
34                                         int nbSteps)
35     {
36         super(model, infoLabel, nbSteps);
37     }
38
39     /**
40      * Initialisation de la création d'une nouvelle figure. détermine le point
41      * de départ de la figure ({@link #startPoint}). initie une nouvelle figure
42      * à la position de l'évènement ({@link Drawing#initiateFigure(Point2D)}),
43      * met à jour le dessin ({@link Drawing#update()}). puis passe à l'étape
44      * suivante en mettant à jour les conseils utilisateurs (
45      * {@link #updateTip()}). Pour la plupart des figures la création commence
46      * par un appui sur le bouton gauche de la souris. A utiliser dans
47      * {@link MouseListener#mousePressed(MouseEvent)} ou bien dans
48      * {@link MouseListener#mouseClicked(MouseEvent)} suivant la figure à créer.
49      * @param e l'évènement souris à utiliser pour initier la création d'une
50      * nouvelle figure à la position de cet évènement
51      */
52     @Override
53     public void startAction(MouseEvent e)
54     {
55         setStartPoint(e);
56         currentFigure = drawingModel.initiateFigure(e.getPoint());
57
58         nextStep();
59
60         drawingModel.setStatus(Status.ADDED);
61         drawingModel.update();
62     }
63
64     /**
65      * Terminaison de la création d'une figure. remet l'étape courante à 0,
66      * détermine la position du point de terminaison de la figure (
67      * {@link #endPoint}). vérifie que la figure ainsi terminée n'est pas de
68      * taille 0 ({@link #checkZeroSizeFigure()}). puis met à jour le dessin (
69      * {@link Drawing#update()}) et les conseils utilisateurs (
70      * {@link #updateTip()}). A utiliser dans un
71      * {@link MouseListener#mousePressed(MouseEvent)} ou bien dans un
72      * {@link MouseListener#mouseClicked(MouseEvent)} suivant la figure à créer.
73      * @param e l'évènement souris à utiliser lors de la terminaison d'une figure
74      */
75     @Override
76     public void endAction(MouseEvent e)
77     {
78         // Remise à zéro de currentStep pour pouvoir réutiliser ce
79         // listener sur une autre figure
80         nextStep();
81
82         setEndPoint(e);
83
84         // à la fin de la figure on la normalise pour qu'elle soit centrée
85         // sur son barycentre et la position du barycentre dans la translation
86         if (currentFigure != null)
87         {
88             currentFigure.normalize();
89         }
90     }
91     else

```

avr 02, 17 16:17

**AbstractCreationListener.java**

Page 2/2

```

91     {
92         System.err.println(getClass().getSimpleName() + "::endAction : null figure");
93     }
94
95     checkZeroSizeFigure();
96     drawingModel.update();
97
98     updateTip();
99 }
100
101 /**
102  * Contrôle de la taille de la figure créée à effectuer à la fin de la
103  * création afin d'éliminer les figures de taille 0:
104  * @return true si une figure de petite taille a été retirée
105  * @see #startPoint
106  * @see #endPoint
107  */
108 protected boolean checkZeroSizeFigure()
109 {
110     if (startPoint.distance(endPoint) < 1.0)
111     {
112         drawingModel.removeLastFigure();
113         System.err.println("Removed zero sized figure");
114         return true;
115     }
116     return false;
117 }
118 }
119

```

avr 02, 17 16:17

**package-info.java**

Page 1/1

```

1  /**
2   * Package contenant les différents widgets (éléments graphiques)
3   */
4  package widgets;

```

```

1 package figures.listeners.creation;
2
3 import java.awt.event.MouseEvent;
4 import java.awt.event.MouseWheelEvent;
5
6 import javax.swing.JLabel;
7
8 import figures.Drawing;
9
10 /**
11  * Listener permettant d'enchaîner les actions souris pour créer des formes
12  * rectangulaires comme des rectangles ou des ellipse (evt des cercles):
13  * <ol>
14  * <li>bouton 1 pressé et maintenu enfoncé</li>
15  * <li>déplacement de la souris avec le bouton enfoncé</li>
16  * <li>relâchement du bouton</li>
17  * </ol>
18  * @author davidroussel
19  */
20 public class RectangularShapeCreationListener extends AbstractCreationListener
21 {
22     /**
23      * Constructeur d'un listener à deux étapes: pressed->drag->release pour
24      * toutes les figures à caractère rectangulaire (Rectangle, Ellipse, evt
25      * Cercle)
26      *
27      * @param model le modèle de dessin à modifier par ce creationListener
28      * @param tipLabel le label dans lequel afficher les conseils utilisateur
29      */
30     public RectangularShapeCreationListener(Drawing model, JLabel tipLabel)
31     {
32         super(model, tipLabel, 2);
33
34         tips[0] = new String("Cliquez et maintenez enfoncé pour initier la figure");
35         tips[1] = new String("Relâchez pour terminer la figure");
36
37         updateTip();
38
39         System.out.println("RectangularShapeCreationListener created");
40     }
41
42     /**
43      * Création d'une nouvelle figure rectangulaire de taille 0 au point de
44      * l'évènement souris, si le bouton appuyé est le bouton gauche.
45      *
46      * @param e l'évènement souris
47      * @see AbstractCreationListener#startAction(MouseEvent)
48      * @see java.awt.event.MouseListener#mousePressed(java.awt.event.MouseEvent)
49      */
50     @Override
51     public void mousePressed(MouseEvent e)
52     {
53         if ((e.getButton() == MouseEvent.BUTTON1) ^ (currentStep == 0))
54         {
55             startAction(e);
56         }
57     }
58
59     /**
60      * Terminaison de la nouvelle figure rectangulaire si le bouton appuyé
61      * était le bouton gauche
62      * @param e l'évènement souris
63      * @see AbstractCreationListener#endAction(MouseEvent)
64      * @see java.awt.event.MouseListener#mouseReleased(java.awt.event.MouseEvent)
65      */
66     @Override
67     public void mouseReleased(MouseEvent e)
68     {
69         if ((e.getButton() == MouseEvent.BUTTON1) ^ (currentStep == 1))
70         {
71             endAction(e);
72         }
73     }
74
75     /** (non-Javadoc)
76      * @see java.awt.event.MouseListener#mouseClicked(java.awt.event.MouseEvent)
77      */
78     @Override
79     public void mouseClicked(MouseEvent e)
80     {
81         // Rien
82     }
83
84     /** (non-Javadoc)
85      * @see java.awt.event.MouseListener#mouseEntered(java.awt.event.MouseEvent)
86      */
87     @Override
88     public void mouseEntered(MouseEvent e)
89     {
90         // Rien

```

```

91     }
92
93     /** (non-Javadoc)
94      * @see java.awt.event.MouseListener#mouseExited(java.awt.event.MouseEvent)
95      */
96     @Override
97     public void mouseExited(MouseEvent e)
98     {
99         // Rien
100     }
101
102     /**
103      * (non-Javadoc)
104      * @see java.awt.event.MouseMotionListener#mouseMoved(java.awt.event.MouseEvent)
105      */
106     @Override
107     public void mouseMoved(MouseEvent e)
108     {
109         // Rien
110     }
111
112     /**
113      * Déplacement du point en bas à droite de la figure rectangulaire. si
114      * l'on se trouve à l'étape 1 (après initialisation de la figure) et que
115      * le bouton enfoncé est bien le bouton gauche
116      * @see java.awt.event.MouseMotionListener#mouseDragged(java.awt.event.MouseEvent)
117      */
118     @Override
119     public void mouseDragged(MouseEvent e)
120     {
121         if (currentStep == 1)
122         {
123             // AbstractFigure figure = drawingModel.getLastFigure();
124             if (currentFigure != null)
125             {
126                 currentFigure.setLastPoint(e.getPoint());
127             }
128             else
129             {
130                 System.err.println(getClass().getSimpleName() + "::mouseDragged: null figure");
131             }
132             drawingModel.update();
133         }
134     }
135
136
137     /** (non-Javadoc)
138      * @see java.awt.event.MouseWheelListener#mouseWheelMoved(java.awt.event.MouseWheelEvent)
139      */
140     @Override
141     public void mouseWheelMoved(MouseWheelEvent e)
142     {
143         // Rien
144     }
145
146
147 }

```

avr 02, 17 16:17

## package-info.java

Page 1/1

```

1  /**
2   * Package contenant les différents widgets (éléments graphiques)
3   */
4   package widgets;

```

avr 02, 17 16:17

## SelectionFigureListener.java

Page 1/2

```

1  /**
2   *
3   */
4   package figures.listeners;
5
6   import java.awt.event.MouseEvent;
7   import java.awt.event.MouseWheelEvent;
8
9   import javax.swing.JLabel;
10
11  import figures.Drawing;
12
13  /**
14   * Listener permettant d'ajouter ou de retirer des figures de la liste des
15   * figures sélectionnées
16   * @author davidroussel
17   */
18  public class SelectionFigureListener extends AbstractFigureListener
19  {
20
21      /**
22       * Constructeur
23       * @param model le modèle de dessin sur lequel on opère
24       * @param infoLabel le label dans lequel afficher les conseils d'utilisation
25       */
26      public SelectionFigureListener(Drawing model, JLabel infoLabel)
27      {
28          super(model, infoLabel, 1);
29
30          tips[0] = new String("Cliquez pour sélectionner/désélectionner une figure");
31          updateTip();
32      }
33
34      /* (non-Javadoc)
35       * @see java.awt.event.MouseListener#mouseClicked(java.awt.event.MouseEvent)
36       */
37      @Override
38      public void mouseClicked(MouseEvent e)
39      {
40          nextStep(); // inutile
41
42          // S'il y a une figure sous le curseur on l'ajoute où on l'enlève
43          // de la sélection suivant son état courant de sélection
44          currentFigure = drawingModel.getFigureAt(e.getPoint());
45
46          if (currentFigure != null)
47          {
48              currentFigure.setSelected(!currentFigure.isSelected());
49
50              drawingModel.updateSelection();
51          }
52      }
53
54      /* (non-Javadoc)
55       * @see java.awt.event.MouseListener#mousePressed(java.awt.event.MouseEvent)
56       */
57      @Override
58      public void mousePressed(MouseEvent e)
59      {
60          // Rien
61      }
62
63      /* (non-Javadoc)
64       * @see java.awt.event.MouseListener#mouseReleased(java.awt.event.MouseEvent)
65       */
66      @Override
67      public void mouseReleased(MouseEvent e)
68      {
69          // Rien
70      }
71
72      /* (non-Javadoc)
73       * @see java.awt.event.MouseListener#mouseEntered(java.awt.event.MouseEvent)
74       */
75      @Override
76      public void mouseEntered(MouseEvent e)
77      {
78          // Rien
79      }
80
81      /* (non-Javadoc)
82       * @see java.awt.event.MouseListener#mouseExited(java.awt.event.MouseEvent)
83       */
84      @Override
85      public void mouseExited(MouseEvent e)
86      {
87          // Rien
88      }
89
90      /* (non-Javadoc)

```

avr 02, 17 16:17

## SelectionFigureListener.java

Page 2/2

```

91  * @see java.awt.event.MouseMotionListener#mouseDragged(java.awt.event.MouseEvent)
92  */
93  @Override
94  public void mouseDragged(MouseEvent e)
95  {
96      // Rien
97  }
98
99  /* (non-Javadoc)
100  * @see java.awt.event.MouseMotionListener#mouseMoved(java.awt.event.MouseEvent)
101  */
102  @Override
103  public void mouseMoved(MouseEvent e)
104  {
105      // Rien
106  }
107
108  /* (non-Javadoc)
109  * @see java.awt.event.MouseWheelListener#mouseWheelMoved(java.awt.event.MouseWheelEvent)
110  */
111  @Override
112  public void mouseWheelMoved(MouseWheelEvent e)
113  {
114      // Rien
115  }
116
117  /* (non-Javadoc)
118  * @see figures.listeners.AbstractFigureListener#startAction(java.awt.event.MouseEvent)
119  */
120  @Override
121  public void startAction(MouseEvent e)
122  {
123      // Rien
124  }
125
126  /* (non-Javadoc)
127  * @see figures.listeners.AbstractFigureListener#endAction(java.awt.event.MouseEvent)
128  */
129  @Override
130  public void endAction(MouseEvent e)
131  {
132      // Rien
133  }
134 }

```

avr 02, 17 16:17

## AbstractTransformShapeListener.java

Page 1/4

```

1  package figures.listeners.transform;
2
3  import java.awt.event.InputEvent;
4  import java.awt.event.MouseEvent;
5  import java.awt.event.MouseListener;
6  import java.awt.event.MouseWheelEvent;
7  import java.awt.geom.AffineTransform;
8  import java.awt.geom.Point2D;
9
10 import javax.swing.JLabel;
11
12 import figures.Drawing;
13 import figures.listeners.AbstractFigureListener;
14 import figures.listeners.creation.AbstractCreationListener;
15
16 /**
17  * Listener permettant de transformer une figure
18  * <ol>
19  * <li>bouton 1 pressé et maintenu enfoncé</li>
20  * <li>déplacement de la souris avec le bouton enfoncé</li>
21  * <li>relâchement du bouton</li>
22  * </ol>
23  * @author davidroussel
24  */
25 public abstract class AbstractTransformShapeListener extends AbstractFigureListener
26 {
27     /**
28      * La transformation initiale de la figure
29      */
30     protected AffineTransform initialTransform;
31
32     /**
33      * Indique si seules les figures sélectionnées sont transformables ou pas
34      */
35     protected boolean onlySelected;
36
37     /**
38      * Le centre de la figure sélectionnée (car on l'utilisera souvent)
39      */
40     protected Point2D center;
41
42     /**
43      * Le modificateur (Ctrl, Shift, Alt, etc.) applicable lors du traitement
44      * des événements souris
45      * @see InputEvent#SHIFT_DOWN_MASK
46      * @see InputEvent#CTRL_DOWN_MASK
47      * @see InputEvent#ALT_DOWN_MASK
48      * @see InputEvent#META_DOWN_MASK
49      */
50     protected int keyMask;
51
52     /**
53      * Valeur par défaut lorsqu'aucun key mask n'est requis
54      */
55     protected static final int NoKeyMask = 0;
56
57     /**
58      * Constructeur d'un listener à deux étapes: pressed->drag->release pour
59      * transformer les figures
60      * @param model le modèle de dessin à modifier par ce listener
61      * @param tipLabel le label dans lequel afficher les conseils utilisateur
62      */
63     public AbstractTransformShapeListener(Drawing model, JLabel tipLabel)
64     {
65         super(model, tipLabel, 2);
66
67         tips[0] = new String("Cliquez et maintenez enfoncé pour transformer la figure");
68         tips[1] = new String("Relâchez pour terminer le déplacement");
69
70         updateTip();
71
72         System.out.println(getClass().getSimpleName() + " created");
73
74         center = null;
75
76         keyMask = NoKeyMask;
77     }
78
79     /**
80      * Vérifie que seul le {@link InputEvent#BUTTON1_MASK} ainsi que le
81      * {@link #keyMask} sont présents dans les modifieurs renvoyés par
82      * {@link MouseEvent#getModifiers()} mais <b>aucun autre</b> modifier
83      * @param modifiers les modifieurs à vérifier
84      * @return true si seuls {@link InputEvent#BUTTON1_MASK} et {@link #keyMask}
85      * sont présents dans les modifieurs, false sinon
86      */
87     public boolean checkModifiers(int modifiers)
88     {
89         return modifiers == (InputEvent.BUTTON1_MASK | keyMask);
90     }

```

avr 02, 17 16:17

## AbstractTransformShapeListener.java

Page 2/4

```

91  /**
92  * Initialisation de la transformation de la figure. Détermine le point de
93  * départ de la transformation de la figure (@link #startPoint) ainsi que
94  * la figure sélectionnée qui peut éventuellement être nulle s'il n'y a pas
95  * de figures sélectionnées ou sous le curseur.
96  * A utiliser dans
97  * {@link MouseListener#mousePressed(MouseEvent)} ou bien dans
98  * {@link MouseListener#mouseClicked(MouseEvent)} suivant la figure à créer.
99  * @see #mousePressed(MouseEvent)
100  * @see #mouseClicked(MouseEvent)
101  */
102  @Override
103  public void startAction(MouseEvent e)
104  {
105      setStartPoint(e);
106
107      currentFigure = drawingModel.getFigureAt(startPoint);
108      if (currentFigure != null)
109      {
110          center = currentFigure.getCenter();
111
112          init();
113
114          nextStep();
115
116          drawingModel.update(); // optional
117      }
118      else
119      {
120          System.err.println(getClass().getSimpleName() + "::startAction: null figure");
121      }
122  }
123
124  /**
125  * Initialisations particulières à l'initialisation du listener
126  * <ul>
127  * <li>Initialisation de transformation initiale</li>
128  * <li>...</li>
129  * </ul>
130  */
131  public abstract void init();
132
133  /**
134  * Terminaison du déplacement d'une figure. remet l'étape courante à 0
135  * détermine la position du point de terminaison du déplacement de la figure
136  * (@link #endPoint). puis met à jour le dessin (
137  * {@link Drawing#update()}) et les conseils utilisateurs (
138  * {@link #updateTip()}). A utiliser dans un
139  * {@link MouseListener#mousePressed(MouseEvent)} ou bien dans un
140  * {@link MouseListener#mouseClicked(MouseEvent)} suivant la figure à créer.
141  * @param e l'évènement souris à utiliser lors de la terminaison d'un figure
142  */
143  @Override
144  public void endAction(MouseEvent e)
145  {
146      if (currentStep == 1)
147      {
148          // Remise à zéro de currentStep pour pouvoir réutiliser ce
149          // listener sur une autre figure
150          nextStep();
151
152          setEndPoint(e);
153
154          currentFigure = null;
155
156          drawingModel.update();
157      }
158  }
159
160  /**
161  * Création d'une nouvelle figure rectangulaire de taille 0 au point de
162  * l'évènement souris. si le bouton appuyé est le bouton gauche.
163  * @param e l'évènement souris
164  * @see AbstractCreationListener#startAction(MouseEvent)
165  * @see java.awt.event.MouseListener#mousePressed(java.awt.event.MouseEvent)
166  */
167  @Override
168  public void mousePressed(MouseEvent e)
169  {
170      currentFigure = drawingModel.getFigureAt(e.getPoint());
171
172      if (currentFigure != null)
173      {
174          if (currentFigure.isSelected() ^
175              (e.getButton() == MouseEvent.BUTTON1) &&
176              checkModifiers(e.getModifiers()))
177          {
178              startAction(e);
179          }
180      }

```

avr 02, 17 16:17

## AbstractTransformShapeListener.java

Page 3/4

```

181     }
182     else
183     {
184         System.err.println(getClass().getSimpleName() + "::mousePressed: null figure");
185     }
186 }
187
188 /**
189  * Terminaison de la nouvelle figure rectangulaire si le bouton appuyé
190  * était le bouton gauche
191  * @param e l'évènement souris
192  * @see AbstractCreationListener#endAction(MouseEvent)
193  * @see java.awt.event.MouseListener#mouseReleased(java.awt.event.MouseEvent)
194  */
195 @Override
196 public void mouseReleased(MouseEvent e)
197 {
198     if (e.getButton() == MouseEvent.BUTTON1) // On se fiche du keymask pour terminer l'action
199     {
200         System.out.println("TransformShapeListener ended...");
201         endAction(e);
202     }
203 }
204
205 /**
206  * (non-Javadoc)
207  * @see java.awt.event.MouseListener#mouseClicked(java.awt.event.MouseEvent)
208  */
209 @Override
210 public void mouseClicked(MouseEvent e)
211 {
212     // Rien
213 }
214
215 /**
216  * (non-Javadoc)
217  * @see java.awt.event.MouseListener#mouseEntered(java.awt.event.MouseEvent)
218  */
219 @Override
220 public void mouseEntered(MouseEvent e)
221 {
222     // Rien
223 }
224
225 /**
226  * (non-Javadoc)
227  * @see java.awt.event.MouseListener#mouseExited(java.awt.event.MouseEvent)
228  */
229 @Override
230 public void mouseExited(MouseEvent e)
231 {
232     // Rien
233 }
234
235 /**
236  * (non-Javadoc)
237  * @see java.awt.event.MouseMotionListener#mouseMoved(java.awt.event.MouseEvent)
238  */
239 @Override
240 public void mouseMoved(MouseEvent e)
241 {
242     // Rien
243 }
244
245 /**
246  * Déplacement du point en bas à droite de la figure rectangulaire. si
247  * l'on se trouve à l'étape 1 (après initialisation du déplacement) et que
248  * le bouton enfoncé est bien le bouton gauche
249  * @see java.awt.event.MouseMotionListener#mouseDragged(java.awt.event.MouseEvent)
250  */
251 @Override
252 public void mouseDragged(MouseEvent e)
253 {
254     if (currentStep == 1)
255     {
256         if (currentFigure != null)
257         {
258             updateDrag(e);
259
260             drawingModel.update();
261         }
262         else
263         {
264             System.err.println(getClass().getSimpleName() + "::mouseDragged: null figure");
265         }
266     }
267 }
268
269 /**
270

```

avr 02, 17 16:17

## AbstractTransformShapeListener.java

Page 4/4

```

271  * Mise à jour de la transformation courante et application
272  * de la transformation initiale ({@link #initialTransformation}) et
273  * de la transformation courante
274  * @param e événement souris
275  */
276  public abstract void updateDrag(MouseEvent e);
277
278  /* (non-Javadoc)
279   * @see java.awt.event.MouseWheelListener#mouseWheelMoved(java.awt.event.MouseWheelEvent)
280   */
281  @Override
282  public void mouseWheelMoved(MouseWheelEvent e)
283  {
284      // Rien
285  }
286  }

```

avr 02, 17 16:17

## MoveShapeListener.java

Page 1/1

```

1  package figures.listeners.transform;
2
3  import java.awt.event.MouseEvent;
4  import java.awt.geom.AffineTransform;
5  import java.awt.geom.Point2D;
6
7  import javax.swing.JLabel;
8
9  import figures.Drawing;
10
11  /**
12   * Listener permettant de déplacer une figure
13   * <ol>
14   * <li>bouton 1 pressé et maintenu enfoncé</li>
15   * <li>déplacement de la souris avec le bouton enfoncé</li>
16   * <li>relâchement du bouton</li>
17   * </ol>
18   * @author davidroussel
19   */
20  public class MoveShapeListener extends AbstractTransformShapeListener
21  {
22      /**
23       * Le dernier point
24       * @note Utilisé pour calculer le déplacement entre l'évènement courant
25       * et l'évènement précédent
26       * @note modifié dans {@link #mouseDragged(MouseEvent)}
27       */
28      private Point2D lastPoint;
29
30      /**
31       * Constructeur d'un listener à deux étapes: pressed->drag->release pour
32       * déplacer toutes les figures
33       * @param model le modèle de dessin à modifier par ce Listener
34       * @param tipLabel le label dans lequel afficher les conseils utilisateur
35       */
36      public MoveShapeListener(Drawing model, JLabel tipLabel)
37      {
38          super(model, tipLabel);
39      }
40
41      /* (non-Javadoc)
42       * @see figures.listeners.transform.AbstractTransformShapeListener#init()
43       */
44      @Override
45      public void init()
46      {
47          lastPoint = startPoint;
48          if (currentFigure != null)
49          {
50              initialTransform = currentFigure.getTranslation();
51              System.out.println("MoveShapeListener2 initialized");
52          }
53          else
54          {
55              System.err.println(getClass().getSimpleName() + "::init: null figure");
56          }
57      }
58
59      /* (non-Javadoc)
60       * @see figures.listeners.transform.AbstractTransformShapeListener#updateDrag(java.awt.event.MouseEvent)
61       */
62      @Override
63      public void updateDrag(MouseEvent e)
64      {
65          // System.out.println("MoveShapeListener2 dragged");
66          Point2D currentPoint = e.getPoint();
67          double dx = currentPoint.getX() - lastPoint.getX();
68          double dy = currentPoint.getY() - lastPoint.getY();
69          AffineTransform translate = AffineTransform.getTranslateInstance(dx, dy);
70          translate.concatenate(initialTransform);
71          currentFigure.setTranslation(translate);
72      }
73  }

```



avr 02, 17 16:17

**package-info.java**

Page 1/1

```

1  /**
2   * Package contenant les différents widgets (éléments graphiques)
3   */
4  package widgets;

```

avr 02, 17 16:17

**AbstractFigureTreeModel.java**

Page 1/5

```

1  package figures.treemodels;
2
3  import java.util.HashSet;
4  import java.util.List;
5  import java.util.Observable;
6  import java.util.Observer;
7  import java.util.Set;
8  import java.util.Vector;
9  import java.util.stream.Collectors;
10 import java.util.stream.Stream;
11
12 import javax.swing.JTree;
13 import javax.swing.event.TreeModelEvent;
14 import javax.swing.event.TreeModelListener;
15 import javax.swing.event.TreeSelectionEvent;
16 import javax.swing.event.TreeSelectionListener;
17 import javax.swing.tree.TreeModel;
18 import javax.swing.tree.TreePath;
19 import javax.swing.tree.TreeSelectionModel;
20
21 import figures.Drawing;
22 import figures.Drawing.Status;
23 import figures.Figure;
24
25 public abstract class AbstractFigureTreeModel implements TreeModel, Observer, TreeSelectionListener
26 {
27     /**
28      * L'élément racine de l'arbre (une simple chaîne de caractères)
29      */
30     protected String rootElement;
31
32     /**
33      * Le modèle de dessin.
34      * On a besoin de garder une référence vers le modèle de dessin lorsque
35      * la liste des figures sélectionnées dans l'arbre change afin que l'on
36      * puisse le notifier des changements
37      */
38     protected Drawing drawing;
39
40     /**
41      * Le JTree utilisé pour visualiser cet arbre.
42      * On a besoin de garder une référence vers cette vue afin de
43      * spécifier (programmatically) quels sont les nœuds sélectionnés
44      * en fonction des figures sélectionnées.
45      * @see #selectedFigures
46      */
47     protected JTree treeView;
48
49     /**
50      * Liste des figures sélectionnées dans l'arbre
51      * (On considérera que seule la sélection des figures a un effet. c'àd.
52      * la sélection du rootNode, ou des autres nœuds parents n'as pas d'effet)
53      */
54     protected Set<TreePath> selectedFigures;
55
56     /**
57      * La liste des listeners de ce modèle
58      */
59     protected Vector<TreeModelListener> treeModelListeners;
60
61     /**
62      * Constructeur de l'arbre des figures
63      * @param drawing le modèle de dessin
64      * @param tree le JTree utilisé pour visualiser cet arbre
65      * @param rootName le nom du nœud racine
66      */
67     public AbstractFigureTreeModel(Drawing drawing, JTree tree, String rootName)
68         throws NullPointerException
69     {
70         this.drawing = drawing;
71         treeView = tree;
72         treeView.addTreeSelectionListener(this);
73         rootElement = new String(rootName);
74         selectedFigures = new HashSet<TreePath>();
75         treeModelListeners = new Vector<TreeModelListener>();
76         if (drawing != null)
77         {
78             drawing.addObserver(this);
79         }
80         else
81         {
82             throw new NullPointerException("FigureTypeTreeModel(null drawing)");
83         }
84     }
85
86     @Override
87     protected void finalize() throws Throwable
88     {
89         drawing.deleteObserver(this);
90         rootElement = null;

```

avr 02, 17 16:17

## AbstractFigureTreeModel.java

Page 2/5

```

91      drawing = null;
92      treeView.removeTreeSelectionListener(this);
93      treeView = null;
94      selectedFigures.clear();
95      selectedFigures = null;
96      treeModelListeners.clear();
97      treeModelListeners = null;
98      super.finalize();
99  }
100
101  /**
102   * Mise à jour par l'observable (en l'occurrence un {@link Drawing})
103   * @param observable le {@link Drawing}
104   * @param data les données à transmettre (non utilisé ici)
105   * @see Observer#update(Observable, Object)
106   */
107  @Override
108  public void update(Observable observable, Object data)
109  {
110      if (observable instanceof Drawing)
111      {
112          synchronized (observable)
113          {
114              drawing = (Drawing) observable;
115              Stream<Figure> stream = drawing.stream();
116
117              // Rebuild a vector from the stream
118              Vector<Figure> figures = stream.sequential()
119                  .collect(Collectors.toCollection(Vector::new));
120
121              if (drawing.hasStatus(Status.REORDERED))
122              {
123                  fireTreeStructureChanged(new TreePath(new Object[] {rootElement}));
124              }
125
126              // Clears currently selected figures
127              selectedFigures.clear();
128
129              /**
130               * Ajout des figures présentes dans #drawing et
131               * de leur TreePath dans #selectedFigures si elles sont
132               * sélectionnées
133               */
134              addFiguresFromDrawing(figures);
135
136              /**
137               * Mise à jour des figures déjà sélectionnées dans le #treeView
138               */
139              updateSelectedPath();
140
141              /**
142               * Remove figures that are no longer in drawings
143               */
144              removeFiguresFromDrawing(figures);
145              // System.out.println("AbstractFigureTreeModel updated : \n" + this);
146          }
147      }
148      else
149      {
150          System.err.println("Observable is not an instance of Drawing");
151      }
152  }
153
154  /**
155   * Ajout des figures de {@link Drawing} à l'arbre (si elles n'y sont pas
156   * déjà) et si les figures sont sélectionnées ajout des paths de sélection
157   * dans {@link #selectedFigures}
158   * @param figures les figures à ajouter
159   * @see Figure#isSelected()
160   * @see la liste des paths des figures sélectionnées
161   * @see {@link #selectedFigures} est vide
162   */
163  protected abstract void addFiguresFromDrawing(List<Figure> figures);
164
165  /**
166   * Retrait des figures qui ne sont plus dans {@link Drawing} de l'arbre
167   * @param figures les figures de {@link Drawing}
168   */
169  protected abstract void removeFiguresFromDrawing(List<Figure> figures);
170
171  /**
172   * Mise à jour des noeuds sélectionnés dans le {@link #treeView} d'après
173   * les paths répertoriés dans {@link #selectedFigures}
174   */
175  protected void updateSelectedPath()
176  {
177      System.out.println("AbstractFigureTreeModel::updateSelectedPath");
178      if (treeView != null)
179      {
180          TreeSelectionModel tsm = treeView.getSelectionModel();

```

avr 02, 17 16:17

## AbstractFigureTreeModel.java

Page 3/5

```

181      if (tsm != null)
182      {
183          TreePath[] treePaths = selectedFigures.toArray(new TreePath[0]);
184          printTreePaths("TreeSelectionModel:updateSelectedPaths", treePaths);
185          tsm.setSelectionPaths(treePaths);
186      }
187      else
188      {
189          System.err.println("AbstractFigureTreeModel:updateSelectedPath : null Selection Model");
190      }
191  }
192  else
193  {
194      System.err.println("AbstractFigureTreeModel:updateSelectedPath : null TreeView");
195  }
196  }
197
198  /**
199   * Méthode à utiliser lorsque la structure de l'arbre change.
200   * Tous les éléments situés en dessous de path sont mis à jour
201   * @param path le chemin en dessous duquel l'arbre a changé
202   */
203  protected void fireTreeStructureChanged(TreePath path)
204  {
205      if (treeModelListeners.size() > 0)
206      {
207          /**
208           * Used to create an event when the node structure has changed in
209           * some way. Identifying the path to the root of the modified
210           * subtree as a TreePath object.
211           */
212          TreeModelEvent e = new TreeModelEvent(this, path);
213          for (TreeModelListener tml : treeModelListeners)
214          {
215              System.out.println("FireTreeStructureChanged(" + e + " to " + tml);
216              tml.treeStructureChanged(e);
217          }
218      }
219  }
220
221  /**
222   * Méthode à utiliser lorsqu'un ou plusieurs noeuds sont ajoutés à
223   * l'arbre
224   * @param path the path to the parent of inserted node(s)
225   * @param newChildIndices an array of the indices of the new inserted nodes
226   * @param newNodeNodes an array of the new inserted nodes (Optional)
227   * @see javax.swing.event.TreeModelListener#treeNodesInserted(TreeModelEvent)
228   */
229  protected void fireTreeNodesInserted(TreePath path,
230      int[] newChildIndices,
231      Object[] newNodeNodes)
232  {
233      if (treeModelListeners.size() > 0)
234      {
235          TreeModelEvent e =
236              new TreeModelEvent(this, path, newChildIndices, newNodeNodes);
237          for (TreeModelListener tml : treeModelListeners)
238          {
239              tml.treeNodesInserted(e);
240          }
241      }
242  }
243
244  /**
245   * Méthode à utiliser lorsqu'un ou plusieurs noeuds sont retirés de l'arbre
246   * @param path the path to the former parent of deleted node
247   * @param oldChildIndices an array of indices (in ascending order) where
248   * the removed nodes used to be
249   * @note if a subtree is removed from the tree this method may only be
250   * invoked once for the root of the removed subtree, not once for
251   * each individual set of siblings removed.
252   */
253  protected void fireNodesRemoved(TreePath path,
254      int[] oldChildIndices,
255      Object[] oldNodes)
256  {
257      if (treeModelListeners.size() > 0)
258      {
259          TreeModelEvent e =
260              new TreeModelEvent(this, path, oldChildIndices, oldNodes);
261          for (TreeModelListener tml : treeModelListeners)
262          {
263              tml.treeNodesRemoved(e);
264          }
265      }
266  }
267
268  /**
269   * Méthode à utiliser lorsqu'un ou plusieurs noeuds sont changés (par
270   * exemple s'il sont sélectionnés programmatiquement)

```

avr 02, 17 16:17

## AbstractFigureTreeModel.java

Page 4/5

```

271  * @param treePathes l'ensemble des {@link TreePath} des noeuds changés
272  */
273  protected void fireNodesChanged(TreePath[] treePathes)
274  {
275      for (int i = 0; i < treePathes.length; i++)
276      {
277          for (TreeModelListener tml : treeModelListeners)
278          {
279              tml.treeNodesChanged(new TreeModelEvent(this, treePathes[i]));
280          }
281      }
282  }
283
284  /**
285   * Recherche d'un noeud terminal dans l'arbre
286   * @param f La figure à rechercher dans l'arbre
287   * @return le {@link TreePath} de la figure recherchée. qui sera vide
288   * si la figure recherchée ne fait pas partie de l'arbre
289   */
290  protected abstract TreePath findLeaf(Figure f);
291
292  @Override
293  public Object getRoot()
294  {
295      return rootElement;
296  }
297
298  @Override
299  public void valueForPathChanged(TreePath path, Object newValue)
300  {
301      System.out
302          .println("**** valueForPathChanged: " + path + " --> " + newValue);
303  }
304
305  @Override
306  public void addTreeModelListener(TreeModelListener l)
307  {
308      if ((l != null) ^ !treeModelListeners.contains(l))
309      {
310          treeModelListeners.add(l);
311      }
312  }
313
314  @Override
315  public void removeTreeModelListener(TreeModelListener l)
316  {
317      if (treeModelListeners.contains(l))
318      {
319          treeModelListeners.remove(l);
320      }
321  }
322
323  /**
324   * Callback déclenché lorsqu'un noeud est sélectionné dans le {@link #treeView}
325   * @param e l'événement de sélection dans le {@link JTree}
326   * @see javax.swing.event.TreeSelectionListener#valueChanged(javax.swing.event.TreeSelectionEven
327   */
328  @Override
329  public void valueChanged(TreeSelectionEvent e)
330  {
331      JTree tree = (JTree) e.getSource();
332      int count = tree.getSelectionCount();
333      TreePath[] paths = tree.getSelectionPaths();
334
335      printTreePathes("TreeSelectionListener::valueChanged", paths);
336
337      drawing.clearSelection();
338
339      for (int i = 0; i < count; i++)
340      {
341          // System.out.println("Selection [" + i + "] = " + paths[i]);
342          Object[] objPath = paths[i].getPath();
343          int pathSize = paths[i].getPathCount();
344          Object node = objPath[pathSize - 1];
345          if (node instanceof Figure)
346          {
347              Figure figure = (Figure) node;
348              figure.setSelected(true);
349          }
350      }
351
352      drawing.updateSelection();
353
354  public void printTreePathes(String message, TreePath[] pathes)
355  {
356      System.out.print(message + " = ");
357      if (pathes != null)
358      {
359

```

avr 02, 17 16:17

## AbstractFigureTreeModel.java

Page 5/5

```

360      for (int i = 0; i < pathes.length; i++)
361      {
362          System.out.print(pathes[i] + ", ");
363      }
364      System.out.println();
365  }
366  }
367  }

```

avr 02, 17 16:17

## FigureTreeModel.java

Page 1/3

```

1  /**
2   *
3   */
4  package figures.treemodels;
5
6  import java.util.Iterator;
7  import java.util.List;
8  import java.util.Vector;
9
10 import javax.swing.JTree;
11 import javax.swing.tree.TreePath;
12
13 import figures.Drawing;
14 import figures.Figure;
15
16 /**
17  * Figure TreeModel dans lequel les noeuds de niveau 1 sont les figures,
18  * Il n'y a pas de noeuds de niveau 2.
19  * @author davidroussel
20  */
21 public class FigureTreeModel extends AbstractFigureTreeModel
22 {
23     /**
24      * La liste des figure dans l'arbre
25      */
26     private List<Figure> figures;
27
28     /**
29      * Constructeur de l'arbre des types de figures
30      * @param drawing le modèle de dessin
31      * @param tree le JTree utilisé pour visualiser cet arbre
32      */
33     public FigureTreeModel(Drawing drawing, JTree tree) throws NullPointerException
34     {
35         super(drawing, tree, "Figures");
36         figures = new Vector<Figure>();
37         update(drawing, null); // force Tree build
38     }
39
40     /**
41      * Ajout des figures de {@link Drawing} à l'arbre (si elles n'y sont pas
42      * déjà).
43      * @param figures les figures à ajouter
44      * @param la liste des paths des figures sélectionnées
45      * ({@link AbstractFigureTreeModel#selectedFigures}) est vide
46      */
47     @Override
48     protected void addFiguresFromDrawing(List<Figure> figures)
49     {
50         for (Iterator<Figure> fit = figures.iterator(); fit.hasNext();)
51         {
52             Figure figure = fit.next();
53
54             if (figure.isSelected())
55             {
56                 TreePath selectedPath = new TreePath(new Object[] {
57                     rootElement,
58                     figure
59                 });
60                 selectedFigures.add(selectedPath);
61             }
62
63             if (!this.figures.contains(figure))
64             {
65                 this.figures.add(figure);
66
67                 TreePath parentPath = new TreePath(new Object[] { rootElement });
68                 int[] indexes = new int[] { this.figures.size() - 1 };
69                 Object[] nodes = new Object[] { figure };
70                 fireTreeNodesInserted(parentPath, indexes, nodes);
71             }
72         }
73
74         printTreePathes("FigureTreeModel:addFiguresFromDrawing["
75             + selectedFigures.size() + "]",
76             selectedFigures.toArray(new TreePath[0]));
77     }
78
79     /**
80      * Retrait des figures qui ne sont plus dans {@link Drawing} de l'arbre
81      * @param figures les figures de {@link Drawing}
82      */
83     @Override
84     protected void removeFiguresFromDrawing(List<Figure> figures)
85     {
86         int figureIndex = 0;
87         for (Iterator<Figure> fit = this.figures.iterator(); fit.hasNext();)
88         {
89             Figure figure = fit.next();
90             if (!figures.contains(figure))

```

avr 02, 17 16:17

## FigureTreeModel.java

Page 2/3

```

91     {
92         fit.remove();
93
94         // Notify the listeners
95         TreePath parentPath = new TreePath(new Object[] { rootElement });
96         int[] childIndex = new int[] { figureIndex };
97         Object[] nodes = new Object[] { figure };
98         fireNodesRemoved(parentPath, childIndex, nodes);
99     }
100     else
101     {
102         figureIndex++;
103     }
104 }
105
106 /**
107  * Recherche d'un noeud terminal dans l'arbre
108  * @param f La figure à rechercher dans l'arbre
109  * @return le {@link TreePath} de la figure recherchée. qui sera vide
110  * si la figure recherchée ne fait pas partie de l'arbre
111  */
112 @Override
113 protected TreePath findLeaf(Figure f)
114 {
115     if (figures.contains(f))
116     {
117         int index = figures.indexOf(f);
118         return new TreePath(new Object[] {
119             rootElement,
120             figures.get(index)
121         });
122     }
123     else
124     {
125         return new TreePath(new Object[0]);
126     }
127 }
128
129 /**
130  * (non-Javadoc)
131  * @see javax.swing.tree.TreeModel#getChild(java.lang.Object, int)
132  */
133 @Override
134 public Object getChild(Object parent, int index)
135 {
136     // System.out.println("getChild(" + parent + ", " + index + ")");
137     if (parent == rootElement)
138     {
139         // return figureTypeFromIndex(index);
140         if ((index ≥ 0) ^ (index < figures.size()))
141         {
142             return figures.get(index);
143         }
144
145         return null;
146     }
147     else
148     {
149         // Figures nodes have no children
150         return null;
151     }
152 }
153
154 /**
155  * (non-Javadoc)
156  * @see javax.swing.tree.TreeModel#getChildCount(java.lang.Object)
157  */
158 @Override
159 public int getChildCount(Object parent)
160 {
161     if (parent == rootElement)
162     {
163         return figures.size();
164     }
165     else
166     {
167         // Figures nodes have no children
168         return 0;
169     }
170 }
171
172 /**
173  * (non-Javadoc)
174  * @see javax.swing.tree.TreeModel#isLeaf(java.lang.Object)
175  */
176 @Override
177 public boolean isLeaf(Object node)
178 {
179     if (node == rootElement)

```

avr 02, 17 16:17

**FigureTreeModel.java**

Page 3/3

```
181     {
182         return false;
183     }
184
185     return true;
186 }
187
188 /**
189  * (non-Javadoc)
190  * @see javax.swing.tree.TreeModel#getIndexOfChild(java.lang.Object,
191  * java.lang.Object)
192  */
193 @Override
194 public int getIndexOfChild(Object parent, Object child)
195 {
196     if (parent == rootElement)
197     {
198         return figures.indexOf(child);
199     }
200
201     return -1;
202 }
203
204 /**
205  * (non-Javadoc)
206  * @see java.lang.Object#toString()
207  */
208 @Override
209 public String toString()
210 {
211     StringBuilder sb = new StringBuilder();
212
213     sb.append(rootElement + "\n");
214     for (Figure figure : figures)
215     {
216         sb.append("+-").append(figure.toString()).append('\n');
217     }
218
219     return sb.toString();
220 }
221 }
```

avr 02, 17 16:17

**package-info.java**

Page 1/1

```
1 /**
2  * Package contenant les différents widgets (éléments graphiques)
3  */
4 package widgets;
```

avr 02, 17 16:17

## FigureFilter.java

Page 1/2

```

1 package filters;
2 import java.util.function.Predicate;
3 import figures.Figure;
4
5 /**
6  * Prédicat permettant de filtrer les figures à partir d'un élément de type T.
7  * T pourra être instancié avec divers types dans les classes filles pour
8  * filtrer :
9  * <ul>
10  * <li>le type de figures: {@link figures.enums.FigureType}</li>
11  * <li>la couleur de remplissage ou de trait: {@link java.awt.Paint}</li>
12  * <li>le type de trait: {@link figures.enums.LineType}</li>
13  * </ul>
14  * @author davidroussel
15  */
16
17 public abstract class FigureFilter<T> implements Predicate<Figure>
18 {
19     /**
20      * L'élément sur lequel filter les figures
21      */
22     protected T element;
23
24     /**
25      * Constructeur par défaut
26      */
27     public FigureFilter()
28     {
29         element = null;
30     }
31
32     /**
33      * Constructeur d'un figure filter
34      * @param element l'élément de référence du prédicat
35      */
36     public FigureFilter(T element)
37     {
38         this.element = element;
39     }
40
41     /**
42      * Accesseur à l'élément du filtre
43      * @return l'élément du filtre
44      */
45     public T getElement()
46     {
47         return element;
48     }
49
50     /**
51      * Test du prédicat
52      * @param f la figure à tester
53      * @return vrai si un élément de la figure f correspond à l'élément contenu
54      * dans ce prédicat (par exemple figure.getType() == element pour filtrer
55      * les types de figures)
56      * @see java.util.function.Predicate#test(java.lang.Object)
57      */
58     @Override
59     public abstract boolean test(Figure f);
60
61     /**
62      * Comparaison avec un autre objet
63      * @param obj l'objet à comparer
64      * @return true si l'autre objet est un filtre sur le même type d'élément
65      */
66     @Override
67     public boolean equals(Object obj)
68     {
69         if (obj == null)
70         {
71             return false;
72         }
73
74         if (obj == this)
75         {
76             return true;
77         }
78
79         if (obj instanceof FigureFilter<?>)
80         {
81             FigureFilter<?> ff = (FigureFilter<?>) obj;
82             if ((ff.element != null) ^ (element != null))
83             {
84                 if (ff.element.getClass() != element.getClass())
85                 {
86                     @SuppressWarnings("unchecked")
87                     FigureFilter<T> fft = (FigureFilter<T>) ff;
88                     return element.equals(fft.element);
89                 }
90             }

```

avr 02, 17 16:17

## FigureFilter.java

Page 2/2

```

91         else
92         {
93             if ((element != null) ^ (ff.element != null))
94             {
95                 return false;
96             }
97             else
98             {
99                 return true;
100             }
101         }
102     }
103
104     return false;
105 }
106
107 /**
108  * Chaîne de caractères représentant le filtre
109  * @return une chaîne de caractère représentant le filtre
110  */
111 @Override
112 public String toString()
113 {
114     return new String(getClass().getSimpleName() + "<"
115         + (element != null ? element.getClass().getSimpleName() : "null")
116         + ">(" + (element != null ? element.toString() : "") + ")");
117 }
118
119 }

```

avr 02, 17 16:17

## FigureFilters.java

Page 1/3

```

1 package filters;
2
3 import java.util.Collection;
4 import java.util.Iterator;
5 import java.util.Vector;
6 import java.util.function.Predicate;
7
8 import figures.Figure;
9
10 /**
11  * Collection de filtres
12  * @author davidroussel
13  */
14 public class FigureFilters<T> implements Collection<FigureFilter<T>>, Predicate<Figure>
15 {
16     /**
17      * Vecteur de filtres
18      */
19     private Vector<FigureFilter<T>> filters;
20
21     /**
22      * Constructeur par défaut
23      */
24     public FigureFilters()
25     {
26         filters = new Vector<FigureFilter<T>>();
27     }
28
29     /**
30      * Test du prédicat
31      * @param f la figure à tester
32      * @return true si l'un au moins des prédicats de la collection est vrai,
33      * false sinon
34      * @see filters.FigureFilter#test(figures.Figure)
35      */
36     @Override
37     public boolean test(Figure f)
38     {
39         boolean result = false;
40
41         for (FigureFilter<T> ff : this)
42         {
43             boolean thisResult = ff.test(f);
44             // System.out.println(ff + (thisResult ? "passed": "denied"));
45             result |= thisResult;
46         }
47
48         // System.out.println(this + (result ? "passed": "denied"));
49
50         return result;
51     }
52
53     /**
54      * Taille de la collection
55      * @return la taille de la collection
56      */
57     @Override
58     public int size()
59     {
60         return filters.size();
61     }
62
63     /**
64      * Teste si la collection est vide
65      * @return true si la collection est vide
66      */
67     @Override
68     public boolean isEmpty()
69     {
70         return filters.isEmpty();
71     }
72
73     /**
74      * Test de contenu d'un objet dans la collection de filtres
75      * @param o l'objet recherché dans la collection de filtres
76      * @return true si l'objet est contenu dans la collection de filtres
77      */
78     @Override
79     public boolean contains(Object o)
80     {
81         return filters.contains(o);
82     }
83
84     /**
85      * Itérateur de la collection de {@link FigureFilter}
86      * @return l'itérateur sur les filtres de la collection
87      */
88     @Override
89     public Iterator<FigureFilter<T>> iterator()
90     {

```

avr 02, 17 16:17

## FigureFilters.java

Page 2/3

```

91         return filters.iterator();
92     }
93
94     /**
95      * Conversion en tableau d'objets
96      * @return un tableau d'objets contenant les éléments de la collection
97      */
98     @Override
99     public Object[] toArray()
100     {
101         return filters.toArray();
102     }
103
104     /**
105      * Conversion en tableau générique
106      * @param a un tableau générique spécimen
107      * @return un tableau générique contenant les éléments de la collection
108      */
109     @Override
110     @SuppressWarnings("hiding")
111     public <T> T[] toArray(T[] a)
112     {
113         return filters.toArray(a);
114     }
115
116     /**
117      * Ajout d'un nouveau filtre à la collection uniquement si celle ci ne
118      * contient pas déjà ce filtre
119      * @param filter le filtre à ajouter
120      * @return true si le filtre a été ajouté n'était pas déjà présent dans la
121      * collection et qu'il a été ajouté
122      */
123     @Override
124     public boolean add(FigureFilter<T> filter)
125     {
126         if (!contains(filter))
127         {
128             return filters.add(filter);
129         }
130         else
131         {
132             return false;
133         }
134     }
135
136     /**
137      * Retrait d'un objet de la collection
138      * @param o l'objet à retirer de la collection
139      * @return true si l'objet a été retiré de la collection
140      */
141     @Override
142     public boolean remove(Object o)
143     {
144         return filters.remove(o);
145     }
146
147     /**
148      * Test si une collection est entièrement contenue dans la collection
149      * @param c la collection à tester
150      * @return true si la collection c est entièrement contenue dans la
151      * collection
152      */
153     @Override
154     public boolean containsAll(Collection<?> c)
155     {
156         return filters.containsAll(c);
157     }
158
159     /**
160      * Ajout d'une collection de {@link FigureFilters} à la collection courante
161      * @param c la collection de {@link FigureFilter} à ajouter
162      * @return true si au moins un élément de la collection c a été ajouté
163      * à la collection courante
164      */
165     @Override
166     public boolean addAll(Collection<? extends FigureFilter<T>> c)
167     {
168         boolean added = false;
169         for (FigureFilter<T> ff : c)
170         {
171             if (!contains(ff))
172             {
173                 added |= add(ff);
174             }
175         }
176
177         return added;
178     }
179
180     /**

```

avr 02, 17 16:17

## FigureFilters.java

Page 3/3

```

181  * Retrait de tous les éléments d'une collection de la collection courante
182  * @param c la collection à retirer de la collection courante
183  * @return true si la collection courante a été modifiée par cette opération
184  */
185  @Override
186  public boolean removeAll(Collection<?> c)
187  {
188      return filters.removeAll(c);
189  }
190
191  /**
192   * Conservation dans la collection courante uniquement des éléments présents
193   * dans la collection c
194   * @param c la collection qui détermine les éléments à conserver dans la
195   * collection courante
196   * @return true si la collection courante a été modifiée par cette opération
197   */
198  @Override
199  public boolean retainAll(Collection<?> c)
200  {
201      boolean retained = filters.retainAll(c);
202
203      // remove doubles
204
205      return retained;
206  }
207
208  /**
209   * Effacement de la collection
210   */
211  @Override
212  public void clear()
213  {
214      filters.clear();
215  }
216
217  /**
218   * Représentation de la collection de filtres
219   * @return une chaîne de caractères représentant tous les filtres
220   */
221  @Override
222  public String toString()
223  {
224      StringBuilder sb = new StringBuilder();
225
226      sb.append(getClass().getSimpleName());
227      sb.append("[");
228      sb.append(filters.size());
229      sb.append("]\n");
230      for (FigureFilter<T> ff : filters)
231      {
232          sb.append(ff.toString() + "\n");
233      }
234
235      return sb.toString();
236  }
237
238
239 }

```

avr 02, 17 16:17

## FlyweightFactory.java

Page 1/2

```

1  package utils;
2
3  import java.util.HashMap;
4
5  /**
6   * Flyweight gérant les différents éléments utilisés dans la zone de dessin.
7   * Utilisable avec les {@link Paint} et avec les {@link BasicStroke} des figures
8   * Gère les éléments dans une HashMap<Integer, T> dont la clé correspond au
9   * hashCode de l'élément correspondant. Lorsque l'on demande un élément à la
10  * Factory, celui-ci le recherche dans sa table de hachage : Si l'élément n'est
11  * pas déjà présent dans la table de hachage il est ajouté, puis renvoyé. S'il
12  * est déjà présent dans la table de hachage il est directement renvoyé et celui
13  * demandé est alors destructible par le garbage collector.
14  */
15  * @author davidroussel
16  */
17  public class FlyweightFactory<T>
18  {
19      /**
20       * La table de hachage contenant les différentes paires <hashCode,elt> et
21       * dont les clés sont les hashCode des différents éléments.
22       */
23      protected HashMap<Integer, T> map;
24
25      /**
26       * Constructeur d'un FlyweightFactory.
27       * Initialise la {@link HashMap}
28       */
29      public FlyweightFactory()
30      {
31          map = new HashMap<Integer, T>();
32      }
33
34      /**
35       * Obtention d'un élément à partir son hashCode plutôt que par l'élément
36       * lui-même
37       * @param hash le hashCode de l'élément demandé
38       * @return l'élément correspondant au hashCode demandé ou bien null si aucun
39       * élément avec ce hashCode n'est contenu dans la factory
40       * @note cette méthode est nécessaire lorsque l'on veut stocker dans la
41       * factory des éléments qui ne réimplémentent pas la méthode hashCode.
42       * Auquel cas on fournit soi-même un code de hachage.
43       */
44      protected T get(int hash)
45      {
46          Integer key = Integer.valueOf(hash);
47          if (map.containsKey(key))
48          {
49              return map.get(key);
50          }
51
52          return null;
53      }
54
55      /**
56       * Ajout d'un élément à la factory en fournissant un hashCode particulier
57       * @param hash le hashCode voulu pour cet élément
58       * @param element l'élément à ajouter
59       * @return true si aucun élément avec ce hashCode n'était contenu dans la
60       * factory et que le couple hash/value a bien été ajouté à la factory
61       * @note cette méthode est nécessaire lorsque l'on veut stocker dans la
62       * factory des éléments qui ne réimplémentent pas la méthode hashCode.
63       * Auquel cas on fournit soi-même un code de hachage.
64       */
65      protected boolean put(int hash, T element)
66      {
67          Integer key = Integer.valueOf(hash);
68          if (!map.containsKey(key))
69          {
70              if (element != null)
71              {
72                  map.put(key, element);
73                  System.out.println("Added " + element
74                      + " to the flyweight factory which contains "
75                      + map.size() + " elements");
76                  return true;
77              }
78              else
79              {
79                  System.err.println("FlyweightFactory::put(...): null element");
80              }
81          }
82          return false;
83      }
84
85      /**
86       * Obtention d'un élément (nouveau ou pas) : Lorsque l'élément demandé est
87       * déjà présent dans la table on le renvoie directement sinon celui-ci est
88       * ajouté à la table avant d'être renvoyé
89       * @param element l'élément demandé (celui-ci pourra être détruit par le

```



avr 02, 17 16:17

## FlyweightFactory.java

Page 2/2

```

91  * garbage collector si il en existe déjà un équivalent dans la table
92  * @return l'élément demandé en provenance de la table
93  */
94  public T get(T element)
95  {
96      if (element != null)
97      {
98          int hash = element.hashCode();
99          T result = get(hash);
100          if (result == null)
101          {
102              put(hash, element);
103              result = get(hash);
104          }
105          return result;
106      }
107      return null;
108  }
109
110  /**
111   * Nettoyage de tous les éléments
112   */
113  public void clear()
114  {
115      map.clear();
116  }
117
118  /**
119   * Nettoyage avant destruction de la factory
120   */
121  @Override
122  protected void finalize()
123  {
124      clear();
125  }
126  }

```

avr 02, 17 16:17

## IconFactory.java

Page 1/1

```

1  package utils;
2
3  import java.net.URL;
4
5  import javax.swing.ImageIcon;
6
7  /**
8   * Classe contenant une FlyweightFactory pour les les icônes. afin de pouvoir
9   * réutiliser une même icône (chargée à partir d'un fichier image contenu dans
10   * le package "images") à plusieurs endroits de l'interface graphique.
11   * @author davidroussel
12   */
13  public class IconFactory
14  {
15      /**
16       * le répertoire de base pour chercher les images
17       */
18      private final static String ImageBase = "/images/";
19
20      /**
21       * L'extension par défaut pour chercher les fichiers images
22       */
23      private final static String ImageType = ".png";
24
25      /**
26       * La factory stockant et fournissant les icônes
27       */
28      static private FlyweightFactory<ImageIcon> iconFactory =
29          new FlyweightFactory<ImageIcon>();
30
31      /**
32       * Méthode d'obtention d'une icône pour un nom donné
33       * @param name le nom de l'icône que l'on recherche
34       * @return l'icône correspondant au nom demandé si un fichier avec ce nom
35       * est trouvé dans le package/répertoire "images" ou bien null si aucune
36       * image correspondant à ce nom n'est trouvée.
37       */
38      static public ImageIcon getIcon(String name)
39      {
40          // checks if there is an icon with this name in the "images" directory
41          if (name.length() > 0)
42          {
43              int hash = name.hashCode();
44              ImageIcon icon = iconFactory.get(hash);
45              if (icon == null)
46              {
47                  URL url = IconFactory.class.getResource(ImageBase + name + ImageType);
48                  if (url != null)
49                  {
50                      icon = new ImageIcon(url);
51                      if (icon != null ^
52                          icon.getImageLoadStatus() == java.awt.MediaTracker.COMPLETE)
53                      {
54                          icon.setDescription(name);
55                          iconFactory.put(hash, icon);
56                      }
57                  }
58                  else
59                  {
60                      System.err.println("IconFactory::getIcon(" + name
61                          + "): could'nt find file " + ImageBase + name
62                          + ImageType);
63                  }
64              }
65              return iconFactory.get(hash);
66          }
67          else
68          {
69              return icon;
70          }
71      }
72      else
73      {
74          System.err.println("IconFactory::getIcon(<EMPTY NAME>");
75      }
76
77      return null;
78  }
79  }

```

avr 02, 17 16:17

IconItem.java

Page 1/1

```

1 package utils;
2 import javax.swing.ImageIcon;
3
4 /**
5  * Class defining an item Name associated to an Icon
6  * @author davidroussel
7  */
8
9 public class IconItem
10 {
11     /**
12      * Combobox item name
13      */
14     private String caption;
15
16     /**
17      * Combobox item icon
18      * @note typically reflects the item name in a file named <caption>.png
19      */
20     private ImageIcon icon;
21
22     /**
23      * Constructor from caption only
24      * @param caption the caption of this item
25      */
26     public IconItem(String caption)
27     {
28         this.caption = caption;
29         icon = IconFactory.getIcon(caption);
30         if (icon == null)
31         {
32             System.err.println("IconItem(" + caption
33                               + "): could not find corresponding icon");
34         }
35     }
36
37     /**
38      * Caption accessor
39      * @return the caption of this item
40      */
41     public String getCaption()
42     {
43         return caption;
44     }
45
46     /**
47      * Icon accessor
48      * @return the icon of this item
49      */
50     public ImageIcon getIcon()
51     {
52         return icon;
53     }
54 }

```

avr 02, 17 16:17

package-info.java

Page 1/1

```

1 /**
2  * Package contenant les différents widgets (éléments graphiques)
3  */
4 package widgets;

```

avr 02, 17 16:17

## PaintFactory.java

Page 1/2

```

1 package utils;
2
3 import java.awt.Color;
4 import java.awt.Component;
5 import java.awt.Paint;
6 import java.util.HashMap;
7 import java.util.Map;
8
9 import javax.swing.JColorChooser;
10
11 /**
12  * Classe contenant une FlyweightFactory pour les {@link Paint} afin de pouvoir
13  * réutiliser un même {@link Paint} à plusieurs endroits du programme
14  * @author davidroussel
15  */
16 public class PaintFactory
17 {
18     /**
19      * Map associant des noms de couleurs standard à des {@link Paint} standards
20      */
21     private static final Map<String, Paint> standardPaints = fillStandardPaints();
22
23     /**
24      * Construction de la map des {@link Paint} standards
25      * @return une map contenant les {@link Paint} standards
26      */
27     private static Map<String, Paint> fillStandardPaints()
28     {
29         Map<String, Paint> map = new HashMap<String, Paint>();
30         map.put("Black", Color.black);
31         map.put("Blue", Color.blue);
32         map.put("Cyan", Color.cyan);
33         map.put("Green", Color.green);
34         map.put("Magenta", Color.magenta);
35         map.put("None", null);
36         map.put("Orange", Color.orange);
37         map.put("Pink", Color.pink);
38         map.put("Red", Color.red);
39         map.put("White", Color.white);
40         map.put("Yellow", Color.yellow);
41
42         return map;
43     }
44
45     /**
46      * Flyweight factory stockant tous les {@link Paint} déjà requis
47      */
48     private static FlyweightFactory<Paint> paintFactory =
49         new FlyweightFactory<Paint>();
50
51     /**
52      * Obtention d'un {@link Paint} de la factory
53      * @param paint le paint recherché
54      * @return le paint recherché extrait de la factory
55      */
56     public static Paint getPaint(Paint paint)
57     {
58         if (paint != null)
59         {
60             return paintFactory.get(paint);
61         }
62
63         return null;
64     }
65
66     /**
67      * Obtention d'un paint de la factory par son nom en le recherchant dans les
68      * {@link #standardPaints}
69      * @param paintName le nom de la couleur requise
70      * @return le paint recherché extrait de la factory
71      */
72     public static Paint getPaint(String paintName)
73     {
74         if (paintName.length() > 0)
75         {
76             if (standardPaints.containsKey(paintName))
77             {
78                 return paintFactory.get(standardPaints.get(paintName));
79             }
80
81             return null;
82         }
83     }
84
85     /**
86      * Obtention d'un paint de la factory en déclenchant une boîte de dialogue
87      * de choix d'une couleur
88      * @param component le composant AWT à l'origine de la boîte de dialogue
89      * @param title le titre de la boîte de dialogue
90      * @param initialColor la couleur initiale de la boîte de dialogue de choix

```

avr 02, 17 16:17

## PaintFactory.java

Page 2/2

```

91     * de couleurs
92     * @return
93     */
94     public static Paint getPaint(Component component,
95                                 String title,
96                                 Color initialColor)
97     {
98         if (component != null)
99         {
100             Color color = JColorChooser.showDialog(component, title, initialColor);
101             if (color != null)
102             {
103                 return paintFactory.get(color);
104             }
105         }
106
107         return null;
108     }
109 }

```

avr 02, 17 16:17

## StrokeFactory.java

Page 1/1

```

1 package utils;
2
3 import java.awt.BasicStroke;
4
5 import figures.enums.LineType;
6
7 /**
8  * Classe contenant une FlyweightFactory pour les {@link BasicStroke} afin de pouvoir
9  * réutiliser un même {@link BasicStroke} à plusieurs endroits du programme
10  * @author davidroussel
11  */
12 public class StrokeFactory
13 {
14     /**
15      * Flyweight factory stockant tous les {@link BasicStroke} déjà requis
16      */
17     private static FlyweightFactory<BasicStroke> strokeFactory =
18         new FlyweightFactory<BasicStroke>();
19
20     /**
21      * Obtention d'un {@link BasicStroke} de la factory
22      * @param stroke le point recherché
23      * @return le stroke recherché
24      */
25     public static BasicStroke getStroke(BasicStroke stroke)
26     {
27         if (stroke != null)
28         {
29             return strokeFactory.get(stroke);
30         }
31
32         return null;
33     }
34
35     /**
36      * Obtention d'un {@link BasicStroke} à partir d'un type de trait et
37      * d'une épaisseur de trait
38      * @param type le type de trait (NONE, SOLID ou DASHED)
39      * @param width l'épaisseur du trait
40      * @return une {@link BasicStroke} correspondant au type et à l'épaisseur
41      * de trait en provenance de la factory
42      */
43     public static BasicStroke getStroke(LineType type, float width)
44     {
45         switch (type)
46         {
47             default:
48                 return null;
49             case SOLID:
50                 return getStroke(new BasicStroke(width,
51                     BasicStroke.CAP_ROUND,
52                     BasicStroke.JOIN_ROUND));
53             case DASHED:
54                 final float dash1[] = { 2 * width };
55                 return getStroke(new BasicStroke(width,
56                     BasicStroke.CAP_ROUND,
57                     BasicStroke.JOIN_ROUND,
58                     width, dash1, 0.0f));
59         }
60     }
61 }

```

avr 02, 17 16:17

## Vector2D.java

Page 1/3

```

1 package utils;
2
3 import java.awt.geom.AffineTransform;
4 import java.awt.geom.Point2D;
5
6 /**
7  * Vector 2D class relating two points
8  * @author davidroussel
9  */
10 public class Vector2D
11 {
12     /**
13      * Vector's origin
14      */
15     protected Point2D start;
16
17     /**
18      * Vector's end
19      */
20     protected Point2D end;
21
22     /**
23      * Constructor from 1 point (the origin is supposed to be (0, 0))
24      * @param p the end point
25      */
26     public Vector2D(Point2D p)
27     {
28         this(null, p);
29     }
30
31     /**
32      * Constructor from two points
33      * @param p1 the start point
34      * @param p2 the end point
35      */
36     public Vector2D(Point2D p1, Point2D p2)
37     {
38         start = p1;
39         end = p2;
40     }
41
42     /**
43      * Constructeur de copie
44      * @param vector le vecteur à copier
45      */
46     public Vector2D(Vector2D vector)
47     {
48         start = vector.start;
49         end = vector.end;
50     }
51
52     /**
53      * Start point getter
54      * @return the start
55      */
56     public Point2D getStart()
57     {
58         if (start == null)
59         {
60             return new Point2D.Double(0.0, 0.0);
61         }
62         else
63         {
64             return start;
65         }
66     }
67
68     /**
69      * Start point setter
70      * @param start the start to set
71      */
72     public void setStart(Point2D start)
73     {
74         this.start = start;
75     }
76
77     /**
78      * End Point getter
79      * @return the end
80      */
81     public Point2D getEnd()
82     {
83         return end;
84     }
85
86     /**
87      * End point setter
88      * @param end the end to set
89      */
90     public void setEnd(Point2D end)

```

avr 02, 17 16:17

## Vector2D.java

Page 2/3

```

91 {
92     this.end = end;
93 }
94
95 /**
96  * Delta X of the vector
97  * @return The delta X of the vector
98  */
99 protected double getX()
100 {
101     return end.getX() - (start == null ? 0.0 : start.getX());
102 }
103
104 /**
105  * Delta Y of the vector
106  * @return The delta Y of the vector
107  */
108 protected double getY()
109 {
110     return end.getY() - (start == null ? 0.0 : start.getY());
111 }
112
113 /**
114  * Dot product with vector v
115  * @param v the vector to compute dot product with
116  * @return the value of the dot product
117  */
118 public double dotProduct(Vector2D v)
119 {
120     return ((start == null ? 0.0 : start.getX()) * end.getX()) +
121           ((start == null ? 0.0 : start.getY()) * end.getY());
122 }
123
124 /**
125  * Cross product's norm
126  * @param v the vector to compute cross product's norm
127  * @return the value of the cross product's norm
128  */
129 public double crossProductNorm(Vector2D v)
130 {
131     return (getX()*v.getY()) - (v.getX()*getY());
132 }
133
134 /**
135  * Vector's norm
136  * @return the vector's norm
137  */
138 public double norm()
139 {
140     return Math.sqrt(dotProduct(this));
141 }
142
143 /**
144  * Compute normalized vector's
145  * @return normalized vector
146  */
147 public Vector2D normalize()
148 {
149     double norm = norm();
150
151     return new Vector2D(new Point2D.Double(getX()/ norm, getY() / norm));
152 }
153
154 /**
155  * Angle between vectors
156  * @param v the vector to compute angle with
157  * @return the angle between current vector and vector v
158  */
159 public double angle(Vector2D v)
160 {
161     Vector2D vn1 = normalize();
162     Vector2D vn2 = v.normalize();
163
164     return Math.atan2(vn2.getY(),vn2.getX()) -
165           Math.atan2(vn1.getY(),vn1.getX());
166 }
167
168 /**
169  * The endPoint as in {@link #end} - {@link #start}
170  * @return the end point
171  */
172 public Point2D toPoint2D()
173 {
174     return new Point2D.Double(end.getX() - start.getX(),
175                               end.getY() - start.getY());
176 }
177
178 /**
179  * Apply Affine transform to vector centered on {@link #start}
180  * @param transform the affine transform to apply

```

avr 02, 17 16:17

## Vector2D.java

Page 3/3

```

181 */
182 public void transformEnd(AffineTransform transform)
183 {
184     Point2D pVector = toPoint2D();
185     if (transform != null)
186     {
187         Point2D tPVector = new Point2D.Double();
188         transform.transform(pVector, tPVector);
189
190         setEnd(new Point2D.Double(start.getX() + tPVector.getX(),
191                                   start.getY() + tPVector.getY()));
192     }
193 }
194 }

```

avr 02, 17 16:17

## DrawingPanel.java

Page 1/6

```

1 package widgets;
2
3 import java.awt.Color;
4 import java.awt.Cursor;
5 import java.awt.Dimension;
6 import java.awt.Graphics;
7 import java.awt.Graphics2D;
8 import java.awt.Point;
9 import java.awt.RenderingHints;
10 import java.awt.event.ComponentAdapter;
11 import java.awt.event.ComponentEvent;
12 import java.awt.event.MouseEvent;
13 import java.awt.event.MouseListener;
14 import java.awt.event.MouseMotionListener;
15 import java.awt.geom.Point2D;
16 import java.text.DecimalFormat;
17 import java.util.Observable;
18 import java.util.Observer;
19
20 import javax.swing.JLabel;
21 import javax.swing.JPanel;
22
23 import figures.Drawing;
24 import figures.Figure;
25 import figures.listeners.AbstractFigureListener;
26 import figures.listeners.creation.AbstractCreationListener;
27
28 /**
29  * Panel de dessin des figures (Vue): mis à jour par modèle des figures (
30  * {@link Drawing}) au travers d'un observateur. On attache des Listeners
31  * (Contrôleurs) à ce Panel pour :
32  * <dl>
33  * <dt>Attachements statiques :</dt>
34  * <dd>Mettre à jour les coordonnées du pointeur de la souris dans la barre
35  * d'état : {@link #coordLabel}</dd>
36  * <dd>Mettre à jour le panneau d'informations relatif aux figures située sous
37  * le pointeur de la souris : {@link #infoPanel}</dd>
38  * <dt>Attachements dynamiques :</dt>
39  * <dd>Pour chaque type de figure à créer on attache un
40  * {@link AbstractCreationListener} ou plus exactement un de ses descendants
41  * pour traduire les événements souris en instructions pour le modèle de dessin
42  * lors de la création d'une nouvelle figure.
43  * </dl>
44  *
45  * @author davidroussel
46  */
47 public class DrawingPanel extends JPanel implements Observer, MouseListener,
48     MouseMotionListener
49 {
50     /**
51      * Taille effective du panel. Ce panel n'ayant pas de Layout Manager. il est
52      * important de conserver une taille effective qui puisse être renvoyée dans
53      * la méthode {@link #getPreferredSize()} et modifiée par un
54      * {@link java.awt.event.ComponentListener} tel que le
55      * {@link ResizeListener} ci-dessous.
56      */
57     protected Dimension size;
58
59     /**
60      * Contrôleur de changement de taille afin de mettre à jour
61      * {@link DrawingPanel#size} utilisé dans
62      * {@link DrawingPanel#getPreferredSize()}.
63      *
64      * @author davidroussel
65      */
66     protected class ResizeListener extends ComponentAdapter
67     {
68         /**
69          * Action à réaliser lorsque le composant change de taille
70          */
71         @Override
72         public void componentResized(ComponentEvent e)
73         {
74             size = e.getComponent().getSize();
75         }
76     }
77
78     /**
79      * Le modèle (les figures) à dessiner
80      */
81     private Drawing drawingModel;
82
83     /**
84      * Le label (à part dans la GUI) dans lequel afficher les coordonnées du
85      * pointeur de la souris
86      */
87     private JLabel coordLabel;
88
89     /**
90      * L' {@link InfoPanel} dans lequel afficher les informations à propos de

```

avr 02, 17 16:17

## DrawingPanel.java

Page 2/6

```

91  * la figure sous le curseur.
92  */
93  private InfoPanel infoPanel;
94
95  /**
96  * Chaîne de caractère à afficher par défaut dans le {@link #coordLabel}
97  */
98  public final static String defaultCoordString = new String("x: ____ y: ____");
99
100  /**
101  * Le formatteur à utiliser pour formater les nombres dans le
102  * {@link #coordLabel} et dans l' {@link #infoPanel}
103  */
104  private final static DecimalFormat coordFormat = new DecimalFormat("000");
105
106  /**
107  * état indiquant s'il faut envoyer les coordonnées de la souris ou la
108  * figure au dessus de laquelle se trouve la souris. Lorsque le curseur sort
109  * du widget (mouseExited) on cesse d'envoyer les coordonnées de la souris
110  * et lorsqu'elle entre (mouseEntered) on recommence à envoyer les
111  * coordonnées de la souris.
112  */
113  private boolean sendInfoState;
114
115  /**
116  * Constructeur de la zone de dessin à partir d'un modèle de dessin.
117  *
118  * @param drawing le modèle de dessin
119  * @param coordLabel le label à mettre à jour avec les coordonnées du
120  * curseur de la souris
121  * @param infoPanel le panneau d'information des figures à mettre à jour
122  * avec les informations relative à la figure située sous le
123  * curseur de la souris
124  */
125  public DrawingPanel(Drawing drawing, JLabel coordLabel, InfoPanel infoPanel)
126  {
127      setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
128      size = new Dimension(800, 600);
129      setPreferredSize(size);
130      addComponentListener(new ResizeListener());
131
132      setBackground(Color.WHITE);
133      setLayout(null);
134      setDoubleBuffered(true);
135
136      drawingModel = drawing;
137      if (drawing != null)
138      {
139          drawingModel.addObserver(this);
140      }
141      else
142      {
143          System.err.println("DrawingPanel caution: null drawing");
144      }
145
146      this.coordLabel = coordLabel;
147
148      if (this.coordLabel != null)
149      {
150          this.coordLabel.setText(defaultCoordString);
151      }
152      else
153      {
154          System.err.println("DrawingPanel : null coordLabel");
155      }
156
157      this.infoPanel = infoPanel;
158
159      if (this.infoPanel != null)
160      {
161          this.infoPanel.resetLabels();
162      }
163      else
164      {
165          System.err.println("DrawingPanel : null infoPanel");
166      }
167
168      // DrawingPanel est son propre listener d'événements souris
169      addMouseListener(this);
170      addMouseMotionListener(this);
171  }
172
173  @Override
174  protected void finalize() throws Throwable
175  {
176      drawingModel.deleteObserver(this);
177      super.finalize();
178  }
179
180  /**

```

avr 02, 17 16:17

## DrawingPanel.java

Page 3/6

```

181  * Accès à la taille effective du panel qui peut changer si celui-ci est
182  * agrandi (avec la fenêtre dans lequel il est par exemple). Cette méthode
183  * permet d'ajuster les scrollbars d'un container qui contiendrait ce panel
184  * lorsque la taille de celui-ci change.
185  *
186  * @return la taille effective du panel de dessin
187  * @see javax.swing.JComponent#getPreferredSize()
188  */
189  @Override
190  public Dimension getPreferredSize()
191  {
192      return size;
193  }
194
195  /**
196  * Mise en place du modèle de dessin. Met en place un nouveau modèle et s'il
197  * est non null ajoute ce panel comme observateur du modèle
198  *
199  * @param drawing le modèle de dessin à mettre en place
200  */
201  public void setDrawing(Drawing drawing)
202  {
203      // retrait du précédent modèle de dessin (s'il existe)
204      if (drawingModel != null)
205      {
206          drawing.deleteObserver(this);
207      }
208
209      // Mise en place du nouveau modèle de dessin
210      drawingModel = drawing;
211      if (drawingModel != null)
212      {
213          drawingModel.addObserver(this);
214      }
215  }
216
217  /**
218  * Mise en place du label dans lequel afficher les coordonnées du pointeur
219  * de la souris.
220  *
221  * @param coordLabel le label dans lequel afficher les coordonnées du
222  * pointeur de la souris.
223  */
224  public void setCoordLabel(JLabel coordLabel)
225  {
226      this.coordLabel = coordLabel;
227  }
228
229  /**
230  * Mise en place du panel d'information dans lequel afficher les infos sur
231  * la figure située sous le curseur
232  *
233  * @param infoPanel l'@link InfoPanel à mettre en place
234  */
235  public void setInfoPanel(InfoPanel infoPanel)
236  {
237      this.infoPanel = infoPanel;
238  }
239
240  /**
241  * Dessin du panel. Effacement de celui-ci puis dessin des figures.
242  * @param g le contexte graphique
243  * @see javax.swing.JComponent#paintComponent(java.awt.Graphics)
244  */
245  @Override
246  protected void paintComponent(Graphics g)
247  {
248      super.paintComponent(g); // Inutile
249
250      // caractéristiques graphiques : mise en place de l'antialiasing
251      Graphics2D g2D = (Graphics2D) g;
252      g2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
253          RenderingHints.VALUE_ANTIALIAS_ON);
254
255      // taille de la zone de dessin
256      Dimension d = getSize();
257      // on commence par effacer le fond
258      g2D.setColor(getBackground());
259      g2D.fillRect(0, 0, d.width, d.height);
260
261      // Puis on dessine l'ensemble des figures
262      if (drawingModel != null)
263      {
264          /*
265           * Application d'un Consumer<Figure> en tant que lambda expression
266           * sur le flux (éventuellement filtré) des figures permettant
267           * de dessiner les figures
268           */
269          drawingModel.stream().forEach((Figure f) -> f.draw(g2D));
270      }

```

avr 02, 17 16:17

## DrawingPanel.java

Page 4/6

```

271  /*
272  * Soulignement des figures sélectionnées (s'il y en a).
273  * Le soulignement est séparé du dessin des figures elles mêmes
274  * de manière à apparaître par dessus les figures dessinées
275  */
276  if (drawingModel.hasSelection())
277  {
278      drawingModel.stream().forEach((Figure f) -> f.drawSelection(g2D));
279  }
280
281  else
282  {
283      System.err.println(getClass().getSimpleName() + "::paintComponent : null model");
284  }
285
286  /**
287  * Mise en place d'un nouveau listener de figure
288  *
289  * @param fl le nouveau listener
290  */
291  public void addFigureListener(AbstractFigureListener fl)
292  {
293      if (fl != null)
294      {
295          addMouseListener(fl);
296          addMouseMotionListener(fl);
297          // System.out.println("CreationListener " + cl + " added");
298      }
299      else
300      {
301          System.err.println("DrawingPanel.addFigureListener(null)");
302      }
303  }
304
305  /**
306  * Retrait d'un listener de figure
307  *
308  * @param fl le creationListener à retirer
309  */
310  public void removeFigureListener(AbstractFigureListener fl)
311  {
312      if (fl != null)
313      {
314          removeMouseListener(fl);
315          removeMouseMotionListener(fl);
316          // System.out.println("CreationListener " + cl + " removed");
317      }
318  }
319
320  /**
321  * Mise à jour déclenchée par un @link Observable#notifyObservers() : en
322  * l'occurrence le modèle de dessin (@link Drawing) lorsque celui ci est
323  * modifié. Cette mise à jour déclenche une requête de redessin du panel.
324  *
325  * @param observable l'observable avant déclenché cette MAJ
326  * @param data les données (evt) transmises par l'observable (non utilisé ici)
327  * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
328  */
329  @Override
330  public void update(Observable observable, Object data)
331  {
332      if (observable instanceof Drawing)
333      {
334          // Le modèle à changé il faut redessiner les figures
335          repaint();
336      }
337  }
338
339  /**
340  * Rafraichissement des panneaux d'information lors du déplacement de la
341  * souris
342  *
343  * @param e l'évènement souris associé
344  */
345  @Override
346  public void mouseDragged(MouseEvent e)
347  {
348      // Déplacement de la souris (btn enfoncé) : MAJ des coordonnées
349      // de la souris dans le coordLabel et infoPanel
350      refreshCoordLabel(e.getPoint());
351      refreshInfoPanel(e.getPoint());
352  }
353
354  /**
355  * Rafraichissement des panneaux d'information lors du déplacement (bouton
356  * enfoncé) de la souris
357  *
358  * @param e l'évènement souris associé
359  */
360

```

avr 02, 17 16:17

## DrawingPanel.java

Page 5/6

```

361 @Override
362 public void mouseMoved(MouseEvent e)
363 {
364     // Déplacement de la souris : MAJ des coordonnées
365     // de la souris dans le coordLabel et infoPanel
366     Point p = e.getPoint();
367     refreshCoordLabel(p);
368     refreshInfoPanel(p);
369 }
370
371 @Override
372 public void mouseClicked(MouseEvent e)
373 {
374     // Rien
375 }
376
377 /**
378  * Reprise du rafraichissement des panneaux d'information lorsque la souris
379  * rentre dans ce panel.
380  *
381  * @param e l'évènement souris associé
382  */
383 @Override
384 public void mouseEntered(MouseEvent e)
385 {
386     sendInfoState = true;
387     refreshCoordLabel(e.getPoint());
388     refreshInfoPanel(e.getPoint());
389 }
390
391 /**
392  * Arrêt du rafraichissement des panneaux d'information et effacement de ces
393  * panneaux lorsque la souris sort du panel.
394  *
395  * @param e l'évènement souris associé
396  */
397 @Override
398 public void mouseExited(MouseEvent e)
399 {
400     // Rien si ce n'est de remettre les coordonnées dans la barre d'état
401     // à x = ____ y = ____
402     sendInfoState = false;
403     refreshCoordLabel(e.getPoint());
404     infoPanel.resetLabels();
405 }
406
407 @Override
408 public void mousePressed(MouseEvent e)
409 {
410     // Rien
411 }
412
413 @Override
414 public void mouseReleased(MouseEvent e)
415 {
416     // Rien
417 }
418
419 /**
420  * Rafraichissement du {@link #coordLabel} (s'il est non null) avec de
421  * nouvelles coordonnées ou bien avec la {@link #defaultCoordString} si l'on
422  * affiche pas les coordonnées
423  *
424  * @param x l'abscisse des coordonnées à afficher
425  * @param y l'ordonnée des coordonnées à afficher
426  */
427 private void refreshCoordLabel(Point p)
428 {
429     if ((coordLabel != null) ^ (p != null))
430     {
431         if (sendInfoState)
432         {
433             String xs = coordFormat.format(p.getX());
434             String ys = coordFormat.format(p.getY());
435             coordLabel.setText("x: " + xs + " y: " + ys);
436         }
437         else
438         {
439             coordLabel.setText(defaultCoordString);
440         }
441     }
442 }
443
444 /**
445  * Rafraichissement du panneau d'information {@link #infoPanel}
446  *
447  * @param p la position du curseur pour déclencher la recherche de figures
448  * sous ce curseur
449  */
450 private void refreshInfoPanel(Point2D p)

```

avr 02, 17 16:17

## DrawingPanel.java

Page 6/6

```

451 {
452     if ((infoPanel != null) ^ sendInfoState)
453     {
454         Figure selectedFigure = drawingModel.getFigureAt(p);
455
456         if (selectedFigure != null)
457         {
458             infoPanel.updateLabels(selectedFigure);
459         }
460         else
461         {
462             infoPanel.resetLabels();
463         }
464     }
465 }
466 }

```



avr 02, 17 16:17

## EditorFrame.java

Page 1/16

```

1 package widgets;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Component;
6 import java.awt.Dimension;
7 import java.awt.HeadlessException;
8 import java.awt.Paint;
9 import java.awt.Toolkit;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.InputEvent;
12 import java.awt.event.ItemEvent;
13 import java.awt.event.ItemListener;
14 import java.awt.event.KeyEvent;
15 import java.util.ArrayList;
16 import java.util.EventObject;
17 import java.util.List;
18
19 import javax.swing.AbstractAction;
20 import javax.swing.AbstractButton;
21 import javax.swing.Action;
22 import javax.swing.Box;
23 import javax.swing.BoxLayout;
24 import javax.swing.ImageIcon;
25 import javax.swing.JButton;
26 import javax.swing.JCheckBoxMenuItem;
27 import javax.swing.JColorChooser;
28 import javax.swing.JComboBox;
29 import javax.swing.JFrame;
30 import javax.swing.JLabel;
31 import javax.swing.JMenu;
32 import javax.swing.JMenuBar;
33 import javax.swing.JMenuItem;
34 import javax.swing.JOptionPane;
35 import javax.swing.JPanel;
36 import javax.swing.JScrollPane;
37 import javax.swing.JSeparator;
38 import javax.swing.JSpinner;
39 import javax.swing.JTabbedPane;
40 import javax.swing.JToolBar;
41 import javax.swing.KeyStroke;
42 import javax.swing.SpinnerNumberModel;
43 import javax.swing.SwingConstants;
44 import javax.swing.event.ChangeEvent;
45 import javax.swing.event.ChangeListener;
46
47 import figures.Drawing;
48 import figures.enums.FigureType;
49 import figures.enums.LineType;
50 import figures.enums.PaintToType;
51 import figures.listeners.AbstractFigureListener;
52 import figures.listeners.SelectionFigureListener;
53 import figures.listeners.creation.AbstractCreationListener;
54 import figures.listeners.transform.AbstractTransformShapeListener;
55 import figures.listeners.transform.MoveShapeListener;
56 import utils.IconFactory;
57 import utils.PaintFactory;
58 import widgets.enums.OperationMode;
59
60 /**
61  * Classe de la fenêtre principale de l'éditeur de figures
62  * @author davidroussel
63  */
64 @SuppressWarnings("serial")
65 public class EditorFrame extends JFrame
66 {
67     /**
68      * Le nom de l'éditeur
69      */
70     protected static final String EditorName = "Figure Editor v4.0";
71
72     /**
73      * Le modèle de dessin sous-jacent;
74      */
75     protected Drawing drawingModel;
76
77     /**
78      * Indique si l'éditeur est en mode Création de figures ou édition
79      * de figures (mode initial : création de figures)
80      */
81     protected OperationMode operationMode = OperationMode.CREATION;
82
83     /**
84      * La zone de dessin dans laquelle seront dessinées les figures.
85      * On a besoin d'une référence à la zone de dessin (contrairement aux
86      * autres widgets) car il faut lui affecter un xxxCreationListener en
87      * fonction de la figure choisie dans la liste des figures possibles.
88      */
89     protected DrawingPanel drawingPanel;
90

```

avr 02, 17 16:17

## EditorFrame.java

Page 2/16

```

91 /**
92  * Le creationListener à mettre en place dans le drawingPanel en fonction
93  * du type de figure choisie;
94  */
95 protected AbstractCreationListener creationListener;
96
97 /**
98  * Le listener à mettre en place dans le drawingPanel lorsque l'on
99  * est en mode édition de figures pour déplacer les figures sélectionnées
100  */
101 protected AbstractTransformShapeListener moveListener;
102
103 /**
104  * Le listener à mettre en place dans le drawingPanel lorsque l'on
105  * est en mode édition de figures pour faire tourner les figures
106  * sélectionnées
107  */
108 protected AbstractTransformShapeListener rotateListener;
109
110 /**
111  * Le listener à mettre en place dans le drawingPanel lorsque l'on
112  * est en mode édition de figures pour changer l'échelle des figures
113  * sélectionnées
114  */
115 protected AbstractTransformShapeListener scaleListener;
116
117 /**
118  * Le listener de sélection des figures à mettre en place lorsque l'on
119  * est en mode édition.
120  */
121 protected AbstractFigureListener selectionListener;
122
123 /**
124  * Le label dans la barre d'état en bas dans lequel on affiche les
125  * conseils utilisateur pour créer une figure
126  */
127 protected JLabel infoLabel;
128
129 /**
130  * L'index de l'élément sélectionné par défaut pour le type de figure
131  */
132 private final static int defaultFigureTypeIndex = 0;
133
134 /**
135  * Les noms des couleurs de remplissage à utiliser pour remplir
136  * la [labeled]combobox des couleurs de remplissage
137  */
138 protected final static String[] fillColorNames =
139 { "Black", "White", "Red", "Orange", "Yellow", "Green", "Cyan", "Blue",
140   "Magenta", "Others", "None" };
141
142 /**
143  * Les couleurs de remplissage à utiliser en fonction de l'élément
144  * sélectionné dans la [labeled]combobox des couleurs de remplissage
145  */
146 protected final static Paint[] fillPaints =
147 { Color.black, Color.white, Color.red, Color.orange, Color.yellow,
148   Color.green, Color.cyan, Color.blue, Color.magenta, null, // Color
149   // selected
150   // bv a
151   // JColorChooser
152   null // No Color
153 };
154
155 /**
156  * L'index de l'élément sélectionné par défaut dans les couleurs de
157  * remplissage
158  */
159 private final static int defaultFillColorIndex = 0; // black
160
161 /**
162  * L'index de la couleur de remplissage à choisir avec un
163  * {@link JColorChooser} fournit par la {@link PaintFactory}
164  */
165 private final static int specialFillColorIndex = 9;
166
167 /**
168  * Les noms des couleurs de trait à utiliser pour remplir
169  * la [labeled]combobox des couleurs de trait
170  */
171 protected final static String[] edgeColorNames = { "Magenta", "Red",
172   "Orange", "Yellow", "Green", "Cyan", "Blue", "Black", "Others" };
173
174 /**
175  * Les couleurs de trait à utiliser en fonction de l'élément
176  * sélectionné dans la [labeled]combobox des couleurs de trait
177  */
178 protected final static Paint[] edgePaints =
179 { Color.magenta, Color.red, Color.orange, Color.yellow, Color.green,
180   Color.cyan, Color.blue, Color.black, null // Color selected by a

```

avr 02, 17 16:17

## EditorFrame.java

Page 3/16

```

181         };
182         // JColorChooser
183
184     /**
185      * L'index de l'élément sélectionné par défaut dans les couleurs de
186      * trait
187      */
188     private final static int defaultEdgeColorIndex = 6; // blue;
189
190     /**
191      * L'index de la couleur de remplissage à choisir avec un
192      * {@link JColorChooser} fournit par la {@link PaintFactory}
193      */
194     private final static int specialEdgeColorIndex = 8;
195
196     /**
197      * L'index de l'élément sélectionné par défaut dans les types de
198      * trait
199      */
200     private final static int defaultEdgeTypeIndex = 1; // solid
201
202     /**
203      * La largeur de trait par défaut
204      */
205     private final static int defaultEdgeWidth = 4;
206
207     /**
208      * Largeur de trait minimum
209      */
210     private final static int minEdgeWidth = 1;
211
212     /**
213      * Largeur de trait maximum
214      */
215     private final static int maxEdgeWidth = 30;
216
217     /**
218      * l'incrément entre deux largeurs de trait
219      */
220     private final static int stepEdgeWidth = 1;
221
222     /**
223      * Action déclenchée lorsque l'on clique sur le bouton quit ou sur l'item
224      * de menu quit
225      */
226     private final Action quitAction = new QuitAction();
227
228     /**
229      * Action déclenchée lorsque l'on clique sur le bouton undo ou sur l'item
230      * de menu undo
231      */
232     private final Action undoAction = new UndoAction();
233
234     /**
235      * Action réalisée lorsque l'on souhaite refaire une action qui vient
236      * d'être annulée
237      */
238     private final Action redoAction = new RedoAction();
239
240     /**
241      * Action déclenchée lorsque l'on clique sur le bouton clear ou sur l'item
242      * de menu clear
243      */
244     private final Action clearAction = new ClearAction();
245
246     /**
247      * Action déclenchée lorsque l'on clique sur le bouton about ou sur l'item
248      * de menu about
249      */
250     private final Action aboutAction = new AboutAction();
251
252     /**
253      * Action déclenchée lorsque l'on sélectionne de mode édition des figures
254      */
255     private final Action toggleCreateEditAction = new ToggleCreateEditAction();
256
257     /**
258      * Action déclenchée pour mettre filter ou non les figures
259      */
260     private final Action filterAction = new FilterAction();
261
262     /**
263      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
264      * des cercles
265      */
266     private final Action circleFilterAction =
267         new ShapeFilterAction(FigureType.CIRCLE);
268
269     /**
270      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage

```

Dimanche avril 02, 2017

./711/EditorFrame.java

avr 02, 17 16:17

## EditorFrame.java

Page 4/16

```

271     * des ellipse
272     */
273     private final Action ellipseFilterAction =
274         new ShapeFilterAction(FigureType.ELLIPSE);
275
276     /**
277      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
278      * des rectangles
279      */
280     private final Action rectangleFilterAction =
281         new ShapeFilterAction(FigureType.RECTANGLE);
282
283     /**
284      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
285      * des rectangles arrondis
286      */
287     private final Action rRectangleFilterAction =
288         new ShapeFilterAction(FigureType.ROUNDED_RECTANGLE);
289
290     /**
291      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
292      * des polygones
293      */
294     private final Action polyFilterAction =
295         new ShapeFilterAction(FigureType.POLYGON);
296
297     /**
298      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
299      * des polygones réguliers
300      */
301     private final Action ngonFilterAction =
302         new ShapeFilterAction(FigureType.NGON);
303
304     /**
305      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
306      * des type de lignes vides
307      */
308     private final Action noneLineFilterAction =
309         new LineFilterAction(LineType.NONE);
310
311     /**
312      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
313      * des type de lignes pleines
314      */
315     private final Action solidLineFilterAction =
316         new LineFilterAction(LineType.SOLID);
317
318     /**
319      * Action déclenchée lorsque l'on clique sur l'item de menu de filtrage
320      * des type de lignes pointillées
321      */
322     private final Action dashedLineFilterAction =
323         new LineFilterAction(LineType.DASHED);
324
325     /**
326      * Action déclenchée pour mettre filter ou non les figures suivant
327      * la couleur de replissage courante
328      */
329     private final Action fillColorFilterAction = new FillColorFilterAction();
330
331     /**
332      * Action déclenchée pour mettre filter ou non les figures suivant
333      * la couleur de trait courante
334      */
335     private final Action edgeColorFilterAction = new EdgeColorFilterAction();
336
337     /**
338      * Action réalisée pour détruire les figures sélectionnées
339      */
340     private final Action deleteAction = new DeleteAction();
341
342     /**
343      * Action réalisée pour monter les figures sélectionnées en tête de liste
344      * des figures
345      */
346     private final Action moveUpAction = new MoveUpAction();
347
348     /**
349      * Action réalisée pour descendre les figures sélectionnées en fin de liste
350      * des figures
351      */
352     private final Action moveDownAction = new MoveDownAction();
353
354     /**
355      * Action réalisée pour appliquer le style courant (couleur de remplissage,
356      * couleur de trait et style de trait) aux figures sélectionnées
357      */
358     private final Action styleAction = new StyleAction();
359
360     /**

```

42/56

avr 02, 17 16:17

## EditorFrame.java

Page 5/16

```

361  * Constructeur de la fenêtre de l'éditeur.
362  * Construit les widgets et assigne les actions et autres listeners
363  * aux widgets.
364  * @throws HeadlessException
365  */
366  public EditorFrame() throws HeadlessException
367  {
368      drawingModel = new Drawing();
369
370      operationMode = OperationMode.CREATION;
371
372      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
373      boolean isMacOS = System.getProperty("os.name").startsWith("Mac OS");
374
375      /*
376       * Construire l'interface graphique en utilisant WindowBuilder:
377       * Menu Contextuel -> Open With -> WindowBuilder Editor puis
378       * aller dans l'onglet Design
379       */
380      setPreferredSize(new Dimension(650, 450));
381      creationListener = null;
382
383      setTitle(EditorName);
384      if (!isMacOS)
385      {
386          setIconImage(Toolkit.getDefaultToolkit().
387              .getImage(EditorFrame.class.getResource("/images/Logo.png")));
388      }
389
390      // -----
391      // Toolbar en haut
392      // -----
393      JToolBar toolBar = new JToolBar();
394      toolBar.setFloatable(false);
395      getContentPane().add(toolBar, BorderLayout.NORTH);
396
397      JButton btnCancel = new JButton("Undo");
398      btnCancel.setAction(undoAction);
399      toolBar.add(btnCancel);
400
401      JButton btnRedo = new JButton("Redo");
402      btnRedo.setAction(redoAction);
403      toolBar.add(btnRedo);
404
405      Component toolBoxSpringer = Box.createHorizontalGlue();
406      toolBar.add(toolBoxSpringer);
407
408      JButton btnAbout = new JButton("About");
409      btnAbout.setAction(aboutAction);
410      toolBar.add(btnAbout);
411
412      JButton btnClose = new JButton("Close");
413      btnClose.setAction(quitAction);
414      toolBar.add(btnClose);
415
416      // -----
417      // Barre d'état en bas
418      // -----
419      JPanel bottomPanel = new JPanel();
420      getContentPane().add(bottomPanel, BorderLayout.SOUTH);
421      bottomPanel.setLayout(new BoxLayout(bottomPanel, BoxLayout.X_AXIS));
422
423      infoLabel = new JLabel(AbstractFigureListener.defaultTip);
424      bottomPanel.add(infoLabel);
425
426      Component horizontalGlue = Box.createHorizontalGlue();
427      bottomPanel.add(horizontalGlue);
428
429      JLabel coordsLabel = new JLabel(DrawingPanel.defaultCoordString);
430      bottomPanel.add(coordsLabel);
431
432      // -----
433      // Panneau de contrôle à gauche
434      // -----
435      JPanel leftPanel = new JPanel();
436      leftPanel.setPreferredSize(new Dimension(220, 10));
437      leftPanel.setAlignmentY(Component.TOP_ALIGNMENT);
438      getContentPane().add(leftPanel, BorderLayout.WEST);
439
440      JListedComboBox figureTypeCombobox = new JListedComboBox("Shape",
441          FigureType
442              .stringValues(),
443          defaultFigureTypeIndex,
444          (ItemListener) null);
445
446      figureTypeCombobox.setAlignmentX(Component.CENTER_ALIGNMENT);
447      figureTypeCombobox.setPreferredSize(new Dimension(80, 32));
448      leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
449      leftPanel.add(figureTypeCombobox);
450
451      JPanel edgeWidthPanel = new JPanel();

```

avr 02, 17 16:17

## EditorFrame.java

Page 6/16

```

451      edgeWidthPanel.setPreferredSize(new Dimension(80, 32));
452      leftPanel.add(edgeWidthPanel);
453      edgeWidthPanel
454          .setLayout(new BoxLayout(edgeWidthPanel, BoxLayout.X_AXIS));
455      SpinnerNumberModel snm =
456          new SpinnerNumberModel(defaultEdgeWidth,
457              minEdgeWidth,
458              maxEdgeWidth,
459              stepEdgeWidth);
460
461      JTabbedPane tabbedPane = new JTabbedPane(SwingConstants.TOP);
462      tabbedPane.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
463      tabbedPane.setAlignmentY(Component.TOP_ALIGNMENT);
464      leftPanel.add(tabbedPane);
465
466      InfoPanel infoPanel = new InfoPanel();
467      infoPanel.setAlignmentY(Component.TOP_ALIGNMENT);
468      tabbedPane.addTab("Info", new ImageIcon(EditorFrame.class.getResource("/images/Details_small.png")),
469          infoPanel, "Selected Figure");
470
471      // -----
472      // Zone de dessin
473      // -----
474      JScrollPane scrollPane = new JScrollPane();
475      getContentPane().add(scrollPane, BorderLayout.CENTER);
476
477      drawingPanel = new DrawingPanel(drawingModel, coordsLabel, infoPanel);
478      scrollPane.setViewportView(drawingPanel);
479
480      // -----
481      // Barre de menus
482      // -----
483      JMenuBar menuBar = new JMenuBar();
484      setJMenuBar(menuBar);
485
486      JMenu mnFile = new JMenu("Drawing");
487      menuBar.add(mnFile);
488
489      JMenuItem mntmCancel = new JMenuItem("Cancel");
490      mntmCancel.setAction(undoAction);
491      mnFile.add(mntmCancel);
492
493      JMenuItem mntmRedo = new JMenuItem("Redo");
494      mntmRedo.setAction(redoAction);
495      mnFile.add(mntmRedo);
496
497      JMenuItem mntmClear = new JMenuItem("Clear");
498      mntmClear.setAction(clearAction);
499      mnFile.add(mntmClear);
500
501      JMenu mnEdition = new JMenu("Edition");
502      menuBar.add(mnEdition);
503
504      JMenu mnFilter = new JMenu("Filter");
505      menuBar.add(mnFilter);
506
507      JCheckBoxMenuItem chckbxmntmFiltering =
508          new JCheckBoxMenuItem("Filtering");
509      chckbxmntmFiltering.setAction(filterAction);
510      mnFilter.add(chckbxmntmFiltering);
511
512      JMenu mnFigures = new JMenu("Figures");
513      mnFilter.add(mnFigures);
514
515      JMenu mnColors = new JMenu("Colors");
516      mnFilter.add(mnColors);
517
518      JMenu mnStrokes = new JMenu("Strokes");
519      mnFilter.add(mnStrokes);
520
521      JSeparator separator = new JSeparator();
522      mnFile.add(separator);
523
524      JMenuItem mntmQuit = new JMenuItem("Quit");
525      mntmQuit.setAction(quitAction);
526      mnFile.add(mntmQuit);
527
528      JMenu mnHelp = new JMenu("Help");
529      menuBar.add(mnHelp);
530
531      JMenuItem mntmAbout = new JMenuItem("About...");
532      mntmAbout.setAction(aboutAction);
533      mnHelp.add(mntmAbout);
534
535      // -----
536      // Ajout des contrôleurs aux widgets
537      // pour connaître les Listeners applicable à un widget
538      // dans WindowBuilder, sélectionnez un widget de l'UI puis Menu
539      // Contextuel -> Add event handler
540      // -----

```

avr 02, 17 16:17

## EditorFrame.java

Page 7/16

```

540 moveListener = new MoveShapelListener(drawingModel, infoLabel);
541 scaleListener = null; // TODO new ScaleShapelListener(drawingModel, infoLabel);
542 rotateListener = null; // TODO new RotateShapelListener(drawingModel, infoLabel);
543 selectionListener = new SelectionFigureListener(drawingModel, infoLabel);
544
545 figureTypeCombobox.addItemListener(new ShapeItemListener(FigureType
546     .fromInteger(figureTypeCombobox.getSelectedIndex())));
547
548 /**
549  * Action pour quitter l'application
550  * @author davidroussel
551  */
552 private class QuitAction extends AbstractAction // implements QuitHandler
553 {
554     /**
555      * Constructeur de l'action pour quitter l'application.
556      * Met en place le raccourci clavier, l'icône et la description
557      * de l'action
558      */
559     public QuitAction()
560     {
561         putValue(NAME, "Quit");
562         /*
563          * Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()
564          * = InputEvent.CTRL_MASK on win/linux
565          * = InputEvent.META_MASK on mac os
566          */
567         putValue(ACCELERATOR_KEY,
568             KeyStroke.getKeyStroke(KeyEvent.VK_Q, Toolkit.getDefaultToolkit().getMenuShortc
569 utKeyMask()));
570         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Quit"));
571         putValue(SMALL_ICON, IconFactory.getIcon("Quit_small"));
572         putValue(SHORT_DESCRIPTION, "Quits the application");
573     }
574
575     /**
576      * Opérations réalisées par l'action
577      * @param e l'évènement déclenchant l'action. Peut provenir d'un bouton
578      * ou d'un item de menu
579      */
580     @Override
581     public void actionPerformed(ActionEvent e)
582     {
583         doQuit();
584     }
585
586     /**
587      * Action réalisée pour quitter dans un {@link Action}
588      */
589     private void doQuit()
590     {
591         /*
592          * Action à effectuer lorsque l'action "undo" est cliquée :
593          * sortir avec un System.exit() (pas très propre, mais fonctionne)
594          */
595         System.exit(0);
596     }
597
598 /**
599  * Action réalisée pour effacer la dernière action dans le dessin
600  */
601 private class UndoAction extends AbstractAction
602 {
603     /**
604      * Constructeur de l'action effacer la dernière action sur le dessin
605      * Met en place le raccourci clavier, l'icône et la description
606      * de l'action
607      */
608     public UndoAction()
609     {
610         putValue(NAME, "Undo");
611         putValue(ACCELERATOR_KEY,
612             KeyStroke.getKeyStroke(KeyEvent.VK_Z,
613                 Toolkit.getDefaultToolkit().
614                     getMenuShortcutKeyMask()));
615         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Undo"));
616         putValue(SMALL_ICON, IconFactory.getIcon("Undo_small"));
617         putValue(SHORT_DESCRIPTION, "Undo last drawing");
618     }
619
620     /**
621      * Opérations réalisées par l'action
622      * @param e l'évènement déclenchant l'action. Peut provenir d'un bouton
623      * ou d'un item de menu
624      */
625     @Override
626     public void actionPerformed(ActionEvent e)
627     {
628

```

avr 02, 17 16:17

## EditorFrame.java

Page 8/16

```

629 // TODO Compléter ...
630 }
631
632 /**
633  * Action réalisée pour refaire la dernière action (qui a été annulée)
634  * dans le dessin
635  */
636 private class RedoAction extends AbstractAction
637 {
638     public RedoAction()
639     {
640         putValue(NAME, "Redo");
641         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Redo"));
642         putValue(SMALL_ICON, IconFactory.getIcon("Redo_small"));
643         putValue(ACCELERATOR_KEY,
644             KeyStroke.getKeyStroke(KeyEvent.VK_Z,
645                 InputEvent.SHIFT_MASK
646                     | Toolkit.getDefaultToolkit().
647                         getMenuShortcutKeyMask()));
648         putValue(SHORT_DESCRIPTION, "Redo last drawing");
649     }
650
651     @Override
652     public void actionPerformed(ActionEvent e)
653     {
654         // TODO Compléter ...
655     }
656
657 /**
658  * Action réalisée pour effacer toutes les figures du dessin
659  */
660 private class ClearAction extends AbstractAction
661 {
662     /**
663      * Constructeur de l'action pour effacer toutes les figures du dessin
664      * Met en place le raccourci clavier, l'icône et la description
665      * de l'action
666      */
667     public ClearAction()
668     {
669         putValue(NAME, "Clear");
670         putValue(ACCELERATOR_KEY,
671             KeyStroke.getKeyStroke(KeyEvent.VK_X,
672                 Toolkit.getDefaultToolkit().
673                     getMenuShortcutKeyMask()));
674         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Clear"));
675         putValue(SMALL_ICON, IconFactory.getIcon("Clear_small"));
676         putValue(SHORT_DESCRIPTION, "Erase all drawings");
677     }
678
679     /**
680      * Opérations réalisées par l'action
681      * @param e l'évènement déclenchant l'action. Peut provenir d'un bouton
682      * ou d'un item de menu
683      */
684     @Override
685     public void actionPerformed(ActionEvent e)
686     {
687         /*
688          * Action à effectuer lorsque l'action "clear" est cliquée :
689          * Effacer toutes les figures du dessin
690          */
691         // TODO Compléter ...
692     }
693
694 /**
695  * Action réalisée pour afficher la boîte de dialogue "A propos ..."
696  */
697 private class AboutAction extends AbstractAction // implements AboutHandler
698 {
699     /**
700      * Constructeur de l'action pour afficher la boîte de dialogue
701      * "A propos ..." Met en place le raccourci clavier, l'icône et la
702      * description de l'action
703      */
704     public AboutAction()
705     {
706         putValue(ACCELERATOR_KEY,
707             KeyStroke.getKeyStroke(KeyEvent.VK_I,
708                 Toolkit.getDefaultToolkit().
709                     getMenuShortcutKeyMask()));
710         putValue(LARGE_ICON_KEY, IconFactory.getIcon("About"));
711         putValue(SMALL_ICON, IconFactory.getIcon("About_small"));
712         putValue(NAME, "About");
713         putValue(SHORT_DESCRIPTION, "App information");
714     }
715
716 }
717
718

```

avr 02, 17 16:17

## EditorFrame.java

Page 9/16

```

719 /**
720  * Opérations réalisées par l'action
721  * @param e l'évènement déclenchant l'action. Peut provenir d'un bouton
722  * ou d'un item de menu
723  */
724 @Override
725 public void actionPerformed(ActionEvent e)
726 {
727     doAbout(e);
728 }
729
730 /**
731  * Action réalisée pour "A propos" dans un {@link Action}
732  * @param e l'évènement ayant déclenché l'action
733  */
734 private void doAbout(EventObject e)
735 {
736     /**
737      * Action à effectuer lorsque l'action "about" est cliquée :
738      * Ouvrir un MessageDialog (JOptionPane.showMessageDialog(...)) de
739      * type JOptionPane.INFORMATION_MESSAGE
740      */
741     Object source = e.getSource();
742     Component component =
743         source instanceof Component ? (Component) source : null;
744     JOptionPane.showMessageDialog(component,
745                                 EditorName,
746                                 "About ...",
747                                 JOptionPane.INFORMATION_MESSAGE);
748 }
749
750 /**
751  * Action réalisée lorsque l'on passe en mode édition des figures
752  */
753 private class ToggleCreateEditAction extends AbstractAction
754 {
755     /**
756      * Liste des "boutons" pouvant déclencher cette action.
757      * De manière à ce que lorsqu'un bouton déclenche l'action
758      * les autres boutons soient eux aussi mis dans l'état correspondant
759      * à l'action
760      */
761     private List<AbstractButton> buttons;
762
763     /**
764      * Constructeur de l'action pour mettre en place ou enlever un filtre
765      * pour filtrer les types de figures
766      */
767     public ToggleCreateEditAction()
768     {
769         putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_TAB, InputEvent.ALT_MASK));
770         putValue(NAME, "Edition");
771         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Edition"));
772         putValue(SMALL_ICON, IconFactory.getIcon("Edition_small"));
773         putValue(SHORT_DESCRIPTION, "Edition des figures");
774
775         buttons = new ArrayList<AbstractButton>();
776     }
777
778     /**
779      * Ajout d'un bouton déclenchant cette action
780      * @param button le bouton à ajouter à la liste des boutons
781      * @return true si le bouton a été ajouté à la liste des boutons
782      * déclenchant cette action. false si le bouton était déjà présent
783      * dans la liste des actions et n'a pas été ajouté
784      */
785     public boolean registerButton(AbstractButton button)
786     {
787         if (!buttons.contains(button))
788         {
789             return buttons.add(button);
790         }
791         return false;
792     }
793
794     /**
795      * Opérations réalisées par l'action
796      * @param event l'évènement déclenchant l'action. Peut provenir d'un
797      * bouton ou d'un item de menu
798      */
799     @Override
800     public void actionPerformed(ActionEvent event)
801     {
802         AbstractButton button = (AbstractButton) event.getSource();
803         boolean selected = button.getModel().isSelected();
804
805         /**
806          * TODO Parcourir tous les "boutons" pour s'assurer qu'ils sont
807          * bien dans l'état voulu
808          */
809     }
810 }

```

avr 02, 17 16:17

## EditorFrame.java

Page 10/16

```

809 /**
810  *
811  */
812 /**
813  * Si on est en mode :
814  * - Creation : on met en place le creationListener courant dans
815  * drawingPanel pour créer la prochaine figure et on enlève de
816  * drawingPanel tous les listeners pour modifier les figures
817  * - Edition On retire le creationListener de drawingPanel puis on
818  * met en places les listeners dans drawingPanel pour
819  * - pouvoir sélectionner/désélectionner des figures
820  * - déplacer des figures
821  * - tourner des figures
822  * - changer l'échelle des figures
823  */
824
825 }
826
827 /**
828  *
829  */
830 /**
831  * Action réalisée pour filtrer ou pas le flux de figures
832  */
833 private class FilterAction extends AbstractAction
834 {
835     /**
836      * Constructeur de l'action pour mettre en place ou enlever un filtre
837      * pour filtrer les types de figures
838      */
839     public FilterAction()
840     {
841         putValue(NAME, "Filter");
842         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Filter"));
843         putValue(SMALL_ICON, IconFactory.getIcon("Filter_small"));
844         putValue(SHORT_DESCRIPTION, "Set/unset filtering");
845         putValue(ACCELERATOR_KEY,
846                 KeyStroke.getKeyStroke(KeyEvent.VK_F,
847                                         Toolkit.getDefaultToolkit().
848                                             getMenuShortcutKeyMask()));
849     }
850
851     /**
852      * Opérations réalisées par l'action
853      * @param event l'évènement déclenchant l'action. Peut provenir d'un
854      * bouton ou d'un item de menu
855      */
856     @Override
857     public void actionPerformed(ActionEvent event)
858     {
859         AbstractButton button = (AbstractButton) event.getSource();
860         boolean selected = button.getModel().isSelected();
861
862         // TODO Compléter ...
863     }
864
865     /**
866      *
867      */
868     /**
869      * Action réalisée pour ajouter ou retirer un filtre de type de figure
870      */
871     private class ShapeFilterAction extends AbstractAction
872     {
873         /**
874          * Le type de figure
875          */
876         private FigureType type;
877
878         /**
879          * Constructeur de l'action pour mettre en place ou enlever un filtre
880          * pour filtrer les types de figures
881          */
882         public ShapeFilterAction(FigureType type)
883         {
884             this.type = type;
885             String name = type.toString();
886             putValue(LARGE_ICON_KEY, IconFactory.getIcon(name));
887             putValue(SMALL_ICON, IconFactory.getIcon(name + "_small"));
888             putValue(NAME, name);
889             putValue(SHORT_DESCRIPTION, "Set/unset " + name + " filter");
890         }
891
892         /**
893          * Opérations réalisées par l'action
894          * @param event l'évènement déclenchant l'action. Peut provenir d'un
895          * bouton ou d'un item de menu
896          */
897         @Override
898         public void actionPerformed(ActionEvent event)
899         {
900         }
901     }
902 }

```

avr 02, 17 16:17

## EditorFrame.java

Page 11/16

```

899     AbstractButton button = (AbstractButton) event.getSource();
900     boolean selected = button.getModel().isSelected();
901
902     // TODO Compléter ...
903 }
904
905 /**
906  * Action réalisée pour ajouter ou retirer un filtre de type trait de figure
907  */
908 private class LineFilterAction extends AbstractAction
909 {
910     /**
911      * Le type de trait de la figure
912      */
913     private LineType type;
914
915     /**
916      * Constructeur de l'action pour mettre en place ou enlever un filtre
917      * pour filtrer les types de figures
918      */
919     public LineFilterAction(LineType type)
920     {
921         this.type = type;
922         String name = type.toString();
923         putValue(LARGE_ICON_KEY, IconFactory.getIcon(name));
924         putValue(SMALL_ICON, IconFactory.getIcon(name + "_small"));
925         putValue(NAME, name);
926         putValue(SHORT_DESCRIPTION, "Set/unset " + name + " filter");
927     }
928
929     /**
930      * Opérations réalisées par l'action
931      * @param e l'évènement déclenchant l'action. Peut provenir d'un
932      * bouton ou d'un item de menu
933      */
934     @Override
935     public void actionPerformed(ActionEvent event)
936     {
937         AbstractButton button = (AbstractButton) event.getSource();
938         boolean selected = button.getModel().isSelected();
939
940         // TODO Compléter ...
941     }
942 }
943
944 /**
945  * Action pour mettre en place un filtre basé sur la couleur de remplissage
946  * courante
947  */
948 private class FillColorFilterAction extends AbstractAction
949 {
950     /**
951      * Constructeur de l'action
952      * Met en place le raccourci clavier, l'icône et la description
953      * de l'action
954      */
955     public FillColorFilterAction()
956     {
957         putValue(NAME, "Fill Color");
958         putValue(LARGE_ICON_KEY, IconFactory.getIcon("FillColor"));
959         putValue(SMALL_ICON, IconFactory.getIcon("FillColor_small"));
960         putValue(SHORT_DESCRIPTION, "Set/Unset Fill Color Filter");
961     }
962
963     /**
964      * Opérations réalisées par l'action
965      * @param e l'évènement déclenchant l'action. Peut provenir d'un bouton
966      * ou d'un item de menu
967      */
968     @Override
969     public void actionPerformed(ActionEvent e)
970     {
971         AbstractButton button = (AbstractButton) e.getSource();
972         boolean selected = button.getModel().isSelected();
973
974         // TODO Compléter ...
975     }
976 }
977
978 /**
979  * Action pour mettre en place un filtre basé sur la couleur de trait
980  * courante
981  */
982 private class EdgeColorFilterAction extends AbstractAction
983 {
984     /**
985      * Constructeur de l'action
986      * Met en place le raccourci clavier, l'icône et la description
987      * de l'action

```

avr 02, 17 16:17

## EditorFrame.java

Page 12/16

```

899     public EdgeColorFilterAction()
900     {
901         putValue(NAME, "Edge Color");
902         putValue(LARGE_ICON_KEY, IconFactory.getIcon("EdgeColor"));
903         putValue(SMALL_ICON, IconFactory.getIcon("EdgeColor_small"));
904         putValue(SHORT_DESCRIPTION, "Set/Unset edge color filter");
905     }
906
907     /**
908      * Opérations réalisées par l'action
909      * @param e l'évènement déclenchant l'action. Peut provenir d'un bouton
910      * ou d'un item de menu
911      */
912     @Override
913     public void actionPerformed(ActionEvent e)
914     {
915         AbstractButton button = (AbstractButton) e.getSource();
916         boolean selected = button.getModel().isSelected();
917
918         // TODO Compléter ...
919     }
920 }
921
922 /**
923  * Action réalisée pour détruire les figures sélectionnées
924  * @author davidroussel
925  */
926 private class DeleteAction extends AbstractAction
927 {
928     public DeleteAction()
929     {
930         putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_X, 0));
931         putValue(NAME, "Delete");
932         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Delete"));
933         putValue(SMALL_ICON, IconFactory.getIcon("Delete_small"));
934         putValue(SHORT_DESCRIPTION, "Delete selected figures");
935     }
936
937     @Override
938     public void actionPerformed(ActionEvent e)
939     {
940         // TODO Compléter ...
941     }
942 }
943
944 /**
945  * Action réalisée pour remonter les figures sélectionnées dans la liste
946  * des figures
947  */
948 private class MoveUpAction extends AbstractAction
949 {
950     public MoveUpAction()
951     {
952         putValue(ACCELERATOR_KEY,
953                 KeyStroke.getKeyStroke(KeyEvent.VK_UP,
954                                         Toolkit.getDefaultToolkit()
955                                             .getMenuShortcutKeyMask()));
956
957         putValue(NAME, "Up");
958         putValue(LARGE_ICON_KEY, IconFactory.getIcon("MoveUp"));
959         putValue(SMALL_ICON, IconFactory.getIcon("MoveUp_small"));
960         putValue(SHORT_DESCRIPTION, "Move selected figures up");
961     }
962
963     @Override
964     public void actionPerformed(ActionEvent e)
965     {
966         // TODO Compléter ...
967     }
968 }
969
970 /**
971  * Action réalisée pour descendre les figures sélectionnées dans la liste
972  * des figures
973  */
974 private class MoveDownAction extends AbstractAction
975 {
976     public MoveDownAction()
977     {
978         putValue(ACCELERATOR_KEY,
979                 KeyStroke.getKeyStroke(KeyEvent.VK_DOWN,
980                                         Toolkit.getDefaultToolkit()
981                                             .getMenuShortcutKeyMask()));
982
983         putValue(NAME, "Down");
984         putValue(LARGE_ICON_KEY, IconFactory.getIcon("MoveDown"));
985         putValue(SMALL_ICON, IconFactory.getIcon("MoveDown_small"));
986         putValue(SHORT_DESCRIPTION, "Move selected figures down");
987     }
988
989     @Override

```

avr 02, 17 16:17

## EditorFrame.java

Page 13/16

```

1079     public void actionPerformed(ActionEvent e)
1080     {
1081         // TODO Compléter ...
1082     }
1083 }
1084
1085 /**
1086  * Action réalisée pour appliquer le style courant aux figures
1087  * sélectionnées,
1088  * A savoir :
1089  * <ul>
1090  * <li>La couleur de remplissage courante</li>
1091  * <li>La couleur de trait courante</li>
1092  * <li>Le type de trait courant (style et épaisseur)</li>
1093  * </ul>
1094  */
1095 private class StyleAction extends AbstractAction
1096 {
1097     public StyleAction()
1098     {
1099         putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_S, 0));
1100         putValue(NAME, "Style");
1101         putValue(LARGE_ICON_KEY, IconFactory.getIcon("Style"));
1102         putValue(SMALL_ICON, IconFactory.getIcon("Style_small"));
1103         putValue(SHORT_DESCRIPTION,
1104             "Apply current style to selected figures");
1105     }
1106
1107     @Override
1108     public void actionPerformed(ActionEvent e)
1109     {
1110         // TODO Compléter ...
1111     }
1112 }
1113
1114 /**
1115  * Contrôleur d'évènement permettant de modifier le type de figures à
1116  * dessiner.
1117  * @note dépend de #drawingModel et #infoLabel qui doivent être non
1118  * null avant instantiation
1119  */
1120 private class ShapeItemListener implements ItemListener
1121 {
1122     /**
1123      * Constructeur valant du contrôleur
1124      * Initialise le type de dessin dans {@link EditorFrame#drawingModel}
1125      * et crée le {@link AbstractCreationListener} correspondant.
1126      * @param initialIndex l'index du type de forme sélectionné afin de
1127      * mettre en place le bon creationListener dans le
1128      * {@link EditorFrame#drawingPanel}.
1129      */
1130     public ShapeItemListener(FigureType type)
1131     {
1132         // Mise en place du type de figure
1133         drawingModel.setFigureType(type);
1134
1135         // Mise en place du type de creationListener
1136         creationListener =
1137             type.getCreationListener(drawingModel, infoLabel);
1138         drawingPanel.addFigureListener(creationListener);
1139     }
1140
1141     @Override
1142     public void itemStateChanged(ItemEvent e)
1143     {
1144         JComboBox<?> items = (JComboBox<?>) e.getSource();
1145         int index = items.getSelectedIndex();
1146         int stateChange = e.getStateChange();
1147         FigureType figureType = FigureType.fromInteger(index);
1148         switch (stateChange)
1149         {
1150             case ItemEvent.SELECTED:
1151             {
1152                 // Mise en place d'un nouveau type de figure
1153                 drawingModel.setFigureType(figureType);
1154                 AbstractCreationListener newCreationListener =
1155                     figureType.getCreationListener(drawingModel, infoLabel);
1156                 if (operationMode == OperationMode.CREATION)
1157                 {
1158                     // Mise en place d'un nouveau type de creationListener
1159                     // Après avoir retiré l'ancien dans le drawingPanel
1160                     drawingPanel.removeFigureListener(creationListener);
1161                     drawingPanel.addFigureListener(newCreationListener);
1162                 }
1163                 creationListener = newCreationListener;
1164                 break;
1165             }
1166         }
1167     }
1168 }

```

avr 02, 17 16:17

## EditorFrame.java

Page 14/16

```

1169     /**
1170      * Contrôleur d'évènements permettant de modifier la couleur du trait.
1171      * @note utilise #drawingModel qui doit être non null avant instantiation
1172      * @note A associer comme listener au JJ[Labeled]ComboBox des couleurs de
1173      * remplissage ou de trait
1174      */
1175     private class ColorItemListener implements ItemListener
1176     {
1177         /**
1178          * Ce à quoi s'applique la couleur choisie.
1179          * Soit au remplissage, soit au trait.
1180          */
1181         private PaintToType applyTo;
1182
1183         /**
1184          * La dernière couleur choisie (pour le {@link JColorChooser})
1185          */
1186         private Color lastColor;
1187
1188         /**
1189          * Le tableau des couleurs possibles
1190          */
1191         private Paint[] colors;
1192
1193         /**
1194          * L'index de la couleur spéciale à choisir avec un
1195          * {@link JColorChooser}
1196          */
1197         private final int customColorIndex;
1198
1199         /**
1200          * L'index de la dernière couleur sélectionnée dans le combobox
1201          * Afin de pouvoir y revenir si jamais le {@link JColorChooser} est
1202          * annulé.
1203          */
1204         private int lastSelectedIndex;
1205
1206         /**
1207          * la couleur choisie
1208          */
1209         private Paint paint;
1210
1211         /**
1212          * Constructeur du contrôleur d'évènements d'un combobox permettant
1213          * de choisir la couleur de remplissage
1214          * @param colors le tableau des couleurs possibles
1215          * @param selectedIndex l'index de l'élément actuellement sélectionné
1216          * @param customColorIndex l'index de la couleur spéciale parmi les
1217          * colors à définir à l'aide d'un {@link JColorChooser}.
1218          * @param applyTo Ce à quoi s'applique la couleur (le remplissage ou
1219          * bien le trait)
1220          */
1221         public ColorItemListener(Paint[] colors,
1222             int selectedIndex,
1223             int customColorIndex,
1224             PaintToType applyTo)
1225         {
1226             this.colors = colors;
1227             lastSelectedIndex = selectedIndex;
1228             this.customColorIndex = customColorIndex;
1229             this.applyTo = applyTo;
1230             lastColor = (Color) colors[selectedIndex];
1231             paint = colors[selectedIndex];
1232
1233             applyTo.applyPaintTo(paint, drawingModel);
1234         }
1235
1236         /**
1237          * Actions à réaliser lorsque l'élément sélectionné du combobox change
1238          * @param e l'évènement de changement d'item du combobox
1239          */
1240         @Override
1241         public void itemStateChanged(ItemEvent e)
1242         {
1243             JComboBox<?> combo = (JComboBox<?>) e.getSource();
1244             int index = combo.getSelectedIndex();
1245
1246             if ((index ≥ 0) ^ (index < colors.length))
1247             {
1248                 if (e.getStateChange() == ItemEvent.SELECTED)
1249                 {
1250                     // New color has been selected
1251                     if (index == customColorIndex) // Custom color from chooser
1252                     {
1253                         Paint chosenColor = PaintFactory
1254                             .getPaint(combo,
1255                                 "Choose " + applyTo.toString() + " Color",
1256                                 lastColor);
1257                         if (chosenColor ≠ null)
1258                     }

```

avr 02, 17 16:17

## EditorFrame.java

Page 15/16

```

1259         {
1260             paint = chosenColor;
1261         }
1262         else
1263         {
1264             // ColorChooser has been cancelled we should go
1265             // back to last selected index
1266             combo.setSelectedIndex(lastSelectedIndex);
1267
1268             // paint does not change
1269         }
1270
1271         else // regular color
1272         {
1273             paint = colors[index];
1274         }
1275
1276         lastColor = (Color) paint;
1277         applyTo.applyPaintTo(paint, drawingModel);
1278
1279         else if (e.getStateChange() == ItemEvent.DESELECTED)
1280         {
1281             // Old color has been deselected
1282             if ((index >= 0) ^ (index < customColorIndex))
1283             {
1284                 lastColor = (Color) edgePaints[index];
1285                 lastSelectedIndex = index;
1286             }
1287         }
1288         else
1289         {
1290             System.err.println("Unknown " + applyTo.toString()
1291                 + " color index: " + index);
1292         }
1293     }
1294 }
1295
1296 /**
1297  * Contrôleur d'événements permettant de modifier le type de trait (normal,
1298  * pointillé, sans trait)
1299  * @note utilise #drawingModel qui doit être non null avant instantiation
1300  * @note à associer comme listener au J[Labelled]Combobox des types de traits
1301  */
1302 private class EdgeTypeListener implements ItemListener
1303 {
1304     /**
1305      * Le type de trait à mettre en place
1306      */
1307     private LineType edgeType;
1308
1309     public EdgeTypeListener(LineType type)
1310     {
1311         edgeType = type;
1312         drawingModel.setEdgeType(edgeType);
1313     }
1314
1315     @Override
1316     public void itemStateChanged(ItemEvent e)
1317     {
1318         JComboBox<?> items = (JComboBox<?>) e.getSource();
1319         int index = items.getSelectedIndex();
1320
1321         if (e.getStateChange() == ItemEvent.SELECTED)
1322         {
1323             // actions à réaliser lorsque le type de trait change
1324             LineType type = LineType.fromInteger(index);
1325             drawingModel.setEdgeType(type);
1326         }
1327     }
1328 }
1329
1330 /**
1331  * Contrôleur d'événement permettant de modifier la taille du trait
1332  * en fonction des valeurs d'un JSpinner
1333  * @note à associer comme listener au JSpinner de l'épaisseur de trait
1334  */
1335 private class EdgeWidthListener implements ChangeListener
1336 {
1337     /**
1338      * Constructeur du contrôleur d'événements contrôlant l'épaisseur du
1339      * trait
1340      * @param initialValue la valeur initiale de la largeur du trait à
1341      * appliquer au dessin (EditorFrame#drawingModel)
1342      */
1343     public EdgeWidthListener(int initialValue)
1344     {
1345         drawingModel.setEdgeWidth(initialValue);
1346     }
1347 }
1348

```

avr 02, 17 16:17

## EditorFrame.java

Page 16/16

```

1349 /**
1350  * Actions à réaliser lorsque la valeur du spinner change
1351  * @param e l'évènement de changement de valeur du spinner
1352  */
1353 @Override
1354 public void stateChanged(ChangeEvent e)
1355 {
1356     JSpinner spinner = (JSpinner) e.getSource();
1357     SpinnerNumberModel spinnerModel =
1358         (SpinnerNumberModel) spinner.getModel();
1359
1360     drawingModel.setEdgeWidth(spinnerModel.getNumber().floatValue());
1361 }
1362
1363 /**
1364  * Action pour ...
1365  * @author davidroussel
1366  */
1367 @SuppressWarnings("unused")
1368 private class EmptyAction extends AbstractAction
1369 {
1370     /**
1371      * Constructeur de l'action pour ...
1372      * Met en place le raccourci clavier, l'icône et la description
1373      * de l'action
1374      */
1375     public EmptyAction()
1376     {
1377         String name = "XXX";
1378         putValue(NAME, name);
1379         /*
1380          * Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()
1381          * = InputEvent.CTRL_MASK on win/linux
1382          * = InputEvent.META_MASK on mac os
1383          */
1384         putValue(ACCELERATOR_KEY,
1385             KeyStroke.getKeyStroke(KeyEvent.VK_X,
1386                 Toolkit.getDefaultToolkit()
1387                     .getMenuShortcutKeyMask()));
1388         putValue(LARGE_ICON_KEY, IconFactory.getIcon(name));
1389         putValue(SMALL_ICON, IconFactory.getIcon(name + "_small"));
1390         putValue(SHORT_DESCRIPTION, "Description de l'action");
1391     }
1392
1393     /**
1394      * Opérations réalisées par l'action
1395      * @param e l'évènement déclenchant l'action. Peut provenir d'un bouton
1396      * ou d'un item de menu
1397      */
1398     @Override
1399     public void actionPerformed(ActionEvent e)
1400     {
1401         AbstractButton button = (AbstractButton) e.getSource();
1402         boolean selected = button.getModel().isSelected();
1403
1404         // drawingModel.awesomeMethod(...)
1405     }
1406 }
1407
1408

```



avr 02, 17 16:17

## OperationMode.java

Page 1/2

```

1 package widgets.enums;
2
3 /**
4  * Différents modes de fonctionnement de l'UI
5  * @author davidrousse
6  */
7 public enum OperationMode
8 {
9     /**
10      * Creation mode dans le quel on crée de nouvelles figures
11      */
12     CREATION,
13
14     /**
15      * Transformation mode dans lequel on effectue des transformations
16      * géométriques (déplacement, rotation, facteur d'échelle) sur
17      * les figures sélectionnées
18      */
19     TRANSFORMATION;
20
21     /**
22      * Nombre d'éléments dans cet enum
23      */
24     public static final int NbOperationModes = 2;
25
26     /**
27      * Conversion d'un entier en {@link OperationMode}
28      *
29      * @param i l'entier à convertir en {@link OperationMode}
30      * @return l'OperationMode correspondant à l'entier
31      */
32     public static OperationMode fromInteger(int i)
33     {
34         switch (i)
35         {
36             case 0:
37                 return CREATION;
38             case 1:
39                 return TRANSFORMATION;
40             default:
41                 return CREATION;
42         }
43     }
44
45     /**
46      * Index du mode
47      * @return l'index du mode
48      * @throws AssertionError si le mode est inconnu
49      */
50     public int toInteger() throws AssertionError
51     {
52         switch (this)
53         {
54             case CREATION:
55                 return 0;
56             case TRANSFORMATION:
57                 return 1;
58         }
59
60         throw new AssertionError("OperationMode Unknown assertion " + this);
61     }
62
63     /**
64      * Représentation sous forme de chaîne de caractères
65      * @return une chaîne de caractères représentant la valeur de cet enum
66      * @throws AssertionError si le mode est inconnu
67      */
68     @Override
69     public String toString() throws AssertionError
70     {
71         switch (this)
72         {
73             case CREATION:
74                 return new String("Creation");
75             case TRANSFORMATION:
76                 return new String("Edition");
77         }
78
79         throw new AssertionError("OperationMode Unknown assertion " + this);
80     }
81
82     /**
83      * Mode suivant dans l'ordre des modes
84      * @return le mode suivant le mode courant
85      * @throws AssertionError si le mode est inconnu
86      */
87     public OperationMode nextMode() throws AssertionError
88     {
89         switch (this)
90         {

```

avr 02, 17 16:17

## OperationMode.java

Page 2/2

```

91     {
92         case CREATION:
93             return TRANSFORMATION;
94         case TRANSFORMATION:
95             return CREATION;
96     }
97
98     throw new AssertionError("OperationMode Unknown assertion " + this);
99 }
100
101 }
102

```

avr 02, 17 16:17

**package-info.java**

Page 1/1

```

1  /**
2   * Package contenant les différents widgets (éléments graphiques)
3   */
4  package widgets;

```

avr 02, 17 16:17

**TreeType.java**

Page 1/1

```

1  package widgets.enums;
2
3  /**
4   * Les types d'arbre pour représenter les figures dans un {@link javax.swing.JTree}
5   * @author davidroussel
6   */
7  public enum TreeType
8  {
9      /**
10       * Simple liste de figures
11       */
12     FIGURE,
13     /**
14      * Groupement des figures par type de figure
15      */
16     FIGURE_TYPE,
17     /**
18      * Groupement des figures par type de couleur de remplissage
19      */
20     FILL_COLOR,
21     /**
22      * Groupement des figures par type de couleur de trait
23      */
24     EDGE_COLOR,
25     /**
26      * Groupement des figures par type de trait
27      */
28     EDGE_TYPE;
29
30     /**
31      * Nombre d'éléments dans cet enum
32      */
33     public static final int NbTreeTypes = 5;
34
35     /**
36      * Conversion d'un entier en {@link TreeType}
37      *
38      * @param i l'entier à convertir en TreeType
39      * @return le TreeType correspondant à l'entier
40      */
41     public static TreeType fromInteger(int i)
42     {
43         switch (i)
44         {
45             case 0:
46                 return FIGURE;
47             case 1:
48                 return FIGURE_TYPE;
49             case 2:
50                 return FILL_COLOR;
51             case 3:
52                 return EDGE_COLOR;
53             case 4:
54                 return EDGE_TYPE;
55             default:
56                 return FIGURE;
57         }
58     }
59
60     /**
61      * Représentation sous forme de chaîne de caractères
62      * @return une chaîne de caractères représentant la valeur de cet enum
63      */
64     @Override
65     public String toString() throws AssertionError
66     {
67         switch (this)
68         {
69             case FIGURE:
70                 return new String("Figure");
71             case FIGURE_TYPE:
72                 return new String("Figure Type");
73             case FILL_COLOR:
74                 return new String("Fill Color");
75             case EDGE_COLOR:
76                 return new String("Edge Color");
77             case EDGE_TYPE:
78                 return new String("Edge Type");
79         }
80
81         throw new AssertionError("TreeType Unknown assertion " + this);
82     }
83 }
84

```

avr 02, 17 16:17

## InfoPanel.java

Page 1/6

```

1 package widgets;
2
3 import java.awt.BasicStroke;
4 import java.awt.Color;
5 import java.awt.GridBagConstraints;
6 import java.awt.GridBagLayout;
7 import java.awt.Insets;
8 import java.awt.Paint;
9 import java.awt.geom.Point2D;
10 import java.awt.geom.Rectangle2D;
11 import java.text.DecimalFormat;
12 import java.util.HashMap;
13 import java.util.Map;
14
15 import javax.swing.ImageIcon;
16 import javax.swing.JLabel;
17 import javax.swing.JPanel;
18 import javax.swing.SwingConstants;
19 import javax.swing.border.LineBorder;
20
21 import figures.Figure;
22 import figures.enums.FigureType;
23 import figures.enums.LineType;
24 import utils.IconFactory;
25 import utils.PaintFactory;
26
27 public class InfoPanel extends JPanel
28 {
29     /**
30      * Une chaîne vide pour remplir les champs lorsque la souris n'est au dessus
31      * d'aucune figure
32      */
33     private static final String emptyString = new String();
34
35     /**
36      * Une icône vide pour remplir les champs avec icône lorsque la souris
37      * n'est au dessus d'aucune figure
38      */
39     private static final ImageIcon emptyIcon = IconFactory.getIcon("None");
40
41     /**
42      * Le formatteur à utiliser pour formater les coordonnées
43      */
44     private final static DecimalFormat coordFormat = new DecimalFormat("000");
45
46     /**
47      * Le label contenant le nom de la figure
48      */
49     private JLabel lblFigureName;
50
51     /**
52      * Le label contenant l'icône correspondant à la figure
53      */
54     private JLabel lblTypeicon;
55
56     /**
57      * La map contenant les différentes icônes des types de figures
58      */
59     private Map<FigureType, ImageIcon> figureIcons;
60
61     /**
62      * Le label contenant l'icône de la couleur de remplissage
63      */
64     private JLabel lblFillColor;
65
66     /**
67      * Le label contenant l'icône de la couleur du contour
68      */
69     private JLabel lblEdgecolor;
70
71     /**
72      * Map contenant les icônes relatives aux différentes couleurs (de contour
73      * ou de remplissage)
74      */
75     private Map<Paint, ImageIcon> paintIcons;
76
77     /**
78      * Le label contenant le type de contour
79      */
80     private JLabel lblStrokeType;
81
82     /**
83      * Map contenant les icônes relatives aux différents types de traits de
84      * contour
85      */
86     private Map<LineType, ImageIcon> lineTypeIcons;
87
88     /**
89      * Le label contenant l'abscisse du point en haut à gauche de la figure
90      */

```

Dimanche avril 02, 2017

../11/InfoPanel.java

avr 02, 17 16:17

## InfoPanel.java

Page 2/6

```

91 private JLabel lblTlx;
92
93 /**
94  * Le label contenant l'ordonnée du point en haut à gauche de la figure
95  */
96 private JLabel lblTly;
97
98 /**
99  * Le label contenant l'abscisse du point en bas à droite de la figure
100  */
101 private JLabel lblBrx;
102
103 /**
104  * Le label contenant l'ordonnée du point en bas à droite de la figure
105  */
106 private JLabel lblBry;
107
108 /**
109  * Le label contenant la largeur de la figure
110  */
111 private JLabel lblDx;
112
113 /**
114  * Le label contenant la hauteur de la figure
115  */
116 private JLabel lblDy;
117
118 /**
119  * Le label contenant l'abscisse du barycentre de la figure
120  */
121 private JLabel lblCx;
122
123 /**
124  * Le label contenant l'ordonnée du barycentre de la figure
125  */
126 private JLabel lblCy;
127
128 /**
129  * Create the panel.
130  */
131 public InfoPanel()
132 {
133     // -----
134     // Initialisation des maps
135     // -----
136     figureIcons = new HashMap<FigureType, ImageIcon>();
137     for (int i = 0; i < FigureType.NbFigureTypes; i++)
138     {
139         FigureType type = FigureType.fromInteger(i);
140         figureIcons.put(type, IconFactory.getIcon(type.toString()));
141     }
142
143     paintIcons = new HashMap<Paint, ImageIcon>();
144     String[] colorStrings = {
145         "Black",
146         "Blue",
147         "Cyan",
148         "Green",
149         "Magenta",
150         "None",
151         "Orange",
152         "Others",
153         "Red",
154         "White",
155         "Yellow"
156     };
157
158     for (int i = 0; i < colorStrings.length; i++)
159     {
160         Paint paint = PaintFactory.getPaint(colorStrings[i]);
161         if (paint != null)
162         {
163             paintIcons.put(paint, IconFactory.getIcon(colorStrings[i]));
164         }
165     }
166
167     lineTypeIcons = new HashMap<LineType, ImageIcon>();
168     for (int i = 0; i < LineType.NbLineTypes; i++)
169     {
170         LineType type = LineType.fromInteger(i);
171         lineTypeIcons.put(type, IconFactory.getIcon(type.toString()));
172     }
173
174     // -----
175     // Création de l'UI
176     // -----
177     setBorder(new LineBorder(new Color(0, 0, 0), 1, true));
178     GridBagLayout gridBagLayout = new GridBagLayout();
179     gridBagLayout.columnWidths = new int[] { 80, 60, 60 };
180     gridBagLayout.rowHeights = new int[] { 30, 32, 32, 32, 20, 20, 20, 20, 20 };

```

../11/InfoPanel.java

51/56

avr 02, 17 16:17

## InfoPanel.java

Page 3/6

```

181 gridBagLayout.columnWeights = new double[]{0.0, 0.0, 0.0};
182 gridBagLayout.rowWeights = new double[]{0.0, 0.0, 0.0, 0.0};
183 setLayout(gridBagLayout);
184
185 lblFigureName = new JLabel("Figure Name");
186 lblFigureName.setHorizontalAlignment(SwingConstants.CENTER);
187 GridBagConstraints gbc_lblFigureName = new GridBagConstraints();
188 gbc_lblFigureName.insets = new Insets(5, 5, 5, 0);
189 gbc_lblFigureName.gridwidth = 3;
190 gbc_lblFigureName.gridx = 0;
191 gbc_lblFigureName.gridy = 0;
192 add(lblFigureName, gbc_lblFigureName);
193
194 JLabel lblType = new JLabel("type");
195 GridBagConstraints gbc_lblType = new GridBagConstraints();
196 gbc_lblType.anchor = GridBagConstraints.EAST;
197 gbc_lblType.insets = new Insets(0, 0, 5, 5);
198 gbc_lblType.gridx = 0;
199 gbc_lblType.gridy = 1;
200 add(lblType, gbc_lblType);
201
202 lblTypeicon = new JLabel(IconFactory.getIcon("Polygon"));
203 lblTypeicon.setHorizontalAlignment(SwingConstants.CENTER);
204 GridBagConstraints gbc_lblTypeicon = new GridBagConstraints();
205 gbc_lblTypeicon.insets = new Insets(0, 0, 5, 0);
206 gbc_lblTypeicon.gridwidth = 2;
207 gbc_lblTypeicon.gridx = 1;
208 gbc_lblTypeicon.gridy = 1;
209 add(lblTypeicon, gbc_lblTypeicon);
210
211 JLabel lblFill = new JLabel("fill");
212 GridBagConstraints gbc_lblFill = new GridBagConstraints();
213 gbc_lblFill.anchor = GridBagConstraints.EAST;
214 gbc_lblFill.insets = new Insets(0, 0, 5, 5);
215 gbc_lblFill.gridx = 0;
216 gbc_lblFill.gridy = 2;
217 add(lblFill, gbc_lblFill);
218
219 lblFillColor = new JLabel(IconFactory.getIcon("White"));
220 GridBagConstraints gbc_lblFillColor = new GridBagConstraints();
221 gbc_lblFillColor.gridwidth = 2;
222 gbc_lblFillColor.insets = new Insets(0, 0, 5, 0);
223 gbc_lblFillColor.gridx = 1;
224 gbc_lblFillColor.gridy = 2;
225 add(lblFillColor, gbc_lblFillColor);
226
227 JLabel lblStroke = new JLabel("stroke");
228 GridBagConstraints gbc_lblStroke = new GridBagConstraints();
229 gbc_lblStroke.anchor = GridBagConstraints.EAST;
230 gbc_lblStroke.insets = new Insets(0, 0, 5, 5);
231 gbc_lblStroke.gridx = 0;
232 gbc_lblStroke.gridy = 3;
233 add(lblStroke, gbc_lblStroke);
234
235 lblEdgecolor = new JLabel(IconFactory.getIcon("Black"));
236 GridBagConstraints gbc_lblStrokecolor = new GridBagConstraints();
237 gbc_lblStrokecolor.insets = new Insets(0, 0, 5, 5);
238 gbc_lblStrokecolor.gridx = 1;
239 gbc_lblStrokecolor.gridy = 3;
240 add(lblEdgecolor, gbc_lblStrokecolor);
241
242 lblStrokeType = new JLabel(IconFactory.getIcon("Solid"));
243 GridBagConstraints gbc_lblStrokeType = new GridBagConstraints();
244 gbc_lblStrokeType.insets = new Insets(0, 0, 5, 0);
245 gbc_lblStrokeType.gridx = 2;
246 gbc_lblStrokeType.gridy = 3;
247 add(lblStrokeType, gbc_lblStrokeType);
248
249 JLabel lblX = new JLabel("x");
250 lblX.setFont(lblX.getFont().deriveFont(lblX.getFont().getSize() - 3f));
251 GridBagConstraints gbc_lblX = new GridBagConstraints();
252 gbc_lblX.insets = new Insets(0, 0, 5, 5);
253 gbc_lblX.gridx = 1;
254 gbc_lblX.gridy = 4;
255 add(lblX, gbc_lblX);
256
257 JLabel lblY = new JLabel("y");
258 lblY.setFont(lblY.getFont().deriveFont(lblY.getFont().getSize() - 3f));
259 GridBagConstraints gbc_lblY = new GridBagConstraints();
260 gbc_lblY.insets = new Insets(0, 0, 5, 0);
261 gbc_lblY.gridx = 2;
262 gbc_lblY.gridy = 4;
263 add(lblY, gbc_lblY);
264
265 JLabel lblTopLeft = new JLabel("top left");
266 lblTopLeft.setFont(lblTopLeft.getFont().deriveFont(lblTopLeft.getFont().getSize() - 3f));
267 GridBagConstraints gbc_lblTopLeft = new GridBagConstraints();
268 gbc_lblTopLeft.anchor = GridBagConstraints.EAST;
269 gbc_lblTopLeft.insets = new Insets(0, 0, 5, 5);
270 gbc_lblTopLeft.gridx = 0;

```

avr 02, 17 16:17

## InfoPanel.java

Page 4/6

```

271 gbc_lblTopLeft.gridy = 5;
272 add(lblTopLeft, gbc_lblTopLeft);
273
274 lblTlx = new JLabel("tlx");
275 lblTlx.setFont(lblTlx.getFont().deriveFont(lblTlx.getFont().getSize() - 3f));
276 GridBagConstraints gbc_lblTlx = new GridBagConstraints();
277 gbc_lblTlx.insets = new Insets(0, 0, 5, 5);
278 gbc_lblTlx.gridx = 1;
279 gbc_lblTlx.gridy = 5;
280 add(lblTlx, gbc_lblTlx);
281
282 lblTly = new JLabel("tly");
283 lblTly.setFont(lblTly.getFont().deriveFont(lblTly.getFont().getSize() - 3f));
284 GridBagConstraints gbc_lblTly = new GridBagConstraints();
285 gbc_lblTly.insets = new Insets(0, 0, 5, 0);
286 gbc_lblTly.gridx = 2;
287 gbc_lblTly.gridy = 5;
288 add(lblTly, gbc_lblTly);
289
290 JLabel lblBottomRight = new JLabel("bottom right");
291 lblBottomRight.setFont(lblBottomRight.getFont().deriveFont(lblBottomRight.getFont().getSize()
292 ) - 3f));
293 GridBagConstraints gbc_lblBottomRight = new GridBagConstraints();
294 gbc_lblBottomRight.anchor = GridBagConstraints.EAST;
295 gbc_lblBottomRight.insets = new Insets(0, 0, 5, 5);
296 gbc_lblBottomRight.gridx = 0;
297 gbc_lblBottomRight.gridy = 6;
298 add(lblBottomRight, gbc_lblBottomRight);
299
300 lblBrx = new JLabel("brx");
301 lblBrx.setFont(lblBrx.getFont().deriveFont(lblBrx.getFont().getSize() - 3f));
302 GridBagConstraints gbc_lblBrx = new GridBagConstraints();
303 gbc_lblBrx.insets = new Insets(0, 0, 5, 5);
304 gbc_lblBrx.gridx = 1;
305 gbc_lblBrx.gridy = 6;
306 add(lblBrx, gbc_lblBrx);
307
308 lblBry = new JLabel("bry");
309 lblBry.setFont(lblBry.getFont().deriveFont(lblBry.getFont().getSize() - 3f));
310 GridBagConstraints gbc_lblBry = new GridBagConstraints();
311 gbc_lblBry.insets = new Insets(0, 0, 5, 0);
312 gbc_lblBry.gridx = 2;
313 gbc_lblBry.gridy = 6;
314 add(lblBry, gbc_lblBry);
315
316 JLabel lblDimensions = new JLabel("dimensions");
317 lblDimensions.setFont(lblDimensions.getFont().deriveFont(lblDimensions.getFont().getSize() -
318 3f));
319 GridBagConstraints gbc_lblDimensions = new GridBagConstraints();
320 gbc_lblDimensions.anchor = GridBagConstraints.EAST;
321 gbc_lblDimensions.insets = new Insets(0, 0, 5, 5);
322 gbc_lblDimensions.gridx = 0;
323 gbc_lblDimensions.gridy = 7;
324 add(lblDimensions, gbc_lblDimensions);
325
326 lblDx = new JLabel("dx");
327 lblDx.setFont(lblDx.getFont().deriveFont(lblDx.getFont().getSize() - 3f));
328 GridBagConstraints gbc_lblDx = new GridBagConstraints();
329 gbc_lblDx.insets = new Insets(0, 0, 5, 5);
330 gbc_lblDx.gridx = 1;
331 gbc_lblDx.gridy = 7;
332 add(lblDx, gbc_lblDx);
333
334 lblDy = new JLabel("dy");
335 lblDy.setFont(lblDy.getFont().deriveFont(lblDy.getFont().getSize() - 3f));
336 GridBagConstraints gbc_lblDy = new GridBagConstraints();
337 gbc_lblDy.insets = new Insets(0, 0, 5, 0);
338 gbc_lblDy.gridx = 2;
339 gbc_lblDy.gridy = 7;
340 add(lblDy, gbc_lblDy);
341
342 JLabel lblCenter = new JLabel("center");
343 lblCenter.setFont(lblCenter.getFont().deriveFont(lblCenter.getFont().getSize() - 3f));
344 GridBagConstraints gbc_lblCenter = new GridBagConstraints();
345 gbc_lblCenter.anchor = GridBagConstraints.EAST;
346 gbc_lblCenter.insets = new Insets(0, 0, 0, 5);
347 gbc_lblCenter.gridx = 0;
348 gbc_lblCenter.gridy = 8;
349 add(lblCenter, gbc_lblCenter);
350
351 lblCx = new JLabel("cx");
352 lblCx.setFont(lblCx.getFont().deriveFont(lblCx.getFont().getSize() - 3f));
353 GridBagConstraints gbc_lblCx = new GridBagConstraints();
354 gbc_lblCx.insets = new Insets(0, 0, 0, 5);
355 gbc_lblCx.gridx = 1;
356 gbc_lblCx.gridy = 8;
357 add(lblCx, gbc_lblCx);
358
359 lblCy = new JLabel("cy");
360 lblCy.setFont(lblCy.getFont().deriveFont(lblCy.getFont().getSize() - 3f));

```

avr 02, 17 16:17

## InfoPanel.java

Page 5/6

```

359     GridBagConstraints gbc_lblCy = new GridBagConstraints();
360     gbc_lblCy.gridx = 2;
361     gbc_lblCy.gridy = 8;
362     add(lblCy, gbc_lblCy);
363
364 }
365
366 /**
367  * Mise à jour de tous les labels avec les informations de figure
368  * @param figure la figure dont il faut extraire les informations
369  */
370 public void updateLabels(Figure figure)
371 {
372     // titre de la figure
373     lblFigureName.setText(figure.toString());
374
375     // Icône du type de figure
376     lblTypeIcon.setIcon(figureIcons.get(figure.getType()));
377
378     // Icône de la couleur de remplissage
379     ImageIcon fillColorIcon = paintIcons.get(figure.getFillPaint());
380     if (fillColorIcon == null)
381     {
382         fillColorIcon = IconFactory.getIcon("Others");
383     }
384     lblFillColor.setIcon(fillColorIcon);
385
386     // Icône de la couleur de trait
387     ImageIcon edgeColorIcon = paintIcons.get(figure.getEdgePaint());
388     if (edgeColorIcon == null)
389     {
390         edgeColorIcon = IconFactory.getIcon("Others");
391     }
392     lblEdgeColor.setIcon(edgeColorIcon);
393
394     // Icône du type de trait
395     BasicStroke stroke = figure.getStroke();
396     ImageIcon lineTypeIcon = null;
397     if (stroke == null)
398     {
399         lineTypeIcon = lineTypeIcons.get(LineType.NONE);
400     }
401     else
402     {
403         float[] dashArray = stroke.getDashArray();
404         if (dashArray == null)
405         {
406             lineTypeIcon = lineTypeIcons.get(LineType.SOLID);
407         }
408         else
409         {
410             lineTypeIcon = lineTypeIcons.get(LineType.DASHED);
411         }
412     }
413     lblStrokeType.setIcon(lineTypeIcon);
414
415     // Données numériques
416     Rectangle2D bounds = figure.getBounds2D();
417     Point2D center = figure.getCenter();
418
419     double minX = bounds.getMinX();
420     double maxX = bounds.getMaxX();
421     double minY = bounds.getMinY();
422     double maxY = bounds.getMaxY();
423     double width = maxX - minX;
424     double height = maxY - minY;
425
426     lblTlx.setText(coordFormat.format(minX));
427     lblTly.setText(coordFormat.format(minY));
428     lblBrx.setText(coordFormat.format(maxX));
429     lblBry.setText(coordFormat.format(maxY));
430
431     lblDx.setText(coordFormat.format(width));
432     lblDy.setText(coordFormat.format(height));
433
434     lblCx.setText(coordFormat.format(center.getX()));
435     lblCy.setText(coordFormat.format(center.getY()));
436 }
437
438 /**
439  * Effacement de tous les labels
440  */
441 public void resetLabels()
442 {
443     // titre de la figure
444     lblFigureName.setText(emptyString);
445
446     // Icône du type de figure
447     lblTypeIcon.setIcon(emptyIcon);
448

```

avr 02, 17 16:17

## InfoPanel.java

Page 6/6

```

449     // Icône de la couleur de remplissage
450     lblFillColor.setIcon(emptyIcon);
451
452     // Icône de la couleur de trait
453     lblEdgeColor.setIcon(emptyIcon);
454
455     // Icône du type de trait
456     lblStrokeType.setIcon(emptyIcon);
457
458     // Données numériques
459     lblTlx.setText(emptyString);
460     lblTly.setText(emptyString);
461     lblBrx.setText(emptyString);
462     lblBry.setText(emptyString);
463
464     lblDx.setText(emptyString);
465     lblDy.setText(emptyString);
466
467     lblCx.setText(emptyString);
468     lblCy.setText(emptyString);
469 }
470 }

```

avr 02, 17 16:17

## JLabeledComboBox.java

Page 1/3

```

1 package widgets;
2
3 import java.awt.Component;
4 import java.awt.Dimension;
5 import java.awt.Font;
6 import java.awt.event.ItemListener;
7
8 import javax.swing.BoxLayout;
9 import javax.swing.ImageIcon;
10 import javax.swing.JComboBox;
11 import javax.swing.JLabel;
12 import javax.swing.JList;
13 import javax.swing.JPanel;
14 import javax.swing.ListCellRenderer;
15 import javax.swing.SwingConstants;
16
17 import utils.IconItem;
18
19 /**
20  * Classe contenant un titre et une liste déroulante utilisant des JLabel avec
21  * des icônes pour les éléments de la liste déroulante
22  */
23 public class JLabeledComboBox extends JPanel
24 {
25     /** Le titre de cette liste */
26     private String title;
27
28     /**
29      * Les textes et icônes pour les items
30      */
31     private IconItem[] items;
32
33     /**
34      * La combobox utilisée à l'intérieur pour pouvoir ajouter des listener
35      * par la suite
36      */
37     private JComboBox<IconItem> combobox;
38
39     /**
40      * Constructeur
41      * @param title le titre du panel
42      * @param captions les légendes des éléments de la liste
43      * @param selectedIndex l'élément sélectionné initialement
44      * @param listener le listener à appeler quand l'élément sélectionné de la
45      * liste change
46      * @see #createImageIcon(String)
47      */
48     public JLabeledComboBox(String title, String[] captions, int selectedIndex,
49                             ItemListener listener)
50     {
51         setAlignmentX(Component.LEFT_ALIGNMENT);
52
53         this.title = title;
54         items = new IconItem[captions.length];
55
56         for (int i = 0; i < captions.length; i++)
57         {
58             items[i] = new IconItem(captions[i]);
59         }
60
61         setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
62
63         // Creates the title
64         JLabel label = new JLabel((this.title != null ? this.title : "text"));
65         label.setHorizontalAlignment(SwingConstants.LEFT);
66         add(label);
67
68         // Creates the Combobox
69         combobox = new JComboBox<IconItem>(items);
70         combobox.setAlignmentX(Component.LEFT_ALIGNMENT);
71         combobox.setEditable(false);
72         int index;
73         if ((selectedIndex < 0) ∨ (selectedIndex > captions.length))
74         {
75             index = 0;
76         }
77         else
78         {
79             index = selectedIndex;
80         }
81         combobox.setSelectedIndex(index);
82         combobox.addItemListener(listener);
83         // Mise en place du renderer pour les éléments de la liste
84         JLabelRenderer renderer = new JLabelRenderer();
85         renderer.setPreferredSize(new Dimension(100, 32));
86         combobox.setRenderer(renderer);
87         // Ajout de la liste
88         add(combobox);
89     }
90

```

avr 02, 17 16:17

## JLabeledComboBox.java

Page 2/3

```

91 /**
92  * Ajout d'un nouveau listener déclenché lorsque un élément est sélectionné
93  * @param aListener le nouveau listener à ajouter.
94  */
95 public void addItemListener(ItemListener aListener)
96 {
97     if (combobox != null)
98     {
99         combobox.addItemListener(aListener);
100     }
101     else
102     {
103         System.err.println(getClass().getSimpleName() + "::addItemListener : null combobox");
104     }
105 }
106
107 /**
108  * Obtention de l'index de l'élément sélectionné dans la combobox
109  * @return l'index de l'élément sélectionné dans la combobox
110  */
111 public int getSelectedIndex()
112 {
113     return combobox.getSelectedIndex();
114 }
115
116 /**
117  * Renderer pour les Labels du combobox
118  */
119 protected class JLabelRenderer extends JLabel
120     implements ListCellRenderer<IconItem>
121 {
122     /** fonte pour les items à problèmes */
123     private Font pbFont;
124
125     /**
126      * Constructeur
127      */
128     public JLabelRenderer()
129     {
130         setOpaque(true);
131         setHorizontalAlignment(LEFT);
132         setVerticalAlignment(CENTER);
133     }
134
135     /*
136      * (non-Javadoc)
137      * @see
138      * javax.swing.ListCellRenderer#getListCellRendererComponent(javax.swing
139      * .JList, java.lang.Object, int, boolean, boolean)
140      */
141     @Override
142     public Component getListCellRendererComponent(
143         JList<? extends IconItem> list, IconItem value, int index,
144         boolean isSelected, boolean cellHasFocus)
145     {
146         if (isSelected)
147         {
148             setBackground(list.getSelectionBackground());
149             setForeground(list.getSelectionForeground());
150         }
151         else
152         {
153             setBackground(list.getBackground());
154             setForeground(list.getForeground());
155         }
156
157         // Mise en place de l'icône et du texte dans le label
158         // Si l'icône est null afficher un label particulier avec
159         // setPbText
160         ImageIcon itemIcon = value.getIcon();
161         String itemString = value.getCaption();
162         setIcon(itemIcon);
163         if (itemIcon != null)
164         {
165             setText(itemString);
166             setFont(list.getFont());
167         }
168         else
169         {
170             setPbText(itemString + " (pas d'image)", list.getFont());
171         }
172
173         return this;
174     }
175
176     /**
177      * Mise en place du texte s'il y a un pb pour cet item
178      * @param pbText le texte à afficher
179      * @param normalFont la fonte à utiliser (italique)
180      */

```

avr 02, 17 16:17

**JLabeledComboBox.java**

Page 3/3

```
181     protected void setPbText(String pbText, Font normalFont)
182     {
183         if (pbFont == null)
184         { // lazily create this font
185             pbFont = normalFont.deriveFont(Font.ITALIC);
186         }
187         setFont(pbFont);
188         setText(pbText);
189     }
190 }
191 }
```

avr 02, 17 16:17

**package-info.java**

Page 1/1

```
1  /**
2   * Package contenant les différents widgets (éléments graphiques)
3   */
4  package widgets;
```

avr 02, 17 16:17

## TreesPanel.java

Page 1/2

```

1 package widgets;
2
3 import java.awt.BorderLayout;
4 import java.awt.event.ItemEvent;
5 import java.awt.event.ItemListener;
6 import java.util.Observer;
7
8 import javax.swing.DefaultComboBoxModel;
9 import javax.swing.JComboBox;
10 import javax.swing.JLabel;
11 import javax.swing.JPanel;
12 import javax.swing.JScrollPane;
13 import javax.swing.JTree;
14 import javax.swing.ScrollPaneConstants;
15 import javax.swing.tree.TreeModel;
16
17 import figures.Drawing;
18 import figures.treemodels.FigureTreeModel;
19 import widgets.enums.TreeType;
20
21 public class TreesPanel extends JPanel
22 {
23     /**
24      * Le type d'arbre que l'on veut utiliser
25      * @see TreeType
26      */
27     private TreeType treeType;
28
29     /**
30      * Le modèle d'arbre à créer en fonction du {@link #treeType}
31      */
32     private TreeModel model;
33
34     /**
35      * Le modèle de dessin
36      */
37     private Drawing drawing;
38
39     /**
40      * Le {@link JTree} à utiliser pour visualiser l'arbre
41      */
42     private JTree tree;
43
44     /**
45      * Change le type d'arbre et crée le TreeModel associé
46      * @param treeType the treeType to set
47      */
48     public void setTreeType(TreeType treeType)
49     {
50         System.out.println("setTreeType(" + treeType + ")");
51         this.treeType = treeType;
52
53         if (model != null)
54         {
55             drawing.deleteObserver((Observer) model);
56             model = null;
57         }
58
59         if ((drawing != null) ^ (tree != null))
60         {
61             switch (this.treeType)
62             {
63                 case FIGURE:
64                     model = new FigureTreeModel(drawing, tree);
65                     break;
66                 case FIGURE_TYPE:
67                     model = null; // TODO new FigureTypeTreeModel(drawing, tree);
68                     break;
69                 case FILL_COLOR:
70                     model = null; // TODO new FillColorTreeModel(drawing, tree);
71                     break;
72                 case EDGE_COLOR:
73                     model = null; // TODO new EdgeColorTreeModel(drawing, tree);
74                     break;
75                 case EDGE_TYPE:
76                     model = null; // TODO new EdgeTypeTreeModel(drawing, tree);
77                     break;
78                 default:
79                     model = null;
80                     break;
81             }
82         }
83         else
84         {
85             System.out.println("FigureTypeTreeModel not set up because "
86                 + "null drawing or null JTree");
87         }
88
89         if (model != null)
90         {

```

avr 02, 17 16:17

## TreesPanel.java

Page 2/2

```

91         tree.setModel(model);
92     }
93     else
94     {
95         System.err.println(getClass().getSimpleName() + "::setTreeType: null model");
96     }
97 }
98
99 /**
100  * Sets the drawing
101  * @param drawing the drawing to set
102  */
103 public void setDrawing(Drawing drawing)
104 {
105     // System.out.println("Setting up Drawing" + drawing + " in
106     // TreesPanel");
107     this.drawing = drawing;
108     if (drawing != null)
109     {
110         setTreeType(treeType);
111     }
112     else
113     {
114         System.err.println(getClass().getSimpleName() + "::setDrawing: null drawing");
115     }
116 }
117
118 /**
119  * Create the panel.
120  */
121 public TreesPanel()
122 {
123     int treeTypeIndex = 0;
124     treeType = TreeType.fromInteger(treeTypeIndex);
125     model = null;
126     setLayout(new BorderLayout(0, 0));
127
128     JPanel treeModePanel = new JPanel();
129     add(treeModePanel, BorderLayout.NORTH);
130     treeModePanel.setLayout(new BorderLayout(0, 0));
131
132     JLabel lblTreeMode = new JLabel("Tree mode");
133     treeModePanel.add(lblTreeMode, BorderLayout.WEST);
134
135     JComboBox<TreeType> treeComboBox = new JComboBox<TreeType>();
136     treeComboBox.setMaximumRowCount(TreeType.NbTreeTypes);
137     treeComboBox
138         .setModel(new DefaultComboBoxModel<TreeType>(TreeType.values()));
139     treeComboBox.setSelectedIndex(treeTypeIndex);
140     treeComboBox.addItemListener(new ItemListener()
141     {
142         @Override
143         public void itemStateChanged(ItemEvent e)
144         {
145             @SuppressWarnings("unchecked")
146             JComboBox<TreeType> combo = (JComboBox<TreeType>) e.getSource();
147             if (e.getStateChange() == ItemEvent.SELECTED)
148             {
149                 Object selectedItem = combo.getSelectedItem();
150                 if (selectedItem instanceof TreeType)
151                 {
152                     setTreeType((TreeType) selectedItem);
153
154                     System.out.println("Setting tree type to " +
155                         selectedItem);
156                 }
157             }
158         }
159     });
160     treeModePanel.add(treeComboBox);
161
162     JScrollPane treeScrollPane = new JScrollPane();
163     treeScrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
164     add(treeScrollPane, BorderLayout.CENTER);
165
166     tree = new JTree();
167     treeScrollPane.setViewportView(tree);
168 }
169 }

```