```java
// src/ui/Selection.java
package ui;

import figure.Point_2D;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;

/**
 * Represente une zone de selection rectangulaire sur le canvas.
 */
public class Selection
{
    final static float dash1[] = {4.0f};
    final static BasicStroke dashed = new BasicStroke(1.0f, BasicStroke.CAP_BUTT,
                                        BasicStroke.JOIN_MITER, 4.0f, dash1, 0.0f);

    protected Color bg, stroke;
    protected Point_2D topleft;
    protected int width,height;

    public Selection(Color colorStroke, Color colorBackground, int x, int y, int width, int height) {
        this.bg = colorBackground;
        this.stroke = colorStroke;
        this.topleft = new Point_2D(x, y);
        this.width = width;
        this.height = height;
    }

    public Selection(Color colorStroke, Color colorBackground, Point_2D a, Point_2D b) {
        this(colorStroke, colorBackground, Math.min(a.getX(), b.getX()), Math.min(a.getY(), b.getY()),
                                           Math.abs(a.getX()-b.getX()), Math.abs(a.getY()-b.getY()));
    }

    /**
     * Dessine la selection
     * @param g
     */
    public void draw(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g.setColor(bg);
        g.fillRect(topleft.getX(), topleft.getY(), width, height);
        g.setColor(stroke);
        g2d.setStroke(dashed);
        g.drawRect(topleft.getX(), topleft.getY(), width, height);
    }

    public boolean contain(Point_2D p) {
        if ( topleft.getX() <= p.getX() && p.getX() <= topleft.getX()+width
          && topleft.getY() <= p.getY() && p.getY() <= topleft.getY()+height) {
            return true;
        }
        return false;
    }
}
// src/ui/CanvasKeyListener.java
package ui;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

import figure.FigureGraphic;

/**
```

```
 * Ecoute les evenements du clavier sur la zone de dessin
 */
public class CanvasKeyListener extends KeyAdapter {

    public CanvasArea canvas;
    public Env env;

    public CanvasKeyListener(CanvasArea canvas, Env env) {
        this.canvas = canvas;
        this.env = env;
    }

    public void keyReleased(KeyEvent e) {
        FigureGraphic figure;
        CanvasMouseListener cml = env.getCanvasMouseListener();
        switch(e.getKeyCode()) {
        case 10: // ENTER
            figure = cml.getBuildingFigure();
            if(figure!=null && figure.canBeFinishedWithKey()) cml.finishBuildingFigure();
            canvas.repaint();
            return;
        case 27: // ESCAPE
            figure = cml.getBuildingFigure();
            if(figure !=null) {
                cml.finishBuildingFigure();
                env.remove(figure);
            }
            canvas.repaint();
            return;
        case 127: // DELETE
            env.removeSelected();
            return;
        }
        switch(e.getKeyChar()) {
        case 'a':
            if(env.getFigures().size() == env.getSelected().size())
                env.unselectAll();
            else
                env.selectAll();
            return;
        }
    }

}

// src/ui/MenuBar.java
package ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.*;
import javax.swing.filechooser.FileFilter;

/**
 * Barre de menu
 */
@SuppressWarnings("serial")
public class MenuBar extends JMenuBar {

    protected File openedFile = null;
    protected Window parent;
    protected Env env;

    public MenuBar(final Window parent, final Env env) {
        this.parent = parent;
```

```
        this.env = env;
        JMenu file = new JMenu("Fichier");
        JMenuItem nouveau = new JMenuItem("Nouveau");
        JMenuItem open = new JMenuItem("Ouvrir");
        JMenuItem save = new JMenuItem("Sauvegarder");
        JMenuItem saveAs = new JMenuItem("Sauvegarder sous");
        nouveau.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if(!parent.confirm("Abandonner le travail actuel ?") )
                    return;
                env.empty();
                env.canvas.repaint();
            }
        });
        open.addActionListener(new OpenFileListener(this));
        save.addActionListener(new SaveFileListener(this, false));
        saveAs.addActionListener(new SaveFileListener(this, true));
        file.add(nouveau);
        file.add(open);
        file.add(save);
        file.add(saveAs);
        add(file);
    }

    protected void open(File f) {
        if(parent.confirm("Abandonner le travail actuel ?") )
            openedFile = env.openFromFile(f) ? f : null;
    }

    protected void save(File f) {
        openedFile = env.saveToFile(f) ? f : null;
    }

    protected class ShapeEditorFileFilter extends FileFilter {
        public String getDescription() {
            return "Shape Editor File (.sef)";
        }
        public boolean accept(File f) {
            return f.isDirectory() || f.getName().endsWith(".sef");
        }
    }

    protected class OpenFileListener implements ActionListener {

        MenuBar menu;
        public OpenFileListener(MenuBar menu) {
            this.menu = menu;
        }

        public void actionPerformed(ActionEvent e) {
            final JFileChooser fc = new JFileChooser();
            fc.setDialogTitle("Ouvrir");
            fc.setFileFilter(new ShapeEditorFileFilter());
            fc.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    if(e.getActionCommand().equals("ApproveSelection")) {
                        menu.open(fc.getSelectedFile());
                    }
                }
            });
            fc.showOpenDialog(menu.parent);
        }
    }

    protected class SaveFileListener implements ActionListener {

        MenuBar menu;
```

```
        boolean saveAs;
        public SaveFileListener(MenuBar menu, boolean saveAs) {
            this.menu = menu;
            this.saveAs = saveAs;
        }
        public void actionPerformed(ActionEvent e) {
            if(!saveAs && menu.openedFile!=null) {
                menu.save(menu.openedFile);
                return;
            }

            final JFileChooser fc = new JFileChooser();
            fc.setApproveButtonText("Enregistrer");
            fc.setDialogTitle("Enregistrer");
            fc.setFileFilter(new ShapeEditorFileFilter());
            fc.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    if(e.getActionCommand().equals("ApproveSelection")) {
                        File selected = fc.getSelectedFile();
                        if(!selected.getName().endsWith(".sef"))
                            selected = new File(selected.getAbsolutePath()+".sef");
                        menu.save(selected);
                    }
                }
            });
            fc.showSaveDialog(menu.parent);
        }
    }
}

// src/ui/CanvasMouseListener.java
package ui;

import java.awt.Color;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import ui.Mode;

import figure.*;

/**
 * Ecoute les evenements de la souris sur la zone de dessin
 */
public class CanvasMouseListener implements MouseListener, MouseMotionListener {

    boolean mouseIsDown = false;
    Env env;
    CanvasArea canvas;
    Point_2D lastPosition;

    FigureGraphic buildingFigure = null;

    public CanvasMouseListener(CanvasArea c, Env env) {
        this.canvas = c;
        this.env = env;
    }

    /**
     * @return la figure en cours de construction
     */
    public FigureGraphic getBuildingFigure() {
        return buildingFigure;
    }

    /**
```

```
     * Termine la figure actuelle
     */
    public void finishBuildingFigure() {
        buildingFigure.setBuilding(false);
        buildingFigure.onFigureFinish();
        buildingFigure = null;
    }

    /**
     * Fonction appelee quand le mode a change
     * @param mode
     */
    public void onToolChanged(Mode mode) {
        if(buildingFigure==null) return;
        env.getFigures().remove(buildingFigure);
        finishBuildingFigure();
        canvas.repaint();
    }

    /**
     * Enregistre une position comme derniere position
     * et retourne le deplacement associe au dernier enregistrement
     * @return
     */
    private Point_2D amassMove(int x, int y) {
        Point_2D move = new Point_2D(0, 0);
        if(lastPosition!=null) {
            move.setX(x - lastPosition.getX());
            move.setY(y - lastPosition.getY());
        }
        lastPosition = new Point_2D(x, y);
        return move;
    }

    public void mouseClicked(MouseEvent e) {
        Mode mode = canvas.getMode();
        switch(mode) {
        case MOVE:
        case SELECT:
            env.selectOneByPosition(new Point_2D(e.getX(), e.getY()));
            env.getToolbox().move.doClick();
            canvas.repaint();
        }
    }
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}

    public void mousePressed(MouseEvent e) {
        mouseIsDown = true;
        Mode mode = canvas.getMode();
        canvas.setSelection(null);
        FigureGraphic figure = null;
        if(mode==Mode.MOVE) {
            figure = env.getOneByPosition(new Point_2D(e.getX(), e.getY()));
            if(figure!=null && !figure.isSelected()) env.selectFigure(figure);
        }
        amassMove(e.getX(), e.getY());
        switch(mode) {
        case MOVE:
            if(figure==null) {
                env.unselectAll();
                env.getToolbox().select.doClick();
            }
            for(FigureGraphic f : env.getFigures())
                if(f.isSelected())
                    f.setTransparent(true);
            break;
```

```
        default:
            if(buildingFigure!=null) {
                if(buildingFigure.canBeFinishedWithMouse()) {
                    finishBuildingFigure();
                }
                else {
                    buildingFigure.onPressPoint(e.getX(), e.getY());
                }
            }
            else {
                try {
                    buildingFigure = mode.getDrawClass().newInstance();
                    buildingFigure.init(env, e.getX(), e.getY());
                    env.addFigure(buildingFigure);
                    env.onSelectionChanged();
                }
                catch (Exception exception) {}
            }
        }
        canvas.repaint();
    }

    public void mouseReleased(MouseEvent e) {
        mouseIsDown = false;
        Mode mode = canvas.getMode();
        canvas.setSelection(null);
        switch(mode) {
        case SELECT:
            if(env.countSelected()>0)
                env.getToolbox().move.doClick();
            for(FigureGraphic f : env.getFigures())
                f.setTransparent(false);
            break;
        case MOVE:
            Point_2D move = amassMove(e.getX(), e.getY());
            env.moveSelected(move.getX(), move.getY());
            for(FigureGraphic f : env.getFigures())
                f.setTransparent(false);
            break;
        default:
            if(buildingFigure!=null) {
                boolean canBeFinished = buildingFigure.canBeFinishedWithMouse();
                buildingFigure.onReleasePoint(e.getX(), e.getY());
                if(canBeFinished) finishBuildingFigure();
            }
        }
        canvas.repaint();
    }

    public void mouseDragged(MouseEvent e) {
        Mode mode = canvas.getMode();
        boolean mustRepaint = true;
        if(mode!=Mode.SELECT) canvas.setSelection(null);
        switch(mode) {
        case SELECT:
            Selection s = new Selection(new Color(120,120,120,150), new Color(120,120,120,50),
                                        new Point_2D(e.getX(), e.getY()), lastPosition);
            env.selectPoints(s);
            canvas.setSelection(s);
            break;
        case MOVE:
            Point_2D move = amassMove(e.getX(), e.getY());
            env.moveSelected(move.getX(), move.getY());
            break;
        default:
            if(buildingFigure!=null) buildingFigure.onMovePoint(e.getX(), e.getY());
            else mustRepaint = false;
```

```
            }
            if(mustRepaint) canvas.repaint();
        }

        public void mouseMoved(MouseEvent e) {
            if (buildingFigure != null) {
                buildingFigure.onMovePoint(e.getX(), e.getY());
                canvas.repaint();
            }
        }

    }

    // src/ui/Mode.java
    package ui;

    import figure.*;

    /**
     * Represente le mode de dessin actuel
     */
    public enum Mode {
        /**
         * mode Deplacement
         */
        MOVE,
        /**
         * mode Selection
         */
        SELECT,
        /**
         * mode Cercle
         */
        DRAW_CIRCLE,
        /**
         * mode Triangle
         */
        DRAW_TRIANGLE,
        /**
         * mode Rectangle
         */
        DRAW_RECTANGLE,
        /**
         * mode Polygon
         */
        DRAW_POLYGON;

        public Class<? extends FigureGraphic> getDrawClass() {
            switch(this) {
            case DRAW_CIRCLE: return Circle.class;
            case DRAW_TRIANGLE: return Triangle.class;
            case DRAW_RECTANGLE: return Rectangle.class;
            case DRAW_POLYGON: return Polygon.class;
            }
            return null;
        }
    }
    // src/ui/Window.java
    package ui;
    import java.awt.*;
    import java.awt.event.WindowAdapter;
    import java.awt.event.WindowEvent;

    import javax.swing.*;

    /**
     * La fenetre de l'application
```

```
 */
@SuppressWarnings("serial")
public class Window extends JFrame {

    private CanvasArea canvas;
    private MenuBar menu;
    public ToolBox toolbox;
    public SelectionPanel selectionPanel;

    private Env env = new Env(this);

    public Window() {
        setBounds(0, 0, 800, 600);
        setMinimumSize(new Dimension(400, 300));
        setLocationRelativeTo(null);
        setTitle("Shape Editor");
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                if(confirm("Quitter et abandonner ce dessin ?"))
                    System.exit(0);
            }
        });

        Container pane = getContentPane();
        pane.setLayout(new GridBagLayout());
        GridBagConstraints constraints = new GridBagConstraints();

        menu = new MenuBar(this, env);
        toolbox = new ToolBox(this, env);
        selectionPanel = new SelectionPanel(this, env);
        canvas = new CanvasArea(env);
        CanvasMouseListener cml = new CanvasMouseListener(canvas, env);
        CanvasKeyListener ckl = new CanvasKeyListener(canvas, env);

        env.setToolbox(toolbox);
        env.setSelectionPanel(selectionPanel);
        env.setCanvas(canvas);
        env.setCanvasMouseListener(cml);

        canvas.addMouseListener(cml);
        canvas.addMouseMotionListener(cml);
        canvas.addKeyListener(ckl);

        setJMenuBar(menu);
        constraints.insets = new Insets(2, 2, 2, 2);
        constraints.fill = GridBagConstraints.BOTH;
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.weightx = 0;
        constraints.weighty = 0;
        pane.add(toolbox, constraints);

        constraints.gridx = 0;
        constraints.gridy = 1;
        pane.add(selectionPanel, constraints);

        constraints.gridx = 1;
        constraints.gridy = 0;
        constraints.gridheight = 2;
        constraints.weightx = 1.0;
        constraints.weighty = 1.0;
        pane.add(canvas, constraints);

        setVisible(true);
    }
```

```
    /**
     * Pose une question a l'utilisateur de type Oui/Non a travers une popup
     * @param message : la question
     * @param title : un titre
     * @return la reponse booleenne
     */
    public boolean confirm(String message, String title) {
        return JOptionPane.OK_OPTION == JOptionPane.showConfirmDialog(this, message, title,
                                        JOptionPane.OK_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE);
    }

    /**
     * Pose une question a l'utilisateur de type Oui/Non a travers une popup
     * @param message : la question
     * @return la reponse booleenne
     */
    public boolean confirm(String message) {
        return confirm(message, null);
    }

    /**
     * Affiche un message d'erreur dans une popup
     * @param message : le message
     */
    public void error(String message) {
        JOptionPane.showConfirmDialog(this, message, null, JOptionPane.CLOSED_OPTION, JOptionPane.ERROR_MES
    }

    public static void main(String[] args) {
        new Window();
    }
}

// src/ui/ToolBox.java
package ui;

import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.*;

import ui.Mode;

/**
 * Boite a outils qui regroupe les boutons des differents modes de l'application ainsi que les choix de cou
 */
@SuppressWarnings("serial")
public class ToolBox extends JPanel {

    public JButton select = new JButton(new ImageIcon("select.png"));
    public JButton move = new JButton(new ImageIcon("move.png"));
    public JButton newCircle = new JButton(new ImageIcon("circle.png"));
    public JButton newTriangle = new JButton(new ImageIcon("triangle.png"));
    public JButton newRectangle = new JButton(new ImageIcon("rectangle.png"));
    public JButton newPolygon = new JButton(new ImageIcon("polygon.png"));
    protected List<JButton> buttons = new ArrayList<JButton>();

    protected Canvas bgColor = new Canvas();
```

```
protected Canvas strokeColor = new Canvas();
protected JLabel bgLabel = new JLabel("fond");
protected JLabel strokeLabel = new JLabel("contour");

protected Env env;
protected Window window;

private JButton addImageButton(JButton b) {
    Dimension d = new Dimension(36, 36);
    b.setPreferredSize(d);
    b.setMinimumSize(d);
    b.setMaximumSize(d);
    buttons.add(b);
    return b;
}
private void select(JButton button) {
    for(JButton b : buttons) {
        b.setSelected(false);
        b.setBackground(null);
    }
    button.setSelected(true);
    button.setBackground(Color.white);
}

public ToolBox(Window window, Env env) {
    this.env = env;
    this.window = window;
    setBorder(BorderFactory.createEtchedBorder());
    GridBagLayout l = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    setLayout(l);

    c.gridx = 0;
    c.gridy = 0;
    c.anchor = GridBagConstraints.WEST;

    JPanel panel = new JPanel();

    panel.add(addImageButton(select));
    panel.add(addImageButton(move));

    add(panel, c);
    c.gridy++;
    panel = new JPanel();
    panel.add(addImageButton(newCircle));
    panel.add(addImageButton(newTriangle));
    panel.add(addImageButton(newRectangle));
    panel.add(addImageButton(newPolygon));
    add(panel, c);

    c.gridy++;
    c.anchor = GridBagConstraints.CENTER;

    panel = new JPanel();
    panel.add(strokeColor);
    panel.add(strokeLabel);
    panel.add(bgColor);
    panel.add(bgLabel);
    add(panel, c);
    bgColor.setSize(20, 20);
    strokeColor.setSize(20, 20);
    bgColor.setBackground(env.getBackgroundColor());
    strokeColor.setBackground(env.getStrokeColor());

    final Window w = window;
    final Env e = env;
    MouseListener clickBg = new MouseListener() {
```

```
            public void mouseReleased(MouseEvent arg0) {}
            public void mousePressed(MouseEvent arg0) {}
            public void mouseExited(MouseEvent arg0) {}
            public void mouseEntered(MouseEvent arg0) {}
            public void mouseClicked(MouseEvent arg0) {
                Color c = JColorChooser.showDialog(w, "Couleur de fond", e.getBackgroundColor());
                if(c==null) return;
                e.setBackgroundColor(c);
                bgColor.setBackground(c);
            }
        };
        bgLabel.addMouseListener(clickBg);
        bgColor.addMouseListener(clickBg);

        MouseListener clickStroke = new MouseListener() {
            public void mouseReleased(MouseEvent arg0) {}
            public void mousePressed(MouseEvent arg0) {}
            public void mouseExited(MouseEvent arg0) {}
            public void mouseEntered(MouseEvent arg0) {}
            public void mouseClicked(MouseEvent arg0) {
                Color c = JColorChooser.showDialog(w, "Couleur de contour", e.getStrokeColor());
                if(c==null) return;
                e.setStrokeColor(c);
                strokeColor.setBackground(c);
            }
        };
        strokeLabel.addMouseListener(clickStroke);
        strokeColor.addMouseListener(clickStroke);

        select.addActionListener(new SelectListener());
        move.addActionListener(new MoveListener());
        newCircle.addActionListener(new NewCircleListener());
        newTriangle.addActionListener(new NewTriangleListener());
        newRectangle.addActionListener(new NewRectangleListener());
        newPolygon.addActionListener(new NewPolygonListener());
        select(move);
    }

    class SelectListener extends ButtonListener {
        public SelectListener() {
            super(Mode.SELECT);
        }
        public void actionPerformed(ActionEvent e) {
            super.actionPerformed(e);
        }
    }
    class MoveListener extends ButtonListener {
        public MoveListener() {
            super(Mode.MOVE);
        }
        public void actionPerformed(ActionEvent e) {
            super.actionPerformed(e);
        }
    }
    class NewCircleListener extends ButtonListener {
        public NewCircleListener() {
            super(Mode.DRAW_CIRCLE);
        }
        public void actionPerformed(ActionEvent e) {
            super.actionPerformed(e);
        }
    }
    class NewTriangleListener extends ButtonListener {
        public NewTriangleListener() {
            super(Mode.DRAW_TRIANGLE);
        }
        public void actionPerformed(ActionEvent e) {
```

```
            super.actionPerformed(e);
        }
    }
    class NewRectangleListener extends ButtonListener {
        public NewRectangleListener() {
            super(Mode.DRAW_RECTANGLE);
        }
        public void actionPerformed(ActionEvent e) {
            super.actionPerformed(e);
        }
    }
    class NewPolygonListener extends ButtonListener {
        public NewPolygonListener() {
            super(Mode.DRAW_POLYGON);
        }
        public void actionPerformed(ActionEvent e) {
            super.actionPerformed(e);
        }
    }

    class ButtonListener implements ActionListener {
        Mode mode;
        public ButtonListener(Mode mode) {
            this.mode = mode;
        }
        public void actionPerformed(ActionEvent e) {
            if(e.getSource() instanceof JButton)
                select((JButton)e.getSource());
            env.getCanvas().setMode(mode);
            env.getCanvasMouseListener().onToolChanged(mode);
        }
    }
}

// src/ui/Env.java
package ui;

import java.awt.Color;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import figure.*;

/**
 * Variable d'environnement qui contient tous le contexte du dessin actuel
 */
public class Env {

    protected Data data = new Data();

    protected Window window;

    protected CanvasArea canvas;
    protected CanvasMouseListener canvasMouseListener;

    protected ToolBox toolbox;
    protected SelectionPanel selectionPanel;

    protected Color bg = new Color(150, 150, 250);
    protected Color stroke = Color.BLACK;
```

```
/**
 * Exportable data
 */
@SuppressWarnings("serial")
protected static class Data implements Serializable {
    /**
     * La liste des figures est trie dans l'ordre de priorite d'affichage
     */
    List<FigureGraphic> figures = new ArrayList<FigureGraphic>();
}

public Env(Window window) {
    this.window = window;
}

public CanvasMouseListener getCanvasMouseListener() {
    return canvasMouseListener;
}
public void setCanvasMouseListener(CanvasMouseListener canvasMouseListener) {
    this.canvasMouseListener = canvasMouseListener;
}

public Color getBackgroundColor() {
    return bg;
}
public Color getStrokeColor() {
    return stroke;
}
public void setBackgroundColor(Color c) {
    bg = c;
}
public void setStrokeColor(Color c) {
    stroke = c;
}

public void setData(Data d) {
    data = d;
    canvas.repaint();
}

public void setCanvas(CanvasArea c) {
    canvas = c;
}
public CanvasArea getCanvas() {
    return canvas;
}

public void setToolbox(ToolBox t) {
    toolbox = t;
}
public ToolBox getToolbox() {
    return toolbox;
}
public void setSelectionPanel(SelectionPanel t) {
    selectionPanel = t;
}
public SelectionPanel getSelectionPanel() {
    return selectionPanel;
}

public List<FigureGraphic> getFigures() {
    return data.figures;
}
public void setFigures(List<FigureGraphic> figures) {
    data.figures = figures;
}
```

```java
/**
 * Trie les figures de telle sorte que les figures selectionnees apparaissent en premier.
 * Outre la selection, l'ordre est conserve.
 */
public void sortFigures() {
    List<FigureGraphic> newfigures = new ArrayList<FigureGraphic>();
    for(FigureGraphic f : data.figures)
        if(f.isSelected())
            newfigures.add(f);
    for(FigureGraphic f : data.figures)
        if(!f.isSelected())
            newfigures.add(f);
    data.figures = newfigures;
}

/**
 * Sauvegarde le dessin actuel dans un fichier
 * @param f le fichier
 * @return true si sauvegarde avec succes, false sinon
 */
public boolean saveToFile(File f) {
    try {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(f));
        oos.writeObject(data);
        return true;
    }
    catch(Exception e) {
        e.printStackTrace();
        window.error("Impossible de sauvegarder le dessin dans le fichier choisi");
    }
    return false;
}

/**
 * Ouvrir un dessin depuis un fichier
 * @param f le fichier a ouvrir
 * @return true si ouvert avec succes, false si l'ouverture du dessin a echoue
 *                                      (fichier incompatible, erreur de fichier)
 */
public boolean openFromFile(File f) {
    try {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(f));
        setData((Data) ois.readObject());
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        window.error("Impossible d'ouvrir le fichier");
    }
    return false;
}

/**
 * Ajoute une figure au dessin
 */
public void addFigure(FigureGraphic f) {
    getFigures().add(0, f);
}

/**
 * Reinitialise le dessin (supprime toutes les figures)
 */
public void empty() {
    data = new Data();
}

/**
 * compare deux listes
```

```
     * @param f
     * @param g
     * @return true si les deux listes sont egales
     */
    public boolean listsAreSame(List<FigureGraphic> f, List<FigureGraphic> g) {
        return f.size() == g.size() && f.containsAll(g);
    }

    private List<FigureGraphic> lastSelection = new ArrayList<FigureGraphic>();
    /**
     * Fonction appele quand la selection a change
     */
    public void onSelectionChanged() {
        List<FigureGraphic> s = new ArrayList<FigureGraphic>( getSelected() );
        if(!listsAreSame(s, lastSelection)) {
            canvas.repaint();
            selectionPanel.onSelectionChanged();
        }
        lastSelection = s;
    }

    // Selection

    /**
     * Annule la selection actuelle
     */
    private void emptySelection() {
        for(FigureGraphic f : getFigures())
            setSelected(f, false);
    }

    /**
     * Dï¿½selectionne toutes les figures
     */
    public void unselectAll() {
        emptySelection();
        onSelectionChanged();
    }

    /**
     * Recherche la premiere figure qui contient le point p
     * @param p
     * @return la premiere figure trouvee sous le point p , null si aucune figure trouvee
     */
    public FigureGraphic getOneByPosition(Point_2D p) {
        for(FigureGraphic f : getFigures())
            if(f.contain(p))
                return f;
        return null;
    }

    private void setSelected(FigureGraphic figure, boolean value) {
        figure.setSelected(value);
        figure.setTransparent(value && canvasMouseListener.mouseIsDown);
    }

    /**
     * Recupere les figures selectionnees
     * @return une liste de toutes les figures selectionnees
     */
    public List<FigureGraphic> getSelected() {
        List<FigureGraphic> figures = new ArrayList<FigureGraphic>();
        for(FigureGraphic f : getFigures())
            if(f.isSelected())
                figures.add(f);
        return figures;
    }
```

```java
/**
 * Selectionne une et une seule figure
 * @param figure : la figure a selectionner
 */
public void selectFigure(FigureGraphic figure) {
    emptySelection();
    setSelected(figure, true);
    sortFigures();
    onSelectionChanged();
}

/**
 * Selectionne la premiere figure qui contient p
 * @param p : le point
 * @return la figure selectionne ou null
 */
public FigureGraphic selectOneByPosition(Point_2D p) {
    emptySelection();
    FigureGraphic figure = getOneByPosition(p);
    if(figure!=null) {
        setSelected(figure, true);
        sortFigures();
    }
    onSelectionChanged();
    return figure;
}

/**
 * Selectionne un ensemble de figures grace a une selection.
 * Le centre des figures est determinant pour savoir si elles appartiennent a cette selection.
 * @param selection
 */
public void selectPoints(Selection selection) {
    for(FigureGraphic f : getFigures())
        setSelected(f, selection.contain(f.getCenter()));
    sortFigures();
    onSelectionChanged();
}

/**
 * Selectionne toutes les figures.
 */
public void selectAll() {
    for(FigureGraphic f : getFigures())
        f.setSelected(true);
    onSelectionChanged();
}

/**
 * @return le nombre de figures selectionnees
 */
public int countSelected() {
    return getSelected().size();
}

/**
 * Deplace toutes les figures selectionnees
 * @param dx : deplacement en x
 * @param dy : deplacement en y
 */
public void moveSelected(int dx, int dy) {
    for(FigureGraphic f : getSelected())
        f.move(dx, dy);
}

/**
```

```
     * Supprime une figure du dessin
     * @param figure
     */
    public void remove(Figure figure) {
        getFigures().remove(figure);
        onSelectionChanged();
    }

    /**
     * Supprime les figures selectionnees du dessin
     */
    public void removeSelected() {
        List<FigureGraphic> figures = getFigures();
        for(FigureGraphic f : getSelected())
            figures.remove(f);
        onSelectionChanged();
    }

}

// src/ui/SelectionPanel.java
package ui;

import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.*;

import figure.FigureGraphic;

/**
 * Panneau de type accordeon qui affichent les details sur les figures actuellement selectionnees
 * avec la possibilite d'editer ces figures.
 */
@SuppressWarnings("serial")
public class SelectionPanel extends JScrollPane {
    protected Window window;
    protected Env env;
    protected JPanel panel;
    protected List<FigureObj> objs;

    public SelectionPanel(Window window, Env env) {
        this.window = window;
        this.env = env;
        setViewportView(panel = new JPanel());
        panel.setLayout(new GridBagLayout());
        onSelectionChanged();
    }

    /**
     * Fonction appelee quand la selection a change
     */
    public void onSelectionChanged() {
        panel.setVisible(false);
        panel.removeAll();
```

```
        GridBagConstraints constraints = new GridBagConstraints();
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.anchor = GridBagConstraints.NORTH;
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.weightx = 1;
        constraints.weighty = 1;
        JLabel title = new JLabel("Figure(s) selectionnee(s)");
        Font font = title.getFont();
        title.setFont(font.deriveFont(font.getStyle() ^ Font.BOLD));
        panel.add(title, constraints);
        int y = 1;
        objs = new ArrayList<SelectionPanel.FigureObj>();
        for(FigureGraphic figure : env.getSelected()) {
            constraints.gridy = y;
            FigureObj obj = new FigureObj(this, figure);
            panel.add(obj, constraints);
            ++ y;
            objs.add(obj);
        }
        if(objs.size()>0) {
            FigureObj obj = objs.get(0);
            obj.setOpened(true);
        }
        panel.setVisible(true);
    }

    protected void closeAll() {
        for(FigureObj obj : objs) {
            obj.setOpened(false);
            obj.button.setEnabled(true);
        }
    }

    public class FigureObj extends JPanel {
        public SelectionPanel parent;
        public FigureGraphic figure;
        public JButton button;
        public JPanel options;

        public void setOpened(boolean opened) {
            options.setVisible(opened);
            button.setVisible(!opened);
        }
        public void open() {
            parent.closeAll();
            setOpened(true);
        }
        public FigureObj(final SelectionPanel parent, final FigureGraphic figure) {
            this.parent = parent;
            this.figure = figure;
            setLayout(new BorderLayout());
            button = new JButton(figure.getName());
            button.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    open();
                }
            });
            add(button, BorderLayout.NORTH);
            options = new JPanel();
            options.setLayout(new GridBagLayout());
            GridBagConstraints constraints = new GridBagConstraints();
            constraints.fill = GridBagConstraints.HORIZONTAL;
            constraints.anchor = GridBagConstraints.NORTH;
            constraints.gridwidth = 1;
            constraints.gridx = 0;
            constraints.gridy = 0;
```

```
final JTextField nameField = new JTextField(figure.getName());

options.add(new JLabel("Nom: "), constraints);
++ constraints.gridx;
options.add(nameField, constraints);
++constraints.gridy;
constraints.gridx = 0;

options.add(new JLabel("Forme: "), constraints);
++ constraints.gridx;
options.add(new JLabel(figure.getShapeName()), constraints);
++constraints.gridy;
constraints.gridx = 0;

JLabel bgLabel = new JLabel("Fond: ");
final Canvas bgColor = new Canvas();
bgColor.setSize(20, 20);
bgColor.setBackground(figure.getBackgroundColor());
options.add(bgLabel, constraints);
++ constraints.gridx;
options.add(bgColor, constraints);
++constraints.gridy;
constraints.gridx = 0;

JLabel strokeLabel = new JLabel("Contour: ");
final Canvas strokeColor = new Canvas();
strokeColor.setSize(20, 20);
strokeColor.setBackground(figure.getStrokeColor());
options.add(strokeLabel, constraints);
++ constraints.gridx;
options.add(strokeColor, constraints);
++constraints.gridy;
constraints.gridx = 0;

add(options, BorderLayout.CENTER);

MouseListener clickBg = new MouseListener() {
    public void mouseReleased(MouseEvent arg0) {}
    public void mousePressed(MouseEvent arg0) {}
    public void mouseExited(MouseEvent arg0) {}
    public void mouseEntered(MouseEvent arg0) {}
    public void mouseClicked(MouseEvent arg0) {
        Color c = JColorChooser.showDialog(window, "Couleur de fond", figure.getBackgroundColor
        if(c==null) return;
        figure.setBackgroundColor(c);
        bgColor.setBackground(c);
        env.getCanvas().repaint();
    }
};
MouseListener clickStroke = new MouseListener() {
    public void mouseReleased(MouseEvent arg0) {}
    public void mousePressed(MouseEvent arg0) {}
    public void mouseExited(MouseEvent arg0) {}
    public void mouseEntered(MouseEvent arg0) {}
    public void mouseClicked(MouseEvent arg0) {
        Color c = JColorChooser.showDialog(window, "Couleur de contour", figure.getStrokeColor(
        if(c==null) return;
        figure.setStrokeColor(c);
        strokeColor.setBackground(c);
        env.getCanvas().repaint();
    }
};

bgLabel.addMouseListener(clickBg);
bgColor.addMouseListener(clickBg);
strokeLabel.addMouseListener(clickStroke);
```

```
            strokeColor.addMouseListener(clickStroke);

            nameField.addKeyListener(new KeyAdapter() {
                public void keyReleased(KeyEvent e) {
                    String name = nameField.getText();
                    if(name.length()==0) return;
                    figure.setName(name);
                    button.setText(name);
                    env.getCanvas().repaint();
                }
            });

            setOpened(false);
        }
    }
}

// src/ui/CanvasArea.java
package ui;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.RenderingHints;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import figure.*;

/**
 * La zone de dessin
 */
@SuppressWarnings("serial")
public class CanvasArea extends Canvas {
    protected Env env;
    private Mode mode = Mode.MOVE;
    protected Selection selection = null;

    public CanvasArea(Env env) {
        this.env = env;
        setBackground(Color.white);
    }

    public void setMode(Mode m) {
        this.mode = m;
    }
    public Mode getMode() {
        return this.mode;
    }

    public void setSelection(Selection s) {
        selection = s;
    }

    @Override
    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        List<FigureGraphic> figures = new ArrayList<FigureGraphic>(env.getFigures());
        Collections.reverse(figures);
        for(FigureGraphic f : figures)
            f.draw(g);
        if(selection!=null)
            selection.draw(g);
```

```
    }

    /**
     * Double buffering implementation
     */
    @Override
    public void update(Graphics g) {
        Graphics offgc;
        Image offscreen = null;
        @SuppressWarnings("deprecation")
        Dimension d = size();
        // create the offscreen buffer and associated Graphics
        offscreen = createImage(d.width, d.height);
        offgc = offscreen.getGraphics();
        // clear the exposed area
        offgc.setColor(getBackground());
        offgc.fillRect(0, 0, d.width, d.height);
        offgc.setColor(getForeground());
        // do normal redraw
        paint(offgc);
        // transfer offscreen to window
        g.drawImage(offscreen, 0, 0, this);
    }
}

// src/figure/Rectangle.java
package figure;

import figure.FigureGraphic;
import figure.Point_2D;

import java.awt.Color;
import java.awt.Graphics;
import java.io.Serializable;

import ui.Env;

@SuppressWarnings("serial")
/**
 * Class for rectangle figure
 */
public class Rectangle extends FigureGraphic implements Serializable
{
    private static long nbOfRectangles = 0;

    protected Point_2D a, b;

    public Rectangle(String name) {
        super(name);
        ++nbOfRectangles;
    }

    /**
    * Constructor that will generate by itself the name
    */
    public Rectangle() {
        this("rect_"+(nbOfRectangles+1));
    }

    /**
    * Constructor
    * @param String name
    * @param Color stroke
    * @param Color bg background color
    * @param int x position
    * @param int y position
    * @param int w width
```

```
     * @param int h height
     */
    public Rectangle(String name, Color stroke, Color bg, int x, int y, int w, int h) {
        this(name);
        setColors(stroke, bg);
        setFirstPoint(x, y);
        setSecondPoint(x+w, y+h);
    }

    /**
     * Create by first click
     * @param env
     * @param x
     * @param y
     */
    public void init(Env env, int x, int y) {
        setColors(env);
        a = new Point_2D(x, y);
        b = new Point_2D(x, y);
        setSelected(true);
        setBuilding(true);
    }

    public Point_2D getCenter() {
        return new Point_2D ((a.getX()+b.getX())/2, (a.getY()+b.getY())/2);
    }

    public int getWidth() {
        return Math.abs(a.getX() - b.getX());
    }

    public int getHeight() {
        return Math.abs(a.getY() - b.getY());
    }

    public Point_2D getTopLeft() {
        return new Point_2D(Math.min(a.getX(), b.getX()), Math.min(a.getY(), b.getY()));
    }

    public void move(int dx, int dy) {
        a.move(dx, dy);
        b.move(dx, dy);
    }

    public void draw(Graphics g) {
        int width = getWidth();
        int height = getHeight();
        Point_2D topleft = getTopLeft();
        g.setColor(getBgForCurrentState());
        g.fillRect(topleft.getX(), topleft.getY(), width, height);
        g.setColor(getStrokeForCurrentState());
        g.drawRect(topleft.getX(), topleft.getY(), width, height);
        afterDraw(g);
    }

    public boolean contain(Point_2D p) {
        int width = getWidth();
        int height = getHeight();
        Point_2D topleft = getTopLeft();
        if ( topleft.getX() <= p.getX() && p.getX() <= topleft.getX()+width
          && topleft.getY() <= p.getY() && p.getY() <= topleft.getY()+height) {
            return true;
        }
        return false;
    }

    public void setFirstPoint(int x, int y) {
```

```
        a = new Point_2D(x, y);
    }
    public void setSecondPoint(int x, int y) {
        b = new Point_2D(x, y);
    }

    private boolean canBeFinished() {
        return getWidth()>THRESHOLD_BUILDING_PX && getHeight()>THRESHOLD_BUILDING_PX;
    }
    public boolean canBeFinishedWithKey() {
        return canBeFinished();
    }
    public boolean canBeFinishedWithMouse() {
        return canBeFinished();
    }

    @Override
    public void onFigureFinish() {
    }

    @Override
    public void onPressPoint(int x, int y) {
    }

    @Override
    public void onReleasePoint(int x, int y) {
        setSecondPoint(x, y);
    }

    @Override
    public void onMovePoint(int x, int y) {
        setSecondPoint(x, y);
    }

    @Override
    public String getShapeName() {
        return "Rectangle";
    }
}
// src/figure/Circle.java
package figure;

import figure.FigureGraphic;
import figure.Point_2D;

import java.awt.Color;
import java.awt.Graphics;
import java.io.Serializable;

import ui.Env;

@SuppressWarnings("serial")
/**
 * Class for circle figure
 */
public class Circle extends FigureGraphic implements Serializable
{
    private static long nbOfCircles = 0;

    protected Point_2D center;
    protected int radius;

    public Circle(String name) {
        super(name);
        ++nbOfCircles;
    }
```

```java
/**
* Constructor that will generate by itself the name
*/
public Circle() {
    this("circle_"+(nbOfCircles+1));
}

/**
* Constructor
* @param String name
* @param Color stroke
* @param Color bg background color
* @param int x position
* @param int y position
* @param int radius of the cercle
*/
public Circle(String name, Color stroke, Color bg, int x, int y, int radius) {
    super(name);
    setColors(stroke, bg);
    setRadius(radius);
    setCenter(x, y);
}

public void init(Env env, int x, int y) {
    setColors(env);
    setCenter(x, y);
    setSelected(true);
    setBuilding(true);
}

public Point_2D getCenter() {
    return center;
}
public void setCenter(int x, int y) {
    center = new Point_2D(x, y);
}

public void move(int dx, int dy) {
    center.move(dx,dy);
}

public void setRadius(int rad) {
    radius = rad;
}
public int getRadius() {
    return radius;
}

public void fitRadiusWithPoint(int x, int y) {
    x -= center.getX();
    y -= center.getY();
    int radius = (int)Math.sqrt(x*x+y*y);
    setRadius(radius);
}

public void draw(Graphics g) {
    Point_2D c = getCenter();
    g.setColor(getBgForCurrentState());
    g.fillOval(c.x-radius, c.y-radius, radius*2, radius*2);
    g.setColor(getStrokeForCurrentState());
    g.drawOval(c.x-radius, c.y-radius, radius*2, radius*2);
    afterDraw(g);
}

public boolean contain(Point_2D p) {
    return (Point_2D.distance(center, p) < radius);   }
```

```
    public double getSurface () {
        return Math.PI*radius*radius;
    }

    private boolean canBeFinished() {
        return 2*radius > THRESHOLD_BUILDING_PX;
    }
    public boolean canBeFinishedWithKey() {
        return canBeFinished();
    }
    public boolean canBeFinishedWithMouse() {
        return canBeFinished();
    }

    @Override
    public void onFigureFinish() {

    }

    @Override
    public void onPressPoint(int x, int y) {
        setCenter(x, y);
    }

    @Override
    public void onReleasePoint(int x, int y) {
        fitRadiusWithPoint(x, y);
    }

    @Override
    public void onMovePoint(int x, int y) {
        fitRadiusWithPoint(x, y);
    }

    @Override
    public String getShapeName() {
        return "Cercle";
    }
}
// src/figure/Point_2D.java
package figure;^M
^M
import java.io.Serializable;^M
^M
@SuppressWarnings("serial")^M
/**^M
* A point on the x,y axis^M
*/^M
public class Point_2D implements Serializable^M
{^M
    protected int x;^M
    protected int y;^M
^M
    public Point_2D ()^M
    {^M
        x = 0;^M
        y = 0;^M
    }^M
^M
    public Point_2D ( int x, int y)^M
    {^M
        this.x = x;^M
        this.y = y;^M
    }^M
^M
    public Point_2D (Point_2D p)^M
    {^M
```

```
        x = p.x;^M
        y = p.y;^M
    }^M
^M
    // Accesseurs^M
     ^M
    public int getX () { return x; }^M
    public int getY () { return y; }^M
    public void setX (int val) { x = val; }^M
    public void setY (int val) { y = val; }^M
^M
    /**^M
     * Affichage contenu^M
     */^M
^M
    public String toString()^M
    {^M
        return new String ("("+getX()+", "+getY()+")");^M
    }^M
^M
    public void move(int dx, int dy)^M
    {^M
        this.x += dx;^M
        this.y += dy;^M
    }^M
^M
    public static double distance(Point_2D p1, Point_2D p2)^M
    {^M
        int dx = p1.x-p2.x;^M
        int dy = p1.y-p2.y;^M
^M
        return Math.sqrt((dx*dx)+(dy*dy));^M
    }^M
^M
    public double distance(Point_2D p)^M
    {^M
        int dx = x-p.x;^M
        int dy = y-p.y;^M
^M
        return ((int)Math.sqrt((dx*dx)+(dy*dy)));^M
    }^M
^M
    public static boolean equal (Point_2D p1, Point_2D p2)^M
    {^M
        return (Point_2D.distance(p1,p2) < 1);^M
    }^M
^M
^M
}
// src/figure/Polygon.java
package figure;

import figure.FigureGraphic;
import figure.Point_2D;

import java.awt.Color;
import java.awt.Graphics;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import ui.Env;

@SuppressWarnings("serial")
/**
 * Class for polygon figure
 */
```

```java
public class Polygon extends FigureGraphic implements Serializable
{
    // Taille de la poignee de terminaison
    protected static final int FINISH_HANDLE_RADIUS_PX = 15;

    private static long nbOfPolygons = 0;

    protected List<Point_2D> points = new ArrayList<Point_2D>();
    protected int[] xAll, yAll;

    public Polygon(String name) {
        super(name);
        ++nbOfPolygons;
    }

    /**
    * Constructor that will generate by itself the name
    */
    public Polygon() {
        this("poly_"+(nbOfPolygons+1));
    }

    /**
    * Constructor
    * @param String name
    * @param Color stroke
    * @param Color bg background color
    */
    public Polygon(String name, Color stroke, Color bg) {
        this(name);
        setColors(stroke, bg);
    }
    /**
     * Create by first click
     * @param env
     * @param x
     * @param y
     */
    public void init(Env env, int x, int y) {
        setColors(env);
        addPoint(x, y);
        addPoint(x, y);
        setSelected(true);
        setBuilding(true);
    }

    public Point_2D getCenter() {
        int xSum=0, ySum=0;
        for (Point_2D point : points) {
            xSum += point.x;
            ySum += point.y;
        }
        return new Point_2D( xSum / points.size(), ySum / points.size());
    }

    public void move(int dx, int dy) {
        for (Point_2D point : points)
            point.move(dx,dy);
        updateXYAll();
    }
    public void addPoint(int x, int y) {
        points.add(new Point_2D(x,y));
        updateXYAll();
    }

    /**
    * Change last added point
```

```
*/
public void editLastPoint(Point_2D p) {
    int size = points.size();
    if(size==0) {
        points.add(p);
    }
    else {
        points.set(points.size()-1, p);
    }
    updateXYAll();
}
/**
* Change last added point
*/
public void editLastPoint(int x, int y) {
    editLastPoint(new Point_2D(x, y));
}

protected void updateXYAll() {
    xAll = new int[points.size()];
    yAll = new int[points.size()];

    int i=0;
    for (Point_2D point : points) {
        xAll[i] = point.x;
        yAll[i] = point.y;
        i++;
    }
}

public void draw(Graphics g) {
    if(isBuilding()) {
        g.setColor(Color.black);
        Point_2D last = null, first = null;
        for(Point_2D p : points) {
            if(last!=null)
                g.drawLine(last.x, last.y, p.x, p.y);
            else
                first = p;
            last = p;
        }
        if(drawTerminaisonEnabled() && canBeFinished()) {
            // Dessine une poignee de terminaison
            if(canBeFinishedWithMouse()) // mouseover
                g.setColor(new Color(255, 100, 100));
            else
                g.setColor(Color.white);
            g.fillOval(first.x-FINISH_HANDLE_RADIUS_PX/2, first.y-FINISH_HANDLE_RADIUS_PX/2,
                    FINISH_HANDLE_RADIUS_PX, FINISH_HANDLE_RADIUS_PX);
            g.setColor(new Color(0, 0, 0));
            g.drawOval(first.x-FINISH_HANDLE_RADIUS_PX/2, first.y-FINISH_HANDLE_RADIUS_PX/2,
                    FINISH_HANDLE_RADIUS_PX, FINISH_HANDLE_RADIUS_PX);
        }

    }
    else {
        g.setColor(getBgForCurrentState());
        g.fillPolygon(xAll, yAll, points.size());
        g.setColor(getStrokeForCurrentState());
        g.drawPolygon(xAll, yAll, points.size());
    }
    afterDraw(g);
}

public boolean contain(Point_2D p) {
    int nvert = points.size();
    int[] verty = yAll;
```

```
        int[] vertx = xAll;
        int testx = p.x;
        int testy = p.y;

        int i, j;
        boolean c = false;
        for (i = 0, j = nvert-1; i < nvert; j = i++) {
          if ( ((verty[i]>testy) != (verty[j]>testy)) &&
                 (testx < (vertx[j]-vertx[i]) * (testy-verty[i]) / (verty[j]-verty[i]) + vertx[i]) )
               c = !c;
        }

        return c;
    }

    protected boolean drawTerminaisonEnabled() {
        return true;
    }

    private boolean canBeFinished() {
        return points.size() > 3;
    }
    public boolean canBeFinishedWithKey() {
        return canBeFinished();
    }
    public boolean canBeFinishedWithMouse() {
        return canBeFinished() && points.get(0).distance(points.get(points.size()-1))<FINISH_HANDLE_RADIUS_
    }

    public void closePath() {
        points.remove(points.size()-1);
    }

    @Override
    public void onFigureFinish() {
        closePath();
    }

    @Override
    public void onPressPoint(int x, int y) {
        addPoint(x, y);
    }
    @Override
    public void onReleasePoint(int x, int y) {

    }

    @Override
    public void onMovePoint(int x, int y) {
        editLastPoint(x, y);
    }
    @Override
    public String getShapeName() {
        return "Polygone";
    }
}
// src/figure/FigureGraphic.java
package figure;^M
import java.awt.*;^M
import java.io.Serializable;^M
^M
import ui.Env;^M
^M
@SuppressWarnings("serial")^M
/**^M
 * Abstract class for figures that propose functions for UI interaction^M
 */^M
```

```
public abstract class FigureGraphic implements Figure, Serializable^M
{^M
    protected Color colorStroke, colorBackground;^M
    protected String name;^M
    ^M
    /**^M
     * Seuil de terminaison d'une figure en pixel^M
     * Cela correspond au seuil de visibilite d'une figure / du rapprochement de deux points. ^M
     * En dessous, on considere que c'est trop petit pour etre valide^M
     */^M
    protected static final int THRESHOLD_BUILDING_PX = 8;^M
    ^M
    /**^M
     * A selected figure should display differently^M
     */^M
    protected boolean selected = false;^M
    ^M
    protected boolean transparent = false;^M
    ^M
    protected boolean building = false; // L'objet est en train d'etre construit^M

    public boolean isBuilding() {^M
        return building;^M
    }^M
^M
    public void setBuilding(boolean building) {^M
        this.building = building;^M
    }^M
^M
    public FigureGraphic (String name, Color colorStroke, Color colorBackground) {^M
        this.colorStroke = colorStroke;   ^M
        this.colorBackground = colorBackground;   ^M
        this.name = name;^M
    }^M
^M
    public FigureGraphic (String name) {^M
        this(name, Color.black, Color.white);^M
    }^M
    ^M
    public void setColors(Env env) {^M
        setColors(env.getStrokeColor(), env.getBackgroundColor());^M
    }^M
    public void setColors(Color stroke, Color bg) {^M
        colorStroke = stroke;^M
        colorBackground = bg;^M
    }^M
^M
    public Color getStrokeColor()^M
    {^M
        return colorStroke;   ^M
    }^M
^M
    public Color getBackgroundColor()^M
    {^M
        return colorBackground;   ^M
    }^M
^M
    public void setStrokeColor(Color c) {^M
        colorStroke = c;^M
    }^M
^M
    public void setBackgroundColor(Color c) {^M
        colorBackground = c;^M
    }^M
    ^M
    /**^M
    * Distance between Figure f1 and f1^M
```

```
     * @param Figure f1^M
     * @param Figure f2^M
     * @return double the distance^M
     */^M
    public static double distance(Figure f1, Figure f2)^M
    {^M
        return Point_2D.distance(f1.getCenter(), f2.getCenter());^M
    }^M
^M
    protected void drawCenter(Graphics g) {^M
        Point_2D c = getCenter();^M
        g.drawLine(c.x-1, c.y, c.x+1, c.y);^M
        g.drawLine(c.x, c.y-1, c.x, c.y+1);^M
    }^M
    ^M
    /**^M
    * Draw the name of the figure on the Graphics^M
    * @param Graphics g^M
    */^M
    protected void drawName(Graphics g) {^M
        Point_2D c = getCenter();^M
        g.drawString(name, c.x+2, c.y+12);^M
    }^M
    ^M
    public abstract void draw(Graphics g);^M
^M
    /**^M
    * A method called after the draw method^M
    * @param Graphics g^M
    */^M
    public void afterDraw(Graphics g) {^M
        if(isSelected()) {^M
            drawCenter(g);^M
            drawName(g);^M
        }^M
    }^M
^M
    public String getName() {^M
        return name;^M
    }^M
    public void setName(String name) {^M
        this.name = name;^M
    }^M
    ^M
    @Override^M
    public String toString() {^M
        return getName();^M
    }^M
^M
    /**^M
    * The getter for the stroke color^M
    */^M
    public Color getStrokeForCurrentState() {^M
        Color c = colorStroke;^M
        return transparent || building ? new Color(c.getRed(), c.getGreen(), c.getBlue(), c.getAlpha()/2) :
    }^M
^M
    /**^M
    * The getter for the background color^M
    */^M
    public Color getBgForCurrentState() {^M
        Color c = colorBackground;^M
        return transparent || building ? new Color(c.getRed(), c.getGreen(), c.getBlue(),  (c.getAlpha()*2)
    }^M
^M
    public void setSelected(boolean s) {^M
        selected = s;^M
```

```
    }^M
    public void setTransparent(boolean o) {^M
        transparent = o;^M
    }^M
    public boolean isSelected() {^M
        return selected;^M
    }^M
    ^M
    /**^M
     * Created by first click^M
     * @param Env env^M
     * @param int x^M
     * @param int y^M
     */^M
    public abstract void init(Env env, int x, int y);^M
    /**^M
    * Is there enough information to complete the figure construction with the mouse^M
    */^M
    public abstract boolean canBeFinishedWithMouse();^M
    /**^M
    * Is there enough information to complete the figure construction with the a keyboard key^M
    */^M
    public abstract boolean canBeFinishedWithKey();^M
    /**^M
    * An event that is called when the figure is finished^M
    */^M
    public abstract void onFigureFinish();^M
    /**^M
    * When figure is in construction mode, perform a certain task when click^M
    */^M
    public abstract void onPressPoint(int x, int y);^M
    /**^M
    * When figure is in construction mode, perform a certain task when click release^M
    */^M
    public abstract void onReleasePoint(int x, int y);^M
    /**^M
    * When the point is moved^M
    */^M
    public abstract void onMovePoint(int x, int y);^M
    /**^M
    * Name of the type of the figure^M
    */^M
    public abstract String getShapeName();^M
}
// src/figure/Triangle.java
package figure;

import java.awt.Color;
import java.io.Serializable;

@SuppressWarnings("serial")
/**
 *  Class for the triangle figure
 */
public class Triangle extends Polygon implements Serializable
{
    private static long nbOfTriangles = 0;

    public Triangle(String name) {
        super(name);
        nbOfTriangles ++;
    }

    /**
    * Constructor that will generate by itself the name
    */
    public Triangle() {
```

```
        this("tri_"+(nbOfTriangles+1));
    }

    /**
    * Constructor
    * @param String name
    * @param Color stroke
    * @param Color bg background color
    * @param int x1 position
    * @param int y1 position
    * @param int x2 width
    * @param int y2 height
    * @param int x3 width
    * @param int y3 height
    */
    public Triangle(String name, Color stroke, Color bg, int x1, int y1, int x2, int y2, int x3, int y3) {
        this(name);
        setColors(stroke, bg);
        addPoint(x1, y1);
        addPoint(x2, y2);
        addPoint(x3, y3);
    }

    public boolean canBeFinished() {
        return points.size()==4;
    }
    public boolean canBeFinishedWithKey() {
        return canBeFinished();
    }
    public boolean canBeFinishedWithMouse() {
        return canBeFinished();
    }

    @Override
    protected boolean drawTerminaisonEnabled() {
        return false;
    }

    @Override
    public String getShapeName() {
        return "Triangle";
    }
}
// src/figure/Figure.java
package figure;

import figure.Point_2D;

import java.awt.Graphics;

/**
* An interface implemented by other figures.
* By implementing this interface figures can be used for various
* operations like drawing, drag&drop, selecting, etc.
*/
public interface Figure
{
    /**
    * Center of the figure
    * @return Point_2D
    */
    public abstract Point_2D getCenter();

    /**
    * Move the figure by dx and dy
    * @param int dx delta x
    * @Param int dy delta y
```

```
     * @return void
     */
    public abstract void move(int dx, int dy);

    /**
    * Tell if the figure is in that point p
    * @param Point_2D p point
    * @return boolean
    */
    public abstract boolean contain(Point_2D p);

    /**
    * Get the name of the figure
    */
    public abstract String toString();

    /**
    * Drag the figure on the screen
    */
    public abstract void draw(Graphics gx);
}
```