Solutions – Exercices Dessin et Graphisme

D.1: Dessin de figures colorées (sans MVC)

```
public class DrawPanel extends javax.swing.JPanel {

    private String text = "Hello world";

    public void setText(String text) {
        this.text = text;
    }

    @Override
    public void paintComponent(Graphics g)
    {
        // Colorier l'arrière-fond du panneau en jaune
        g.setColor(Color.YELLOW);
        int w = getWidth();
        int h = getHeight();
        g.fillRect(0, 0, w, h);
        // Tracer quatre ovales de couleur non prédéfinie
        Color c = new Color(150, 200, 200);
        g.setColor(c);
        g.fillOval(0, 0, w/2, h/2);
        g.fillOval(0, h/2, w/2, h/2);
        g.fillOval(w/2, 0, w/2, h/2);
        g.fillOval(w/2, h/2, w/2, h/2);
        // Dessiner deux diagonales rouges
        g.setColor(Color.RED);
        g.drawLine(0, 0, w, h);
        g.drawLine(0, h, w, 0);
        // Dessiner un rectangle relient les centres des ovales
        g.drawRect(w/4, h/4, w/2, h/2);
        // Dessiner un texte au milieu
        g.setColor(Color.BLACK);
        g.drawString(text, w/2, h/2);
    }
}

MainFrame:
    private void textTextFieldActionPerformed(java.awt.event.ActionEvent evt) {
        drawPanel.setText(textTextField.getText());
        drawPanel.repaint();
    }
```

D.2: Grille (sans MVC)

```
public class DrawPanel extends javax.swing.JPanel {

    private int rows = 7;
    private int cols = 12;

    @Override
    public void paintComponent(Graphics g) {
        // calculer la taille d'une cellule - version imprécise
        //p.ex: si vous avez beaucoup de colonnes, il reste souvent un bord à droite
//        int cellWidth  = (getWidth() - 1)  / cols;
//        int cellHeight = (getHeight() - 1) / rows;

        // calculer la taille d'une cellule - version précise
        double cellWidth = (getWidth() - 1) / (double) cols;
        double cellHeight = (getHeight() - 1) / (double) rows;

        // changer la couleur
        g.setColor(Color.WHITE);
        // supprimer tout (dessiner un rectangle blanc)
        g.fillRect(0, 0, getWidth(), getHeight());

        // changer la couleur
        g.setColor(Color.BLACK);
        // dessiner les lignes horizontales
        for (int i = 0; i <= rows; i++) {
            g.drawLine(0, (int) (i * cellHeight), getWidth(), (int) (i *
cellHeight));
        }
        // dessiner les lignes verticales
        for (int i = 0; i <= cols; i++) {
            g.drawLine((int) (i * cellWidth), 0, (int) (i * cellWidth),
getHeight());
        }
    }

    public void setRows(int pRows) {
        rows = pRows;
    }

    public void setCols(int pCols) {
        cols = pCols;
    }


public class MainFrame extends javax.swing.JFrame {

    public MainFrame() {
        initComponents();
        // initialiser les valeurs par défaut des lignes et des colonnes
        drawPanel.setRows(Integer.valueOf(rowsTextField.getText()));
        drawPanel.setCols(Integer.valueOf(colsTextField.getText()));
    }
    private void drawButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // modifier la grille
        drawPanel.setRows(Integer.valueOf(rowsTextField.getText()));
        drawPanel.setCols(Integer.valueOf(colsTextField.getText()));
        drawPanel.repaint();
    }
```

Exercice D.3: Echiquier (sans MVC)

```
public class DrawPanel extends javax.swing.JPanel {

    @Override
    public void paintComponent(Graphics g)
    {
        // calculer le côté d'un carreau
        int side = Math.min(getWidth(),getHeight());
        int squareSide = side / 8;

        // (c) calcul des espaces des bords
        int offsetLeft = 0; //(getWidth()  - 8*squareSide) / 2;
        int offsetTop  = 0; //(getHeight() - 8*squareSide) / 2;

        //effacer le dessin actuel
        g.setColor(Color.WHITE);
        g.fillRect(0,0, getWidth(),getHeight());

        //dessiner l'échiquier
        for (int r = 0 ; r<8 ; r++)
            for (int c = 0 ; c<8 ; c++) {
                if ((r+c)%2==0) g.setColor(Color.WHITE);
                else            g.setColor(Color.GRAY);
                g.fillRect(offsetLeft+c*squareSide,
                           offsetTop +r*squareSide, squareSide, squareSide);
                // (b) Dessiner la bordure.
                g.setColor(Color.BLACK);
                g.drawRect(offsetLeft+c*squareSide,
                           offsetTop +r*squareSide, squareSide, squareSide);

            }
    }
```

D.4: Color Mixer (sans MVC)

```
public class DrawPanel extends javax.swing.JPanel {

    private int red = 200, green = 200, blue = 200, alpha = 200;

    public void setRed(int red) {
        this.red = red;
    }

    public void setGreen(int green) {
        this.green = green;
    }

    public void setBlue(int blue) {
        this.blue = blue;
    }

    public void setAlpha(int alpha) {
        this.alpha = alpha;
    }

    @Override
    protected void paintComponent(Graphics g) {
        //effacer le fond
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, getWidth(), getHeight());

        //tracer les diagonales
        g.setColor(Color.BLACK);
        double xStep = getWidth() / 20.0;
        double yStep = getHeight() / 20.0;
        for (int i = 0; i < 20; i++) {
            //diagonales haut-droite --> bas-gauche
            g.drawLine((int) (i * xStep), 0, 0, (int) (i * yStep));
            g.drawLine((int) (i * xStep), getHeight() - 1, getWidth() - 1, (int) (i
* yStep));
            //diagonales haut-gauche --> bas-droite
            g.drawLine((int) (i * xStep), 0, getWidth() - 1, getHeight() - 1 - (int)
(i * yStep));
            g.drawLine(0, (int) (i * yStep), getWidth() - 1 - (int) (i * xStep),
getHeight() - 1);
        }

        //solution alternative
//        g.setColor(Color.BLACK);
//        int cellWidth  =  getWidth()/20;
//        int cellHeight = getHeight()/20;
//        for(int i = 0; i <= 40; i++)
//        {
//            g.drawLine(i*cellWidth, getHeight(), -20*cellWidth+cellWidth*i, 0);
//            g.drawLine(0, i*cellHeight, getWidth(), -20*cellHeight+cellHeight*i);
```

```java
//        }
        //Tracer les carrés coloriés
        int vertSide = getHeight() / 3;
        int horSide = getWidth() / 3;

        g.setColor(new Color(red, 0, 0, alpha));
        g.fillRect(0, 0, horSide, vertSide);
        g.setColor(new Color(red, green, 0, alpha));
        g.fillRect(horSide, 0, horSide, vertSide);
        g.setColor(new Color(0, green, 0, alpha));
        g.fillRect(2 * horSide, 0, horSide, vertSide);

        g.setColor(new Color(red, 0, blue, alpha));
        g.fillRect(horSide / 2, vertSide, horSide / 2, vertSide);

        g.setColor(new Color(red, green, blue, alpha));
        g.fillRect(horSide, vertSide, horSide, vertSide);
        g.setColor(new Color(0, green, blue, alpha));
        g.fillRect(2 * horSide, vertSide, horSide / 2, vertSide);

        g.setColor(new Color(0, 0, blue, alpha));
        g.fillRect(horSide, 2 * vertSide, horSide, vertSide);
    }

public class MainFrame extends javax.swing.JFrame {

    public MainFrame() {
        initComponents();
        updateView();
    }

    private void updateView() {
        drawPanel.setRed(redSlider.getValue());
        drawPanel.setGreen(greenSlider.getValue());
        drawPanel.setBlue(blueSlider.getValue());
        drawPanel.setAlpha(alphaSlider.getValue());
        drawPanel.repaint();
    }

    private void redSliderStateChanged(javax.swing.event.ChangeEvent evt) {
        updateView();
    }

    private void greenSliderStateChanged(javax.swing.event.ChangeEvent evt) {
        updateView();
    }

    private void blueSliderStateChanged(javax.swing.event.ChangeEvent evt) {
        updateView();
    }

    private void alphaSliderStateChanged(javax.swing.event.ChangeEvent evt) {
        updateView();
    }
```

D.5: Damier

```
public class Piece {
    private Color color = null;
    private int col;
    private int row;

    public Piece(Color pColor, int pCol, int pRow) {
        color = pColor;
        row = pRow;
        col = pCol;
    }

    public void moveTo(int pCol, int pRow) {
        col = pCol;
        row = pRow;
    }

    public int getCol() {
        return col;
    }

    public Color getColor() {
        return color;
    }

    public int getRow() {
        return row;
    }

    public void draw(Graphics g, int offsetLeft, int offsetTop, int squareSide) {
        g.setColor(color);
        //dessiner les pions
        g.fillOval(offsetLeft + col * squareSide + 1,
                offsetTop + row * squareSide + 1,
                squareSide - 2, squareSide - 2);
    }
}

public class Checkers {
    private ArrayList<Piece> alPieces = new ArrayList<>();

    public void init() {
        //placer les pions du joueur 0 (bleu)
        for (int r = 0 ; r < 3 ; r++)
            for (int c = 0 ; c < 8 ; c++)
                if ((r+c)%2 != 0)
                    alPieces.add(new Piece(Color.BLUE, c, r));
        //placer les pions du joueur 1 (rouge)
        for (int r = 5 ; r < 8 ; r++)
            for (int c = 0 ; c < 8 ; c++)
                if ((r+c)%2 != 0)
                alPieces.add(new Piece(Color.RED, c, r));
}

public Piece getPieceAt(int col, int row) {
    Piece result = null;
    for (int i = 0 ; i < alPieces.size() ; i++)
         if (row == alPieces.get(i).getRow() && col == alPieces.get(i).getCol())
                result = alPieces.get(i);
    return result;
}

public boolean move(int startCol, int startRow, int destCol, int destRow) {
    boolean result = true;
    Piece p = getPieceAt(startCol,startRow);

    if (p == null)    // pion n'existe pas, mouvement non valide
        result = false;
    else
        //pion existe
        if (destRow<0 || destRow>=8 || destCol<0 || destCol>=8) {
            //supprimer le pion:
            int i = alPieces.indexOf(p);
            alPieces.remove(i);
        }
        else if (getPieceAt(destCol,destRow) == null) {  //case de dest. libre?
            //déplacer le pion
            p.moveTo(destCol,destRow);
        }
        else result = false;  //mouvement non valide

    return result;
}

public void draw(Graphics g, int width, int height) {
    // calculer le côté d'un carreau
    int side = Math.min(width,height);
    int squareSide = side / 8;

    // (c) calcul des espaces des bords
    int offsetLeft = (width  - 8*squareSide) / 2;
    int offsetTop  = (height - 8*squareSide) / 2;

    //dessiner l'échiquier
    for (int r=0 ; r<8 ; r++)
    {
        for (int c=0 ; c<8 ; c++) {
            if ((r+c)%2 == 0) g.setColor(Color.WHITE);
            else            g.setColor(Color.GRAY);
            g.fillRect(offsetLeft + c*squareSide,
                       offsetTop  + r*squareSide, squareSide, squareSide);
            // (b) Dessiner la bordure de la case
            g.setColor(Color.BLACK);
```

```
                g.drawRect(offsetLeft + c*squareSide,
                           offsetTop  + r*squareSide, squareSide, squareSide);
            }
        }

        // dessiner les pions
        for (int i=0; i<alPieces.size(); i++) {
            alPieces.get(i).draw(g, offsetLeft, offsetTop, squareSide);
        }

    }

}

public class DrawPanel extends javax.swing.JPanel
{
    /** une référence vers le modèle */
    private Checkers checkers = null;

    public void setCheckers(Checkers pCheckers) {
        checkers = pCheckers;
    }

    @Override
    public void paintComponent(Graphics g) {
        //effacer le dessin actuel
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, getWidth(), getHeight());

        if (checkers!=null) {
            checkers.draw(g, getWidth(), getHeight());
        }
    }
}



public class MainFrame extends javax.swing.JFrame {

    private Checkers checkers = null;

    public MainFrame() {
        initComponents();
    }

    private void moveButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if (checkers!=null) {
            // essayer de déplacer le pion
            if (checkers.move(Integer.valueOf(startColTextField.getText()),
                              Integer.valueOf(startRowTextField.getText()),
                              Integer.valueOf(destColTextField.getText()),
                              Integer.valueOf(destRowTextField.getText())))
                errorLabel.setText("");
            else
                errorLabel.setText("Invalid Move!");
            // redessiner le damier
            drawPanel.repaint();
        }
    }

    private void newButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // initialiser le damier
        checkers = new Checkers();
        checkers.init();
        drawPanel.setCheckers(checkers);
        // redessiner le damier
        drawPanel.repaint();
    }
```

D.6: Vector Line Painter

```
public class Line {
    /** point de départ de la ligne */
    private Point from;
    /** point d'arrivée de la ligne */
    private Point to;
    /** la couleur de la ligne */
    private Color color;

    public Line(Point pFrom, Point pTo, Color pColor) {
        from   = pFrom;
        to     = pTo;
        color  = pColor;
    }

    public Line(int fromX, int fromY, int toX, int toY, Color pColor) {
        from   = new Point(fromX, fromY);
        to     = new Point(toX  , toY  );
        color  = pColor;
    }

    public Point getFrom() {
        return from;
    }

    public Point getTo() {
        return to;
    }

    public Color getColor() {
        return color;
    }

    public void draw(Graphics g) {
        g.setColor(color);
        g.drawLine(from.x, from.y, to.x, to.y);
    }

}

public class Lines {
    private ArrayList<Line> alLines = new ArrayList<>();

    public void add(Line pLine) {
        alLines.add(pLine);
    }

    public void draw(Graphics g) {
        // dessiner les lignes
        for (int i=0 ; i<alLines.size() ; i++)
                alLines.get(i).draw(g);
```

```java
    }
}


public class DrawPanel extends javax.swing.JPanel
{
    private Lines lines = null;

    public void setLines(Lines lines) {
        this.lines = lines;
    }

    public void paintComponent(Graphics g)    {
        // dessiner un fond blanc
        g.setColor(Color.white);
        g.fillRect(0,0,getWidth(),getHeight());

        if (lines!=null)
            lines.draw(g);
    }
}

public class MainFrame extends javax.swing.JFrame {
    private Lines lines = new Lines();
    private Color color = Color.BLACK;

    public MainFrame() {
        initComponents();
        drawPanel.setLines(lines);
    }

    private void addButtonActionPerformed(java.awt.event.ActionEvent evt) {
        //Ajouter une ligne et redessiner
        Line line = new Line(Integer.valueOf(fromXTextField.getText()),
                             Integer.valueOf(fromYTextField.getText()),
                             Integer.valueOf(toXTextField.getText()),
                             Integer.valueOf(toYTextField.getText()),
                             color);
        lines.add(line);
        drawPanel.repaint();
    }

    private void newButtonActionPerformed(java.awt.event.ActionEvent evt) {
        lines = new Lines();
        drawPanel.setLines(lines);
        drawPanel.repaint();
    }

    private void chooseColorButtonActionPerformed(java.awt.event.ActionEvent evt) {
        color = JColorChooser.showDialog(this,"Choix d'une couleur", color);
    }
}
```

D.7: Histogramme

```
public class NumberInfos {

    private ArrayList<NumberInfo> alNumberInfos = new ArrayList<>();

    private String title;

    public NumberInfos(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public void add(double value, String label) {
        alNumberInfos.add(new NumberInfo(value, label));
    }

    public double getTotal() {
        double result = 0;
        for (int i = 0; i < alNumberInfos.size(); i++) {
            result = result + alNumberInfos.get(i).getValue();
        }
        return result;
    }

    public double getMaximum() {
        double result = 0;
        for (int i = 0; i < alNumberInfos.size(); i++) {
            if (result < alNumberInfos.get(i).getValue()) {
                result = alNumberInfos.get(i).getValue();
            }
        }
        return result;
    }

    public Object[] toArray() {
        return alNumberInfos.toArray();
    }

    public int size() {
        return alNumberInfos.size();
    }

    public NumberInfo get(int i) {
        return alNumberInfos.get(i);
    }

    public void draw(Graphics g, int pWidth, int pHeight) {
        int height = pHeight - 1; //pour éviter de dépasser en bas du panneau
        int width = pWidth - 1; //pour éviter de dépasser à droite du panneau

        if (size() > 0) {
            double barWidth = (double) width / size();
            for (int i = 0; i < size(); i++) {
                //choix de la couleur (alternativement orange et jaune)
                if (i % 2 == 0) {
                    g.setColor(Color.YELLOW);
                } else {
                    g.setColor(Color.ORANGE);
                }
                //hauteur d'un rectangle
                double barHeight = (double) get(i).getValue() / getMaximum() *
height;
                //affichage du fond des rectangles
                g.fillRect((int) (i * barWidth), height - (int) barHeight,
                        (int) barWidth, (int) barHeight);
                //afficher la bordure en rouge
                g.setColor(Color.RED);
                g.drawRect((int) (i * barWidth), height - (int) barHeight,
                        (int) barWidth, (int) barHeight);
                //afficher le libellé et la valeur
                g.setColor(Color.BLUE);
                g.drawString(get(i).getLabel(), (int) (i * barWidth + 2), height -
4);
                g.drawString(String.valueOf(get(i).getValue()),
                        (int) (i * barWidth + 2), height - 18);
            }
            //afficher le titre
            g.setColor(Color.DARK_GRAY);
            g.drawString(title, 3, 11);
            g.setColor(Color.BLACK);
            g.drawString(title, 2, 10);
        }

    }
}


public class DrawPanel extends javax.swing.JPanel {
    private NumberInfos numberInfos = null;

    @Override
    public void paintComponent(Graphics g)
    {
        g.setColor(Color.white);
        g.fillRect(0,0,getWidth(),getHeight());

        if (numberInfos!=null)
              numberInfos.draw(g, getWidth(), getHeight());
    }
    public void setNumberInfos(NumberInfos pNumberInfos)
    {
        numberInfos=pNumberInfos;
    }

public class MainFrame extends javax.swing.JFrame {

    private NumberInfos numberInfos = null;

    public MainFrame() {
        initComponents();
    }

    private void updateView() {
        // display the list data
        numberInfoJList.setListData(numberInfos.toArray());
        // request repainting of the diagram
        repaint();
    }

    private void test() {
        // pour la démonstation
        numberInfos.add(63, "JAN");
        numberInfos.add(65, "FEB");
        numberInfos.add(58, "MAR");
        numberInfos.add(77, "APR");
        numberInfos.add(111, "MAY");
        numberInfos.add(124, "JUN");
        numberInfos.add(104, "JUL");
        numberInfos.add(99, "AUG");
        numberInfos.add(80, "SEP");
        numberInfos.add(66, "OCT");
        numberInfos.add(103, "NOV");
        numberInfos.add(80, "DEC");
    }

    private void newButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // create a new list
        numberInfos = new NumberInfos(titleTextField.getText());
        // pass its reference to the drawPanel
        drawPanel.setNumberInfos(numberInfos);
        // enable add button
        addButton.setEnabled(true);
        //test
        test();
        // update the view
        updateView();
    }

    private void addButtonActionPerformed(java.awt.event.ActionEvent evt) {
        numberInfos.add(Double.valueOf(valueTextField.getText()),
                labelTextField.getText());
        updateView();
    }
```

D.8: Squares

```
public class MainFrame extends javax.swing.JFrame {

    private Squares squares = new Squares();

    public MainFrame() {
        initComponents();
        drawPanel.setSquares(squares);
    }

    private void createButtonActionPerformed(java.awt.event.ActionEvent evt) {
        int x = Integer.valueOf(xTextField.getText());
        int y = Integer.valueOf(yTextField.getText());
        int side = Integer.valueOf(sideTextField.getText());
        Color color = new Color(
                0,
                (int) ((double) x / (drawPanel.getWidth() - side) * 255),
                (int) ((double) y / (drawPanel.getHeight() - side) * 255)
        );

        Square s = new Square(x, y, side, color);
        squares.add(s);

        drawPanel.repaint();
    }

    private void randomButtonActionPerformed(java.awt.event.ActionEvent evt) {
        int maxSide = 30;
        int minSide = 10;

        for (int i = 0; i < 100; i++) {
            int side = (int) (Math.random() * (maxSide - minSide + 1)) + minSide;
            int x = (int) (Math.random() * (drawPanel.getWidth() - side));
            int y = (int) (Math.random() * (drawPanel.getHeight() - side));
            int green = (int) ((double) x / (drawPanel.getWidth() - side) * 255);
            int blue = (int) ((double) y / (drawPanel.getHeight() - side) * 255);

            Color color = new Color(0, green, blue);

            Square s = new Square(x, y, side, color);
            squares.add(s);
        }
        drawPanel.repaint();
    }

    private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
        squares.clear();
        drawPanel.repaint();
    }

}
```

D.9: Traceur de polynômes

```java
public class Polynomial {
    private double xmin;
    private double xmax;
    private double ymin;
    private double ymax;

    private ArrayList<Double> alCoefficients = new ArrayList<>();

    public void add(double pCoeff) {
        alCoefficients.add(pCoeff);
    }

    public void clear() {
        alCoefficients.clear();
    }

    public int size() {
        return alCoefficients.size();
    }

    public void setLimits(double xmin, double xmax, double ymin, double ymax) {
        this.xmin = xmin;
        this.xmax = xmax;
        this.ymin = ymin;
        this.ymax = ymax;
    }

    public double getCoefficient(int i) {
        if (alCoefficients.size() > i) {
            return alCoefficients.get(i);
        } else {
            return 0;
        }
    }

    public double evaluateNaive(double pX) {
        double result = 0;
        for (int i = 0; i < alCoefficients.size(); i++) {
            result = result + alCoefficients.get(i) * Math.pow(pX, i);
        }
        return result;
    }

    public double evaluateHorner(double pX) {
        double result = 0;
        for (int i = alCoefficients.size() - 1; i >= 0; i--) {
            result = result * pX + alCoefficients.get(i);
        }
        return result;
    }
```

```
public void draw(Graphics g, int pWidth, int pHeight, Color pColor) {
    double xfactor = (xmax-xmin)/pWidth;
    double yfactor = (ymax-ymin)/pHeight;
    g.setColor(pColor);
    g.drawString("f(x) = " + toString(), 4, 21);
    for (int xgraph=0; xgraph < pWidth; xgraph++){
        double x = xmin + xgraph * xfactor;
        double y = evaluateHorner(x);
        double ygraph = pHeight - (y-ymin)/yfactor;
        g.setColor(pColor);
        g.drawLine((int)xgraph, (int)ygraph, (int)xgraph, (int)ygraph);
    }
}

@Override
public String toString() {
    String s = " = ";
    if (alCoefficients.size() > 0) {
        for (int i = 0; i < alCoefficients.size(); i++) {
            s = " + " + alCoefficients.get(i) + "x^" + i + s;
        }
        return s;
    } else {
        return "";
    }

}

public class DrawPanel extends javax.swing.JPanel {
    private Polynomial poly = null;

    public DrawPanel() {
        initComponents();
    }

    public void setPoly(Polynomial poly) {
        this.poly = poly;
    }

    @Override
    protected void paintComponent(Graphics g) {
        g.setColor(new Color(249, 243, 194));
        g.fillRect(0,0,getWidth(),getHeight());
        if (poly != null) {
            poly.draw(g, getWidth(), getHeight(), Color.RED);
        }
    }

public class MainFrame extends javax.swing.JFrame {

    private Polynomial poly = null;
```

```
public MainFrame() {
    initComponents();
    poly = new Polynomial();
    drawPanel.setPoly(poly);
    setLimits();
    updateView();
}

private void updateView() {
    drawPanel.repaint();
}

private void setLimits() {
    double xmin = Double.valueOf(xminTextField.getText());
    double xmax = Double.valueOf(xmaxTextField.getText());
    double ymin = Double.valueOf(yminTextField.getText());
    double ymax = Double.valueOf(ymaxTextField.getText());
    poly.setLimits(xmin, xmax, ymin, ymax);
    updateView();
}

private void addButtonActionPerformed(java.awt.event.ActionEvent evt) {
    poly.add(Double.valueOf(coeffTextField.getText()));
    updateView();
}

private void xminTextFieldActionPerformed(java.awt.event.ActionEvent evt) {
    setLimits();
    updateView();
}

private void yminTextFieldActionPerformed(java.awt.event.ActionEvent evt) {
    setLimits();
    updateView();
}

private void xmaxTextFieldActionPerformed(java.awt.event.ActionEvent evt) {
    setLimits();
    updateView();
}

private void ymaxTextFieldActionPerformed(java.awt.event.ActionEvent evt) {
    setLimits();
    updateView();
}

private void newButtonActionPerformed(java.awt.event.ActionEvent evt) {
    poly.clear();
    updateView();
}
```