

```
public class Shape
{
    private Point from;
    private Point to;
    private Color color;

    public Shape(int pX1, int pY1, int pX2, int pY2, Color pColor)
    {
        from = new Point(pX1, pY1);
        to = new Point(pX2, pY2);
        color = pColor;
    }

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

    public Point getFrom()
    {
        return from;
    }

    public void setFrom(Point pFrom)
    {
        from = pFrom;
    }

    public Point getTo()
    {
        return to;
    }

    public void setTo(Point pTo)
    {
        to = pTo;
    }

    public void setColor(Color pColor)
    {
        color = pColor;
    }

    public Color getColor()
    {
        return color;
    }

    public int getTop()
    {
        return Math.min(from.y, to.y);
    }

    public int getBottom()
    {
        return getTop()+getHeight()-1;
    }

    public int getLeft()
    {
        return Math.min(from.x, to.x);
    }

    public int getRight()
    {
        return getLeft()+getWidth()-1;
    }

    public int getWidth()
    {
        return Math.abs(from.x - to.x)+1;
    }

    public int getHeight()
    {
        return Math.abs(from.y - to.y)+1;
    }

    public double getArea()
    {
        return 0; // identique pour une ligne
    }

    public void draw(Graphics g)
    {
        // Ne rien faire
    }

    public String toString()
    {
        // on arrondit pour ne pas avoir trop de décimales
        int a = (int) getArea();
        return "[" + from.x + "," + from.y + "] -> [" + to.x + "," + to.y + "] / a=" + a + "px²";
    }
}
```

```
public class Ellipse extends Shape
{
    public Ellipse(int pX1, int pY1, int pX2, int pY2, Color pColor)
    {
        super(pX1, pY1, pX2, pY2, pColor);
    }

    public Ellipse(Point pFrom, Point pTo, Color pColor)
    {
        super(pFrom, pTo, pColor);
    }

    @Override
    public double getArea()
    {
        return Math.PI * getWidth() / 2.0 * getHeight() / 2.0;
    }

    @Override
    public void draw(Graphics g)
    {
        g.setColor(getColor());
        g.drawOval(getLeft(), getTop(), getWidth(), getHeight());
    }

    @Override
    public String toString()
    {
        return "Ellipse " + super.toString();
    }
}
```

```
public class IsoTriangle extends Shape
{
    public IsoTriangle(int pX1, int pY1, int pX2, int pY2, Color pColor)
    {
        super(pX1, pY1, pX2, pY2, pColor);
    }

    public IsoTriangle(Point pFrom, Point pTo, Color pColor)
    {
        super(pFrom, pTo, pColor);
    }

    @Override
    public double getArea()
    {
        return getWidth() * getHeight() / 2;
    }

    @Override
    public void draw(Graphics g)
    {
        int x0 = getFrom().x;
        int y0 = getFrom().y;
        int dx = getTo().x - x0;
        int dy = getTo().y - y0;

        // facteur indiquant dans quel sens dessiner le triangle
        int fx, fy;

        if (dx * dy < 0)
        {
            fx = 1;
            fy = 0;
        }
        else
        {
            fx = 0;
            fy = 1;
        }

        g.setColor(getColor());

        g.drawLine(x0, y0, x0 + dx * fx, y0 + dy * fy);
        g.drawLine(x0, y0, x0 + dx / (1 + fx), y0 + dy / (1 + fy));
        g.drawLine(x0 + dx * fx, y0 + dy * fy, x0 + dx / (1 + fx), y0 + dy / (1 + fy));
    }

    @Override
    public String toString()
    {
        return "IsoTriangle " + super.toString();
    }
}
```

```
public class Line extends Shape
{
    public Line(int pX1, int pY1, int pX2, int pY2, Color pColor)
    {
        super(pX1, pY1, pX2, pY2, pColor);
    }

    public Line(Point pFrom, Point pTo, Color pColor)
    {
        super(pFrom, pTo, pColor);
    }

    @Override
    public void draw(Graphics g)
    {
        g.setColor(getColor());
        g.drawLine(getFrom().x, getFrom().y, getTo().x, getTo().y);
    }

    @Override
    public String toString()
    {
        return "Line " + super.toString();
    }
}
```

```
public class Rectangle extends Shape
{
    public Rectangle(int pX1, int pY1, int pX2, int pY2, Color pColor)
    {
        super(pX1, pY1, pX2, pY2, pColor);
    }

    public Rectangle(Point pFrom, Point pTo, Color pColor)
    {
        super(pFrom, pTo, pColor);
    }

    @Override
    public double getArea()
    {
        return getWidth() * getHeight();
    }

    @Override
    public void draw(Graphics g)
    {
        g.setColor(getColor());
        g.drawRect(getLeft(), getTop(), getWidth(), getHeight());
    }

    @Override
    public String toString()
    {
        return "Rectangle " + super.toString();
    }
}
```

```
public class RightTriangle extends Shape
{
    public RightTriangle(int pX1, int pY1, int pX2, int pY2, Color pColor)
    {
        super(pX1, pY1, pX2, pY2, pColor);
    }

    public RightTriangle(Point pFrom, Point pTo, Color pColor)
    {
        super(pFrom, pTo, pColor);
    }

    @Override
    public double getArea()
    {
        return getWidth() * getHeight() / 2;
    }

    @Override
    public void draw(Graphics g)
    {
        int x = getLeft();
        int y = getTop();
        int w = getWidth();
        int h = getHeight();

        // facteur indiquant dans quel sens dessiner le triangle
        int fy, fx;
        if (getFrom().y > getTo().y)
            fy = 1;
        else
            fy = 0;
        if (getFrom().x > getTo().x)
            fx = 1;
        else
            fx = 0;

        g.setColor(getColor());

        g.drawLine(x + w * fx, y + h * fy, x + w * (1 - fx), y + h * fy);
        g.drawLine(x + w * (1 - fx), y + h * fy, x + w * (1 - fx), y + h * (1 - fy));
        g.drawLine(x + w * (1 - fx), y + h * (1 - fy), x + w * fx, y + h * fy);
    }

    @Override
    public String toString()
    {
        return "RightTriangle " + super.toString();
    }
}
```

```
public class Shapes
{
    private ArrayList<Shape> alShapes = new ArrayList<>();

    public void add(Shape s)
    {
        alShapes.add(s);
    }

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

    public int getNumberOfShapes(Color color)
    {
        int n = 0;
        for (int i = 0; i < alShapes.size(); i++)
            if (alShapes.get(i).getColor().equals(color))
                n++;
        return n;
    }

    public int getIndexOfBiggestShape()
    {
        if (alShapes.isEmpty())
            return -1; // signale une liste vide

        int s = 0;
        double max = alShapes.get(0).getArea();
        for (int i = 1; i < alShapes.size(); i++)
        {
            if (alShapes.get(i).getArea() > max)
            {
                max = alShapes.get(i).getArea();
                s = i;
            }
        }
        return s;
    }

    public void sortByArea()
    {
        Shape min, current;
        int posMin;
        int n = alShapes.size();
        for (int i = 0; i <= n - 2; i++)
        {
            min = alShapes.get(i);
            posMin = i;
            for (int j = i + 1; j <= n - 1; j++)
            {
                current = alShapes.get(j);
                if (current.getArea() < min.getArea())
                {
                    min = current;
                    posMin = j;
                }
            }
            if (posMin != i)
            {
                alShapes.set(posMin, alShapes.get(i));
                alShapes.set(i, min);
            }
        }
    }

    public void draw(Graphics g)
    {
        // Chaque figure se dessine elle-même
        for (int i = 0; i < alShapes.size(); i++)
            alShapes.get(i).draw(g);
    }

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

```
public class DrawPanel extends javax.swing.JPanel
{
    private Shapes shapes = null;

    public DrawPanel()
    {
        initComponents();
    }

    public void setShapes(Shapes s)
    {
        shapes = s;
    }

    @Override
    public void paintComponent(Graphics g)
    {
        // Effacer le dessin
        int w = getWidth();
        int h = getHeight();
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, w, h);

        // Dessiner les figures si présentes
        if (shapes != null)
            shapes.draw(g);
    }
}
// Skipped: ... initComponents { ... }
// Variables declaration - do not modify//GEN-BEGIN:variables
// End of variables declaration//GEN-END:variables
}
```

```

public class MainFrame extends javax.swing.JFrame
{
    private Shapes shapes = new Shapes();
    private Shape newShape = null;

    // Couleur à utiliser pour de nouvelles lignes
    private Color drawColor = Color.BLUE;

    public MainFrame()
    {
        initComponents();
        // La liste des shapes est fixée directement dans la JList typeList:
        // 0=Line / 1=Rectangle / 2=Ellipse / 3=IsoTriangle / 4=RightTriangle
        // au début on sélectionne directement la ligne
        typeList.setSelectedIndex(0);
        drawPanel.setShapes(shapes);
        updateView();
    }

    public void updateView()
    {
        shapesList.setListData(shapes.toArray());
        nbrShapesLabel.setText(String.valueOf(shapes.getNumberOfShapes(Color.BLUE)));

        // selectionne la figure la plus "grande"
        // désélectionne si la liste est vide
        int i = shapes.getIndexOfBiggestShape();
        shapesList.setSelectedIndex(i);
        repaint();
    }
}
// Skipped: ... initComponents { ... }
private void drawPanelMouseDragged(java.awt.event.MouseEvent evt) { //GEN-FIRST: event_drawPanelMouseDragged
    if (newShape == null)
        return;

    newShape.setTo(evt.getPoint());
    updateView();
} //GEN-LAST: event_drawPanelMouseDragged

private void drawPanelMousePressed(java.awt.event.MouseEvent evt) { //GEN-FIRST: event_drawPanelMousePressed
    int i = typeList.getSelectedIndex();
    if (i == -1)
        return; // plus rien de sélectionné!

    if (i == 0)
        newShape = new Line(evt.getPoint(), evt.getPoint(), drawColor);
    else if (i == 1)
        newShape = new Rectangle(evt.getPoint(), evt.getPoint(), drawColor);
    else if (i == 2)
        newShape = new Ellipse(evt.getPoint(), evt.getPoint(), drawColor);
    else if (i == 3)
        newShape = new IsoTriangle(evt.getPoint(), evt.getPoint(), drawColor);
    else
        newShape = new RightTriangle(evt.getPoint(), evt.getPoint(), drawColor);

    shapes.add(newShape);
    updateView();
} //GEN-LAST: event_drawPanelMousePressed

private void drawPanelMouseReleased(java.awt.event.MouseEvent evt) { //GEN-FIRST: event_drawPanelMouseReleased
    if (newShape == null)
        return;

    newShape.setTo(evt.getPoint());
    newShape = null;
    updateView();
} //GEN-LAST: event_drawPanelMouseReleased

private void changeColorButtonActionPerformed(java.awt.event.ActionEvent evt) //GEN-FIRST: event_changeColorButtonActionPerformed
{ //GEN-HEADEREND: event_changeColorButtonActionPerformed
    drawColor = JColorChooser.showDialog(this, "Choix d'une couleur", drawColor);
    updateView();
} //GEN-LAST: event_changeColorButtonActionPerformed

private void drawPanelMouseClicked(java.awt.event.MouseEvent evt) //GEN-FIRST: event_drawPanelMouseClicked
{ //GEN-HEADEREND: event_drawPanelMouseClicked
    if (evt.getClickCount() >= 2)
    {
        shapes.clear();
        updateView();
    }
} //GEN-LAST: event_drawPanelMouseClicked

private void sortButtonActionPerformed(java.awt.event.ActionEvent evt) //GEN-FIRST: event_sortButtonActionPerformed
{ //GEN-HEADEREND: event_sortButtonActionPerformed
    shapes.sortByArea();
    updateView();
} //GEN-LAST: event_sortButtonActionPerformed
// Skipped: ... Look & Feel
// Variables declaration - do not modify //GEN-BEGIN: variables
private javax.swing.JButton changeColorButton;
private DrawPanel drawPanel;
private javax.swing.JLabel jLabel1;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JLabel nbrShapesLabel;
private javax.swing.JList shapesList;

```

```
private javax.swing.JButton sortButton;  
private javax.swing.JList typeList;  
// End of variables declaration//GEN-END:variables  
}
```