```java
public class PolarPoint extends Point
{
    // L'angle et le rayon n'ont pas besoin d'être stockés

    public PolarPoint(double angle, double radius)
    {
        super((int)(Math.sin(angle)*radius), (int)(Math.cos(angle)*radius));
    }
}
```

```java
public class Dial
{
    private int numberOfTicks;
    private int majorTick;
    private int currentTick;

    public Dial(int numberOfTicks, int majorTick)
    {
        this.numberOfTicks = numberOfTicks;
        this.majorTick = majorTick;
        this.currentTick = 0;
    }

    public int getCurrentTick()
    {
        return currentTick;
    }

    public void setCurrentTick(int currentTick)
    {
        this.currentTick = currentTick;
    }

    public void draw (Graphics g, int x, int y, int width, int height, int currentTick)
    {
        // Juste pour tester les bords
        //   g.setColor(Color.BLUE);
        //   g.drawRect(x, y, width-1, height-1);

        // Calculs
        int radius = Math.min(width, height) / 2;
        int offsetX = x + width/2-radius;
        int offsetY = y + height/2-radius;
        int centerX = offsetX + radius;
        int centerY = offsetY + radius;

        // Dessiner le cadran
        g.setColor(Color.BLACK);
        g.drawOval(offsetX, offsetY, 2*radius, 2*radius);

        // Variables utiles
        Point startPoint, endPoint, labelPoint;
        double angle;

        // Dessine les graduations et les chiffres (labels)
        int lengthTick = radius /10;
        int lengthMajorTick = radius / 5;

        // Les chiffres (labels) doivent être "centrés" à la main
        int xError;
        if (numberOfTicks > 10)
            xError = 7;
        else
            xError = 4;

        for (int i=0; i < numberOfTicks; i++)
        {
            angle = (i / (double) numberOfTicks) * 2 * Math.PI;
            if ((majorTick > 0) && (i % majorTick == 0))
            {
                startPoint = new PolarPoint(angle, radius - lengthMajorTick);
                labelPoint = new PolarPoint(angle, radius - lengthMajorTick-15);
                g.setColor(Color.BLUE);
                g.drawString(String.valueOf(i), centerX+labelPoint.x-xError, centerY-labelPoint.y+5); // chiffres sur le cadran
            }
            else
                startPoint = new PolarPoint(angle, radius - lengthTick);
            endPoint = new PolarPoint(angle, radius);

            g.setColor(Color.BLACK);
            g.drawLine(centerX+startPoint.x, centerY-startPoint.y, centerX+endPoint.x, centerY-endPoint.y);
        }

        // Fait pointer l'aiguille vers le tick actuel
        g.setColor(Color.RED);
        angle = (currentTick / (double) numberOfTicks) * 2 * Math.PI;
        endPoint = new PolarPoint(angle, radius-lengthTick-1);
        g.drawLine(centerX, centerY, centerX+endPoint.x, centerY-endPoint.y);
    }
}
```

```java
public class Chronometer
{
    // Le temps écoulé en dixièmes de secondes écoulées
    private int time = 0;

    // La position d'affichage
    private Point position;

    public Chronometer(Point pPosition)
    {
        position = pPosition;
        time = 0;
    }

    public int getTime()
    {
        return time;
    }

    public void reset()
    {
        time = 0;
    }

    public Point getPosition()
    {
        return position;
    }

    public void setPosition(Point pPosition)
    {
        position = pPosition;
    }

    public int getMinutes()
    {
        return time / 600;
    }

    public int getSeconds()
    {
        return time % 600 / 10;
    }

    public int getTenthsOfSeconds()
    {
        return time % 10;
    }

    public void nextTick()
    {
        time++;
    }

    public String toString()
    {
        return getMinutes() + " minutes, " + getSeconds() + " seconds, " + getTenthsOfSeconds() + "/10 seconds";
    }

    public void draw(Graphics g, int pWidth, int pHeight)
    {
        g.setColor(Color.BLACK);
        g.drawString(getMinutes() + " min " + getSeconds() + " sec " + getTenthsOfSeconds() + " sec/10", position.x, position.y + 20);
    }
}
```

```java
public class AnalogChronometer extends Chronometer
{
    private Dial secondsDial;
    private Dial minutesDial;
    private Dial tenthSecondsDial;

    public AnalogChronometer()
    {
        super(null);
        secondsDial = new Dial(60, 5);
        minutesDial = new Dial(60, 5);
        tenthSecondsDial = new Dial(10, 1);
    }

    @Override
    public void draw(Graphics g, int pWidth, int pHeight)
    {
        // La surface de dessin est diminuée de 5 pixels par côté
        // pour avoir une "marge" sur tout le bord
        int w = pWidth-10;
        int h = pHeight-10;

        // Afficher le (grand) cadran pour les secondes
        secondsDial.draw(g, w/8+5, 5, w*3/4, h*3/4, getSeconds());

        // Afficher un (petit) cadran pour les minutes
        minutesDial.draw(g, 5, h*3/4+5, w/4, h/4, getMinutes());

        // Afficher un (petit) cadran pour les 10èmes de seondes
        tenthSecondsDial.draw(g, w*3/4+5, h*3/4+5, w/4, h/4, getTenthsOfSeconds());
    }
}
```

```java
public class DrawPanel extends javax.swing.JPanel
{
    private Chronometer chronometer;

    public void setChronometer(Chronometer pChrono)
    {
        chronometer = pChrono;
    }

    public DrawPanel()
    {
        initComponents();
    }

    @Override
    public void paintComponent(Graphics g)
    {
        int w = getWidth();
        int h = getHeight();

        g.setColor(Color.WHITE);
        g.fillRect(0, 0, w, h);

        if (chronometer != null)
            chronometer.draw(g, w, h);
    }
// 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 Timer timer;
    private Chronometer chrono;

    public MainFrame()
    {
        initComponents();
        stepButton.setVisible(false);
        chrono = new Chronometer(new Point(10, 20));
        drawPanel.setChronometer(chrono);
        timer = new Timer(100, stepButton.getActionListeners()[0]);
        timer.stop();
    }
// Skipped: ... initComponents { ... }
private void startButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_startButtonActionPerformed
    timer.start();
}//GEN-LAST:event_startButtonActionPerformed

private void stopButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_stopButtonActionPerformed
    timer.stop();
}//GEN-LAST:event_stopButtonActionPerformed

private void resetButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_resetButtonActionPerformed
    timer.stop();
    chrono.reset();
    repaint();
}//GEN-LAST:event_resetButtonActionPerformed

private void stepButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_stepButtonActionPerformed
    chrono.nextTick();
    repaint();
}//GEN-LAST:event_stepButtonActionPerformed

    private void changeViewButtonActionPerformed(java.awt.event.ActionEvent evt)//GEN-FIRST:event_changeViewButtonActionPerformed
    {//GEN-HEADEREND:event_changeViewButtonActionPerformed
        if (chrono instanceof AnalogChronometer)
        {
            chrono = new Chronometer(new Point(10,20));
            changeViewButton.setText("Analog View");
        }
        else
        {
            chrono = new AnalogChronometer();
            changeViewButton.setText("Digital View");
        }
        drawPanel.setChronometer(chrono);
        repaint();
    }//GEN-LAST:event_changeViewButtonActionPerformed
// Skipped: ... Look & Feel
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton changeViewButton;
    private DrawPanel drawPanel;
    private javax.swing.JButton resetButton;
    private javax.swing.JButton startButton;
    private javax.swing.JButton stepButton;
    private javax.swing.JButton stopButton;
    // End of variables declaration//GEN-END:variables
}
```