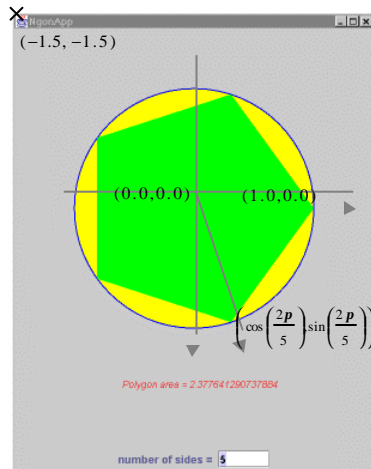


1.00 Clase 18

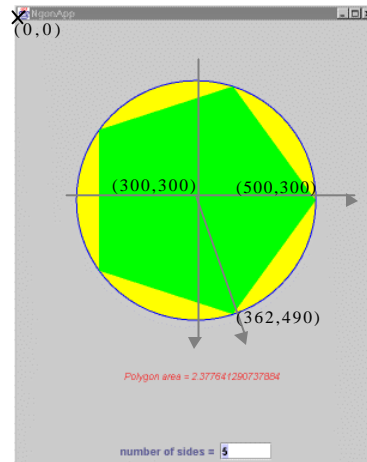
Transformaciones geométricas en la API 2D

Coordenadas transformadas en NgonApp



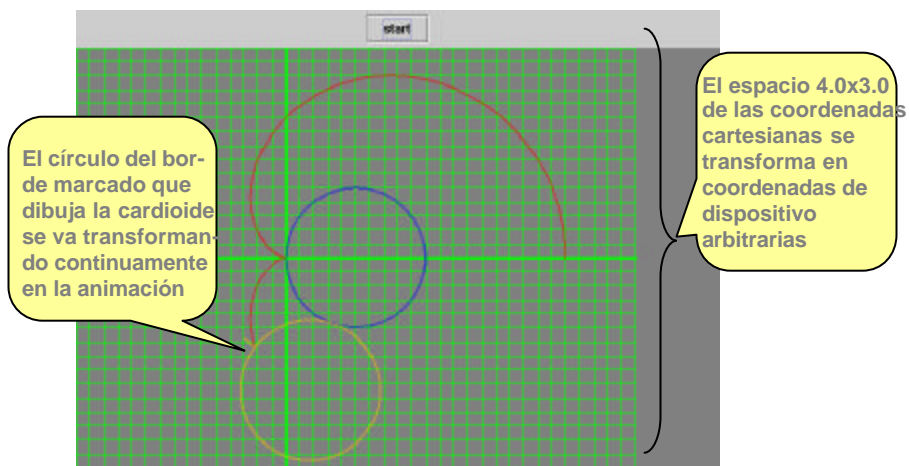
Coordenadas de píxel en NgonApp

```
static final float SCALE=200.0F;  
static final float tx = 1.5F;  
static final float ty = 1.5F;  
float transX( float x ) {  
    return ( x + tx ) * SCALE;  
}  
float transY( float y ) {  
    return ( y + ty ) * SCALE;  
}
```



3

Transformaciones en la curva cardioide



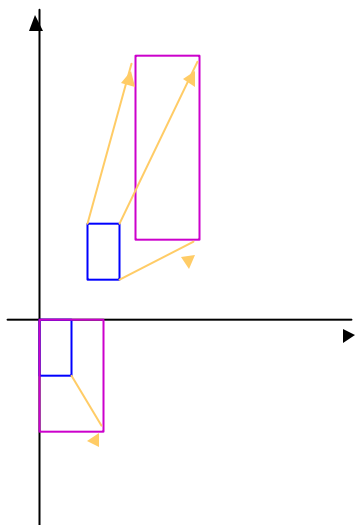
4

Transformaciones afines

- La API 2D proporciona un fuerte soporte para las *transformaciones afines*.
- Una transformación afín mapea las coordenadas 2D preservando la dirección y el paralelismo de las líneas.
- Todas las transformaciones afines pueden representarse mediante una matriz de coma flotante 3x3.
- Existe un número de transformaciones afines "primitivas" que se pueden combinar.

5

Escalas



$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x * x \\ s_y * y \\ 1 \end{bmatrix}$$

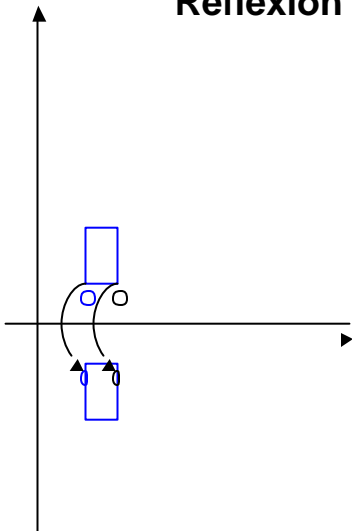
6

Observaciones sobre escalas

- Las operaciones básicas de ajuste de escala tienen lugar con relación al origen. Si la forma geométrica está en el origen, crece. Si está en otra parte, crece y se desplaza.
- s_x , ajuste de la escala en la dimensión x, no tiene que ser igual a s_y , ajuste de la escala en la dimensión y.
- Por ejemplo, para girar una figura verticalmente sobre el eje x ajuste la escala en $s_x=1$, $s_y=-1$.

7

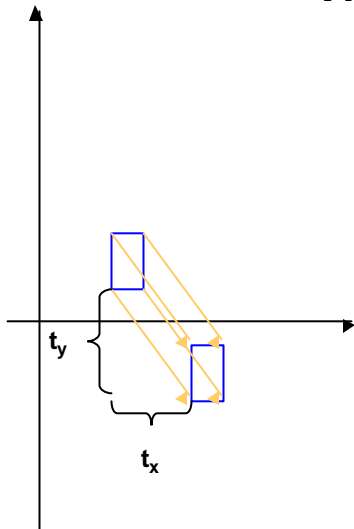
Reflexión en el ajuste de la escala



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ -y \\ 1 \end{bmatrix}$$

8

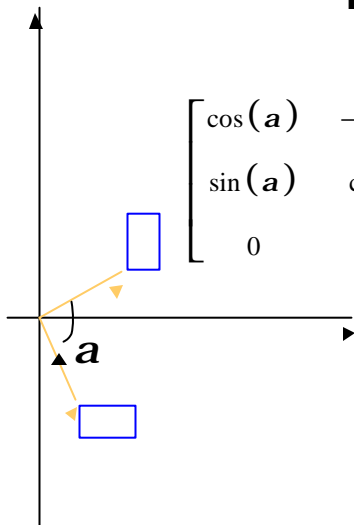
Traslación



$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+t_x \\ y+t_y \\ 1 \end{bmatrix}$$

9

Rotación



$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \cos(a) - y \sin(a) \\ x \sin(a) + y \cos(a) \\ 1 \end{bmatrix}$$

10

Composición de transformaciones

- Supongamos que queremos modificar la escala del punto (x, y) por 2 y luego girar 90 grados.

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right)$$

rotar

modificar escala

11

Composición de transformaciones, 2

Dado que la multiplicación de matrices es asociativa, podemos reescribirla del siguiente modo:

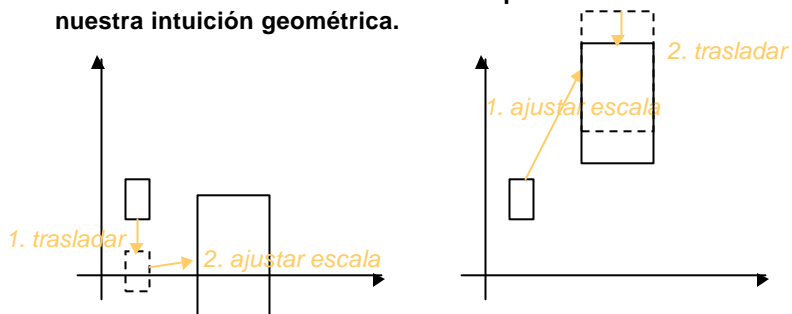
$$\left(\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

12

Composición de transformaciones, 3

- Dado que la multiplicación de matrices normalmente no es conmutativa, el orden de las transformaciones sí importa. Esto coincide con nuestra intuición geométrica.



- Si invertimos la matriz, deshacemos la transformación.

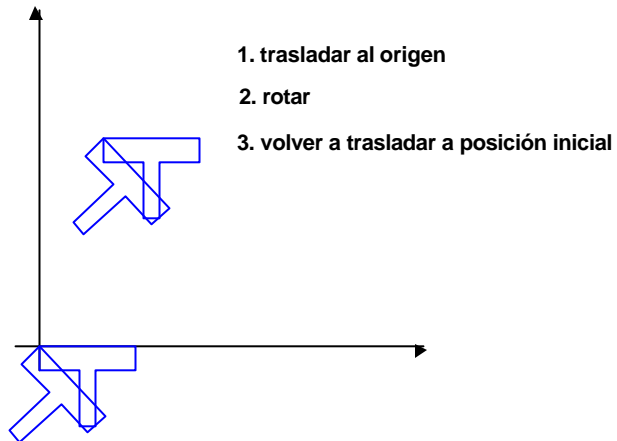
13

Las transformaciones y el origen

- Cuando transformamos una forma geométrica, transformamos cada uno de los puntos que definen la forma, y luego la volvemos a dibujar.
- Si ajustamos la escala o rotamos una forma que no esté anclada en el origen, ésta también se trasladará.
- Si simplemente queremos ajustar su escala o rotarla, entonces deberíamos trasladarla primero al origen, ajustar la escala o rotar y volver a trasladarla a su posición inicial.

14

Las transformaciones y el origen, 2



15

Transformaciones en la API 2D

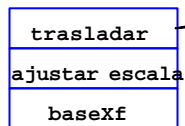
- Las transformaciones están representadas por instancias de la clase `AffineTransform` del paquete `java.awt.geom`.
- Construya una transformación compuesta mediante:
 1. La creación de una nueva instancia de `AffineTransform`
 2. La llamada a métodos para construir una *pila* de transformaciones básicas: último en llegar, primero que se aplica.
 - `translate(double tx, double ty)`
 - `scale(double sx, double sy)`
 - `rotate(double theta)`
 - `rotate(double theta, double x, double y)`
rota sobre (x,y)

16

Ejemplo de transformación

```
baseXf = new AffineTransform();
baseXf.scale( scale, -scale );
baseXf.translate( -uRect.x, -uRect.y );
```

Si ahora aplicamos baseXF, éste hará primero la traslación, luego ajustará la escala. Recuerde que en Java las transformaciones se construyen como una pila: último en llegar, primero que se aplica.



Primero que se aplica

17

De vuelta a la Cardioide

Construyamos nuestro sistema de coordenadas:

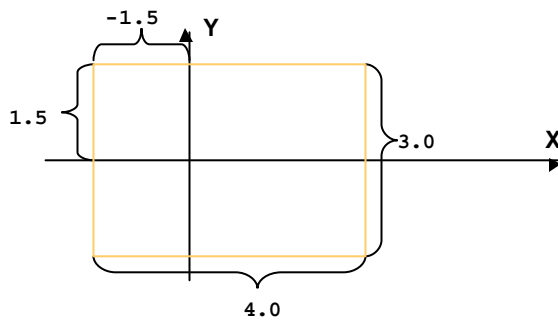
```
public class Cardioid extends JFrame {
    private CardioidGraph graph;
    private Rectangle2D.Float uSpace;

    public static void main( String [] args ) {
        Rectangle2D.Float uS =
            new Rectangle2D.Float(-1.5F,1.5F,4.0F,3.0F);
        Cardioid card = new Cardioid( uS );
        card.setSize( 640, 480 );
        card.setVisible( true );
    }
}
```

18

Cardioid Cartesian Space (espacio cartesiano cardiode)??

`Rectangle2D.Float(-1.5F,1.5F,4.0F,3.0F)`
representa el área de las coordenadas cartesianas en
la que dibujaremos nuestro gráfico:



19

CardioidGraph

```
public Cardioid( Rectangle2D.Float uS ) {  
    uSpace = uS;  
    graph = new CardioidGraph( uSpace );  
    . . .  
  
public class CardioidGraph extends GraphPanel  
    implements ActionListener {  
  
    public CardioidGraph( Rectangle2D.Float uR ) {  
        super( uR );  
        . . .  
    }  
}
```

20

GraphPanel

```
public class GraphPanel extends JPanel {
    protected Rectangle2D.Float uRect;
    protected AffineTransform baseXf = null;
    protected Dimension curDim = null;
    protected double scale;
    private GeneralPath axes = null;

    public GraphPanel( Rectangle2D.Float uR ) {
        uRect = (Rectangle2D.Float) uR.clone();
    }
}
```

21

GraphPanel, paintComponent()

```
public void paintComponent( Graphics g ) {
    super.paintComponent( g );
    Graphics2D g2 = (Graphics2D) g;
    if ( ! getSize().equals( curDim ) )
        doResize();
    drawAxes( g2 );
    drawGrid( g2 );
}
```

22

GraphPanel, doResize()

```
private void doResize() {
    curDim = getSize();
    hScale = curDim.width / uRect.width;
    vScale = curDim.height / uRect.height;
    scale = Math.min( hScale, vScale );
    baseXf = new AffineTransform();
    baseXf.scale( scale, -scale );
    baseXf.translate( -uRect.x, -uRect.y );

    axes = createAxes();
    grid = createGrid();
}
```

23

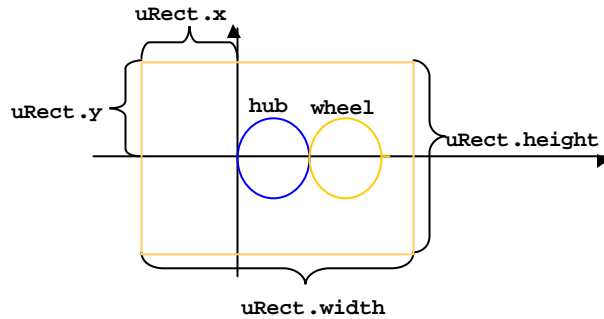
Creación y diseño de los ejes

```
private GeneralPath createAxes() {
    GeneralPath path = new GeneralPath();
    path.moveTo( uRect.x, 0F );
    path.lineTo( uRect.x + uRect.width, 0F );
    path.moveTo( 0F, uRect.y );
    path.lineTo( 0F, uRect.y - uRect.height );
    return path;
}

private void drawAxes( Graphics2D g2 ) {
    g2.setPaint( Color.green );
    g2.setStroke( new BasicStroke(3) );
    g2.draw(baseXf.createTransformedShape(axes));
}
```

24

Inicialización de CardioidGraph



25

Inicialización de CardioidGraph,2

```
public CardioidGraph( Rectangle2D.Float uR ) {  
    super( uR );  
    diam = uRect.width / 4;  
    hub = new Ellipse2D.Float(0F, -diam/2, diam, diam);  
    tick = uRect.width / 40;  
    wheel = new GeneralPath();  
    Shape s = new Ellipse2D.Double(diam, -diam/2,  
                                   diam, diam);  
    wheel.append( s, false );  
    wheel.moveTo( 2*diam, 0F );  
    wheel.lineTo( 2*diam + tick, 0F );  
}
```

26

Timers (temporizadores)

- Swing proporciona una clase utilitaria llamada `Timer` que facilita la construcción de animaciones.
- Los `Timers` marcan tiempos en un intervalo establecido que se puede configurar; estos tiempos se reciben como `ActionEvents`.

```
public class TimerUser
    implements ActionListener {
    private Timer timer = null;
    private int tIval = 100; // intervalo en milisegundos

    public TimerUser()
    { timer = new Timer( tIval, this ); }

    public void start()
    { timer.start(); }

    public void actionPerformed( ActionEvent e )
    { /*repite la misma acción en cada tiempo*/ }
```

27

Métodos de Timer

```
Timer( int tickMillis, ActionListener l )
void start()
void stop()
boolean isRunning()
void setRepeats(boolean repeats)
boolean isRepeats()
void setCoalesce(boolean coalesce)
boolean isCoalesce()
```

28

CardioidGraph, start()

```
public void start() {
    if ( timer != null ) {
        timer.stop();
    }
    timer = new Timer( tIval, this );

    curve = new GeneralPath();
    curve.moveTo( 2F, 0F );

    tCount = 0;
    repaint();
    timer.start();
}
```

29

CardioidGraph, actionPerformed()

```
public void actionPerformed( ActionEvent e ) {
    tCount++;
    double theta = tCount * Math.PI / 180;
    double sint = Math.sin( theta );
    double cost = Math.cos( theta );
    double cx = cost + cost*cost;
    double cy = sint + sint*cost;
    curve.lineTo( (float)cx, (float) cy );
    if ( tCount >= 360 ) {
        timer.stop();
        timer = null;
    }
    repaint();
}
```

30

CardioidGraph, paintComponent()

```
public void paintComponent( Graphics g ) {  
    super.paintComponent( g );  
    Graphics2D g2 = (Graphics2D) g;  
    drawHub( g2 );  
    drawCurve( g2 );  
    drawWheel( g2 );  
}
```

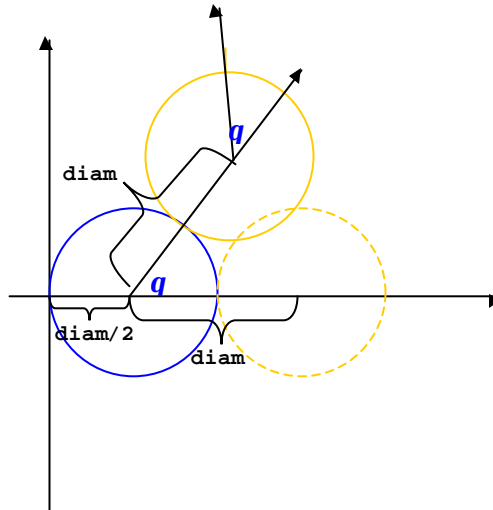
31

CardioidGraph, drawCurve()

```
private void drawCurve( Graphics2D g2 ) {  
    if ( curve == null ) return;  
    // hacer curva translúcida  
    Composite c = g2.getComposite();  
    Composite hc = AlphaComposite.getInstance(  
        AlphaComposite.SRC_OVER, .5F );  
    g2.setComposite( hc );  
  
    g2.setPaint( Color.red );  
    g2.setStroke( new BasicStroke( 2 ) );  
    g2.draw( baseXf.createTransformedShape( curve ) );  
    g2.setComposite( c );  
}
```

32

Geometría de la cardioide



33

CardioidGraph, drawWheel()

```
private void drawWheel( Graphics2D g2 ) {  
    Composite c = g2.getComposite();  
    Composite hc = AlphaComposite.getInstance(  
        AlphaComposite.SRC_OVER, .5F );  
    g2.setComposite( hc );  
    g2.setPaint( Color.orange );  
    g2.setStroke( new BasicStroke( 2 ) );  
    double theta = tCount * Math.PI / 180;  
    AffineTransform whlXf = new AffineTransform(baseXf);  
    whlXf.rotate( theta, diam/2, 0.0 );  
    whlXf.rotate( theta, 3*diam/2, 0.0 );  
    g2.draw( whlXf.createTransformedShape( wheel ) );  
    g2.setComposite( c );  
}
```

34