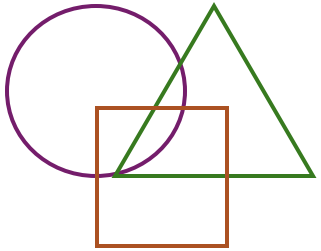


# BSF4ooRexx850 JDOR: Java 2D Drawing for ooRexx

Elif Deger

*RexxLA Symposium, 06.05.2025*

# *Agenda*

1. Introduction to JDOR and ooRexx
2. Java 2D API and AWT basics
3. JDOR Commands
4. Practical Examples
5. Conclusion

# *Java 2D Graphics in ooRexx with JDOR*

**Problem Statement**:

*Java 2D graphics are powerful but have complex syntax. How can Rexx simplify this?*

**Objective**:

*Introduce JDOR, a Rexx command handler that utilizes Java 2D API for accessible graphics programming.*

# *JDOR: Java 2D for ooRexx*

## What is JDOR?

- JDOR = Java 2D Drawing for ooRexx
- A **Rexx command handler** that:
  -> Bridges ooRexx + Java 2D using BSF4ooRexx850
  -> With BSF4ooRexx850, you can use all Java2D classes directly in ooRexx.
  -> Replaces Java verbosity with simple Rexx commands
- Enables drawing shapes, text, images…

# Why ooRexx + Java?

**Key Advantages**

✓ **No Java knowledge, syntax required** : Commands mirror Graphics2D methods
✓ **Full Java 2D power** : Shapes, text, images, transformations
✓ **Seamless integration**: call addJdorHandler  // Connect to Java 2D

# Java Graphics Creation

**Core Components**
**AWT (Abstract Window Toolkit)**

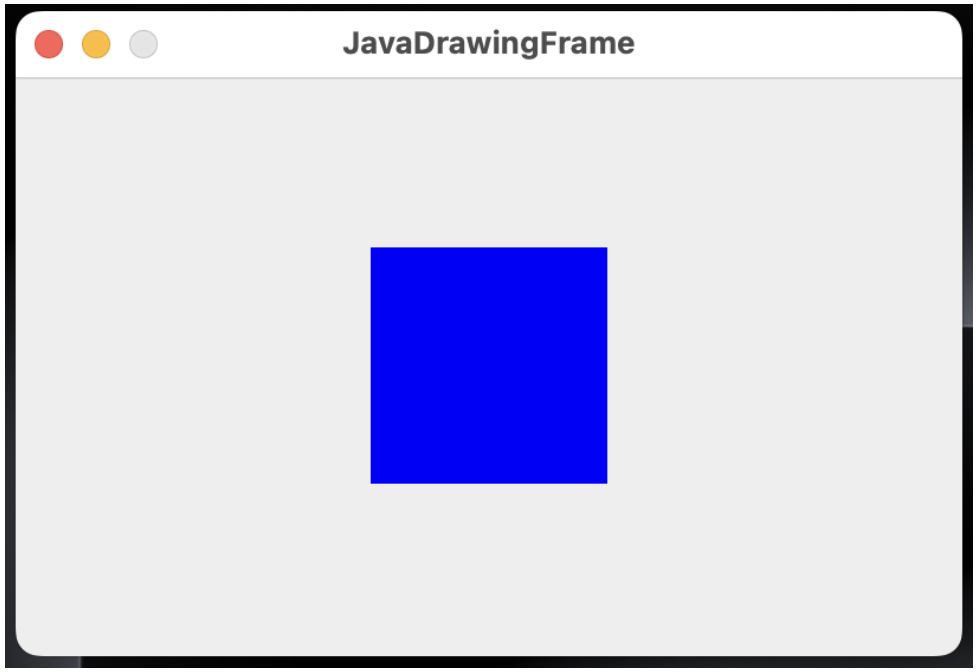
- Supports Graphical User Interface (GUI) programming
- Features:
  -> UI elements (buttons, textboxes)
  -> Layout managers
  -> Event handling

**Java 2D API (extends AWT)**

- Enhances the graphics, text, and imaging capabilities of the AWT
  - Advanced 2D graphics/text/imaging
  - Key class: **Graphics2D**

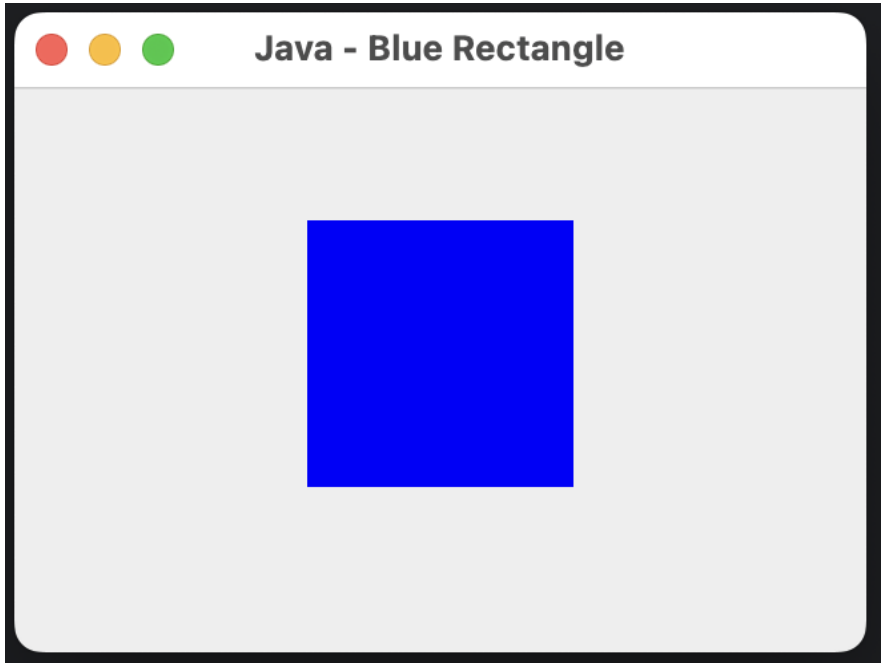
# JDOR in Action!

```
1  call addjdorhandler
2  address jdor
3
4  new 400 245
5  winshow
6
7  goto 150 72
8  color blue
9  fillrect 100 100
10
11 sleep 60
12 ::requires "jdor.cls"
```



# Comparison To Java

```java
import javax.swing.*;
import java.awt.*;

public class Main extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;

        // Set color to blue and draw filled rectangle
        g2d.setColor(Color.BLUE);
        g2d.fillRect( x: 110,  y: 50,  width: 100,  height: 100);
    }

    public static void main(String[] args) {
        // Create the GUI window
        JFrame frame = new JFrame( title: "Java - Blue Rectangle");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize( width: 320,  height: 240);

        // Add drawing panel
        frame.add(new Main());

        // Show the window
        frame.setVisible(true);
    }
}
```



# Some JDOR Commands

| Command | Action | Example |
|---|---|---|
| color | Set RGB/named color | color blue |
| drawRect | Draw rectangle outline | drawRect 50 50 100 60 |
| fillOval | Fill circle | fillOval 50 50 80 80 |
| rotate | Rotate next drawing | rotate 45 |
| loadImage | Import image file | loadImage logo "img.png" |

.
.
.

For more: jdor_doc.html

-> The JDOR commands are documented in the BSF4ooRexx850 installation folder

# JDOR Overview

To begin working with JDOR, the following code block should alyaws be executed:

1 call addjdorhandler -- load and add the java rexx command handler, using default name: jdor

2 address jdor  -- set default environment to jdor

These instructions ensure that the JDOR package is properly loaded and set as the default environment for further operations.

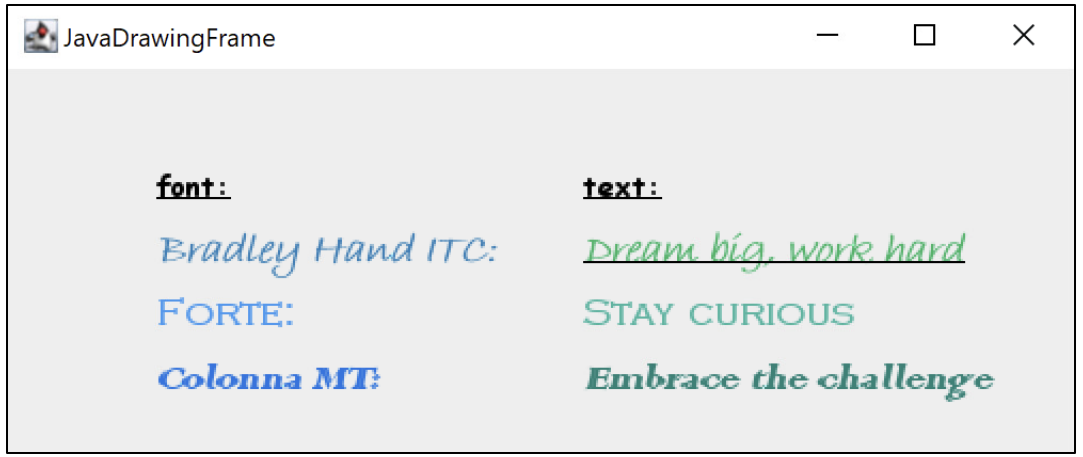
# *Text Rendering with JDOR*

**JDOR Features**

- Font management
- Text rendering
- Color selection
- Positioning with goto

**Commands Used**

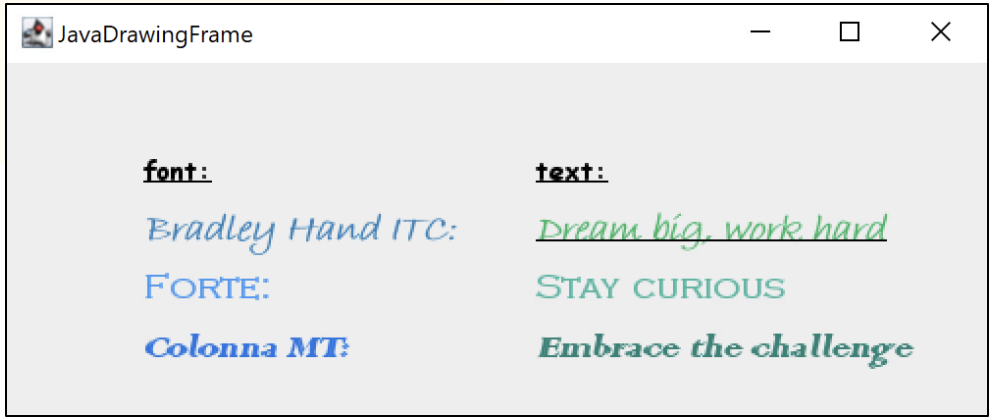
- **fontSize** 14 / **fontStyle** 1 → Set font size and style
- **font** 14_Comic "Comic Sans MS" → Use installed system font
- **goto** 70 60 → Move cursor
- **drawString** "font:" → Draw text
- **stringBounds** → Measure text width
- Multiple fonts displayed:
  - Comic Sans MS
  - Bradley Hand ITC
  - Copperplate Gothic Light
  - Colonna MT




JDOR-text.rxj

# Text Rendering with JDOR

```
call addJdorHandler -- load
address jdor -- set default environment to JDOR

-- setting the colors
color enchanting 41 128 185
color warmSpring 60 154 242
color cerulean 24 117 227
color shallowSea 40 180 99
color lagoon 62 181 161
color mosaicTile 29 130 118

--Creating and showing a new window
win_width = 500
win_height = 180
winSize win_width win_height
winShow

fontSize 14
fontStyle 1 -- 1=BOLD
font 20_Comic "Comic Sans MS"
goto 70 60
color black
font 20_Comic
drawString "font:"
stringBounds "font:"
parse var rc x " " y " " width " " height
say width
color black
drawLine 70+width 60
goto 270 60
drawString "text:"
stringBounds "text:"
parse var rc x " " y " " width " " height
say width
color black
drawLine 270+width 60
--create the 1st
fontSize 20
fontStyle 3 -- 3=BOLD+ITALIC
font 20_Bradley "Bradley Hand ITC"

goto 270 90
color shallowSea
font 20_Bradley
drawString "Dream big, work hard"
stringBounds "Dream big, work hard"
parse var rc x " " y " " width " " height
say width
color black
drawLine 270+width 90

goto 70 90
color enchanting
font 20_Bradley
drawString "Bradley Hand ITC:"
stringBounds "Bradley Hand ITC:"
parse var rc x " " y " " width " " height
say width
--create a 2nd
fontSize 18
fontStyle 1
font 18_Copper "Copperplate Gothic Light"

goto 270 120
color lagoon
font 18_Copper
drawString "Stay curious"
stringBounds "Stay curious"
parse var rc x " " y " " width " " height
say width

goto 70 120
color warmSpring
drawString "Forte:"
stringBounds "Forte:"
parse var rc x " " y " " width " " height
say width
--create a 3rd
fontSize 20
FontStyle 3
font 20_Colonna "Colonna MT"

goto 270 150
color mosaicTile
font 20_Colonna
drawString "Embrace the challenge"
stringBounds "Embrace the challenge"
parse var rc x " " y " " width " " height
say width

goto 70 150
color cerulean
font 20_Colonna
drawString "Colonna MT:"
stringBounds "Colonna MT:"
parse var rc x " " y " " width " " height
say width

sleep 40
::requires "jdor.cls"
```



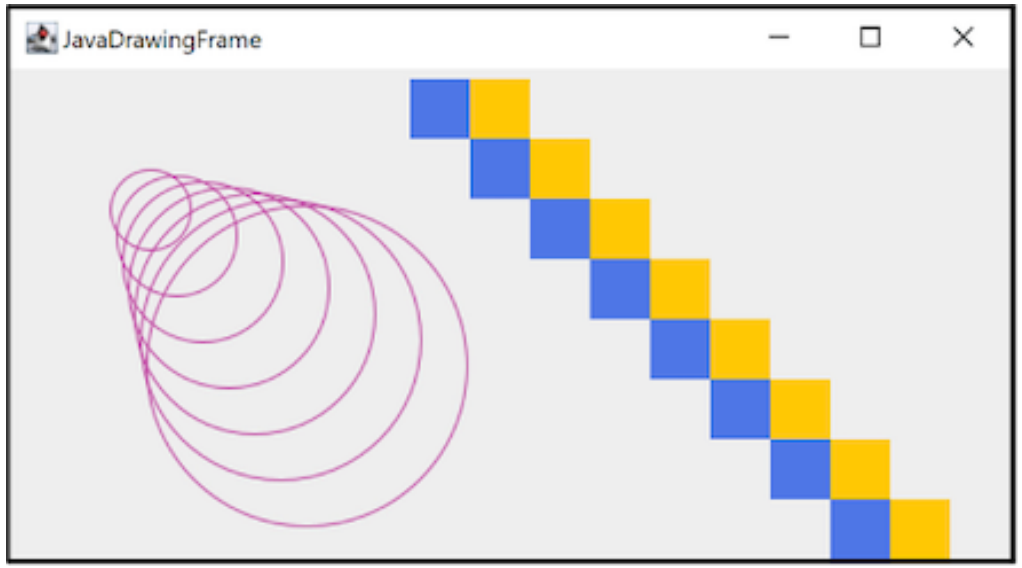
# Drawing with JDOR 1

**JDOR Features**

- Shape creation (ovals, rectangles)
- Fill vs. outline control
- Precise positioning
- Pattern generation

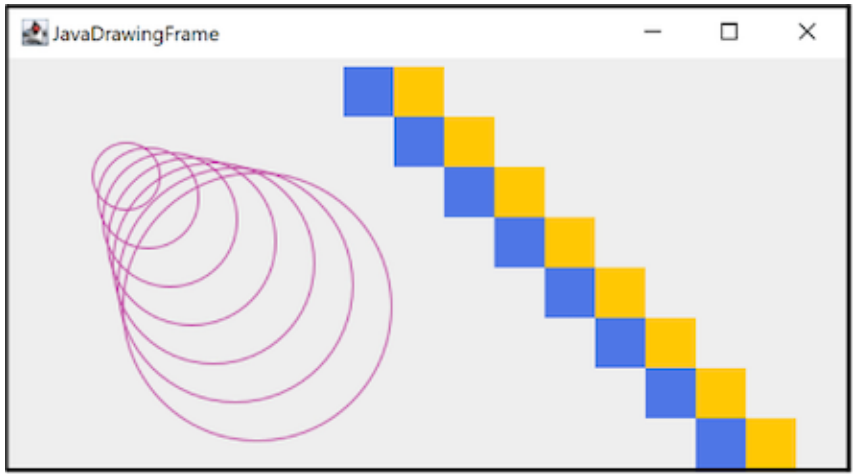
**Commands Used**

- **color** mulberry 192 69 161 → Set custom RGB color
- **goto** x y → Position cursor (e.g., goto 50 50)
- **drawOval** w h / **fillRect** w h → Outline vs. filled shapes
- do i = 1 to 10 → Loop for pattern creation




JDOR-drawing.rxj

# Drawing with JDOR 1




JDOR-drawing.rxj

```
call addjdorhandler
address jdor

--Creating and showing a new window
win_width = 500
win_height = 245
new win_width win_height
winshow

-- Set the color
color mulberry 192 69 161
-- Draw the ovals
goto 50 50
drawOval 40 40
goto 53 53
drawOval 60 60
goto 56 56
drawOval 80 80
goto 59 59
drawOval 100 100
goto 62 62
drawOval 120 120
goto 65 65
drawOval 140 140
goto 68 68
drawOval 160 160

-- Define the size of the rectangles
rect_width = 30
rect_height = 30

-- Set the initial position for the first rectangle
start_x = 200
start_y = 5
-- Draw the pattern of Sapphire colored rectangles
do i = 1 to 10
-- Calculate the position of the current rectangle
  rect_x = start_x + (i - 1) * rect_width
  rect_y = start_y + (i - 1) * rect_height

  goto rect_x rect_y
color sapphire 79 118 231
  fillrect rect_width rect_height
end

-- Define the size of the rectangles
rect_width = 30
rect_height = 30

-- Set the initial position for the first rectangle
start_x = 230
start_y = 5
-- Draw the pattern of orange rectangles
do i = 1 to 10
-- Calculate the position of the current rectangle
rect_x = start_x + (i - 1) * rect_width
rect_y = start_y + (i - 1) * rect_height

goto rect_x rect_y
color orange
fillrect rect_width rect_height
end

sleep 60
::requires "jdor.cls"
```

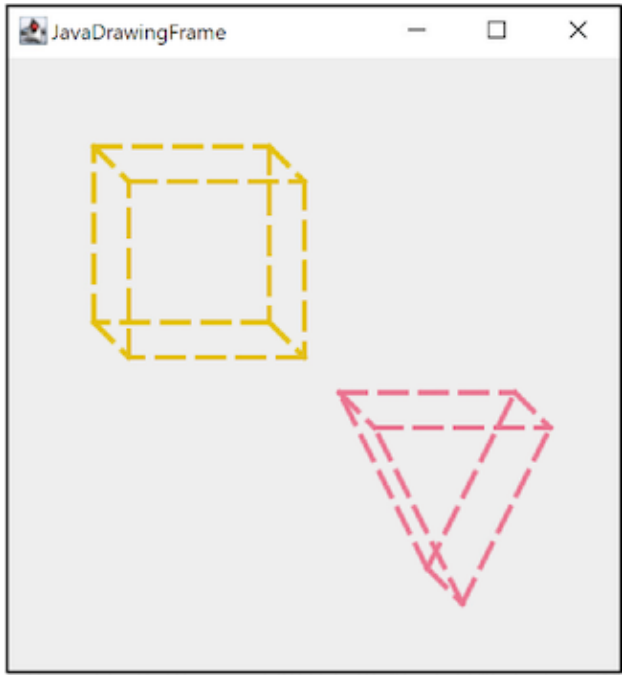
# Drawing with JDOR 2

**JDOR Features**

- 3D wireframe construction
- Perspective techniques
- Custom line styling
- Multi-shape composition

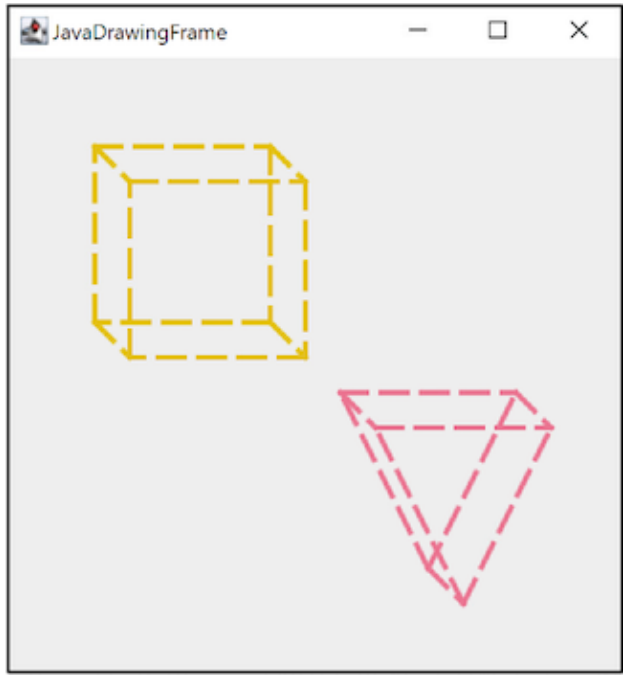
**Commands Used**

- **stroke** stroke1 3 2 0 10 "dashphase_stroke1" 0 → Dashed lines
- **color** LemonLine 228 192 0 → Custom color
- **goto** x y → Position cursor
- **drawLine** x y → Connect vertices




JDOR-CubePyramid.rxj

# Drawing with JDOR 2




JDOR-CubePyramid.rxj

```
call addJdorHandler
address jdor

-- Creating and showing a new window
win_width = 350
win_height = 350
NEW win_width win_height
WINSHOW

-- Setting the colors
color LemonLime 228 192 0
color peonypink 235 117 145

-- Draw the Cube
--Creating / Saving stroke
dashphase_stroke1=bsf.createJavaArrayOf("float.class", 15, 8, 15,8)
STROKE strokeA 3 2 0 10 "dashphase_stroke1" 0
-- Draw the front face of the cube
color LemonLime
goto 50 50
STROKE strokeA
drawLine 150 50
goto 150 50
drawLine 150 150
goto 150 150
drawLine 50 150
goto 50 150
drawLine 50 50

-- Draw the back face of the cube
goto 70 70
drawLine 170 70
goto 170 70
drawLine 170 170
goto 170 170
drawLine 70 170
goto 70 170
drawLine 70 70

-- Connect the corresponding vertices of the front and back faces
goto 50 50
drawLine 70 70
goto 150 50
drawLine 170 70
goto 150 150
drawLine 170 170
goto 50 150
drawLine 70 170

-- Draw the Triangle
-- Draw the front face of the triangle
color peonypink
goto 190 190
drawLine 290 190
goto 240 290
drawLine 190 190
goto 240 290
drawLine 290 190

-- Draw the back face of the triangle
goto 210 210
drawLine 310 210
goto 260 310
drawLine 210 210
goto 260 310
drawLine 310 210

-- Connect the corresponding vertices of the front and back faces
goto 190 190
drawLine 210 210
goto 290 190
drawLine 310 210
goto 240 290
drawLine 260 310

sleep 60
::requires "jdor.cls"
```

# Visualizing with Images

**Key Features**

✓ **Image Loading & Display**

**loadImage** Pyramids_of_Giza "py.jpg"
**drawImage** Pyramids_of_Giza

✓ **Graphical Overlays**

- Shapes: **fillRect**, **fillOval**
- Text: Custom fonts/colors for pyramid labels

✓ **Save Capability**

**saveImage** "Names_of_Giza_Pyramids.png"

JDOR-images.rxj

py.png


Names_of_Giza_Pyramids.png

# Visualizing with Images

```
call addJdorHandler
address jdor

  --Creating and showing a new window
win_width = 500
win_height = 308
winsize win_width win_height
winshow

 -- import the image
loadImage Pyramids_of_Giza "py.png"
drawImage Pyramids_of_Giza

 -- draw and fill rectangle
color powderblue
goto 140 200
drawRect 60 40
fillRect 60 40

 -- draw and fill circle
goto 170 210
color thistle
drawOval 70 70
fillOval 70 70

 -- draw rectangle
goto 260 220
color tropicaldream
drawRect 70 70

 -- 1st Pyramid
fontSize16
fontStyle 1 -- 1=BOLD
font 16_Berlin_S "Berlin Sans FB"
color silkribbon
goto 50 100
drawString "MENKAURE"
stringBounds"MENKAURE"
parse var rc x " " y " " width " " height
say width
color citron
drawLine50 + width 100

 --2nd Pyramid
fontSize 32
font 32_Forte "Forte"
color blazeorange
goto 250 50
drawString "KHUFU"

 -- 3rd Pyramid
fontSize 24
font 24_Arabic_T "Arabic Typesetting"
color jamaicansea
goto 400 110
drawString "KHAFRE"

 -- Save the image
saveImage "Names_of_Giza_Pyramids.png"

sleep 40
::requires "jdor.cls"
```

JDOR-images.rxj

py.png


Names_of_Giza_Pyramids.png

# Rotate, Scale, Translate and Shear

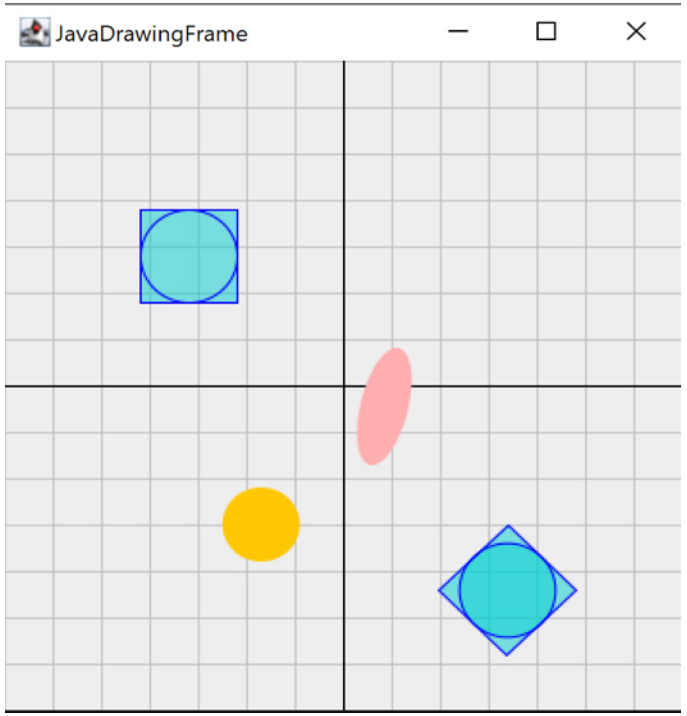
**Key Transformation Methods**

✓ **Rotate**
**rotate** — to specify an angle of rotation in radians

✓ **Scale**
**scale** — to specify a scaling factor in the *x* and *y* directions

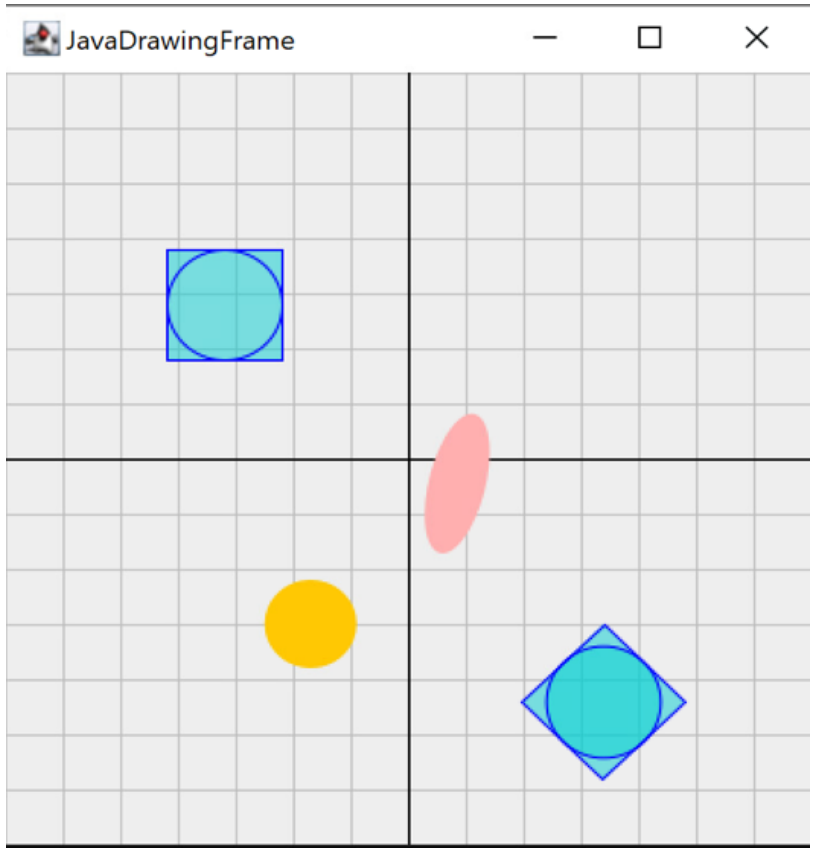
✓ **Shear**
**shear** — to specify a shearing factor in the *x* and *y* directions

✓ **Translate**
translate — to specify a translation offset in the *x* and *y* directions




JDOR-manipulate.rxj

# Rotate, Scale, Translate and Shear




JDOR-manipulate.rxj

```
call addJdorHandler
address jdor
--Creating and showing a new window
win_width = 350
win_height = 350
WINSIZE win_width win_height
WINSHOW

 --setting the colors
color coordinate_system 190 190 190 200
color middle 0 0 0 255

   --drawing the system
color coordinate_system
   do i=0 to win_width by 25
goto i 0
drawline i win_height
   end
   do i=0 to win_height by 25
goto 0 i
drawline win_width i
   end
color middle
goto win_width/2 0
drawline win_width/2 win_height
goto 0 win_height/2
drawline win_width win_height/2

-- Applying methods
-- draw two lines forming a big X
moveTo 70 80          -- currX=70, currY=80

color pantone 0 206 209 127
fillRect 50 50
color blue
drawRect 50 50
color blue
drawOval 50 50
translate 260 250
moveTo 0 0
rotate 45
color pantone
fillRect 50 50
fillOval 50 50
color blue
drawRect 50 50
color blue
drawOval 50 50

"goto 150 15"
drawPolygon 50 50
rotate 45
drawPolygon 50 50
rotate 45

goto 70 70
color orange
fillOval 40 40
"shear -1 0"
color pink
fillOval 40 40

say "press enter to end."; parse pull
sleep 400
::requires "jdor.cls"
```

# *Affine Transformation: Rotating Triangle*

**Key Features**

✓ **Shape Creation**

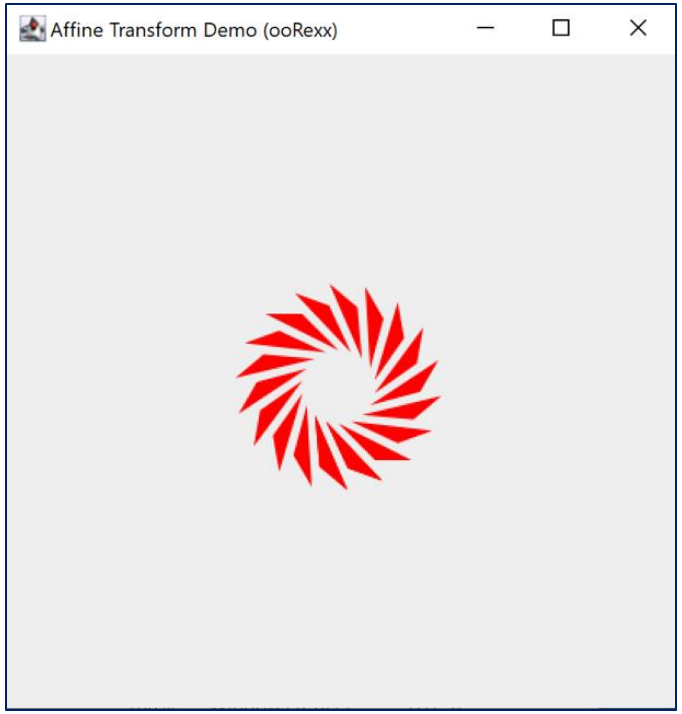
```
shape myP polygon "(20,0,40)" "(40,20,40)" 3  /* Triangle */
```

✓ **Transformations**

```
translate 200 200 /* Reposition */
scale 1.1 1.1 /* Enlarge */
rotate 20 /* Tilt */
```

✓ **Animation Loop**

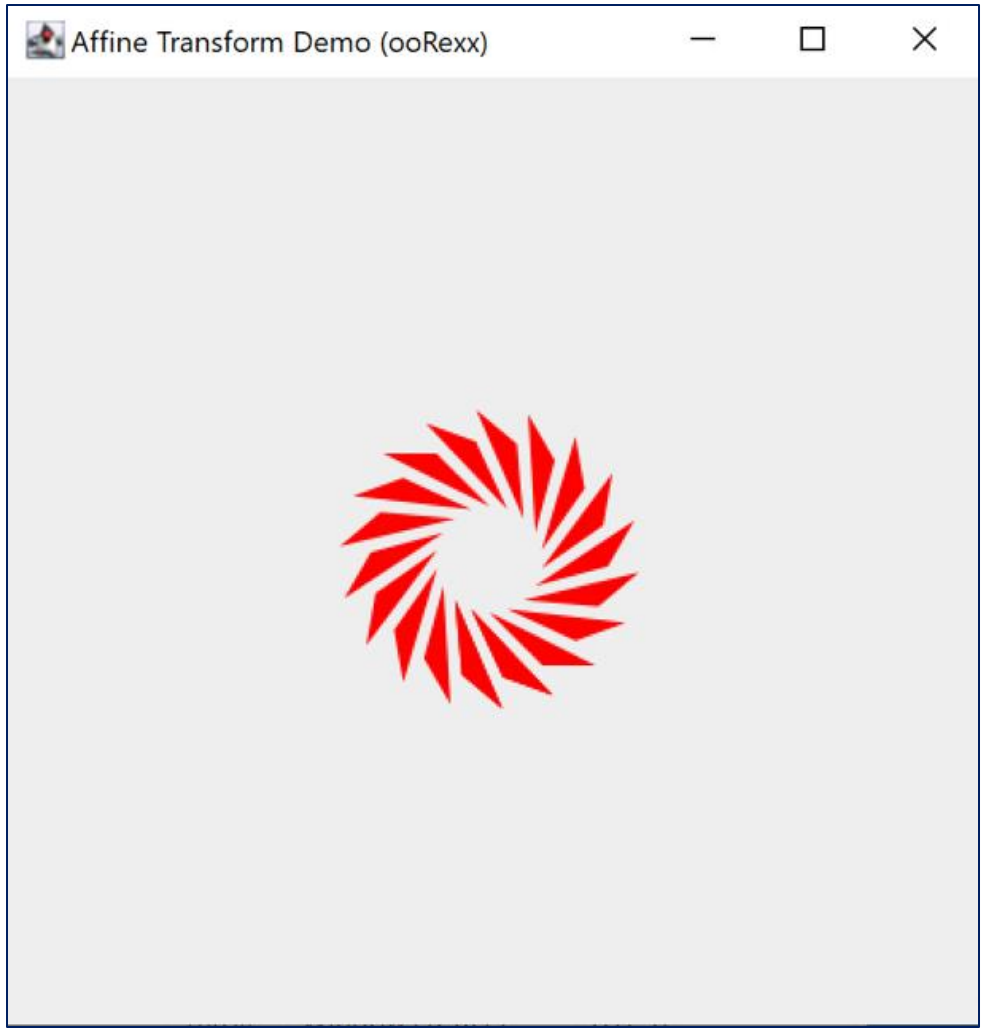
```
do 20             /* Repeat 20x */
  fillShape myP     /* Draw triangle */
  rotate 20        /* Incremental rotation */
end
```




JDOR-AffineTransformation.rxj

# *Affine Transformation: Rotating Triangle*

```
1  jdh=.bsf~new("org.oorexx.handlers.jdor.JavaDrawingHandler")
2  say "JDOR version:" jdh~version
3  call BsfCommandHandler "add", "jdor", jdh
4  address jdor
5
6  newImage 300 300
7  winShow
8  winTitle "Affine Transform Demo (ooRexx)"
9
10 polygonXs="(20,0,40)" -- define three x coordinates for the triangle
11 polygonYs="(40,20,40)" -- define three y coordinates for the triangle
12 shape myP polygon polygonXs polygonYs 3 -- create triangle shape
13
14 translate 200 200 -- move origin (x=200, y=200)
15 scale 1.1 1.1 -- increase the triangle shape size 10%
16 rotate 20 -- rotate by 20 degrees
17 color red -- set color to red
18 do 20
19   fillShape myP -- fill (and show) the triangle shape
20   rotate 20
21 end
22
23 say 'Hit <enter> to proceed (end) ...'
24 parse pull data -- wait until user presses <enter> on the keyboard
```




JDOR-AffineTransformation.rxj

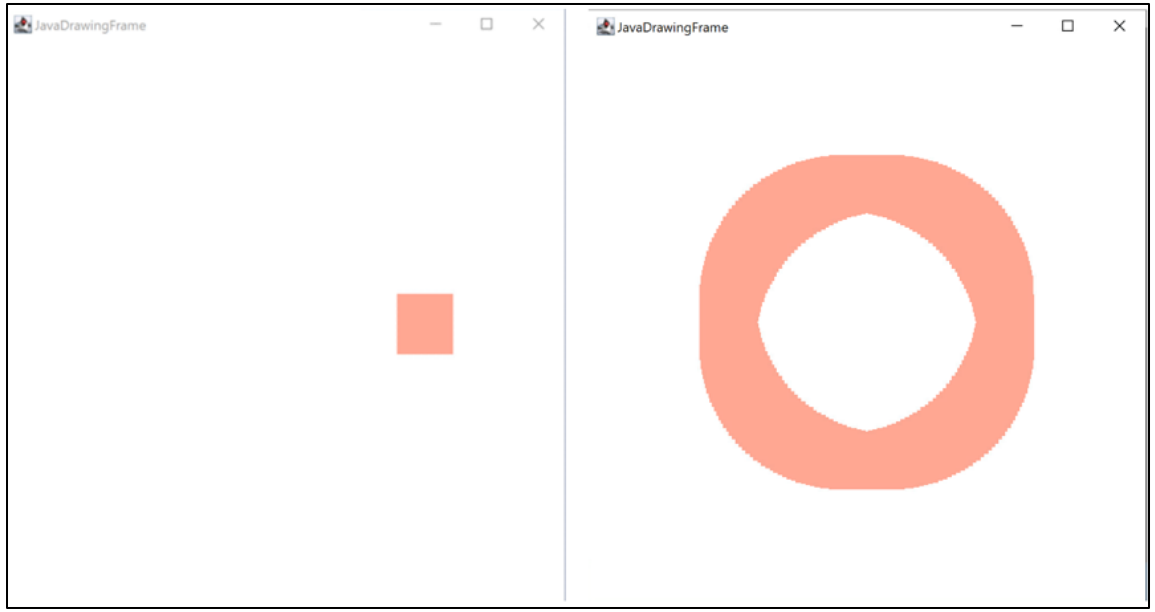
# *Animation with JDOR: Circular Motion*

## Core Components

✓ **Math Integration**
**call** bsf.import "java.lang.math", "calc"  /* Access sin/cos */

✓ **Motion Parameters**
**radius**: Circular path size
**speed**: Angular velocity (degrees/frame)
**square_size**: Object dimensions

✓ **Visual Settings**
**color**: desertSunrise 255 167 146



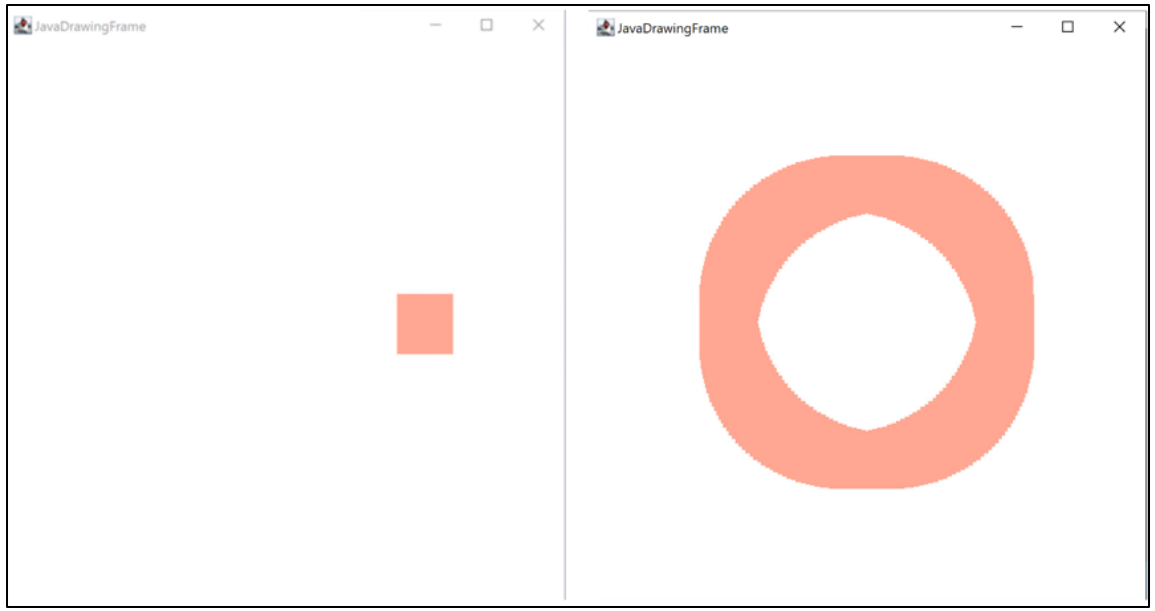

### Key Animation Technique

```
do forever
  currX = centerX + cos(angle)*radius
  currY = centerY + sin(angle)*radius
  goto currX currY
  fillRect square_size square_size
  angle = angle + speed
  sleep 0.01
end
```

JDOR-move.rxj

# *Animation with JDOR: Circular Motion*

```
call addjdorhandler
address jdor
call bsf.import "java.lang.math", "calc"

win_width = 500
win_height = 500
square_size = 50

speed = 2
color desertSunrise 255 167 146

winsize win_width win_height
new win_width win_height
background white
clearRect win_width win_height
winshow
color desertsunrise

centerX = win_width / 2
centerY = win_height / 2
radius = win_width / 4
angle = 0

do forever
getstate
currx = centerx + .calc~cos(.calc~toradians(angle)) * radius - square_size / 2
curry = centery + .calc~sin(.calc~toradians(angle)) * radius - square_size / 2

goto currx curry
fillRect square_size square_size
angle = angle + speed
sleep 0.01
end

::requires "jdor.cls"
```




JDOR-move.rxj

# *Animated Star Pattern with JDOR*

## Core Components

✓ **Math Integration**

**call** bsf.import "java.lang.Math", "calc"  /* For sin/cos */
delta_angle = .calc~toRadians(72)  /* Convert to radians */

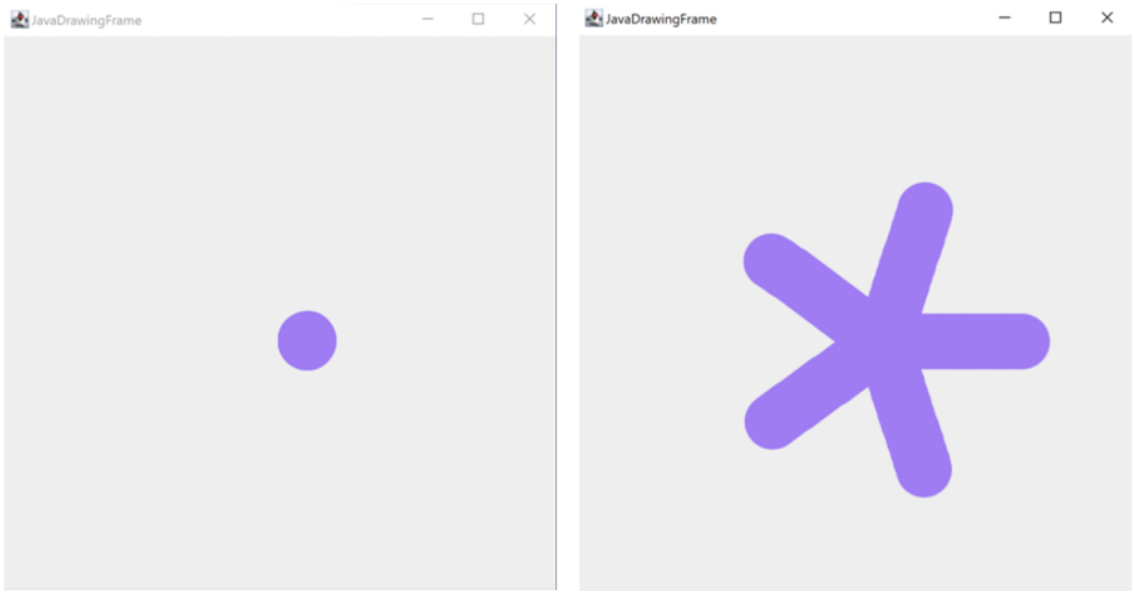
✓ **Star Geometry**

5 points (72° increments) via delta_angle

Circular motion with expanding/retracting radius

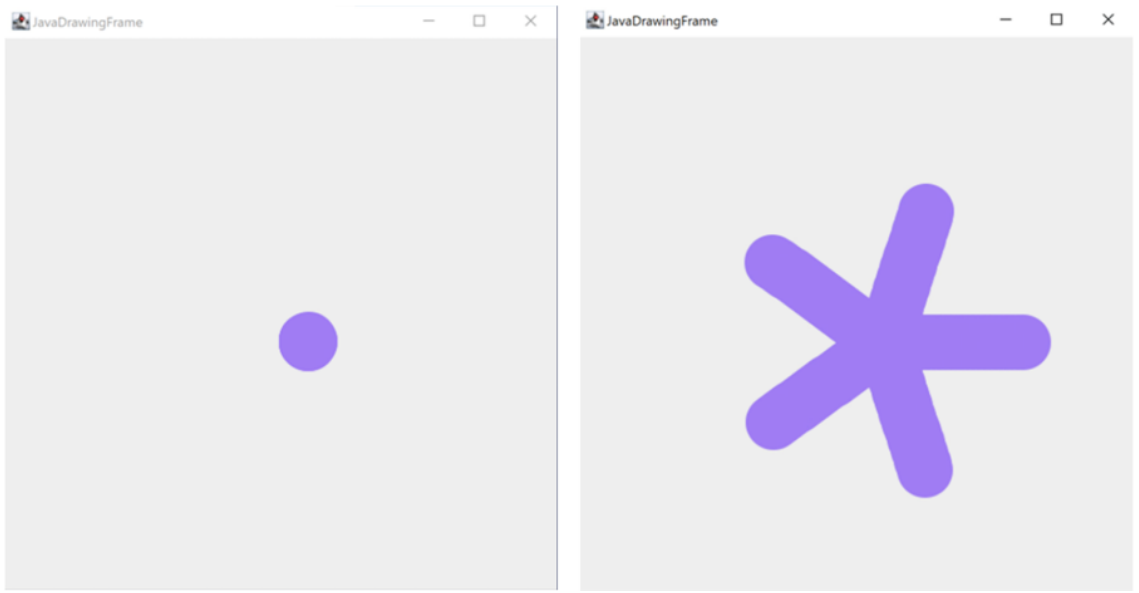
## Key Animation Technique

```
do forever
  currX = centerX + cos(angle)*distance
  currY = centerY + sin(angle)*distance
  goto currX currY
  fillOval star_size star_size
  angle = angle + 72°  /* 5-pointed star */
  distance = distance + speed
  if distance > radius then distance = 0
end
```




JDOR-PurpleStar.rxj

# *Animated Star Pattern with JDOR*




JDOR-PurpleStar.rxj

```
call addjdorhandler
address jdor
call bsf.import "java.lang.Math", "calc"

win_width = 500
win_height = 500
star_size = 50
speed = 1

color daylightlilac 158 124 243
winsize win_width win_height
new win_width win_height
background white
clearoval win_width win_height
winshow

color daylightLilac
centerX = win_width / 2
centerY = win_height / 2
radius = win_width / 4
angle = 0
delta_angle = .calc~toRadians(72)  -- 360 degrees divided by 5 sides of the star
distance = 0

 do forever
  getState
  currX = centerX + .calc~cos(angle) * distance
  curry = centery + .calc~sin(angle) * distance
  goto currx curry
  fillOval star_size star_size
  angle = angle + delta_angle
  if angle > 2 * .calc~pi then angle = angle - 2 * .calc~pi
  distance = distance + speed
  if distance > radius then distance = 0
 end
::requires "jdor.cls"
```

# *Rotating & Ascending Square Animation*

**Core Components**

✓ **Rotation Mechanics**

**rotate** center_x center_y angle (5° increments)

Pivot point: Window center

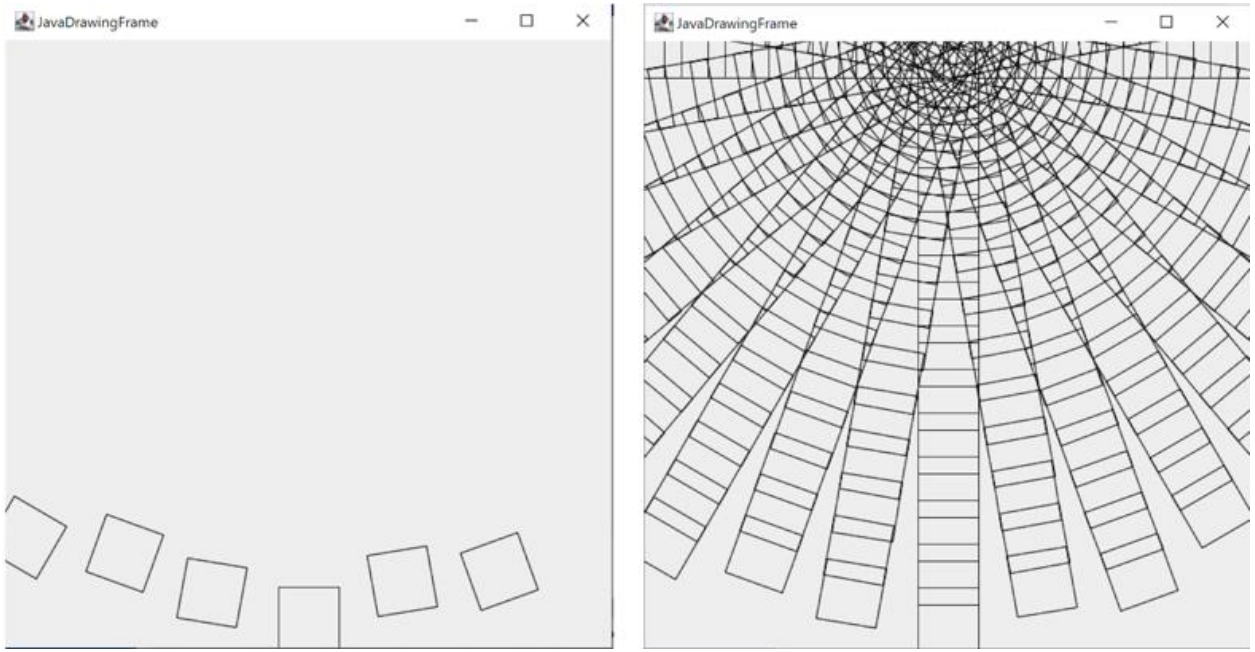
✓ **Motion Control**

Vertical ascent: square_y = square_y - 1

Frame timing: sleep 0.005 (smooth animation)

✓ **Initial Setup**

square_size = 50
center_x = 500 / 2  /* Window center */
square_y = 500 - square_size  /* Start at bottom */



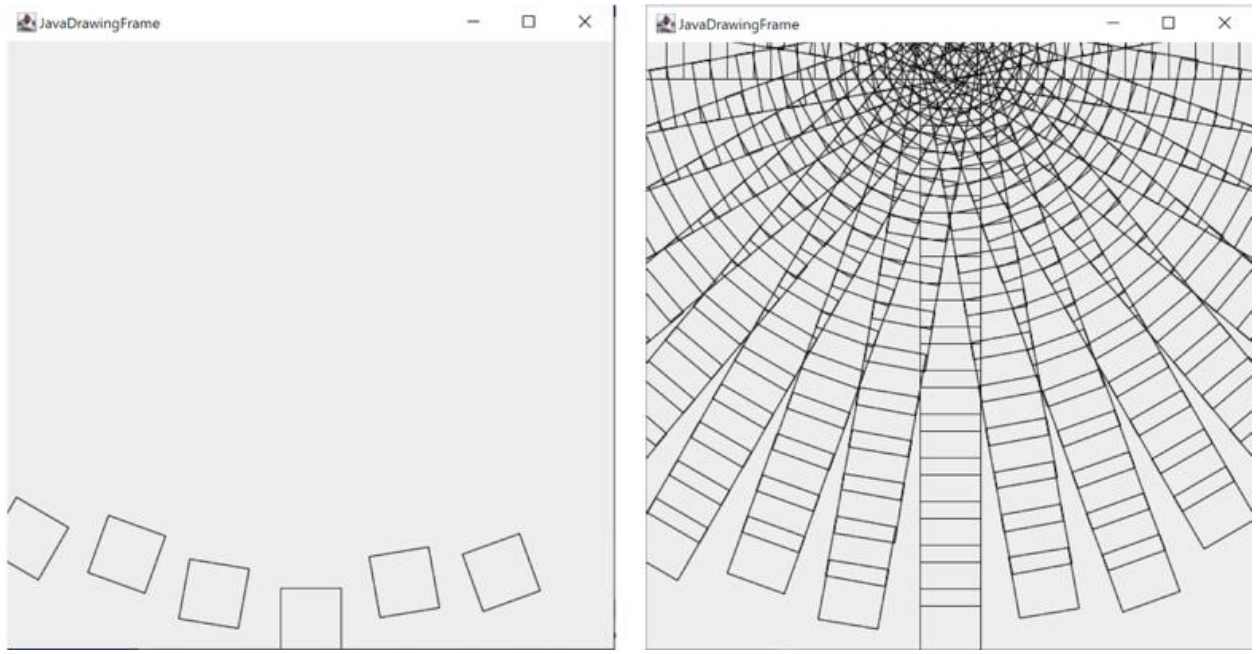

JDOR-RotatingSquare.rxj

**Key Animation Technique**

```
do while square_y > 0
  goto square_x square_y
  drawRect square_size square_size  /* Draw square */
  rotate center_x center_y angle   /* Pivot at window center */
  square_y = square_y - 1        /* Move upward */
  sleep 0.005                  /* Control speed */
end
```

# *Rotating & Ascending Square Animation*

```
call addJdorHandler
address jdor

-- creating and showing a new window
new 500 500
winShow

-- define the initial size of the square
square_size = 50

-- set the rotation angle
angle = 5

-- calculate the center coordinates of the window
center_x = 500 / 2
center_y = 500 / 2

-- calculate the starting position of the square at the bottom of the window
square_x = center_x - square_size / 2
square_y = 500 - square_size
-- draw and rotate the square
do while square_y > 0
  -- draw the square at the current position
  goto square_x square_y
  drawRect square_size square_size
 -- rotate the square
  rotate center_x center_y angle
 -- update the position of the square
  square_y = square_y - 1
  -- pause to observe the rotation
 sleep 0.005
end
sleep 60

::requires "jdor.cls"
```

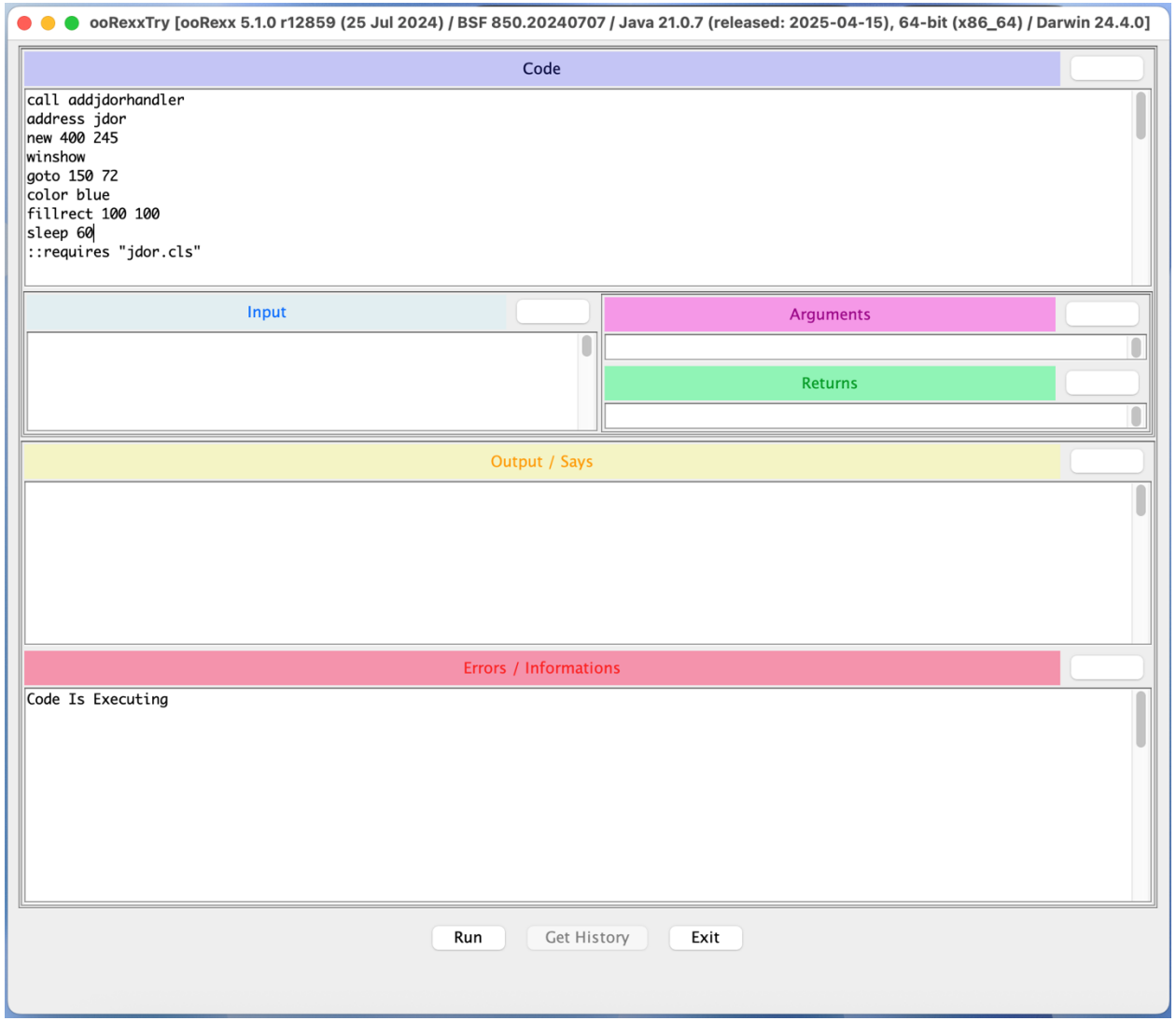



JDOR-RotatingSquare.rxj

# *Get Started Now!*

In 3 steps:

1. Install ooRexx
2. Install Java
3. Install BSF4ooRexx



# Conclusion

JDOR = Java power + Rexx simplicity

Great entry point to graphics programming

Opens new creative opportunities ☺

# References


Apache License, Version 2.0. https://www.apache.org/licenses/LICENSE-2.0

Blauensteiner, F. (2023). JDOR – An introduction to Java 2D's drawing classes with ooRexx and BSF4ooRexx,https://wi.wu.ac.at/rgf/diplomarbeiten/BakkStuff/2023/202302_Blauensteiner_JDOR.pdf

Cowell, J. (1999). The Abstract Windowing Toolkit. In: Essential Visual J++ 6.0 fast . Essential Series. Springer, London. https://doi.org/10.1007/978-1-4471-0565-7_11

Flatscher, R. G. (2022). BSF4ooRexx: Introducing the JDOR Rexx Command Handler for Easy Creation of Bitmaps and Bitmap Manipulations on Windows, Mac and Linux International RexxLA Symposium, 2022-09, https://www.rexxla.org/presentations/2022/202209_JDOR_command_handler.pdf

Flatscher, R. G. (2023). Proposing ooRexx and BSF4ooRexx for Teaching Programming and Fundamental Programming Concepts. ISECON23-Conference

Holt, W. (1999). THE EMBEDDED WINDOW TOOLKIT (Doctoral dissertation, UNIVERSITY OF CALIFORNIA SANTA CRUZ). http://alumni.soe.ucsc.edu/~wholt/thesis.pdf

https://dotnettutorials.net/lesson/abstract-windows-toolkit-awt-in-java/), 2022. Abstract Windows Toolkit (AWT) in Java. Retrieved 10.04.2023 from https://dotnettutorials.net/lesson/abstract-windows-toolkit-awt-in-java/

# References


Linforth, P. (2017) Pyramids, Egypt, Egyptian image. [Image]. Retrieved from https://pixabay.com/photos/pyramids-egypt-egyptian-ancient-2371501/

Oracle (o. D. -a). About the JFC and Swing. Oracle. Retrieved 15.05.2023 from https://docs.oracle.com/javase/tutorial/uiswing/start/about.html

Oracle. (o.D.-b). Class Graphics2D. Oracle. Retrieved 10.04.2023 from https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html

Oracle. (o.D.-c). Coordinates. Oracle. Retrieved 05.05.2023 from https://docs.oracle.com/javase/tutorial/2d/overview/coordinate.html

Oracle. (o.D.-d). Images. Oracle. Retrieved 05.05.2023 from https://docs.oracle.com/javase/tutorial/2d/overview/images.html

Oracle. (o.D.-e). Lesson: Getting Started with Graphics. Oracle. Retrieved 01.05.2023 from https://docs.oracle.com/javase/tutorial/2d/basic2d/index.html

Oracle. (o.D.-f). Lesson: Overview of the Java 2D API Concepts. www.docs.oracle.com. Retrieved 01.05.2023 from https://docs.oracle.com/javase/tutorial/2d/overview/index.html

# References


Oracle. (o.D.-g). Trail: 2D Graphics.Oracle Retrieved 20.05.2023 from https://docs.oracle.com/javase/tutorial/2d/index.html

Oracle. (o.D.-h). Transforming Shapes, Text and Images. Retrieved 13.05.2023 from https://docs.oracle.com/javase/tutorial/2d/advanced/transforming.html

Pixabay Content License. https://pixabay.com/service/license-summary/

Schäling, B. (2010). Programmieren in Java: Aufbau. www.highscore.de.

Sun-Microsystems. (1999). 1.2 Rendering Model. Nickerson Group at University of Washington. Retrieved 15.05.2023 from https://nicklab.gs.washington.edu/java/jdk1.3.1/guide/2d/spec/j2d-intro.fm2.html