1. Write a java program to implement 2D Translation.

package pkg2dtranslation;

import java.awt.EventQueue;

import java.awt.Graphics;

import java.awt.Graphics2D;

import javax.swing.JFrame;

import javax.swing.JPanel;

class Surface extends JPanel{

private void doDrawing(Graphics g){

Graphics g2d=(Graphics2D) g.create();

g2d.fillRect(20, 20, 80, 50);

g2d.translate(150,50);

g2d.fillRect(20, 20, 80, 50);

g2d.dispose();

}

@Override

public void paintComponent(Graphics g){

super.paintComponent(g);

doDrawing(g);

}

}

public class Rectangle extends JFrame {

public Rectangle(){

initUI();

}

private void initUI(){

add(new Surface());

setTitle("Translation");

setSize(300,200);

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable(){

@Override

public void run(){

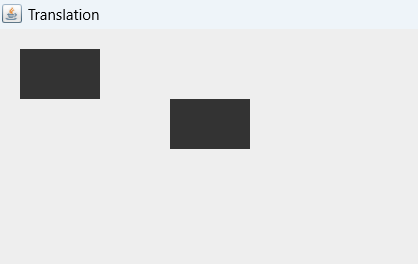
Rectangle rec = new Rectangle();

rec.setVisible(true);

}

});

Output:



1. Write a java program to implement 2D Rotation.

package pkg2drotation;

import java.awt.Color;

import java.awt.EventQueue;

import java.awt.Graphics;

import java.awt.Graphics2D;

import javax.swing.JFrame;

import javax.swing.JPanel;

class Surface extends JPanel {

private void doDrawing(Graphics g) {

Graphics2D g2d = (Graphics2D) g.create();

g2d.setPaint(new Color(150, 150, 150));

g2d.fillRect(20, 20, 80, 50);

g2d.translate(180, -50);

g2d.rotate(Math.PI/4);

g2d.fillRect(80, 80, 80, 50);

g2d.dispose();

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

doDrawing(g);

}

}

public class Main extends JFrame {

public Main() {

initUI();

}

private void initUI() {

setTitle("Rotation");

add(new Surface());

setSize(300, 200);

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

Main ex = new Main();

ex.setVisible(true);

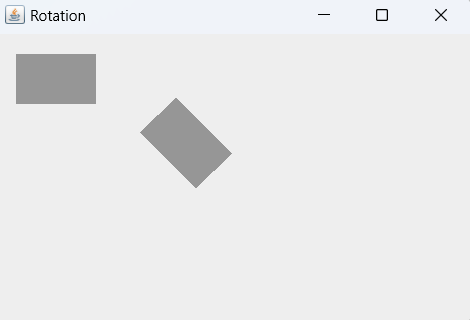
}

});

}

}

Output:



1. Write a java program to implement 2D Scaling.

package pkg2dscaling;

import java.awt.Color;

import java.awt.EventQueue;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.AffineTransform;

import javax.swing.JFrame;

import javax.swing.JPanel;

class Surface extends JPanel {

private void doDrawing(Graphics g) {

Graphics2D g2d = (Graphics2D) g.create();

g2d.setColor(new Color(150, 150, 150));

g2d.fillRect(20, 20, 80, 50);

AffineTransform tx1 = new AffineTransform();

tx1.translate(110, 22);

tx1.scale(0.5, 0.5);

g2d.setTransform(tx1);

g2d.fillRect(0, 0, 80, 50);

AffineTransform tx2 = new AffineTransform();

tx2.translate(170, 20);

tx2.scale(1.5, 1.5);

g2d.setTransform(tx2);

g2d.fillRect(0, 0, 80, 50);

g2d.dispose();

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

doDrawing(g);

}

}

public class Scaling extends JFrame {

public Scaling() {

initUI();

}

private void initUI() {

add(new Surface());

setTitle("Scaling");

setSize(330, 160);

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

Scaling ex = new Scaling();

ex.setVisible(true);

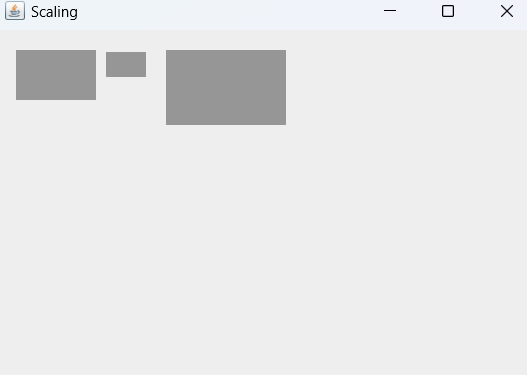
}

});

}

}

Output:



1. Write a java program to implement 2D Shearing.

package pkg2dshearing;

import java.awt.Color;

import java.awt.EventQueue;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Rectangle;

import java.awt.geom.AffineTransform;

import javax.swing.JFrame;

import javax.swing.JPanel;

class Surface extends JPanel {

private void doDrawing(Graphics g) {

Graphics2D g2d = (Graphics2D) g.create();

AffineTransform tx1 = new AffineTransform();

tx1.translate(50, 90);

g2d.setTransform(tx1);

g2d.setPaint(Color.green);

g2d.drawRect(0, 0, 160, 50);

AffineTransform tx2 = new AffineTransform();

tx2.translate(50, 90);

tx2.shear(0, 1);

g2d.setTransform(tx2);

g2d.setPaint(Color.blue);

g2d.draw(new Rectangle(0, 0, 80, 50));

AffineTransform tx3 = new AffineTransform();

tx3.translate(130, 10);

tx3.shear(0, 1);

g2d.setTransform(tx3);

g2d.setPaint(Color.red);

g2d.drawRect(0, 0, 80, 50);

g2d.dispose();

}

@Override

public void paintComponent(Graphics g) { super.paintComponent(g);

doDrawing(g);

}

}

public class Shearing extends JFrame {

public Shearing() {

initUI();

}

private void initUI() {

add(new Surface());

setTitle("Shearing");

setSize(330, 270);

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

Shearing ex = new Shearing();

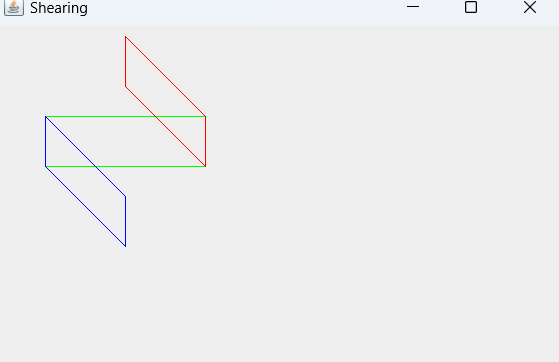
ex.setVisible(true);

}

});

}

Output:



1. Write a java program to implement 2D Reflection along x-axis(rectangle).

package reflection;

import javax.swing.\*;

import java.awt.\*;

public class Reflection extends JPanel {

// Original rectangle

private int originalX = 100;

private int originalY = 100;

private int originalWidth = 200;

private int originalHeight = 100;

// Reflected rectangle

private int reflectedX;

private int reflectedY;

private int reflectedWidth;

private int reflectedHeight;

public Reflection() {

reflectAboutXAxis();

}

private void reflectAboutXAxis() {

// Reflect about x-axis

reflectedX = originalX;

reflectedY = getHeight() - originalY - originalHeight; // Reflecting about x-axis

reflectedWidth = originalWidth;

reflectedHeight = originalHeight;

// Calculate center of the panel

int panelCenterX = getWidth() / 2;

int panelCenterY = getHeight() / 2;

// Calculate center of the rectangles

int originalCenterX = originalX + originalWidth / 2;

int originalCenterY = originalY + originalHeight / 2;

int reflectedCenterX = reflectedX + reflectedWidth / 2;

int reflectedCenterY = reflectedY + reflectedHeight / 2;

// Adjust positions of both rectangles to center them

originalX = panelCenterX - originalWidth / 2;

originalY = panelCenterY - originalHeight / 2;

reflectedX = panelCenterX - reflectedWidth / 2;

reflectedY = panelCenterY - (reflectedCenterY - panelCenterY) - reflectedHeight / 2;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

// Draw original rectangle

g2d.setColor(Color.BLUE);

g2d.drawRect(originalX, originalY, originalWidth, originalHeight);

// Draw reflected rectangle

g2d.setColor(Color.RED);

g2d.drawRect(reflectedX, reflectedY, reflectedWidth, reflectedHeight);

}

public static void main(String[] args) {

JFrame frame = new JFrame("X-Axis Reflection");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(600, 400);

frame.setLocationRelativeTo(null);

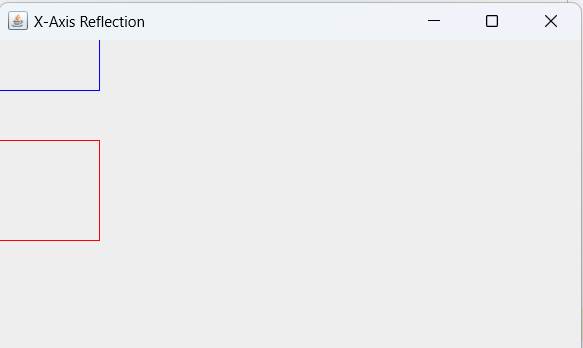
frame.add(new Reflection());

frame.setVisible(true);

}

}

Output:



1. Write a java program to implement 2D Reflection along x-axis(triangle).

package reflectiontriangle;

import javax.swing.\*;

import java.awt.\*;

public class Reflectiontriangle extends JPanel {

// Original points

private int[] originalX = {100, 200, 300, 400};

private int[] originalY = {100, 200, 100, 200};

// Reflected points

private int[] reflectedX;

private int[] reflectedY;

public Reflectiontriangle() {

reflectAboutXAxis();

}

private void reflectAboutXAxis() {

reflectedX = new int[originalX.length];

reflectedY = new int[originalY.length];

for (int i = 0; i < originalX.length; i++) {

reflectedX[i] = originalX[i];

reflectedY[i] = getHeight() - originalY[i]; // Reflecting about x-axis

}

// Calculate offset for centering

int offsetX = (getWidth() - (int) (1.1 \* (originalX[originalX.length - 1] - originalX[0]))) / 2;

int offsetY = (getHeight() - (int) (1.1 \* (originalY[originalY.length - 1] - originalY[0]))) / 2;

// Apply offset to the reflected points

for (int i = 0; i < reflectedX.length; i++) {

reflectedX[i] += offsetX;

reflectedY[i] += offsetY;

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

// Draw original shape

g2d.setColor(Color.BLUE);

g2d.drawPolygon(originalX, originalY, originalX.length);

// Draw reflected shape

g2d.setColor(Color.RED);

g2d.drawPolygon(reflectedX, reflectedY, reflectedX.length);

}

public static void main(String[] args) {

JFrame frame = new JFrame("X-Axis Reflection");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(600, 400);

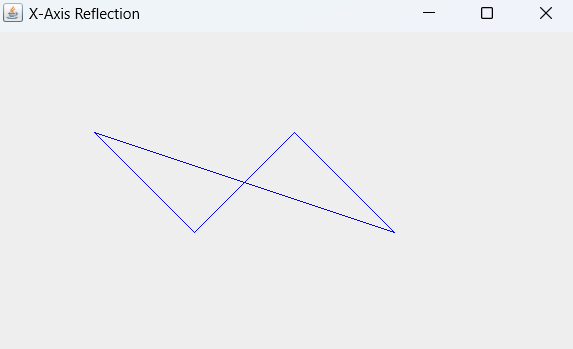
frame.setLocationRelativeTo(null);

frame.add(new Reflectiontriangle());

frame.setVisible(true);

}}

Output:



1. Write a java program to implement 3D Translation.

package \_3dtranslation;

import java.awt.\*;

import javax.swing.\*;

public class \_3DTranslation extends JPanel {

private static final int SIZE = 100;

private static final int[] cube = {

50, 50, 50, // Front face vertices

150, 50, 50,

150, 150, 50,

50, 150, 50,

50, 50, 150, // Back face vertices

150, 50, 150,

150, 150, 150,

50, 150, 150

};

private int[] translatedCube;

public \_3DTranslation() {

translatedCube = new int[cube.length];

System.arraycopy(cube, 0, translatedCube, 0, cube.length);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

drawCube(g, cube);

drawCube(g, translatedCube);

}

private void drawCube(Graphics g, int[] vertices) {

g.setColor(Color.BLUE);

g.drawLine(vertices[0], vertices[1], vertices[3], vertices[4]);

g.drawLine(vertices[3], vertices[4], vertices[6], vertices[7]);

g.drawLine(vertices[6], vertices[7], vertices[9], vertices[10]);

g.drawLine(vertices[9], vertices[10], vertices[0], vertices[1]);

g.drawLine(vertices[1], vertices[2], vertices[4], vertices[5]);

g.drawLine(vertices[4], vertices[5], vertices[7], vertices[8]);

g.drawLine(vertices[7], vertices[8], vertices[10], vertices[11]);

g.drawLine(vertices[10], vertices[11], vertices[1], vertices[2]);

g.drawLine(vertices[0], vertices[1], vertices[5], vertices[6]);

g.drawLine(vertices[5], vertices[6], vertices[2], vertices[3]);

g.drawLine(vertices[2], vertices[3], vertices[7], vertices[8]);

g.drawLine(vertices[7], vertices[8], vertices[1], vertices[0]);

}

public void translate(int dx, int dy, int dz) {

for (int i = 0; i < translatedCube.length; i += 3) {

translatedCube[i] += dx;

translatedCube[i + 1] += dy;

translatedCube[i + 2] += dz;

}

repaint();

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

JFrame frame = new JFrame("3D Translation Example");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

\_3DTranslation panel = new \_3DTranslation();

frame.add(panel);

frame.setSize(300, 300);

frame.setVisible(true);

// Example translation

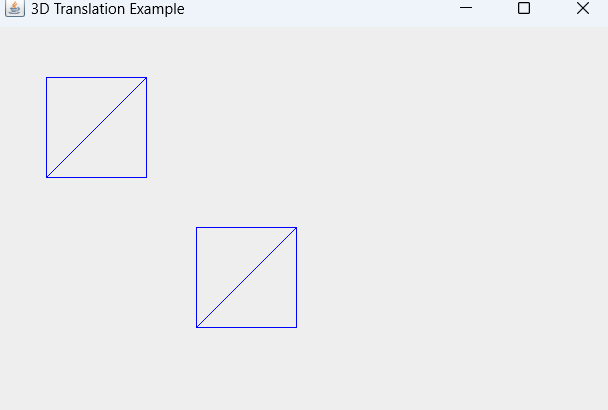
panel.translate(150, 150, 150); // Translate by (150, 150, 150)

});

}

}

Output:



1. Write a java program to implement 3D Shearing.

package \_3dshearing1;

import java.awt.\*;

import javax.swing.\*;

public class \_3DShearing1 extends JPanel {

private static final int SIZE = 100;

private static final int[] cube = {

50, 50, 50, // Front face vertices

150, 50, 50,

150, 150, 50,

50, 150, 50,

50, 50, 150, // Back face vertices

150, 50, 150,

150, 150, 150,

50, 150, 150

};

private final int[] shearedCube;

public \_3DShearing1() {

shearedCube = new int[cube.length];

System.arraycopy(cube, 0, shearedCube, 0, cube.length);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

drawCube(g, cube);

drawCube(g, shearedCube);

}

private void drawCube(Graphics g, int[] vertices) {

g.setColor(Color.BLUE);

g.drawLine(vertices[0], vertices[1], vertices[3], vertices[4]);

g.drawLine(vertices[3], vertices[4], vertices[6], vertices[7]);

g.drawLine(vertices[6], vertices[7], vertices[9], vertices[10]);

g.drawLine(vertices[9], vertices[10], vertices[0], vertices[1]);

g.drawLine(vertices[1], vertices[2], vertices[4], vertices[5]);

g.drawLine(vertices[4], vertices[5], vertices[7], vertices[8]);

g.drawLine(vertices[7], vertices[8], vertices[10], vertices[11]);

g.drawLine(vertices[10], vertices[11], vertices[1], vertices[2]);

g.drawLine(vertices[0], vertices[1], vertices[5], vertices[6]);

g.drawLine(vertices[5], vertices[6], vertices[2], vertices[3]);

g.drawLine(vertices[2], vertices[3], vertices[7], vertices[8]);

g.drawLine(vertices[7], vertices[8], vertices[1], vertices[0]);

}

public void shear(double shx, double shy) {

for (int i = 0; i < shearedCube.length; i += 3) {

shearedCube[i] += shearedCube[i + 2] \* shx;

shearedCube[i + 1] += shearedCube[i + 2] \* shy;

}

repaint();

}

public static void main(String[] args) {

// TODO code application logic here

SwingUtilities.invokeLater(() -> {

JFrame frame = new JFrame("3D Shearing Example");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

\_3DShearing1 panel = new \_3DShearing1();

frame.add(panel);

frame.setSize(300, 300);

frame.setVisible(true);

// Example shearing

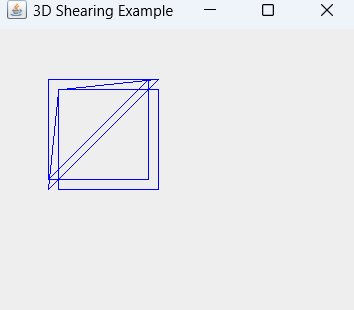
panel.shear(0.2, 0.2); // Shear along both x and y axes

});

}

}

Output:



1. Write a java program to implement DDA Line Drawing Algorithm.

package linedrawdda;

import javax.swing.\*;

import java.awt.\*;

public class LinedrawDDA extends JPanel {

private int x1, y1, x2, y2;

public LinedrawDDA(int x1, int y1, int x2, int y2) {

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setColor(Color.RED);

int dx = x2 - x1;

int dy = y2 - y1;

int steps = Math.max(Math.abs(dx), Math.abs(dy));

float xIncrement = (float) dx / steps;

float yIncrement = (float) dy / steps;

float x = x1;

float y = y1;

for (int i = 0; i <= steps; i++) {

g2d.fillRect(Math.round(x), Math.round(y), 1, 1);

x += xIncrement;

y += yIncrement;

}

}

public static void main(String[] args) {

int x1 = 50, y1 = 50; // Start point

int x2 = 250, y2 = 200; // End point

JFrame frame = new JFrame("DDA Line Drawing Algorithm");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(400, 300);

frame.setLocationRelativeTo(null);

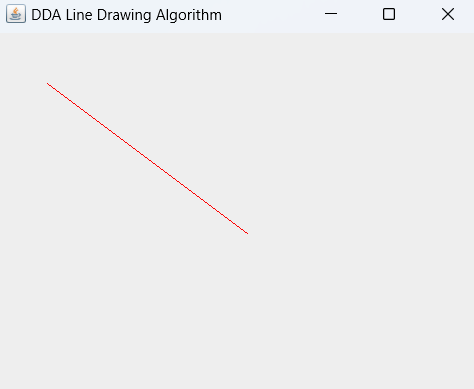
frame.add(new LinedrawDDA(x1, y1, x2, y2));

frame.setVisible(true);

}

}

Output:



10.Write a java program to implement Bresenham's Line Drawing Algorithm.

package linedrawbla;

import javax.swing.\*;

import java.awt.\*;

public class LinedrawBLA extends JPanel {

private int x1, y1, x2, y2;

public LinedrawBLA(int x1, int y1, int x2, int y2) {

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setColor(Color.RED);

int dx = Math.abs(x2 - x1);

int dy = Math.abs(y2 - y1);

int sx = x1 < x2 ? 1 : -1;

int sy = y1 < y2 ? 1 : -1;

int err = dx - dy;

int x = x1;

int y = y1;

while (x != x2 || y != y2) {

g2d.fillRect(x, y, 1, 1);

int err2 = 2 \* err;

if (err2 > -dy) {

err -= dy;

x += sx;

}

if (err2 < dx) {

err += dx;

y += sy;

}

}

}

public static void main(String[] args) {

int x1 = 50, y1 = 50; // Start point

int x2 = 250, y2 = 200; // End point

JFrame frame = new JFrame("Bresenham's Line Drawing Algorithm");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(400, 300);

frame.setLocationRelativeTo(null);

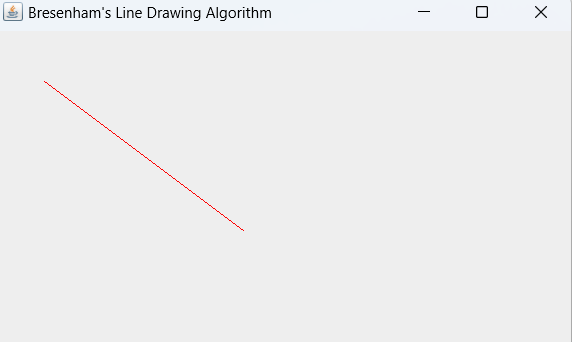
frame.add(new LinedrawBLA(x1, y1, x2, y2));

frame.setVisible(true);

}

}

Output:



11. Write a java program to implement Midpoint Circle Drawing Algorithm.

package midpointcircle;

import javax.swing.\*;

import java.awt.\*;

public class MidpointCircle extends JPanel {

private int radius, centerX, centerY;

public MidpointCircle(int radius, int centerX, int centerY) {

this.radius = radius;

this.centerX = centerX;

this.centerY = centerY;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setColor(Color.BLUE);

int x = 0;

int y = radius;

int p = 1 - radius;

drawCircle(g2d, centerX, centerY, x, y);

while (x < y) {

x++;

if (p < 0) {

p += 2 \* x + 1;

} else {

y--;

p += 2 \* (x - y) + 1;

}

drawCircle(g2d, centerX, centerY, x, y);

}

}

private void drawCircle(Graphics2D g2d, int centerX, int centerY, int x, int y) {

g2d.fillRect(centerX + x, centerY + y, 1, 1);

g2d.fillRect(centerX - x, centerY + y, 1, 1);

g2d.fillRect(centerX + x, centerY - y, 1, 1);

g2d.fillRect(centerX - x, centerY - y, 1, 1);

g2d.fillRect(centerX + y, centerY + x, 1, 1);

g2d.fillRect(centerX - y, centerY + x, 1, 1);

g2d.fillRect(centerX + y, centerY - x, 1, 1);

g2d.fillRect(centerX - y, centerY - x, 1, 1);

}

public static void main(String[] args) {

int radius = 100;

int centerX = 200;

int centerY = 150;

JFrame frame = new JFrame("Midpoint Circle Drawing Algorithm");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

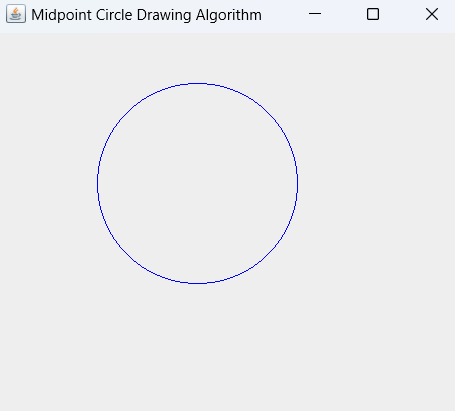
frame.setSize(400, 300);

frame.setLocationRelativeTo(null);

frame.add(new MidpointCircle(radius, centerX, centerY));

frame.setVisible(true);

}} Output:



12.Write a java program to implement Midpoint Ellipse Drawing Algorithm.

package midpoingellipse;

import javax.swing.\*;

import java.awt.\*;

public class MidpoingEllipse extends JPanel {

private int a, b, centerX, centerY;

public MidpoingEllipse(int a, int b, int centerX, int centerY){

this.a = a;

this.b = b;

this.centerX = centerX;

this.centerY = centerY;

}

@Override

protected void paintComponent(Graphics g){

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setColor(Color.RED);

int x = 0;

int y = b;

double d1 = (b \* b) - (a \* a \* b) + (0.25 \* a \* a);

int dx = 2 \* b \* b \* x;

int dy = 2 \* a \* a \* y;

drawEllipse(g2d, centerX, centerY, x, y);

while (dx < dy) {

x++;

if (d1 < 0) {

dx = dx + (2 \* b \* b);

d1 = d1 + dx + (b \* b);

} else {

y--;

dx = dx + (2 \* b \* b);

dy = dy - (2 \* a \* a);

d1 = d1 + dx - dy + (b \* b);

}

drawEllipse(g2d, centerX, centerY, x, y);

}

double d2 = ((b \* b) \* ((x + 0.5) \* (x + 0.5))) + ((a \* a) \* ((y - 1) \* (y - 1))) - (a \* a \* b \* b);

while (y > 0) {

y--;

if (d2 > 0) {

dy = dy - (2 \* a \* a);

d2 = d2 + (a \* a) - dy;

} else {

x++;

dx = dx + (2 \* b \* b);

dy = dy - (2 \* a \* a);

d2 = d2 + dx - dy + (a \* a);

}

drawEllipse(g2d, centerX, centerY, x, y);

}

}

private void drawEllipse(Graphics2D g2d, int centerX, int centerY, int x, int y) {

g2d.fillRect(centerX + x, centerY + y, 1, 1);

g2d.fillRect(centerX - x, centerY + y, 1, 1);

g2d.fillRect(centerX + x, centerY - y, 1, 1);

g2d.fillRect(centerX - x, centerY - y, 1, 1);

}

public static void main(String[] args) {

int a = 50; // Semi-major axis

int b = 100; // Semi-minor axis

int centerX = 150; // Center X coordinate

int centerY = 200; // Center Y coordinate

JFrame frame = new JFrame("Midpoint Ellipse Drawing Algorithm");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(400, 300);

frame.setLocationRelativeTo(null);

frame.add(new MidpoingEllipse(a, b, centerX, centerY));

frame.setVisible(true);

}

}

Output:

