Лабораторна робота №2

Тема: Побудова та анімація зображень за допомогою Java2D

Мета: Ознайомитися з можливостями побудови зображень та їх анімації у Java2D

Теоретичні відомості

Є два види комп'ютерної графіки – растрова та векторна. Растрова графіка представляє зображення як набір пікселів. Векторна графіка використовує геометричні примітиви – точки, лінії, полігони, дуги. Java 2D надає можливість працювати як з растровою, так і з векторною графікою.

Java 2D це API для створення двовимірних зображень за допомогою мови програмування Java. Java 2D API має наступні властивості:

- широкий вибір геометричних примітивів
- визначення перетину фігур, тексту, зображень
- контроль якості рендерингу
- друк документів
- уніфікована модель рендерингу для дисплеїв та принтерів

Java 2D досить потужна технологія, за допомогою якої можна створювати потужні та красиві інтерфейси користувача, ігри, анімації, мультимедійні програми.

Базова програма та малювання

Код базової програми для малювання в Java 2D наведено нижче. Для створення програм в Java 2D достатньо стандартного JDK.

```
import java.awt.*;

import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Skeleton extends JPanel {

	// Всі дії, пов'язані з малюванням, виконуються в цьому методі
	public void paint(Graphics g) {

		// Оскільки Java2D є надбудовою над старішою бібліотекою, необхідно робити це приведення
		Graphics2D g2d = (Graphics2D) g;

		// Далі йде безпосередньо малювання. Для прикладу намалюємо такий рядок
	 g2d.drawString("Привіт, Java 2D!", 50, 50);
    }
```

```
    public static void main(String[] args) {

        // Створюємо нове графічне вікно (формочка). Параметр конструктора - заголовок вікна.
        JFrame frame = new JFrame("Привіт, Java 2D!");
        // Визначаємо поведінку програми при закритті вікна (ЛКМ на "хрестик")
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Визначаємо розмір вікна
        frame.setSize(500, 500);
        // Якщо позиція прив'язана до null, вікно з'явиться в центрі екрану
        frame.setLocationRelativeTo(null);
        // Забороняємо змінювати розміри вікна
        frame.setResizable(false);

        // Додаємо до вікна панель, що і описується даним класом
        // Зауважте, що точка входу в програму - метод main, може бути й в іншому класі
        frame.add(new Skeleton());
        // Показуємо форму. Важливо додати всі компоненти на форму до того, як зробити її видимою.
        frame.setVisible(true);
    }
}
```

Слід зауважити, що розмір вікна включає в себе рамку вікна та заголовок вікна (titlebar). Для того, щоб дізнатися розміри площини для малювання, необхідно виконати наступне:

1. Додати поля в клас Skeleton:

```
private static int maxWidth;
private static int maxHeight;
```

2. Додати до методу main наступне, обов’язково після виконання `frame.setVisible(true)`:

```
Dimension size = frame.getSize();
Insets insets = frame.getInsets();
maxWidth =  size.width - insets.left - insets.right - 1;
maxHeight =  size.height - insets.top - insets.bottom - 1;
```

Тепер можна використовувати отримані значення у методі paint, відповідно початок координат це точка (0,0), максимальні значення координат для видимої точки це (maxWidth, maxHeight).

Перед початком малювання, можна змінити колір фону. Зміна фону відбувається заливкою визначеної прямокутної області, тому все, що було намальовано, затреться. Для зміни кольору фону на чорний, необхідно виконати наступні дії:

```
g2d.setBackground(Color.black);
g2d.clearRect(0, 0, maxWidth, maxHeight);
```

Колір, що використовується для малювання, задається безпосередньо перед тим, як викликати відповідний метод малювання. Для того, щоб змінити колір для малювання, використовується метод

```
g2d.setColor(Color.yellow);
```

Для того, щоб намалювати точку, окремого методу не передбачено. Її можна намалювати, використавши метод для побудови лінії, вказавши однакові початкову та кінцеву точки.

Для того, щоб намалювати лінію, використовується метод

```
g2d.drawLine(0, 50, maxWidth, maxHeight);
```

В результаті отримаємо лінію, зображену на рис. 1.

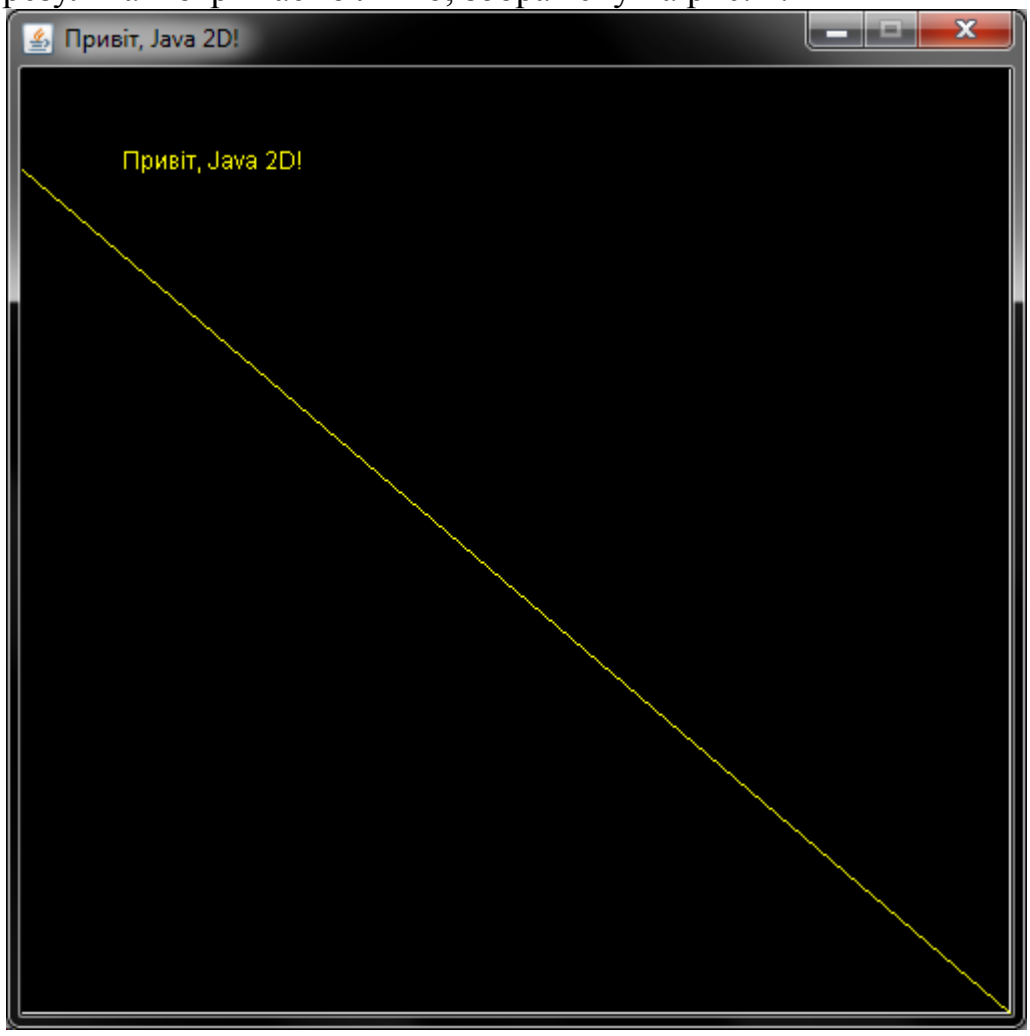

Рис. 1 Намальована лінія поганої якості

Як бачимо, якість малювання лінії погана. Java 2D надає широкий вибір налаштувань якості рендерингу. В даній лабораторній роботі пропонується наступний набір налаштувань (цей код необхідно вставити одразу після виконання `Graphics2D g2d=(Graphics2D)g;`)

```
RenderingHints rh =
```

```
            new RenderingHints(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

        rh.put(RenderingHints.KEY_RENDERING,
                RenderingHints.VALUE_RENDER_QUALITY);

        g2d.setRenderingHints(rh);
```

В результаті, з новими налаштуваннями рендерингу, отримаємо таку лінію:

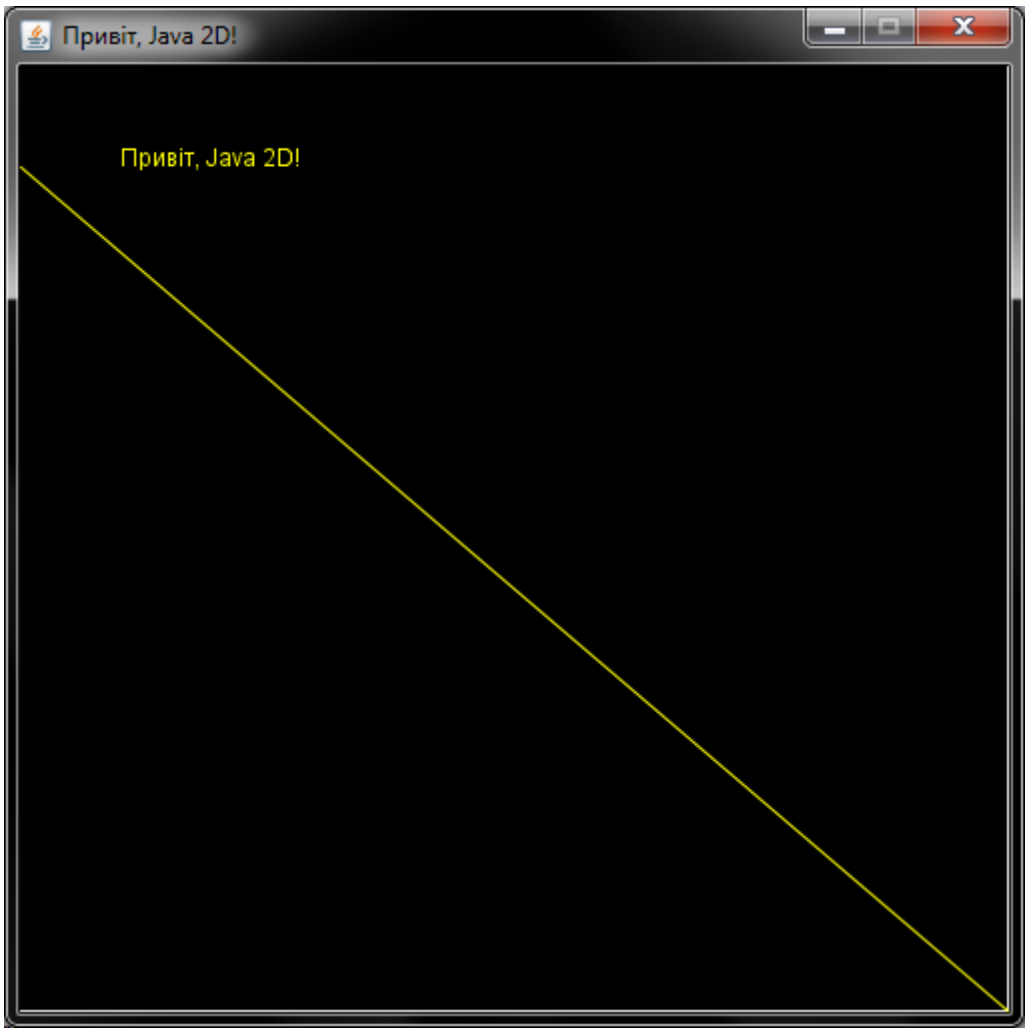

Рис. 2 Намальована лінія хорошої якості

Для побудови складних фігур можна використовувати як стандартні примітиви, так і побудову по точкам.

Примітив у свою чергу можна побудувати не залитим, та залитим. Для побудови контуру примітиву використовується такий метод (для прямокутника):

```
g2d.drawRect(10, 20, 30, 40);
```

Для побудови залитого прямокутника використовується схожий метод:

```
g2d.fillRect(100, 100, 150, 50);
```

Як для заливки, так і для побудови контуру примітиву використовуються значення кольору та пензлика, що були востаннє задані перед малюванням.

Всього доступні такі примітиви (рис. 3)

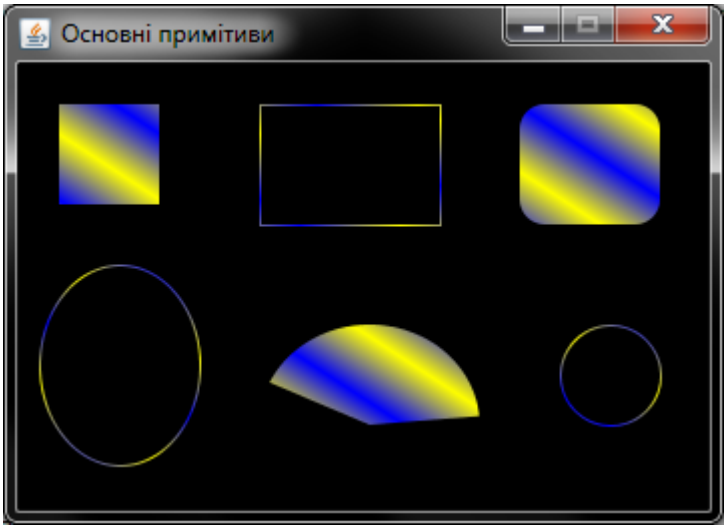

Рис. 3 Базові примітиви

Якщо перед побудовою примітиву був заданий градієнт, примітив буде намальовано з його використанням, а не простим кольором, як раніше. Код для встановлення градієнту та побудови примітивів:

```
GradientPaint gp = new GradientPaint(5, 25,
        Color.YELLOW, 20, 2, Color.BLUE, true);
g2d.setPaint(gp);

g2d.fillRect(20, 20, 50, 50);
g2d.drawRect(120, 20, 90, 60);
g2d.fillRoundRect(250, 20, 70, 60, 25, 25);

g2d.draw(new Ellipse2D.Double(10, 100, 80, 100));
g2d.fillArc(120, 130, 110, 100, 5, 150);
g2d.drawOval(270, 130, 50, 50);
```

Колір можна також задавати безпосередньо числовим значенням, використовуючи конструктор класу Color. Наступний запис кольору еквівалентний наведеному вище:

```
GradientPaint gp = new GradientPaint(5, 25,
        new Color(255,255,0), 20, 2, new Color(0,0,255), true);
g2d.setPaint(gp);
```

В конструктор передаються безпосередньо кількісні складові кожного із кольорів RGB.

Окрім примітивів, можна також малювати ламаною лінією замкнуті контури. Наступний код малює зірку:

```
double points[][] = {
        { 0, 85 }, { 75, 75 }, { 100, 10 }, { 125, 75 },
        { 200, 85 }, { 150, 125 }, { 160, 190 }, { 100, 150 },
        { 40, 190 }, { 50, 125 }, { 0, 85 }
    };

GeneralPath star = new GeneralPath();

g2d.translate(70, 12);
star.moveTo(points[0][0], points[0][1]);

for (int k = 1; k < points.length; k++)
    star.lineTo(points[k][0], points[k][1]);

star.closePath();
g2d.fill(star);
//g2d.draw(star);
```

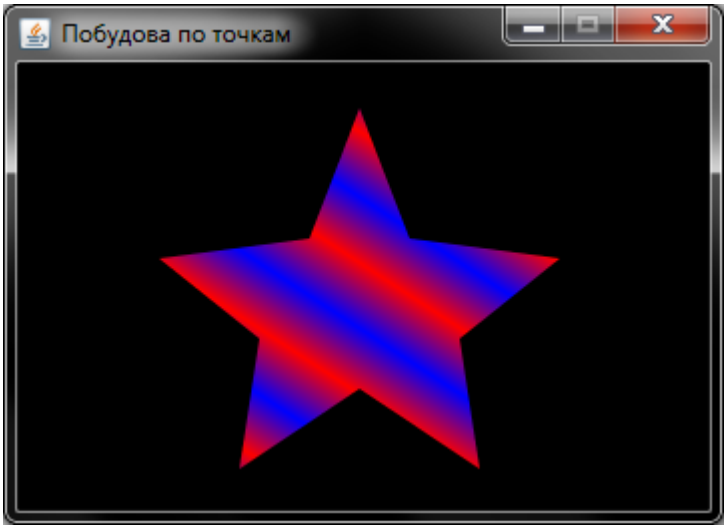

Рис. 4 Побудова контуру по точкам

Після виконання циклу, об’єкт star можна використовувати як звичайний примітив, малюючи лише його контур, чи одразу із заливкою.

Java 2D також надає можливість керувати параметрами пензля для малювання. Зокрема, можна керувати товщиною лінії, типами її кінців, типами з’єднань. Проілюструємо роботу цих параметрів на прикладі:

```
        // Конструктор приймає три параметри: товщину лінії, тип декорації кінців
та з'єднань
        BasicStroke bs1 = new BasicStroke(16, BasicStroke.CAP_BUTT,
            BasicStroke.JOIN_BEVEL);
        g2d.setStroke(bs1);
        g2d.drawLine(20, 30, 250, 30);

        BasicStroke bs2 = new BasicStroke(16, BasicStroke.CAP_ROUND,
            BasicStroke.JOIN_BEVEL);
        g2d.setStroke(bs2);
        g2d.drawLine(20, 80, 250, 80);

        BasicStroke bs3 = new BasicStroke(16, BasicStroke.CAP_SQUARE,
            BasicStroke.JOIN_BEVEL);
        g2d.setStroke(bs3);
        g2d.drawLine(20, 130, 250, 130);
```

```
 // Скидаємо налаштування пензля на стандартні
BasicStroke bs4 = new BasicStroke();
g2d.setStroke(bs4);

g2d.setColor(new Color(150,150,255));
g2d.drawLine(20, 20, 20, 140);
g2d.drawLine(250, 20, 250, 140);
g2d.drawLine(258, 20, 258, 140);
```

Результат виконання можна бачити на рис. 5

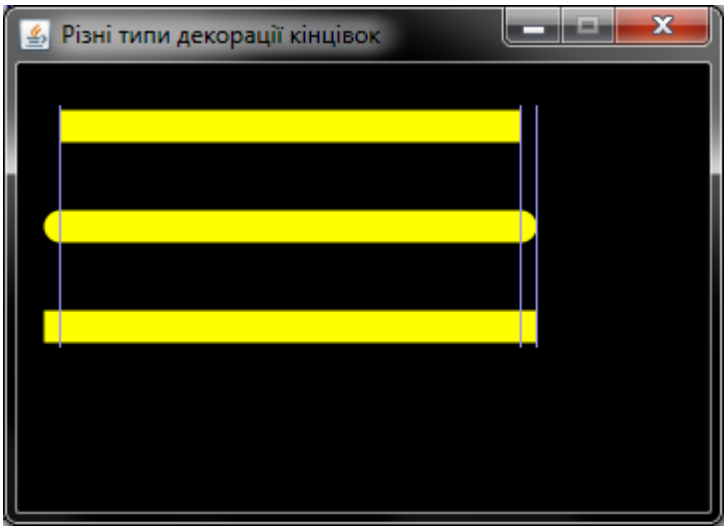

Рис. 5 Різні типи декорації кінцівок

Зображені лінії використовують типи декорації кінцівок, відповідно зверху донизу: CAP_BUTT, CAP_ROUND, та CAP_SQUARE. Вертикальними рисками позначені задані та фактичні межі намальованої лінії. Зауважте, що в усіх трьох випадках задана довжина лінії однакова, від $x_1 = 20$ до $x_2 = 250$.

Розглянемо типи декорації з'єднань:

```
BasicStroke bs1 = new BasicStroke(16, BasicStroke.CAP_ROUND,
    BasicStroke.JOIN_BEVEL);
g2d.setStroke(bs1);
g2d.drawRect(15, 75, 80, 50);

BasicStroke bs2 = new BasicStroke(16, BasicStroke.CAP_ROUND,
    BasicStroke.JOIN_MITER);
g2d.setStroke(bs2);
g2d.drawRect(125, 75, 80, 50);

BasicStroke bs3 = new BasicStroke(16, BasicStroke.CAP_ROUND,
    BasicStroke.JOIN_ROUND);
g2d.setStroke(bs3);
g2d.drawRect(235, 75, 80, 50);
```

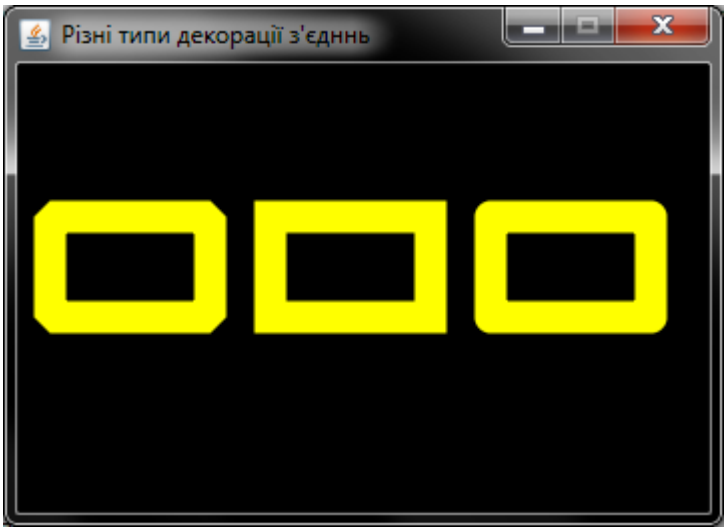

Рис. 6 Різні типи декорації з’єднань

Анімація

Java 2D надає інструменти для перетворення зображення, що дає можливість створювати цікаві анімаційні ефекти. Розглянемо основні трансформації – зсув, поворот, масштабування, та зміну прозорості. Зауважте, що анімаційні перетворення впливають лише на ті фігури, що малюються після відповідних перетворень.

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.GeneralPath;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.Timer;

public class Star extends JPanel implements ActionListener {

       // Масштабування відбувається відносно центру координат,
       // тому малювати фігуру бажано також симетрично центру
       double points[][] = {
            { -100, -15 }, { -25, -25 }, { 0, -90 }, { 25, -25 },
            { 100, -15 }, { 50, 25 }, { 60, 100 }, { 0, 50 },
            { -60, 100 }, { -50, 25 }, { -100, -15 }
        };

    Timer timer;

    // Для анімації повороту
    private double angle = 0;

    // Для анімації масштабування
    private double scale = 1;
    private double delta = 0.01;

    // Для анімації руху
    private double dx = 1;
    private double tx = 0;
    private double dy = 1;
```

```
    private double ty = 6;

       private static int maxWidth;
       private static int maxHeight;

    public Star() {

       // Таймер генеруватиме подію що 10 мс
         timer = new Timer(10, this);
         timer.start();
    }

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

        Graphics2D g2d = (Graphics2D)g;
        // Встановлюємо кольори
        g2d.setBackground(Color.black);
        g2d.setColor(Color.YELLOW);
        g2d.clearRect(0, 0, maxWidth+1, maxHeight+1);

        // Встановлюємо параметри рендерингу
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                             RenderingHints.VALUE_ANTIALIAS_ON);

        g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
                             RenderingHints.VALUE_RENDER_QUALITY);

        // Зсовуємо центр координат в центр вікна
        g2d.translate(maxWidth/2, maxHeight/2);

        // Перетворення для анімації руху.
        g2d.translate(tx, ty);

        // Створення зірки
        GeneralPath star = new GeneralPath();
        star.moveTo(points[0][0], points[0][1]);

        for (int k = 1; k < points.length; k++)
            star.lineTo(points[k][0], points[k][1]);

        star.closePath();

        // Перетворення для анімації повороту. Якщо не задати 2 та 3 параметри –
поворот відбудеться відносно центру координат
        g2d.rotate(angle, star.getBounds2D().getCenterX(),
star.getBounds2D().getCenterY());

        // Перетворення для анімації масштабу
        g2d.scale(scale, 0.99);

        // Перетворення для анімації зміни прозорості
        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
                (float)scale));

        // Далі йдуть всі ті методи, що необхідні для власне малювання малюнку
        g2d.fill(star);

    }

    public static void main(String[] args) {
```

```
        JFrame frame = new JFrame("Приклад анімації");
        frame.add(new Star());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500, 500);
        frame.setResizable(false);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);

             Dimension size = frame.getSize();
        Insets insets = frame.getInsets();
        maxWidth =  size.width - insets.left - insets.right - 1;
        maxHeight =  size.height - insets.top - insets.bottom - 1;
    }

    // Цей метод буде викликано щоразу, як спрацює таймер
    public void actionPerformed(ActionEvent e) {

        if ( scale < 0.01 ) {
            delta = -delta;
        } else if (scale > 0.99) {
            delta = -delta;
        }

        if ( tx < -maxWidth/3 ) {
            dx = -dx;
        } else if ( tx > maxWidth/3 ) {
             dx = -dx;
        }

        if ( ty < -maxHeight/3 ) {
            dy = -dy;
        } else if ( ty > maxHeight/3 ) {
             dy = -dy;
        }

        scale += delta;
        angle += 0.01;
        tx += dx;
        ty += dy;

        repaint();
    }
}
```

Завдання

За допомогою Java 2D намалювати картинку з лабораторної роботи №1 (за варіантом).

Додатково виконати:

1. Хоча б 1 стандартний примітив, та хоча б 1 фігуру, побудовану по точкам (ламаною).
2. Хоча б 1 фігуру залити градієнтною фарбою за вибором (в цьому випадку колір може не співпадати з варіантом із лабораторної роботи № 1).

3. На достатній відстані від побудованого малюнку намалювати прямокутну рамку, всередині якої відбуватиметься анімація. Тип лінії рамки задано за варіантом.
4. Виконати анімацію малюнку, за варіантом. При цьому рамка повинна залишатися статичною. Взаємодія з рамкою не обов’язкова, якщо не передбачено варіантом.

Варіанти завдань

№	Типи анімації	Тип лінії рамки
1	1, 5	JOIN_BEVEL
2	1, 10	JOIN_MITER
3	1, 7	JOIN_ROUND
4	3, 10	JOIN_BEVEL
5	1, 9	JOIN_MITER
6	7, 9	JOIN_ROUND
7	3, 5	JOIN_BEVEL
8	9, 10	JOIN_MITER
9	3, 7	JOIN_ROUND
10	7, 10	JOIN_BEVEL
11	3, 9	JOIN_MITER
12	1, 8	JOIN_ROUND
13	2, 9	JOIN_BEVEL
14	1, 6	JOIN_MITER
15	3, 6	JOIN_ROUND
16	4, 10	JOIN_BEVEL
17	5, 9	JOIN_MITER
18	5, 10	JOIN_ROUND
19	2, 10	JOIN_BEVEL
20	6, 10	JOIN_MITER
21	8, 9	JOIN_ROUND
22	3, 8	JOIN_BEVEL
23	6, 9	JOIN_MITER
24	8, 10	JOIN_ROUND
25	4, 9	JOIN_BEVEL
26	2, 5	JOIN_MITER
27	4, 6	JOIN_ROUND
28	2, 7	JOIN_BEVEL
29	4, 8	JOIN_MITER
30	2, 6	JOIN_ROUND

Типи анімації:

1. Рух по колу проти годинникової стрілки
2. Рух по колу за годинниковою стрілкою

3. Рух по квадрату проти годинникової стрілки
4. Рух по квадрату за годинниковою стрілкою
5. Обертання навколо центру малюнка за годинниковою стрілкою
6. Обертання навколо центру малюнка проти годинникової стрілки
7. Обертання навколо кута малюнка за годинниковою стрілкою
8. Обертання навколо кута малюнка проти годинникової стрілки
9. Зміна прозорості
10. Масштабування

Контрольні запитання

1. Що таке Java 2D?
2. Які основні можливості надає Java 2D?
3. Які є базові примітиви в Java 2D?
4. Які є методи заливки в Java 2D?
5. Які основні перетворення доступні в Java 2D?
6. Які є типи декорації кінців ліній в Java 2D?
7. Які є типи декорації з’єднань в Java 2D?