```java
/**
 * @author Simon Guggisberg
 * @author Patrick Furrer
 * @version 1.0
 * @description Application that creates a window and draws shapes that bounce around in it.
 * @since 2024-03-21
 */


import bouncable.Bouncable;
import display.Displayer;
import factory.*;
import display.Display;
import factory.BorderFactory;

import javax.swing.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.LinkedList;

/**
 * Main class for the Bouncers program.
 */
public class Bouncers {
    private static final String TITLE = "Bouncers";
    private static final int NB_SPAWN = 10;
    private static final int REFRESH_MS = 10;
    private final LinkedList<Bouncable> bouncers = new LinkedList<>();
    private Timer timer;

    private ShapeFactory borderFactory = new BorderFactory();
    private ShapeFactory filledFactory = new FilledFactory();

    /**
     * Creates a new Bouncers object.
     */
    private Bouncers() {
        Displayer instance = Display.getInstance();
        instance.setTitle(TITLE);

        KeyAdapter keyAdapter = new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                super.keyPressed(e);
                switch (e.getKeyChar()) {
                    case 'e': {
                        bouncers.clear();
                    }
                    break;
                    case 'b': {
                        generateShapes(borderFactory);
                    }
                    break;
                    case 'f': {
                        generateShapes(filledFactory);
                    }
                    break;
                    case 'q': {
                        System.exit(0);
```

```
                        break;
                    }
                }
            }
        };
        instance.addKeyListener(keyAdapter);

    }

    /**
     * Generates circles and squares and adds them to the list of bouncers.
     *
     * @param factory the factory to use
     */
    private void generateShapes(ShapeFactory factory) {
        for (int i = 0; i < NB_SPAWN; ++i) {
            bouncers.add(factory.createSquare());
            bouncers.add(factory.createCircle());
        }
    }

    /**
     * Creates a timer and starts it.
     */
    public void run() {
        timer = new Timer(REFRESH_MS, e -> {
            for (Bouncable b : bouncers) {
                b.move();
                b.draw();
            }
            Display.getInstance().repaint();
        });
        timer.start();
    }

    /**
     * Main method.
     *
     * @param args command line arguments
     */
    public static void main(String... args) {
        new Bouncers().run();
    }
}
```

```java
package display;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.KeyAdapter;

import display.renderer.*;
import display.renderer.Renderer;

/**
 * Singleton class for the Display.
 */
public class Display implements Displayer {
    private static final int INIT_WIDTH = 800;
    private static final int INIT_HEIGHT = 600;
    private static Display instance;
    private final JFrame frame;
    private final JPanel panel;
    private Image image;
    private Graphics2D g2d;
    private Renderer filledRenderer;
    private Renderer borderRenderer;

    /**
     * Get the instance of the Display
     *
     * @return the instance
     */
    public static Displayer getInstance() {
        if (instance == null)
            instance = new Display();
        return instance;
    }

    /**
     * Creates a new Display object.
     */
    private Display() {
        this(INIT_WIDTH, INIT_HEIGHT);
    }

    /**
     * Creates a new Display object.
     *
     * @param width  the width
     * @param height the height
     */
    private Display(int width, int height) {
        filledRenderer = new FilledRenderer();
        borderRenderer = new BorderRenderer();
        frame = new JFrame();
        panel = new JPanel();
        frame.add(panel);
        frame.setLayout(new FlowLayout(FlowLayout.LEADING));
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
```

```
        frame.setVisible(true);
        frame.setFocusable(true);
        frame.requestFocusInWindow();
        frame.addComponentListener(new ComponentAdapter() {
            @Override
            public void componentResized(ComponentEvent e) {
                setSize(frame.getContentPane().getWidth(),
                        frame.getContentPane().getHeight());
            }
        });

        panel.setLayout(new BorderLayout());
        frame.setSize(width, height);
        setSize(width, height);
    }

    @Override
    public int getWidth() {
        return frame.getContentPane().getWidth();
    }

    @Override
    public int getHeight() {
        return frame.getContentPane().getHeight();
    }

    @Override
    public void addKeyListener(KeyAdapter keyAdapter) {
        frame.addKeyListener(keyAdapter);
    }

    /**
     * Close the display
     */
    public void close() {
        frame.dispose();
    }

    /**
     * Get the FilledRenderer
     *
     * @return the FilledRenderer
     */
    public Renderer getFilledRenderer() {
        return filledRenderer;
    }

    /**
     * Get the BorderRenderer
     *
     * @return the BorderRenderer
     */
    public Renderer getBorderRenderer() {
        return borderRenderer;
    }

    /**
     * Set the size of the display
     *
```

```
     * @param width  the width
     * @param height the height
     */
    private void setSize(int width, int height) {
        panel.setSize(width, height);
        //panel.setPreferredSize(new Dimension(width, height));
        if (image != null) {
            image.flush();
        }
        image = panel.createImage(width, height);
        g2d = (Graphics2D) image.getGraphics();
    }

    @Override
    public Graphics2D getGraphics() {
        return g2d;
    }

    @Override
    public void repaint() {
        panel.getGraphics().drawImage(image, 0, 0, null);
        image.getGraphics().clearRect(0, 0,
                frame.getContentPane().getWidth(),
                frame.getHeight());
    }

    @Override
    public void setTitle(String title) {
        frame.setTitle(title);
    }
}
```

```java
package display;

import java.awt.*;
import java.awt.event.KeyAdapter;

/**
 * Displayer is the interface for interacting with a display.
 */
public interface Displayer {
    /**
     * Get the width
     * @return the width
     */
    int getWidth();

    /**
     * Get the height
     * @return the height
     */
    int getHeight();

    /**
     * Get the graphics
     * @return the graphics2D
     */
    Graphics2D getGraphics();

    /**
     * Repaint the display
     */
    void repaint();

    /**
     * Set the title of the display
     * @param title the title
     */
    void setTitle(String title);

    /**
     * Add a key listener
     * @param ka the key adapter
     */
    void addKeyListener(KeyAdapter ka);
}
```

```java
package display.renderer;

import java.awt.*;

import bouncable.Bouncable;

/**
 * Renderer is an interface that defines how to display a bouncable object on the screen
 */
public interface Renderer {
    /**
     * Display the bouncable object on the screen
     *
     * @param g the graphics object
     * @param b the bouncable object
     */
    void display(Graphics2D g, Bouncable b);
}
```

```java
package display.renderer;

import bouncable.Bouncable;
import display.Display;

import java.awt.*;

/**
 * BorderRenderer is a Singleton Renderer that draws a Bouncable with its color and shape
but only its border.
 */
public class BorderRenderer extends AbstractRenderer {
    private static Renderer instance;
    final static int STROKE_WIDTH = 2;

    /**
     * Returns the instance of BorderRenderer.
     *
     * @return the instance of BorderRenderer
     */
    public static Renderer getInstance() {
        if (instance == null)
            instance = new BorderRenderer();
        return instance;
    }

    @Override
    public void display(Graphics2D g, Bouncable b) {
        g.setStroke(new BasicStroke(STROKE_WIDTH));
        super.display(g, b);
    }
}
```

```java
package display.renderer;

import bouncable.Bouncable;

import java.awt.*;

/**
 * FilledRenderer is a Singleton Renderer that draws a Bouncable filled with its color and shape.
 */
public class FilledRenderer extends AbstractRenderer {
    private static Renderer instance;

    /**
     * Returns the instance of FilledRenderer.
     *
     * @return the instance of FilledRenderer
     */
    public static Renderer getInstance() {
        if (instance == null)
            instance = new FilledRenderer();
        return instance;
    }

    @Override
    public void display(Graphics2D g, Bouncable b) {
        super.display(g, b);
        g.fill(b.getShape());
    }
}
```

```java
package display.renderer;

import bouncable.Bouncable;
import display.Display;

import java.awt.*;

/**
 * AbstractRenderer is a Renderer that draws a Bouncable with its color and shape.
 */
public abstract class AbstractRenderer implements Renderer {
    @Override
    public void display(Graphics2D g, Bouncable b) {
        g.setColor(b.getColor());
        g.draw(b.getShape());
    }
}
```

```java
package factory;

import bouncable.Circle;
import bouncable.Square;

/**
 * Interface for a shape factory.
 */
public interface ShapeFactory {
    /**
     * Create a square object
     */
    Square createSquare();

    /**
     * Create a Circle object
     */
    Circle createCircle();
}
```

```java
package factory;

import bouncable.Circle;
import bouncable.Square;
import bouncable.border.*;

/**
 * BorderFactory is a ShapeFactory that creates shapes with their border only.
 */
public class BorderFactory implements ShapeFactory {
    @Override
    public Square createSquare() {
        return new SquareBorder();
    }

    @Override
    public Circle createCircle() {
        return new CircleBorder();
    }
}
```

```java
package factory;

import bouncable.Circle;
import bouncable.Square;
import bouncable.filled.*;

/**
 * FilledFactory is a ShapeFactory that creates filled shapes.
 */
public class FilledFactory implements ShapeFactory {
    @Override
    public Square createSquare() {
        return new SquareFilled();
    }

    @Override
    public Circle createCircle() {
        return new CircleFilled();
    }
}
```

```java
package bouncable;

import java.awt.*;
import java.awt.geom.*;

/**
 * Circle is a BouncableShape that is drawn as a circle.
 */
public abstract class Circle extends BouncableShape {
    public Circle(Color color) {
        super(color);
    }

    @Override
    public Shape getShape() {
        return new Ellipse2D.Double(x, y, size, size);
    }
}
```

```java
package bouncable;

import java.awt.*;
import java.awt.geom.*;

/**
 * Square is a BouncableShape that is drawn as a square.
 */
public abstract class Square extends BouncableShape {
    public Square(Color color) {
        super(color);
    }

    @Override
    public Shape getShape() {
        return new Rectangle2D.Double(x, y, size, size);
    }
}
```

```java
package bouncable;

import java.awt.*;

/**
 * Interface for a bouncable object.
 */
public interface Bouncable {
    /**
     * Draw the object.
     */
    void draw();

    /**
     * Move the object.
     */
    void move();

    /**
     * Get the color of the object.
     *
     * @return the color
     */
    Color getColor();

    /**
     * Get the shape of the object.
     *
     * @return the shape
     */
    Shape getShape();
}
```

```java
package bouncable;

import display.Display;

import java.awt.*;
import java.util.Random;

/**
 * Abstract class for a bouncable shape.
 */
public abstract class BouncableShape implements Bouncable {
    private static final Random RANDOM = new Random();
    private static final int MIN_SIZE = 1;
    private static final int MAX_SIZE = 50;
    private static final int MIN_SPEED = 1;
    private static final int MAX_SPEED = 5;
    private final Color color;
    protected int size;
    protected int x;
    protected int y;
    private int dx;
    private int dy;

    /**
     * Creates a new BouncableShape object.
     *
     * @param color the color
     */
    public BouncableShape(Color color) {
        this.x = RANDOM.nextInt(Display.getInstance().getWidth());
        this.y = RANDOM.nextInt(Display.getInstance().getHeight());
        this.size = RANDOM.nextInt(MIN_SIZE, MAX_SIZE);
        this.color = color;
        dx = getRandomSpeed();
        dy = getRandomSpeed();
    }

    /**
     * Get a random speed in ]-maxSpeed, -1] U [1, maxSpeed[
     *
     * @return the random speed
     */
    private static int getRandomSpeed() {
        return RANDOM.nextInt(MIN_SPEED, MAX_SPEED) * (RANDOM.nextBoolean() ? 1 : -1);
    }

    @Override
    public void move() {
        x += dx;
        y += dy;
        int halfSize = size / 2;
        if (x - halfSize < 0 || x + halfSize > Display.getInstance().getWidth()) {
            dx = -dx;
            x = x - halfSize < 0 ? halfSize : Display.getInstance().getWidth() - halfSize;
        }
        if (y - halfSize < 0 || y + halfSize > Display.getInstance().getHeight()) {
            dy = -dy;
            y = y - halfSize < 0 ? halfSize : Display.getInstance().getHeight() - halfSize;
        }
```

```
    }

    @Override
    public Color getColor() {
        return color;
    }
}
```

```java
package bouncable.border;

import bouncable.Circle;
import display.Display;
import display.renderer.BorderRenderer;

import java.awt.*;

/**
 * CircleBorder is a Circle that is drawn with a green border only.
 */
public class CircleBorder extends Circle {
    public CircleBorder() {
        super(Color.GREEN);
    }

    @Override
    public void draw() {
        BorderRenderer.getInstance().display(Display.getInstance().getGraphics(), this);
    }
}
```

```java
package bouncable.border;

import bouncable.Square;
import display.Display;
import display.renderer.BorderRenderer;

import java.awt.*;

/**
 * SquareBorder is a Square that is drawn with a red border only.
 */
public class SquareBorder extends Square {
    public SquareBorder() {
        super(Color.RED);
    }

    @Override
    public void draw() {
        BorderRenderer.getInstance().display(Display.getInstance().getGraphics(), this);
    }
}
```

```java
package bouncable.filled;

import bouncable.Circle;
import display.Display;
import display.renderer.FilledRenderer;

import java.awt.*;

/**
 * CircleFilled is a Circle that is drawn filled with blue.
 */
public class CircleFilled extends Circle {
    public CircleFilled() {
        super(Color.BLUE);
    }

    @Override
    public void draw() {
        FilledRenderer.getInstance().display(Display.getInstance().getGraphics(), this);
    }
}
```

```java
package bouncable.filled;

import bouncable.Square;
import display.Display;
import display.renderer.FilledRenderer;

import java.awt.*;

/**
 * SquareFilled is a Square that is drawn filled with orange.
 */
public class SquareFilled extends Square {
    public SquareFilled() {
        super( Color.ORANGE);
    }

    @Override
    public void draw() {
        FilledRenderer.getInstance().display(Display.getInstance().getGraphics(), this);
    }
}
```