```java
/**
 *  MCR - Laboratoire 2 Bouncers
 *  Rui Manuel Mota Carneiro, Kylian Manzini
 */

import bounceable.Bounceable;
import display.Display;
import factory.BouncerFactory;
import factory.FilledFactory;
import factory.NotFilledFactory;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.LinkedList;
import java.util.Random;
import java.util.concurrent.locks.*;

public class Bouncers {
    private final LinkedList<Bounceable> BOUNCERS = new LinkedList<>();
    private boolean running;

    private final Lock MUTEX = new ReentrantLock();

    final int MAX_SIZE = 50;
    final int MAX_SPEED = 5;
    final int FPS = 25;

    public Bouncers() {
        running = true;
    }

    private void createSquaresBatch(BouncerFactory factory, int amount) {
        Display display = Display.getInstance();
        Random r = new Random();

        for (int i = 0; i < amount; ++i) {
            BOUNCERS.add(factory.createSquare(
                    r.nextInt(display.getWidth() - MAX_SIZE),
                    r.nextInt(display.getHeight() - MAX_SIZE),
                    r.nextInt(MAX_SIZE),
                    r.nextInt(MAX_SPEED) * (r.nextBoolean() ? 1 : -1),
                    r.nextInt(MAX_SPEED) * (r.nextBoolean() ? 1 : -1))
            );
        }
    }

    private void createCircleBatch(BouncerFactory factory, int amount) {
        Display display = Display.getInstance();
        Random r = new Random();

        for (int i = 0; i < amount; ++i) {
            BOUNCERS.add(factory.createCircle(
                    r.nextInt(display.getWidth() - MAX_SIZE),
                    r.nextInt(display.getHeight() - MAX_SIZE),
                    r.nextInt(MAX_SIZE),
                    r.nextInt(MAX_SPEED) * (r.nextBoolean() ? 1 : -1),
                    r.nextInt(MAX_SPEED) * (r.nextBoolean() ? 1 : -1))
            );
        }
    }

    public void run() {

        Display display = Display.getInstance();

        Display.getInstance().addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent keyEvent) {
                switch (keyEvent.getKeyCode()) {
                    case KeyEvent.VK_E -> {
```

```
 71                          MUTEX.lock();
 72                          BOUNCERS.clear();
 73                          MUTEX.unlock();
 74                      }
 75                      case KeyEvent.VK_B -> {
 76                          MUTEX.lock();
 77                          createSquaresBatch(NotFilledFactory.getInstance(), 5);
 78                          createCircleBatch(NotFilledFactory.getInstance(), 5);
 79                          MUTEX.unlock();
 80                      }
 81                      case KeyEvent.VK_F -> {
 82                          MUTEX.lock();
 83                          createCircleBatch(FilledFactory.getInstance(), 5);
 84                          createSquaresBatch(FilledFactory.getInstance(), 5);
 85                          MUTEX.unlock();
 86                      }
 87                      case KeyEvent.VK_Q ->
 88                          running = false;
 89
 90                  }
 91              }
 92          });
 93
 94
 95          long lastTime, currentTime;
 96          lastTime = System.currentTimeMillis() - 40;
 97          while (running) {
 98
 99              currentTime = System.currentTimeMillis();
100              try {
101                  Thread.sleep(Math.max((1000 / FPS) - (currentTime - lastTime), 0));
102              } catch (InterruptedException e) {
103                  e.printStackTrace();
104              }
105              lastTime = System.currentTimeMillis();
106
107
108              MUTEX.lock();
109
110              for (Bounceable bounce : BOUNCERS) {
111                  bounce.move(display.getWidth(), display.getHeight());
112              }
113              for (Bounceable bounce : BOUNCERS) {
114                  bounce.draw();
115              }
116              display.repaint();
117
118              MUTEX.unlock();
119          }
120          System.exit(0);
121      }
122
123      public static void main(String... args) {
124          new Bouncers().run();
125      }
126  }
```

```java
package display;

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

public class Display implements Displayer {

    static final int WINDOW_SIZE = 500;
    static final int MIN_HEIGHT = 100;
    static final int MIN_WIDTH = 200;
    private static Display instance;
    private final JPanel content;
    public final JFrame window;
    private Image img;


    private Display() {
        window = new JFrame();
        window.setSize(WINDOW_SIZE,WINDOW_SIZE);
        window.setMinimumSize(new Dimension(MIN_WIDTH,MIN_HEIGHT));
        window.setTitle("Bouncers");


        content = new JPanel();
        content.setSize(WINDOW_SIZE,WINDOW_SIZE);
        content.setBackground(Color.WHITE);
        window.setContentPane(content);



        // trigger when window is resized
        window.addComponentListener(new ComponentAdapter() {
            @Override
            public void componentResized(ComponentEvent e) {
                img = createImage();
            }
        });

        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);

        img = createImage();
    }


    /**
     * Create or return the Display instance
     */
    public static Display getInstance() {
        if (instance == null) instance = new Display();
        return instance;
    }

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

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

    @Override
    public Graphics2D getGraphics() {
        return (Graphics2D) img.getGraphics();
    }
```

```java

    @Override
    public void repaint() {
        content.getGraphics().drawImage(img,0,0,null);
        Graphics2D g = getGraphics();

        g.clearRect(0, 0, getWidth(), getHeight());
        g.dispose();
    }

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

    @Override
    public void addKeyListener(KeyAdapter ka) {
        window.addKeyListener(ka);
    }

    private Image createImage(){
        return content.createImage(getWidth(),getHeight());
    }
}

```

```java
package display;

import java.awt.Graphics2D;
import java.awt.event.KeyAdapter;

public interface Displayer {
    int getWidth();

    int getHeight();

    Graphics2D getGraphics();

    void repaint();

    void setTitle(String title);
    void addKeyListener(KeyAdapter ka);
}

```

```java
package display.renderer;

import bounceable.Bounceable;

import java.awt.Graphics2D;

public interface Renderer {
    void display(Graphics2D g, Bounceable b);
}
```

```java
package display.renderer;

import bounceable.Bounceable;

import java.awt.Graphics2D;

public class FilledRender implements Renderer {

    static private FilledRender instance;

    private FilledRender(){};

    public static Renderer getInstance() {
        if (instance == null){
            instance = new FilledRender();
        }
        return instance;
    }

    @Override
    public void display(Graphics2D g, Bounceable b) {
        g.setPaint(b.getColor());
        g.fill(b.getShape());
        g.draw(b.getShape());
    }
}

```

```java
package display.renderer;

import bounceable.Bounceable;

import java.awt.*;

public class NotFilledRender implements Renderer {
    static private final BasicStroke STROKE = new BasicStroke(5);

    static private NotFilledRender instance;

    private NotFilledRender(){};

    public static Renderer getInstance() {
        if (instance == null){
            instance = new NotFilledRender();
        }
        return instance;
    }

    @Override
    public void display(Graphics2D g, Bounceable b) {
        g.setStroke(STROKE);
        g.setPaint(b.getColor());
        g.draw(b.getShape());
    }
}

```

```java
package factory;

import bounceable.FilledCircle;
import bounceable.FilledSquare;

import java.awt.Color;

public class FilledFactory extends BouncerFactory {

    static private FilledFactory instance;

    private FilledFactory(){}

    /**
     * Create or return the FilledFactory instance
     */
    public static FilledFactory getInstance() {
        if (instance == null)
            instance = new FilledFactory();
        return instance;
    }

    @Override
    public FilledCircle createCircle(int x, int y, int diameter, int directionX, int directionY) {
        return new FilledCircle(x, y, directionX, directionY, diameter, Color.BLUE);
    }

    @Override
    public FilledSquare createSquare(int x, int y, int size, int directionX, int directionY) {
        return new FilledSquare(x, y, directionX, directionY, size, Color.ORANGE);
    }
}

```

```java
package factory;

import bounceable.Circle;
import bounceable.Square;

public abstract class BouncerFactory {
    public abstract Circle createCircle(int x, int y, int diameter, int directionX, int directionY);
    public abstract Square createSquare(int x, int y, int size, int directionX, int directionY);
}

```

```java
package factory;

import bounceable.NotFilledCircle;
import bounceable.NotFilledSquare;

import java.awt.Color;

public class NotFilledFactory extends BouncerFactory {

    static private NotFilledFactory instance;

    private NotFilledFactory(){}

    /**
     * Create or return the NotFilledFactory instance
     */
    public static NotFilledFactory getInstance() {
        if (instance == null)
            instance = new NotFilledFactory();
        return instance;
    }

    @Override
    public NotFilledCircle createCircle(int x, int y, int diameter, int directionX, int directionY) {
        return new NotFilledCircle(x, y, directionX, directionY, diameter, Color.GREEN);
    }

    @Override
    public NotFilledSquare createSquare(int x, int y, int size, int directionX, int directionY) {
        return new NotFilledSquare(x, y, directionX, directionY, size, Color.RED);
    }
}

```

```java
package bounceable;

import java.awt.Color;
import java.awt.geom.Ellipse2D;

public abstract class Circle extends Bouncer {

    private final int diameter;

    Circle(int x, int y, int directionX, int directionY, int diameter, Color color) {
        super(x, y, directionX, directionY, color);
        this.diameter = diameter;
    }

    @Override
    protected void bounceChecker(int width, int height){
        if (x < 0){
            directionX = -directionX;
            x = 0;
        }
        if (x > width - diameter){
            directionX = -directionX;
            x = width - diameter;
        }
        if (y < 0){
            directionY = -directionY;
            y = 0;
        }
        if (y > height - diameter){
            directionY = -directionY;
            y = height - diameter;
        }
    }


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

```

```java
package bounceable;

import java.awt.Color;
import java.awt.geom.Rectangle2D;

public abstract class Square extends Bouncer {

    private final int length;

    Square(int x, int y, int directionX, int directionY, int length, Color color) {
        super(x, y, directionX, directionY, color);
        this.length = length;
    }

    @Override
    protected void bounceChecker(int width, int height){
        if (x < 0){
            directionX = -directionX;
            x = 0;
        }
        if (x > width - length){
            directionX = -directionX;
            x = width - length;
        }
        if (y < 0){
            directionY = -directionY;
            y = 0;
        }
        if (y > height - length){
            directionY = -directionY;
            y = height - length;
        }
    }

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

```

```java
package bounceable;

import display.Display;
import display.renderer.Renderer;

import java.awt.Color;
import java.util.Vector;

abstract public class Bouncer implements Bounceable {

    protected int directionX;
    protected int directionY;
    protected int x;
    protected int y;

    protected Color color;

    Bouncer(int x, int y, int directionX, int directionY, Color color) {
        this.directionX = directionX;
        this.directionY = directionY;
        this.x = x;
        this.y = y;
        this.color = color;
    }
    Bouncer(Vector<Integer> position, Vector<Integer> direction, Color color) {
        if (position.capacity() != 2 || direction.capacity() != 2)
            throw new RuntimeException("position and direction must be vector of size 2. (x, y)");
        this.directionX = direction.elementAt(0);
        this.directionY = direction.elementAt(1);
        this.x = position.elementAt(0);
        this.y = position.elementAt(1);
        this.color = color;
    }

    /**
     * Check if Bouncable must bounce
     * @param width width of the pane to bounce on
     * @param height height of the pane to bounce on
     */
    abstract protected void bounceChecker(int width, int height);

    @Override
    public void move(int width, int height) {
        bounceChecker(width, height);
        x += directionX;
        y += directionY;
    }

    @Override
    public Color getColor() {
        return this.color;
    }
    @Override
    public void draw() {
        getRenderer().display(Display.getInstance().getGraphics(), this);
    }

    /**
     * Get the correct Renderer according to the type of the caller
     * @return a object that implement the Renderer interface
     */
    protected abstract Renderer getRenderer();

}
```

```java
package bounceable;

import java.awt.Color;
import java.awt.Shape;

public interface Bounceable {
    void draw();
    void move(int width, int height);
    Color getColor();
    Shape getShape();

}
```

```java
package bounceable;

import display.renderer.FilledRender;
import display.renderer.Renderer;

import java.awt.Color;

public class FilledCircle extends Circle{
    public FilledCircle(int x, int y, int directionX, int directionY, int diameter, Color color) {
        super(x, y, directionX, directionY, diameter, color);
    }

    @Override
    protected Renderer getRenderer() {
        return FilledRender.getInstance();
    }
}

```

```java
package bounceable;

import display.renderer.FilledRender;
import display.renderer.Renderer;

import java.awt.Color;

public class FilledSquare extends Square {
    public FilledSquare(int x, int y, int directionX, int directionY, int length, Color color) {
        super(x, y, directionX, directionY, length, color);
    }

    @Override
    protected Renderer getRenderer() {
        return FilledRender.getInstance();
    }
}

```

```java
package bounceable;

import display.renderer.Renderer;
import display.renderer.NotFilledRender;

import java.awt.Color;

public class NotFilledCircle extends Circle{
    public NotFilledCircle(int x, int y, int directionX, int directionY, int diameter, Color color) {
        super(x, y, directionX, directionY, diameter, color);
    }

    @Override
    protected Renderer getRenderer() {
        return NotFilledRender.getInstance();
    }
}

```

```java
package bounceable;

import display.renderer.Renderer;
import display.renderer.NotFilledRender;
import java.awt.*;

public class NotFilledSquare extends Square{
    public NotFilledSquare(int x, int y, int directionX, int directionY, int length, Color color) {
        super(x, y, directionX, directionY, length, color);
    }

    @Override
    protected Renderer getRenderer() {
        return NotFilledRender.getInstance();
    }
}

```