```java
import Shape.Bouncable;
import Shape.Factory.BorderedShapeFactory;
import Shape.Factory.FilledShapeFactory;
import Shape.CustomShape;
import Shape.Factory.ShapeFactory;
import View.Displayer;
import View.MainWindow;

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

/**
 * -------------------------------------------------------------------------------
 * @Labo         : Labo 02 : Singleton
 * @Authors      : Slimani Walid & Steiner Jeremiah
 * @Description  : This file contains the definition of the Bouncers class,
 *                 represents a program that simulates bouncing shapes.
 * @Remarque     : /
 * @Modification : /
 * -------------------------------------------------------------------------------
 **/

public class Bouncers {

    // region Field
    /** List of bouncing shapes. */
    private final LinkedList<Bouncable> bouncers;

    /** Number of shapes to instantiate by each key press. */
    private final static int NBR_BY_CLICK = 10;

    /** Delay for refreshing the display, in milliseconds. */
    private final static int REFRESH_DELAY = 10;
    // endregion

    // region Ctor
    /**
     * @brief Constructs a new instance of the Bouncers class.
     */
    private Bouncers() {
        bouncers = new LinkedList<Bouncable>();
    }
    // endregion

    // region Public methode
    /**
     * @brief The main method of the program.
     * @param args The command-line arguments.
     */
    public static void main(String[] args) {
        new Bouncers().run();
    }

    /**
     * @brief Runs the Bouncers program.
     */
    public void run() {
        MainWindow window = MainWindow.getInstance();
        window.setTitle("Bouncer");

        window.addKeyListener(new KeyAdapter() {
```

```
64              @Override
65              public void keyPressed(KeyEvent e) {
66                  switch (e.getKeyCode()) {
67                      case KeyEvent.VK_B ->
68                          instantiate(BorderedShapeFactory.getInstance(), NBR_BY_CLICK);
69                      case KeyEvent.VK_F ->
70                          instantiate(FilledShapeFactory.getInstance(), NBR_BY_CLICK);
71
72                      case KeyEvent.VK_E -> bouncers.clear();
73                      case KeyEvent.VK_Q -> System.exit(0);
74                  }
75              }
76          });
77
78          new Timer(REFRESH_DELAY, e -> update(window)).start();
79      }
80      // endregion
81
82      // region Private methode
83      /**
84       * @brief Updates the display.
85       * @param window The display window.
86       */
87      private void update(Displayer window)
88      {
89          moveShapes(window);
90      }
91
92      /**
93       * @brief Moves the shapes and updates the display.
94       * @param window The display window.
95       */
96      private void moveShapes(Displayer window) {
97          window.repaint();
98          for (Bouncable customShape : bouncers) {
99              customShape.move();
100             customShape.draw();
101         }
102     }
103
104     /**
105      * @brief Instantiates a specified number of shapes using the given factory.
106      * @param factory The shape factory.
107      * @param nbr The number of shapes to instantiate.
108      */
109     private void instantiate(ShapeFactory factory, int nbr)
110     {
111         for (int i = 0; i < nbr; ++i)
112         {
113             bouncers.add(factory.createCircle());
114             bouncers.add(factory.createSquare());
115         }
116     }
117     // endregion
118 }
```

```java
package Shape.Bordered;

import Shape.Circle;
import Shape.Renderer.BorderedRenderer;

import java.awt.*;

/**
 * -----------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the BorderedCircle class.
 * @Info        : Represents a bordered circle shape.
 * -----------------------------------------------------------------------------
 **/

public class BorderedCircle extends Circle {
    // region Ctor
    /**
     * @brief Constructs a new instance of the BorderedCircle class.
     */
    public BorderedCircle() {
        super();
        renderer = BorderedRenderer.getInstance();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the color of the BorderedCircle.
     * @return The color of the BorderedCircle.
     */
    @Override
    public Color getColor() {
        return Color.GREEN;
    }
    // endregion
}
```

```java
package Shape.Bordered;

import Shape.Renderer.BorderedRenderer;
import Shape.Square;
import java.awt.*;

/**
 * -------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the BorderedSquare class.
 * @Info        : Represents a bordered square shape.
 * -------------------------------------------------------------------------------------
 **/

public class BorderedSquare extends Square {
    // region Ctor
    /**
     * @brief Constructs a new instance of the BorderedSquare class.
     */
    public BorderedSquare() {
        super();
        renderer = BorderedRenderer.getInstance();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the color of the BorderedSquare.
     * @return The color of the BorderedSquare.
     */
    @Override
    public Color getColor() {
        return Color.RED;
    }
    // endregion
}
```

```java
package Shape;

import java.awt.*;

/**
 * ---------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the Bouncable interface.
 * @Info        : Represents an interface for objects that can bounce.
 * ---------------------------------------------------------------------------------
 **/

public interface Bouncable {
    // region Public methode
    /**
     * @brief Draws the object.
     */
    void draw();

    /**
     * @brief Moves the object.
     */
    void move();

    /**
     * @brief Gets the color of the object.
     * @return The color of the object.
     */
    Color getColor();

    /**
     * @brief Gets the shape of the object.
     * @return The shape of the object.
     */
    Shape getShape();
    // endregion
}
```

```java
package Shape;

import java.awt.*;
import java.awt.geom.Ellipse2D;

/**
 * -----------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the Circle abstract class.
 * @Info        : Represents an abstract class for circle shapes.
 * -----------------------------------------------------------------------------
 **/

public abstract class Circle extends CustomShape {
    // region Ctor
    /**
     * @brief Constructs a new instance of the Circle class.
     */
    public Circle() {
        super();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the shape of the circle.
     * @return The shape of the circle.
     */
    @Override
    public Shape getShape() {
        return new Ellipse2D.Double(position.getX(), position.getY(), size, size);
    }
    // endregion
}
```

```java
package Shape;

import Shape.Renderer.Renderer;
import View.Displayer;
import View.MainWindow;

import java.util.Random;

import static java.lang.Math.max;
import static java.lang.Math.min;

/**
 * -------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the CustomShape abstract class.
 * @Info        : Represents an abstract class for custom shapes.
 * -------------------------------------------------------------------------------------
 **/

public abstract class CustomShape implements Bouncable {
    // region Field
    /** The display area associated with the custom shape. */
    protected Displayer displayer;

    /** The random number generator. */
    private final static Random random = new Random();

    /** The size of the custom shape. */
    protected final int size;

    /** The movement vector of the custom shape. */
    private final Vector2D movement;

    /** The speed of the custom shape. */
    private final double speed;

    /** The position vector of the custom shape. */
    protected Vector2D position;

    /** The renderer used to draw the custom shape. */
    protected Renderer renderer;
    // endregion

    // region Ctor
    /**
     * @brief Constructs a new instance of the CustomShape class.
     */
    protected CustomShape() {
        displayer = MainWindow.getInstance();
        size = random.nextInt(5, 50);
        movement = new Vector2D(
                random.nextInt(-10, 10),
                random.nextInt(-10, 10)
        );
        movement.normalize();
        speed = random.nextInt(3, 10);

        // as demo (spawn at center instead of random position as pdf)
        position = new Vector2D(
                displayer.getWidth()/2.,
                displayer.getHeight()/2.
        );
        // if random spawn needed ctrl + '/' to uncomment lines followed:
```

```
64  //        position = new Vector2D(
65  //                random.nextInt(1, displayer.getWidth()),
66  //                random.nextInt(1, displayer.getHeight())
67  //        );
68      }
69      // endregion
70
71      // region Public methode
72      /**
73       * @brief Draws the custom shape.
74       */
75      @Override
76      public void draw() {
77          renderer.display(displayer.getGraphics(), this);
78      }
79
80      /**
81       * @brief Moves the custom shape by applying the current movement.
82       */
83      @Override
84      public void move() {
85          int maxX = displayer.getWidth() - size;
86          int maxY = displayer.getHeight() - size;
87
88          Vector2D newPosition = new Vector2D(position.getX() + movement.getX() * speed,
89                                              position.getY() + movement.getY() * speed);
90
91          if (newPosition.getX() >= maxX || newPosition.getX() <= 0)
92          {
93              movement.swapOnX();
94              newPosition = new Vector2D(min(position.getX() + movement.getX() * speed, maxX),
95                                         max(0, newPosition.getY()));
96          }
97
98          if (newPosition.getY() >= maxY || newPosition.getY() <= 0)
99          {
100             movement.swapOnY();
101             newPosition = new Vector2D (newPosition.getX(), newPosition.getY() < 0 ? 0 :
102                             min(position.getY() + movement.getY() * speed, maxY));
103         }
104
105         position = newPosition;
106     }
107     // endregion
108 }
```

```java
package Shape.Factory;

import Shape.Bordered.BorderedCircle;
import Shape.Bordered.BorderedSquare;

/**
 * -------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the BorderedShapeFactory class.
 * @Info        : Represents a factory for creating bordered shapes.
 * -------------------------------------------------------------------------------------
 **/

public class BorderedShapeFactory extends ShapeFactory{

    // region Intern Static Class (for Singleton)
    /**
     * @brief Represents an inner static class for implementing the Singleton pattern.
     */
    private static class Instance
    {
        /** The singleton instance of the BorderedShapeFactory class. */
        static final BorderedShapeFactory instance = new BorderedShapeFactory();
    }
    // endregion

    // region Public methode
    /**
     * @brief Creates a new bordered circle.
     * @return The created bordered circle.
     */
    @Override
    public BorderedCircle createCircle() {
        return new BorderedCircle();
    }

    /**
     * @brief Creates a new bordered square.
     * @return The created bordered square.
     */
    @Override
    public BorderedSquare createSquare() {
        return new BorderedSquare();
    }

    /**
     * @brief Gets the singleton instance of the BorderedShapeFactory class.
     * @return The singleton instance.
     */
    public static BorderedShapeFactory getInstance() {
        return BorderedShapeFactory.Instance.instance;
    }
    // endregion

}
```

```java
package Shape.Factory;

import Shape.Filled.FilledCircle;
import Shape.Filled.FilledSquare;

/**
 * ---------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the FilledShapeFactory class.
 * @Info        : Represents a factory for creating filled shapes.
 * ---------------------------------------------------------------------------------
 **/

public class FilledShapeFactory extends ShapeFactory{

    // region Intern Static Class (for Singleton)
    /**
     * @brief Represents an inner static class for implementing the Singleton pattern.
     */
    private static class Instance
    {
        /** The singleton instance of the FilledShapeFactory class. */
        static final FilledShapeFactory instance = new FilledShapeFactory();
    }
    // endregion

    // region Public methode
    /**
     * @brief Creates a new filled circle.
     * @return The created filled circle.
     */
    @Override
    public FilledCircle createCircle() {
        return new FilledCircle();
    }

    /**
     * @brief Creates a new filled square.
     * @return The created filled square.
     */
    @Override
    public FilledSquare createSquare() {
        return new FilledSquare();
    }

    /**
     * @brief Gets the singleton instance of the FilledShapeFactory class.
     * @return The singleton instance.
     */
    public static FilledShapeFactory getInstance() {
        return FilledShapeFactory.Instance.instance;
    }
    // endregion
}
```

```java
package Shape.Factory;

import Shape.Circle;
import Shape.Square;

/**
 * -------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the ShapeFactory abstract class.
 * @Info        : Represents an abstract factory for creating shapes.
 * -------------------------------------------------------------------------------------
 **/

public abstract class ShapeFactory {
    // region Public methode
    /**
     * @brief Creates a new circle.
     * @return The created circle.
     */
    public abstract Circle createCircle();

    /**
     * @brief Creates a new square.
     * @return The created square.
     */
    public abstract Square createSquare();
    // endregion
}
```

```java
package Shape.Filled;

import Shape.Circle;
import Shape.Renderer.FilledRenderer;

import java.awt.*;

/**
 * -----------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the FilledCircle class.
 * @Info        : Represents a filled circle shape.
 * -----------------------------------------------------------------------------
 **/

public class FilledCircle extends Circle {
    // region Ctor
    /**
     * @brief Constructs a new instance of the FilledCircle class.
     */
    public FilledCircle() {
        super();
        renderer = FilledRenderer.getInstance();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the color of the FilledCircle.
     * @return The color of the FilledCircle.
     */
    @Override
    public Color getColor() {
        return Color.BLUE;
    }
    // endregion
}
```

```java
package Shape.Filled;

import Shape.Renderer.FilledRenderer;
import Shape.Square;
import java.awt.*;

/**
 * -------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the FilledSquare class.
 * @Info        : Represents a filled square shape.
 * -------------------------------------------------------------------------------------
 **/

public class FilledSquare extends Square {
    // region Ctor
    /**
     * @brief Constructs a new instance of the FilledSquare class.
     */
    public FilledSquare() {
        super();
        renderer = FilledRenderer.getInstance();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the color of the FilledSquare.
     * @return The color of the FilledSquare.
     */
    @Override
    public Color getColor() {
        return Color.ORANGE;
    }
    // endregion
}
```

```java
package Shape.Renderer;

import java.awt.*;
import Shape.Bouncable;

/**
 * ---------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the BorderedRenderer class.
 * @Info        : Represents a renderer for drawing bordered shapes.
 * ---------------------------------------------------------------------------------
 **/

public class BorderedRenderer implements Renderer{

    // region Intern Static Class (for Singleton)
    /**
     * @brief Represents an inner static class for implementing the Singleton pattern.
     */
    private static class Instance
    {
        /** The singleton instance of the BorderedRenderer class. */
        static final BorderedRenderer instance = new BorderedRenderer();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the singleton instance of the BorderedRenderer class.
     * @return The singleton instance.
     */
    public static BorderedRenderer getInstance() {
        return BorderedRenderer.Instance.instance;
    }

    /**
     * @brief Displays the given shape with a border using the provided graphics context.
     * @param g The graphics context.
     * @param b The shape to display.
     */
    @Override
    public void display(Graphics2D g, Bouncable b) {
        g.setColor(b.getColor());
        g.setStroke(new BasicStroke(2));
        g.draw(b.getShape());
    }
    // endregion
}
```

```java
package Shape.Renderer;

import java.awt.*;
import Shape.Bouncable;

/**
 * ---------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the FilledRenderer class.
 * @Info        : Represents a renderer for drawing filled shapes.
 * ---------------------------------------------------------------------------------
 **/

public class FilledRenderer implements Renderer {
    // region Intern Static Class (for Singleton)
    /**
     * @brief Represents an inner static class for implementing the Singleton pattern.
     */
    private static class Instance
    {
        /** The singleton instance of the FilledRenderer class. */
        static final FilledRenderer instance = new FilledRenderer();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the singleton instance of the FilledRenderer class.
     * @return The singleton instance.
     */
    public static FilledRenderer getInstance() {
        return FilledRenderer.Instance.instance;
    }

    /**
     * @brief Displays the given shape filled with color using the provided graphics context.
     * @param g The graphics context.
     * @param b The shape to display.
     */
    @Override
    public void display(Graphics2D g, Bouncable b) {
        g.setColor(b.getColor());
        g.fill(b.getShape());
    }
    // endregion
}
```

```java
package Shape.Renderer;

import Shape.Bouncable;

import java.awt.Graphics2D;

/**
 * ---------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the Renderer interface.
 * @Info        : Represents an interface for rendering shapes.
 * ---------------------------------------------------------------------------------------
 **/

public interface Renderer {
    // region Public methode
    /**
     * @brief Displays the given shape using the provided graphics context.
     * @param g The graphics context.
     * @param b The shape to display.
     */
    void display(Graphics2D g, Bouncable b);
    // endregion
}
```

```java
package Shape;

import java.awt.*;
import java.awt.geom.Rectangle2D;

/**
 * -------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the Square abstract class.
 * @Info        : Represents an abstract class for square shapes.
 * -------------------------------------------------------------------------------------
 **/

public abstract class Square extends CustomShape {
    // region Ctor
    /**
     * @brief Constructs a new instance of the Square class.
     */
    public Square() {
        super();
    }
    // endregion

    // region Public methode
    /**
     * @brief Gets the shape of the square.
     * @return The shape of the square.
     */
    @Override
    public final Shape getShape() {
        return new Rectangle2D.Double(position.getX(), position.getY(), size, size);
    }
    // endregion
}
```

```java
package Shape;

import static java.lang.Math.sqrt;

/**
 * ---------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the Vector2D class.
 * @Info        : Represents a 2D vector.
 * ---------------------------------------------------------------------------------
 **/

public class Vector2D {
    // region Field
    /** The x-coordinate of the vector. */
    private double x;

    /** The y-coordinate of the vector. */
    private double y;
    // endregion

    // region Ctor
    /**
     * @brief Constructs a new instance of the Vector2D class with specified coordinates.
     * @param x The x-coordinate of the vector.
     * @param y The y-coordinate of the vector.
     */
    public Vector2D(double x, double y) {
        this.x = x;
        this.y = y;
    }
    // endregion

    // region Public methode
    /**
     * @brief Normalizes the vector.
     */
    public void normalize()
    {
        double length = sqrt(x * x + y * y);
        x /= length;
        y /= length;
    }

    /**
     * @brief Gets the x-coordinate of the vector.
     * @return The x-coordinate of the vector.
     */
    public double getX() {
        return x;
    }

    /**
     * @brief Gets the y-coordinate of the vector.
     * @return The y-coordinate of the vector.
     */
    public double getY() {
        return y;
    }

    /**
     * @brief Swaps the x-coordinate of the vector.
     */
```

```
64      public void swapOnX()
65      {
66          x *= -1;
67      }
68
69      /**
70       * @brief Swaps the y-coordinate of the vector.
71       */
72      public void swapOnY()
73      {
74          y *= -1;
75      }
76      // endregion
77  }
```

```java
package View;

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

/**
 * -------------------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the Displayer interface.
 * @Info        : Represents an interface for displaying graphics.
 * -------------------------------------------------------------------------------------
 **/

public interface Displayer
{
    // region Public methode
    /**
     * @brief Gets the width of the display area.
     * @return The width of the display area.
     */
    int getWidth();
    /**
     * @brief Gets the height of the display area.
     * @return The height of the display area.
     */
    int getHeight();
    /**
     * @brief Gets the graphics context for drawing on the display area.
     * @return The graphics context.
     */
    Graphics2D getGraphics();
    /**
     * @brief Repaints the display area.
     */
    void repaint();
    /**
     * @brief Sets the title of the display area.
     * @param title The title to set.
     */
    void setTitle(String title);
    /**
     * @brief Adds a key listener to the display area.
     * @param ka The key adapter to add.
     */
    void addKeyListener(KeyAdapter ka);
    // endregion
}
```

```java
package View;

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

/**
 * ---------------------------------------------------------------------------
 * @Authors     : Slimani Walid & Steiner Jeremiah
 * @Description : This file contains the definition of the MainWindow class.
 * @Info        : Represents the main window of the application.
 * ---------------------------------------------------------------------------
 **/

public class MainWindow implements Displayer {
    // region Field
    /** The image associated with the main window. */
    protected Image image;

    /** The JFrame representing the main window. */
    private final JFrame frame;

    /** The container panel of the main window. */
    private final Container panel;
    // endregion

    // region Intern Static Class (for Singleton)
    /**
     * @brief Represents an inner static class for implementing the Singleton pattern.
     */
    private static class Instance
    {
        /** The singleton instance of the MainWindow class. */
        static final MainWindow instance = new MainWindow();
    }
    // endregion

    // region Ctor
    /**
     * @brief Constructs a new instance of the MainWindow class.
     */
    private MainWindow(){
        System.out.println("-- Singleton()");
        int size = 500;
        Dimension dimension = new Dimension(size, size);

        frame = new JFrame();

        Toolkit toolkit = Toolkit.getDefaultToolkit();

        Dimension screenSize = toolkit.getScreenSize();
        frame.setLocation(new Point((int) screenSize.getWidth() / 4, (int) screenSize.getHeight() /
            4));
        frame.setSize(dimension.width, dimension.height);
        frame.setPreferredSize(frame.getSize());

        panel = frame.getContentPane();
        panel.setBackground(Color.LIGHT_GRAY);
        image = panel.createImage(dimension.width, dimension.height);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
63
64          frame.addComponentListener(new ComponentAdapter() {
65              @Override
66              public void componentResized(ComponentEvent e) {
67                  image = panel.createImage(getWidth(),
68                          // Exception on height = 0 => set to 1 in this case
69                          (getHeight() == 0 ? 1 : getHeight())
70                  );
71              }
72          });
73          frame.setVisible(true);
74      }
75      // endregion
76
77      // region Public Methods
78      /**
79       * @brief Gets the singleton instance of the MainWindow class.
80       * @return The singleton instance.
81       */
82      public static MainWindow getInstance() {
83          return Instance.instance;
84      }
85
86      /**
87       * @brief Gets the width of the main window.
88       * @return The width of the main window.
89       */
90      @Override
91      public int getWidth() {
92          return panel.getWidth();
93      }
94
95      /**
96       * @brief Gets the height of the main window.
97       * @return The height of the main window.
98       */
99      @Override
100     public int getHeight() {
101         return panel.getHeight();
102     }
103
104     /**
105      * @brief Gets the graphics context for drawing on the main window.
106      * @return The graphics context.
107      */
108     @Override
109     public Graphics2D getGraphics() {
110         return (Graphics2D) image.getGraphics();
111     }
112
113     /**
114      * @brief Repaints the main window.
115      */
116     @Override
117     public void repaint() {
118         panel.getGraphics().drawImage(image, 0,0,null);
119         getGraphics().clearRect(0, 0, getWidth(), getHeight());
120     }
121
122     /**
123      * @brief Sets the title of the main window.
124      * @param title The title to set.
125      */
```

```
126     @Override
127     public void setTitle(String title) {
128         frame.setTitle(title);
129     }
130 
131     /**
132      * @brief Adds a key listener to the main window.
133      * @param ka The key adapter to add.
134      */
135     @Override
136     public void addKeyListener(KeyAdapter ka) {
137         frame.addKeyListener(ka);
138     }
139     // endregion
140 
141 }
```