```
1 import Shape.Bouncable;
 2 import Shape.Factory.BorderedShapeFactory;
 3 import Shape.Factory.FilledShapeFactory;
 4 import Shape.CustomShape;
 5 import Shape.Factory.ShapeFactory;
 6 import View.Displayer;
   import View.MainWindow;
 8
9 import javax.swing.*;
10 import java.awt.event.KeyAdapter;
11 import java.awt.event.KeyEvent;
12 import java.util.LinkedList;
13
14 /**
15
16
                   : Labo 02 : Singleton
    * @Tabo
17
    * @Authors
                   : Slimani Walid & Steiner Jeremiah
18
   \star @Description : This file contains the definition of the Bouncers class,
19
                      represents a program that simulates bouncing shapes.
20
    * @Remarque
                   : /
21
    * @Modification : /
2.2.
23
24
25 public class Bouncers {
26
27
        // region Field
28
        /** List of bouncing shapes. */
29
       private final LinkedList<Bouncable> bouncers;
30
31
        /** Number of shapes to instantiate by each key press. */
32
        private final static int NBR_BY_CLICK = 10;
33
34
        /** Delay for refreshing the display, in milliseconds. 
 */
35
       private final static int REFRESH_DELAY = 10;
36
        // endregion
37
38
       // region Ctor
39
        \star @brief Constructs a new instance of the Bouncers class.
40
41
42
       private Bouncers() {
43
           bouncers = new LinkedList<Bouncable>();
44
45
        // endregion
46
47
       // region Public methode
48
49
         * @brief The main method of the program.
50
         * @param args The command-line arguments.
51
        */
52
       public static void main(String[] args) {
53
           new Bouncers().run();
54
        }
55
57
        * @brief Runs the Bouncers program.
58
59
        public void run() {
60
           MainWindow window = MainWindow.getInstance();
           window.setTitle("Bouncer");
61
62
63
           window.addKeyListener(new KeyAdapter() {
```

```
Bouncers.java
```

```
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
          \star @param nbr The number of shapes to instantiate.
108
109
        private void instantiate(ShapeFactory factory, int nbr)
110
             for (int i = 0; i < nbr; ++i)</pre>
111
112
113
                 bouncers.add(factory.createCircle());
114
                 bouncers.add(factory.createSquare());
115
116
117
         // endregion
118 }
```

```
1 package Shape.Bordered;
3 import Shape.Circle;
 4 import Shape.Renderer.BorderedRenderer;
 6 import java.awt.*;
8 /**
9
    * @Authors : Slimani Walid & Steiner Jeremiah
10
11
    \star @Description : This file contains the definition of the BorderedCircle class.
12
            : Represents a bordered circle shape.
    * @Info
13
    * ------
14
    **/
15
16 public class BorderedCircle extends Circle {
17
      // region Ctor
18
      /**
19
       * @brief Constructs a new instance of the BorderedCircle class.
20
       */
21
      public BorderedCircle() {
22
          super();
23
          renderer = BorderedRenderer.getInstance();
24
25
      // endregion
26
27
      // region Public methode
28
      /**
29
       * @brief Gets the color of the BorderedCircle.
30
       * @return The color of the BorderedCircle.
31
       */
32
       @Override
33
      public Color getColor() {
34
          return Color.GREEN;
35
36
      // endregion
37 }
```

```
1 package Shape.Bordered;
3 import Shape.Renderer.BorderedRenderer;
4 import Shape.Square;
5 import java.awt.*;
6
7
8
   * -----
   * @Authors : Slimani Walid & Steiner Jeremiah
9
   * @Description : This file contains the definition of the BorderedSquare class.
10
11
            : Represents a bordered square shape.
12
13
14
   public class BorderedSquare extends Square {
15
16
       // region Ctor
17
      /**
18
      * @brief Constructs a new instance of the BorderedSquare class.
19
     public BorderedSquare() {
20
21
          super();
22
          renderer = BorderedRenderer.getInstance();
23
24
      // endregion
25
26
      // region Public methode
27
      /**
28
       * @brief Gets the color of the BorderedSquare.
29
       * @return The color of the BorderedSquare.
30
       */
31
      @Override
32
      public Color getColor() {
33
          return Color.RED;
34
35
      // endregion
36 }
```

```
1 package Shape;
3 import java.awt.*;
 4
 5 /**
   * -----
 6
    * @Authors : Slimani Walid & Steiner Jeremiah
 7
8
   * @Description : This file contains the definition of the Bouncable interface.
9
   * @Info : Represents an interface for objects that can bounce.
10
11
    **/
12
13 public interface Bouncable {
14
      // region Public methode
15
      /**
16
      * @brief Draws the object.
17
       */
18
      void draw();
19
     /**
20
21
      * @brief Moves the object.
22
       */
23
      void move();
24
25
     /**
26
      * @brief Gets the color of the object.
27
       * @return The color of the object.
28
29
      Color getColor();
30
31
      /**
32
      * @brief Gets the shape of the object.
33
       * @return The shape of the object.
34
       */
35
      Shape getShape();
36
      // endregion
37 }
```

```
1 package Shape;
3 import java.awt.*;
4 import java.awt.geom.Ellipse2D;
6 /**
7
    * ------
    * @Authors : Slimani Walid & Steiner Jeremiah
8
9
   * @Description : This file contains the definition of the Circle abstract class.
   * @Info : Represents an abstract class for circle shapes.
10
11
12
13
14 public abstract class Circle extends CustomShape {
15
      // region Ctor
      /**
16
17
      * @brief Constructs a new instance of the Circle class.
18
       */
19
     public Circle() {
20
          super();
21
22
      // endregion
23
     // region Public methode
24
      /**
25
26
       * @brief Gets the shape of the circle.
27
       * @return The shape of the circle.
28
29
      @Override
30
      public Shape getShape() {
31
          return new Ellipse2D.Double(position.getX(), position.getY(), size, size);
32
33
      // endregion
34 }
```

```
1 package Shape;
   import Shape.Renderer.Renderer;
 4 import View.Displayer;
 5 import View.MainWindow;
 7
   import java.util.Random;
 8
9
   import static java.lang.Math.max;
10 import static java.lang.Math.min;
11
12 /**
    * -----
13
14
    * @Authors : Slimani Walid & Steiner Jeremiah
15
     * @Description : This file contains the definition of the CustomShape abstract class.
              : Represents an abstract class for custom shapes.
16
    * @Info
17
18
    **/
19
20 public abstract class CustomShape implements Bouncable {
21
       // region Field
22
       /** The display area associated with the custom shape. */
23
       protected Displayer displayer;
24
25
       /** The random number generator. */
26
       private final static Random random = new Random();
27
28
       /** The size of the custom shape. */
29
       protected final int size;
30
31
       /** The movement vector of the custom shape. */
32
       private final Vector2D movement;
33
34
       /** The speed of the custom shape. */
35
       private final double speed;
36
37
       /** The position vector of the custom shape. */
38
       protected Vector2D position;
39
40
       /** The renderer used to draw the custom shape. */
41
       protected Renderer renderer;
42
       // endregion
43
44
       // region Ctor
45
46
        * @brief Constructs a new instance of the CustomShape class.
47
        */
48
       protected CustomShape() {
49
           displayer = MainWindow.getInstance();
50
           size = random.nextInt(5, 50);
51
           movement = new Vector2D(
52
                   random.nextInt(-10, 10),
53
                   random.nextInt(-10, 10)
54
           );
55
           movement.normalize();
56
           speed = random.nextInt(3, 10);
57
58
           // as demo (spawn at center instead of random position as pdf)
59
           position = new Vector2D(
60
                   displayer.getWidth()/2.,
61
                   displayer.getHeight()/2.
62
           );
63
           // if random spawn needed ctrl + '/' to uncomment lines followed:
```

108 }

```
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
         \star @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)</pre>
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)</pre>
99
100
                 movement.swapOnY();
101
                 newPosition = new Vector2D (newPosition.getX(), newPosition.getY() < 0 ? 0 :</pre>
102
                                 min(position.getY() + movement.getY() * speed, maxY));
103
             }
104
105
            position = newPosition;
106
        // endregion
107
```

```
1 package Shape.Factory;
 3 import Shape.Bordered.BorderedCircle;
 4 import Shape.Bordered.BorderedSquare;
 6 /**
    * ------
 7
               : Slimani Walid & Steiner Jeremiah
 8
    * @Authors
 9
    * @Description : This file contains the definition of the BorderedShapeFactory class.
    \star @Info \phantom{a} : Represents a factory for creating bordered shapes.
10
11
12
13
14 public class BorderedShapeFactory extends ShapeFactory{
15
16
       // region Intern Static Class (for Singleton)
17
       /**
18
        \star @brief Represents an inner static class for implementing the Singleton pattern.
19
20
       private static class Instance
21
22
           /** The singleton instance of the BorderedShapeFactory class. */
23
           static final BorderedShapeFactory instance = new BorderedShapeFactory();
24
25
       // endregion
26
27
       // region Public methode
28
29
        * @brief Creates a new bordered circle.
30
        * @return The created bordered circle.
31
        */
32
       @Override
33
       public BorderedCircle createCircle() {
34
           return new BorderedCircle();
35
36
37
       /**
38
       * @brief Creates a new bordered square.
39
        * @return The created bordered square.
40
        */
41
       @Override
42
       public BorderedSquare createSquare() {
43
           return new BorderedSquare();
44
45
46
       /**
47
        * @brief Gets the singleton instance of the BorderedShapeFactory class.
48
        * @return The singleton instance.
49
        */
50
       public static BorderedShapeFactory getInstance() {
51
           return BorderedShapeFactory.Instance.instance;
52
53
       // endregion
54
55 }
```

```
1 package Shape.Factory;
 3 import Shape.Filled.FilledCircle;
 4 import Shape.Filled.FilledSquare;
 6 /**
    * ------
 7
               : Slimani Walid & Steiner Jeremiah
 8
    * @Authors
 9
   * @Description : This file contains the definition of the FilledShapeFactory class.
    \star @Info : Represents a factory for creating filled shapes.
10
11
12
13
14 public class FilledShapeFactory extends ShapeFactory{
15
16
       // region Intern Static Class (for Singleton)
17
       /**
18
       \star @brief Represents an inner static class for implementing the Singleton pattern.
19
20
       private static class Instance
21
22
           /** The singleton instance of the FilledShapeFactory class. */
23
           static final FilledShapeFactory instance = new FilledShapeFactory();
24
25
       // endregion
26
27
       // region Public methode
28
       /**
29
        * @brief Creates a new filled circle.
30
        * @return The created filled circle.
31
        */
32
       @Override
33
       public FilledCircle createCircle() {
           return new FilledCircle();
34
35
36
37
       /**
38
       * @brief Creates a new filled square.
39
        * @return The created filled square.
40
        */
41
       @Override
42
       public FilledSquare createSquare() {
43
           return new FilledSquare();
44
45
46
       /**
        \star @brief Gets the singleton instance of the FilledShapeFactory class.
47
48
        * @return The singleton instance.
49
        */
50
       public static FilledShapeFactory getInstance() {
           return FilledShapeFactory.Instance.instance;
51
52
53
       // endregion
54 }
```

Shape/Factory/ShapeFactory.java

```
1 package Shape.Factory;
3 import Shape.Circle;
4 import Shape. Square;
6 /**
7
   * ------
   * @Authors : Slimani Walid & Steiner Jeremiah
8
9
   * @Description : This file contains the definition of the ShapeFactory abstract class.
   * @Info : Represents an abstract factory for creating shapes.
10
11
    * ------
12
13
14 public abstract class ShapeFactory {
15
      // region Public methode
      /**
16
17
      * @brief Creates a new circle.
18
      * @return The created circle.
19
20
     public abstract Circle createCircle();
21
22
     /**
23
      * @brief Creates a new square.
24
      * @return The created square.
25
      */
26
     public abstract Square createSquare();
27
      // endregion
28 }
```

Shape/Filled/FilledCircle.java

```
1 package Shape.Filled;
3 import Shape.Circle;
4 import Shape.Renderer.FilledRenderer;
6 import java.awt.*;
8 /**
9
    * @Authors : Slimani Walid & Steiner Jeremiah
10
11
    \star @Description : This file contains the definition of the FilledCircle class.
12
            : Represents a filled circle shape.
   * @Info
13
    * ------
14
   **/
15
16 public class FilledCircle extends Circle {
17
      // region Ctor
18
      /**
19
       * @brief Constructs a new instance of the FilledCircle class.
20
       */
21
      public FilledCircle() {
22
          super();
23
          renderer = FilledRenderer.getInstance();
24
25
     // endregion
26
27
      // region Public methode
28
      /**
29
       * @brief Gets the color of the FilledCircle.
       * @return The color of the FilledCircle.
30
31
       */
32
       @Override
33
      public Color getColor() {
34
          return Color.BLUE;
35
36
      // endregion
37 }
```

```
1 package Shape.Filled;
3 import Shape.Renderer.FilledRenderer;
4 import Shape.Square;
5 import java.awt.*;
6
7
8
   * -----
   * @Authors : Slimani Walid & Steiner Jeremiah
9
   \star @Description : This file contains the definition of the FilledSquare class.
10
11
            : Represents a filled square shape.
12
13
14
   public class FilledSquare extends Square {
15
16
       // region Ctor
17
      /**
18
      * @brief Constructs a new instance of the FilledSquare class.
19
     public FilledSquare() {
20
21
          super();
22
          renderer = FilledRenderer.getInstance();
23
24
      // endregion
25
26
      // region Public methode
27
      /**
28
       * @brief Gets the color of the FilledSquare.
29
       * @return The color of the FilledSquare.
30
       */
31
      @Override
32
      public Color getColor() {
33
          return Color.ORANGE;
34
35
      // endregion
36 }
```

```
1 package Shape.Renderer;
 3 import java.awt.*;
 4 import Shape.Bouncable;
 6 /**
 7
    * @Authors : Slimani Walid & Steiner Jeremiah
 8
 9
    * @Description : This file contains the definition of the BorderedRenderer class.
     \star @Info \phantom{a} : Represents a renderer for drawing bordered shapes.
10
11
12
13
14 public class BorderedRenderer implements Renderer{
15
16
        // region Intern Static Class (for Singleton)
17
        /**
18
        \star @brief Represents an inner static class for implementing the Singleton pattern.
19
20
       private static class Instance
21
22
            /** The singleton instance of the BorderedRenderer class. */
23
            static final BorderedRenderer instance = new BorderedRenderer();
24
25
       // endregion
26
27
       // region Public methode
28
       /**
29
        \star @brief Gets the singleton instance of the BorderedRenderer class.
30
         * @return The singleton instance.
31
        */
32
        public static BorderedRenderer getInstance() {
33
           return BorderedRenderer.Instance.instance;
34
35
36
       /**
37
        \star @brief Displays the given shape with a border using the provided graphics context.
38
         * @param g The graphics context.
39
         * @param b The shape to display.
40
        */
41
        @Override
42
       public void display(Graphics2D g, Bouncable b) {
43
           g.setColor(b.getColor());
44
           g.setStroke(new BasicStroke(2));
45
           g.draw(b.getShape());
46
47
        // endregion
48 }
```

46 }

```
1 package Shape.Renderer;
 3 import java.awt.*;
 4 import Shape.Bouncable;
 6 /**
 7
    * @Authors : Slimani Walid & Steiner Jeremiah
 8
 9
    * @Description : This file contains the definition of the FilledRenderer class.
     * @Info : Represents a renderer for drawing filled shapes.
10
11
12
13
14 public class FilledRenderer implements Renderer {
15
        // region Intern Static Class (for Singleton)
16
17
         \star @brief Represents an inner static class for implementing the Singleton pattern.
18
        */
19
       private static class Instance
20
21
            /** The singleton instance of the FilledRenderer class. */
22
            static final FilledRenderer instance = new FilledRenderer();
23
24
       // endregion
25
26
       // region Public methode
27
28
        \star @brief Gets the singleton instance of the FilledRenderer class.
29
        * @return The singleton instance.
30
        */
31
       public static FilledRenderer getInstance() {
32
            return FilledRenderer.Instance.instance;
33
34
35
       /**
36
        \star @brief Displays the given shape filled with color using the provided graphics context.
37
         * @param g The graphics context.
38
         \star @param b The shape to display.
39
        */
40
        @Override
41
        public void display(Graphics2D g, Bouncable b) {
42
           g.setColor(b.getColor());
43
           g.fill(b.getShape());
44
45
        // endregion
```

Shape/Renderer/Renderer.java

```
1 package Shape.Renderer;
3 import Shape.Bouncable;
4
5 import java.awt.Graphics2D;
 6
7
8
   * -----
   * @Authors : Slimani Walid & Steiner Jeremiah
9
   \star @Description : This file contains the definition of the Renderer interface.
10
11
           : Represents an interface for rendering shapes.
12
13
14
15 public interface Renderer {
16
       // region Public methode
17
      /**
18
       * @brief Displays the given shape using the provided graphics context.
19
       * @param g The graphics context.
20
       * @param b The shape to display.
21
       */
22
      void display(Graphics2D g, Bouncable b);
23
      // endregion
24 }
```

```
1 package Shape;
3 import java.awt.*;
4 import java.awt.geom.Rectangle2D;
6 /**
7
   * ------
   * @Authors : Slimani Walid & Steiner Jeremiah
8
9
   * @Description : This file contains the definition of the Square abstract class.
   * @Info : Represents an abstract class for square shapes.
10
    * -----
11
12
13
14 public abstract class Square extends CustomShape {
15
      // region Ctor
      /**
16
17
      * @brief Constructs a new instance of the Square class.
18
      */
19
     public Square() {
20
         super();
21
22
     // endregion
23
    // region Public methode
24
25
      /**
26
      * @brief Gets the shape of the square.
27
      * @return The shape of the square.
28
29
      @Override
30
      public final Shape getShape() {
31
         return new Rectangle2D.Double(position.getX(), position.getY(), size, size);
32
33
      // endregion
34 }
```

```
1 package Shape;
 3 import static java.lang.Math.sqrt;
 4
 5 /**
    * -----
 6
    * @Authors : Slimani Walid & Steiner Jeremiah
 7
 8
    * @Description : This file contains the definition of the Vector2D class.
 9
    * @Info : Represents a 2D vector.
10
11
12
13 public class Vector2D {
14
       // region Field
15
       /** The x-coordinate of the vector. */
16
       private double x;
17
18
       /** The y-coordinate of the vector. */
19
       private double y;
       // endregion
20
21
22
      // region Ctor
23
       /**
24
        * @brief Constructs a new instance of the Vector2D class with specified coordinates.
25
        \star @param x The x-coordinate of the vector.
26
        * @param y The y-coordinate of the vector.
27
28
       public Vector2D(double x, double y) {
29
           this.x = x;
30
           this.y = y;
31
32
       // endregion
33
34
       // region Public methode
35
36
       * @brief Normalizes the vector.
37
        */
38
       public void normalize()
39
40
           double length = sqrt(x * x + y * y);
41
          x /= length;
42
          y /= length;
43
44
45
46
        \star @brief Gets the x-coordinate of the vector.
47
        * @return The x-coordinate of the vector.
48
        */
49
       public double getX() {
50
           return x;
51
52
53
54
        * @brief Gets the y-coordinate of the vector.
55
        * @return The y-coordinate of the vector.
       */
57
       public double getY() {
58
           return y;
59
       }
60
61
62
        * @brief Swaps the x-coordinate of the vector.
63
        */
```

Shape/Vector2D.java

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

```
View/Displayer.java
```

```
1 package View;
 3 import java.awt.Graphics2D;
4 import java.awt.event.KeyAdapter;
 6 /**
    * ------
 7
    \star @Authors \phantom{a} : Slimani Walid & Steiner Jeremiah
 8
9
   * @Description : This file contains the definition of the Displayer interface.
    \star @Info : Represents an interface for displaying graphics.
10
11
    * -----
12
13
14 public interface Displayer
15
16
       // region Public methode
17
      /**
18
       * @brief Gets the width of the display area.
19
       * @return The width of the display area.
20
       */
21
      int getWidth();
22
      /**
23
      * @brief Gets the height of the display area.
24
       * @return The height of the display area.
25
       */
26
     int getHeight();
27
      /**
28
       * @brief Gets the graphics context for drawing on the display area.
29
       * @return The graphics context.
30
       */
      Graphics2D getGraphics();
31
32
      /**
33
       * @brief Repaints the display area.
34
       */
35
      void repaint();
36
      /**
37
       * @brief Sets the title of the display area.
38
       * @param title The title to set.
39
       */
40
      void setTitle(String title);
41
42
       * @brief Adds a key listener to the display area.
43
       * @param ka The key adapter to add.
44
       */
45
      void addKeyListener(KeyAdapter ka);
46
      // endregion
47 }
```

```
1 package View;
 3 import javax.swing.*;
 4 import java.awt.*;
 5 import java.awt.event.ComponentAdapter;
 6 import java.awt.event.ComponentEvent;
   import java.awt.event.KeyAdapter;
 8
9
   /**
    + -----
10
11
    * @Authors
                  : Slimani Walid & Steiner Jeremiah
    \star @Description : This file contains the definition of the MainWindow class.
12
13
             : Represents the main window of the application.
14
     **/
15
16
17
   public class MainWindow implements Displayer {
18
       // region Field
19
        /** The image associated with the main window. */
20
       protected Image image;
21
       /** The JFrame representing the main window. */
2.2.
2.3
       private final JFrame frame;
24
25
       /** The container panel of the main window. */
26
       private final Container panel;
27
       // endregion
28
29
       // region Intern Static Class (for Singleton)
30
31
        * @brief Represents an inner static class for implementing the Singleton pattern.
32
33
       private static class Instance
34
3.5
            /** The singleton instance of the MainWindow class. */
36
           static final MainWindow instance = new MainWindow();
37
38
       // endregion
39
40
       // region Ctor
41
42
        \star @brief Constructs a new instance of the MainWindow class.
43
44
       private MainWindow() {
45
           System.out.println("-- Singleton()");
46
           int size = 500;
47
           Dimension dimension = new Dimension(size, size);
48
49
           frame = new JFrame();
50
51
           Toolkit toolkit = Toolkit.getDefaultToolkit();
52
53
           Dimension screenSize = toolkit.getScreenSize();
           frame.setLocation(new Point((int) screenSize.getWidth() / 4, (int) screenSize.getHeight() /
54
55
           frame.setSize(dimension.width, dimension.height);
56
           frame.setPreferredSize(frame.getSize());
57
58
           panel = frame.getContentPane();
59
           panel.setBackground(Color.LIGHT_GRAY);
60
           image = panel.createImage(dimension.width, dimension.height);
61
62
           frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
63
             frame.addComponentListener(new ComponentAdapter() {
 64
 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() {
             panel.getGraphics().drawImage(image, 0,0,null);
118
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
          */
```

View/MainWindow.java

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