Graphics Programming – Interactivity and Animation

Zbigniew Dendzik
Institute of Physics, University of Silesia

2025

Introduction

This set illustrates the use of the `MouseMotionListener` interface to handle mouse events (interactivity) and the use of an additional thread to implement animation.

Example I

Below you will find an example illustrating the use of an object-oriented approach to interactive graphics programming. Identify the class members, analyse the purpose of their use and the syntax, and analyse the documentation for each of the library classes used. Answer the question of why and how the inheritance mechanism was used here. Prepare an appropriate compilation unit and test the example.

```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4  import java.awt.geom.*;
5
6  class Ball extends Ellipse2D.Float
7  {
8     Board p;
9     int dx,dy;
10
11    Ball(Board p,int x,int y,int dx,int dy)
12    {
13       this.x=x;
14       this.y=y;
15       this.width=10;
16       this.height=10;
17
18       this.p=p;
19       this.dx=dx;
20       this.dy=dy;
21    }
22
23    void nextStep()
24    {
25       x+=dx;
```

```
26          y+=dy;
27
28          if(getMinX()<0 || getMaxX()>p.getWidth())  dx=-dx;
29          if(getMinY()<0 || getMaxY()>p.getHeight()) dy=-dy;
30
31          p.repaint();
32          Toolkit.getDefaultToolkit().sync();
33      }
34  }
35
36  class BallEngine extends Thread
37  {
38      Ball a;
39
40      BallEngine(Ball a)
41      {
42          this.a=a;
43          start();
44      }
45
46      public void run()
47      {
48          try
49          {
50              while(true)
51              {
52                  a.nextStep();
53                  sleep(15);
54              }
55          }
56          catch(InterruptedException e){}
57      }
58  }
59
60  class Paddle extends Rectangle2D.Float
61  {
62      Paddle(int x)
63      {
64          this.x=x;
65          this.y=170;
66          this.width=60;
67          this.height=10;
68      }
69
70      void setX(int x)
71      {
72          this.x=x;
73      }
74  }
75
76  class Board extends JPanel implements MouseMotionListener
```

```
77  {
78      Paddle b;
79      Ball a;
80      BallEngine s;
81
82      Board()
83      {
84          super();
85          addMouseMotionListener(this);
86
87          b=new Paddle(100);
88          a=new Ball(this,100,100,1,1);
89          s=new BallEngine(a);
90      }
91
92      public void paintComponent(Graphics g)
93      {
94          super.paintComponent(g);
95          Graphics2D g2d=(Graphics2D)g;
96
97          g2d.fill(a);
98          g2d.fill(b);
99      }
100
101     public void mouseMoved(MouseEvent e)
102     {
103         b.setX(e.getX()-50);
104         repaint();
105     }
106
107     public void mouseDragged(MouseEvent e)
108     {
109
110     }
111 }
112
113 public class Program
114 {
115     public static void main(String[] args)
116     {
117         javax.swing.SwingUtilities.invokeLater(new Runnable()
118         {
119             public void run()
120             {
121                 Board p;
122                 p=new Board();
123
124                 JFrame jf=new JFrame();
125                 jf.add(p);
126
127                 jf.setTitle("Graphics test");
```

```
128                 jf.setSize(400,370);
129                 jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
130                 jf.setVisible(true);
131             }
132         });
133     }
134 }
```

Listing 1: Program.java

Ex. 9.1

Extend the above example by adding bouncing of the ball from the paddle. For this purpose, use the documentation of the appropriate library classes (`Rectangle2D.Float` and `Ellipse2D.Float`) – find the appropriate methods and use them to detect collisions between these objects.

Ex. 9.2

Extend the example by adding bricks to the board. To do this, you can write a new class, for example `Brick`, extending the `Rectangle2D.Float` class from the Java2D package. References to objects representing bricks can be stored in an appropriate array or using the library implementation of the `java.util.ArrayList` list (the first solution is faster). Consider how to optimise the collision detection algorithm of the ball with a large number of bricks.

Ex. 9.3

Implement operations for breaking bricks and counting points. As before, collision detection can be implemented using appropriate methods of the `Ball` or `Brick` classes, which these classes inherited from their base classes (`Rectangle2D.Float` and `Ellipse2D.Float`).

Ex. 9.4

Modify the ball reflection operation from the paddle so that the reflection occurs at a sharper angle if the ball is not reflected by the central part of the paddle (remember to rescale the velocity vector so that the reflection does not change the absolute value of the speed, i.e. does not change the kinetic energy of the ball).

Ex. 9.5

Implement a scoring system and extend the example by adding colours, graphics or photos. You can also add sound effects.

Ex. 9.6

Extend the example by adding several boards corresponding to successive levels with increasing difficulty. Add a system of obstacles and bonuses.

Example II

Working in pairs, design an arcade game according to your own idea. Prepare a list of classes, define the relationships between them and the methods you would have to implement in each of them.

Ex. 9.7

Using your skills in object-oriented design and interactive graphics programming, design and write an implementation of a traffic jam simulation. You can use a cellular automaton operating according to the following rules: - the road is a sequence of cells, - each vehicle occupies one cell, - each cell contains at most one vehicle, - the vehicle speed is in the range 0–5 cells per time step, - if the vehicle has speed 5, then in each time step it accelerates by 1 until it reaches speed 5, - if the distance to the preceding vehicle is d, the vehicle slows down to d–1 to avoid collision, - with some probability, some cars slow down for no external reason.

Note that traffic jams can arise without any external cause (obstacle on the road), can persist long after the vehicle that caused it has left the jam area, and can propagate both in the direction of traffic and in the opposite direction. Perform simulations for different values of road load and different values of the probability that a given vehicle will slow down without any external reason.