

```
1:  /*
2:
3:  Jared Dyreson
4:  CWID: 889546529
5:  Cursor.java ->
6:  csrc_compile: TRUE
7:
8:  */
9:
10: import java.awt.*;
11:
12: public class Cursor {
13:     // positions in R^2
14:     private int X = 0, Y = 0;
15:
16:     // our what our cursor looks like and what kind of paint color it uses
17:     private Color cursor_color = Color.YELLOW;
18:     private Color paint_color = Color.WHITE;
19:     // painting flag
20:     private boolean space_is_marked = false;
21:
22:     // getters
23:     public int get_x(){ return this.X; }
24:     public int get_y(){ return this.Y; }
25:     public Color get_color(){ return this.cursor_color; }
26:     public Color get_paint_color(){ return this.paint_color; }
27:     public boolean get_space_flag(){ return space_is_marked; }
28:
29:     // setters
30:     public void set_position(int x, int y){
31:         this.X = x;
32:         this.Y = y;
33:     }
34:
35:     public void set_color(Color new_color){
36:         this.cursor_color = new_color;
37:     }
38:
39:     public void set_paint_color(Color new_color){
40:         this.paint_color = new_color;
41:     }
42:
43:     public void toggle_marker(boolean value){
44:         this.space_is_marked = value;
45:     }
46:     // our constructor is a glorified setter
47:     public Cursor(int x, int y){
48:         this.set_position(x, y);
```

```
49:         }  
50:     }
```

```
1:  /*
2:
3:  Jared Dyreson
4:  CWID: 889546529
5:  JDemoKey.java ->
6:  csrc_compile: TRUE
7:
8:  */
9:
10: import javax.swing.*;
11: import java.awt.*;
12: import java.awt.event.*;
13:
14: public class JDemoKey extends JFrame implements KeyListener{
15:
16:     // the boundaries for the checker board
17:     private final int FRAME_HEIGHT = 500, FRAME_WIDTH = 500, ROWS = 16, COLS = 16;
18:
19:     // we need two parallel arrays to keep track of the colors on the board
20:     // this one keeps track of the colors for the pane
21:     private JPanel[][] tpanel = new JPanel[ROWS][COLS];
22:     // this keeps track of what colors the tpanel object reflects
23:     private Color[][] color_panel = new Color[ROWS][COLS];
24:     // the main container for the checker board layout
25:     private JPanel pane = new JPanel(new GridLayout(ROWS, COLS, 2, 2));
26:
27:     private Color w = Color.WHITE;
28:     private Color b = Color.BLACK;
29:     // so we can keep track of what colors are around us
30:     private Color previous_color, current_color;
31:
32:     private Cursor cursor = new Cursor(0, 0);
33:
34:     // setters
35:     public void set_previous_color(Color c){ this.previous_color = c; }
36:     public void set_current_color(Color c){ this.current_color = c; }
37:
38:     // getters
39:     public Color get_previous_color(){ return this.previous_color; }
40:     public Color get_current_color(){ return this.current_color; }
41:
42:     public JDemoKey(){
43:         super("MS Paint");
44:
45:         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
46:         this.setSize(FRAME_HEIGHT, FRAME_WIDTH);
47:         this.setLayout(new BorderLayout());
48:     }
```

```
49:         for(int x = 0; x < COLS; ++x){
50:             for(int y = 0; y < ROWS; ++y){
51:                 // make the grid and corresponding parallel array with all white tiles
52:                 tpanel[x][y] = new JPanel();
53:                 color_panel[x][y] = this.w;
54:                 this.pane.add(tpanel[x][y]);
55:                 tpanel[x][y].setBackground(color_panel[x][y]);
56:             }
57:         }
58:
59:         this.add(pane, BorderLayout.CENTER);
60:
61:         addKeyListener(this);
62:         // initialize the cursor
63:         this.tpanel[0][0].setBackground(this.cursor.get_color());
64:     }
65:
66:     public void set_position(int x, int y, boolean coloring) throws IndexOutOfBoundsException{
67:
68:         // initial position
69:         int x_naught = this.cursor.get_x();
70:         int y_naught = this.cursor.get_y();
71:
72:         // move the cursor over
73:         this.tpanel[y][x].setBackground(this.cursor.get_color());
74:         // update the current and previous colors
75:         this.set_previous_color(this.color_panel[y_naught][x_naught]);
76:         this.set_current_color(this.color_panel[y][x]);
77:
78:         // we pass in a boolean flag to allow us to use the same function
79:         // with different conditions
80:
81:         if(coloring){
82:             // erase black tiles
83:             if(this.previous_color == Color.BLACK){
84:                 this.color_panel[y_naught][x_naught] = this.w;
85:                 this.tpanel[y_naught][x_naught].setBackground(this.w);
86:                 this.previous_color = Color.WHITE;
87:             }
88:             // mark tiles
89:             else if(this.current_color == Color.WHITE){
90:                 this.color_panel[y_naught][x_naught] = this.b;
91:                 this.tpanel[y_naught][x_naught].setBackground(this.b);
92:                 this.previous_color = Color.BLACK;
93:             }
94:         }
95:         // just continue as normal
96:         else{
```

```
97:         this.tpanel[y_naught][x_naught].setBackground(this.previous_color);
98:     }
99:     // update cursor location
100:    this.cursor.set_position(x, y);
101:    // make the pane reload
102:    this.pane.revalidate();
103:    this.pane.repaint();
104:    // we are only allowing for one pixel being marked
105:    // at a time
106:    this.cursor.toggle_marker(false);
107:
108: }
109:
110: @Override
111: public void keyTyped(KeyEvent event){
112:     // only executes when char is typed
113:     char c = event.getKeyChar();
114: }
115:
116: @Override
117: public void keyPressed(KeyEvent event){
118:
119:     int key_code = event.getKeyCode();
120:
121:     // get the position of the cursor
122:     int x = this.cursor.get_x();
123:     int y = this.cursor.get_y();
124:     // we have a try catch block to basically ignore the fact that we bump against a wall
125:     // the catch does nothing
126:     try{
127:         // if the space key is pressed, we can indicate we want to draw
128:         if(key_code == KeyEvent.VK_SPACE){
129:             this.cursor.toggle_marker(true);
130:         }
131:         // -/+ are flipped because of the frame of reference
132:         // - means we want to go up/right because the indexes start from 0
133:         // + means we want to go down/left because the indexes start from 0
134:
135:         if(key_code == KeyEvent.VK_UP){
136:             this.set_position(x, y-1, this.cursor.get_space_flag());
137:         }
138:         else if(key_code == KeyEvent.VK_DOWN){
139:             this.set_position(x, y+1, this.cursor.get_space_flag());
140:         }
141:         else if(key_code == KeyEvent.VK_LEFT){
142:             this.set_position(x-1, y, this.cursor.get_space_flag());
143:         }
144:         else if(key_code == KeyEvent.VK_RIGHT){
```

```
145:                                     this.set_position(x+1, y, this.cursor.get_space_flag());
146:                                     }
147:                                     }
148:                                     catch(Exception error){}
149:     }
150:
151:     @Override
152:     public void keyReleased(KeyEvent event){ return; }
153:     public static void main(String[] args){
154:         // Auto generated with caffeine and autovt@.service
155:         JDemoKey j = new JDemoKey();
156:         j.setVisible(true);
157:     }
158: }
```

