

2023/2024(1)
EF234302 Object Oriented Programming

Lecture #8b

Event Handling & Inner Class

Misbakhul Munir **IRFAN SUBAKTI**

司馬伊凡

Мисбакхул Мунир **Ирфан Субакти**

Painting

- A `JPanel` has a method:

```
void paintComponent(Graphics g)
```

using which one can define painting of arbitrary graphical shapes

- Library operations from `Graphics` class:

```
g.drawLine(x1, y1, x2, y2);
```

```
g.drawRect(x, y, xSize, ySize);
```

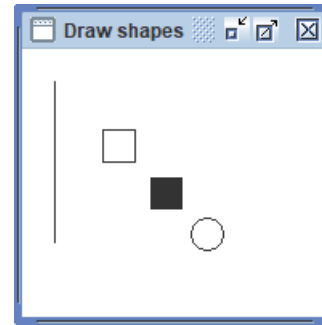
```
g.fillRect(x, y, xSize, ySize);
```

```
g.drawOval(x, y, xSize, ySize);
```

etc.

Painting (continued)

```
MyPainting.java x
1 import java.awt.*;
2 import javax.swing.*;
3 public class MyPainting extends JPanel {
4     public void paintComponent(Graphics g) {
5         int x1, y1, x2, y2;
6         x1 = y1 = x2 = 20;
7         y2 = 120;
8         g.drawLine(x1, y1, x2, y2);
9
10        int x, y;
11        x = y = 50;
12        int xSize, ySize;
13        xSize = ySize = 20;
14        g.drawRect(x, y, xSize, ySize);
15
16        x = y = 80;
17        g.fillRect(x, y, xSize, ySize);
18
19        x = y = 105;
20        g.drawOval(x, y, xSize, ySize);
21    }
22    public static void main(String[] args) {
23        JFrame.setDefaultLookAndFeelDecorated(true);
24        JFrame frame = new JFrame("Draw shapes");
25        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26        frame.setBackground(Color.white);
27        frame.setSize(200, 200);
28        MyPainting panel = new MyPainting();
29        frame.add(panel);
30        frame.setVisible(true);
31    }
32 }
```

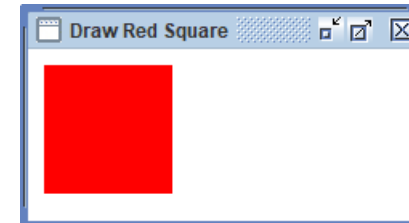
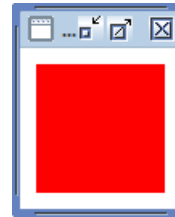


Colour

- Colour is expressed in terms of **RGB** (**R**ed **G**reen **B**lue) values: An integer between 0 and 255 for each of **R**ed, **G**reen and **B**lue pigments
- The higher the number, the higher the colour → think of as brighter/darker
- **Black** – `new Color(0, 0, 0);`
- **White** – `new Color(255, 255, 255);`
- **Bright red** – `new Color(255, 0, 0);`
- `g.setColor` method sets the colour for future painting operations

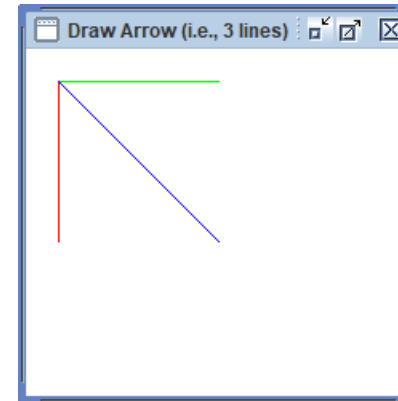
Red Square: Drawing

```
MyRedSquare.java x
1 import java.awt.*;
2 import javax.swing.*;
3 public class MyRedSquare extends JPanel {
4     public MyRedSquare() {
5         setPreferredSize(new Dimension(100, 100));
6         setBackground(Color.white);
7     }
8     public void paintComponent(Graphics g) {
9         super.paintComponent(g); // Mandatory!
10        g.setColor(new Color(255, 0, 0));
11        g.fillRect(10, 10, 80, 80);
12    }
13    public static void main(String[] args) {
14        JFrame.setDefaultLookAndFeelDecorated(true);
15        JFrame frame = new JFrame("Draw Red Square");
16        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17        // We don't use: frame.setSize(250, 250); in here
18        MyRedSquare panel = new MyRedSquare();
19        frame.add(panel);
20        frame.pack();
21        frame.setVisible(true);
22    }
23 }
```



Arrow: Drawing

```
MyPainting2.java x
1 import java.awt.*;
2 import javax.swing.*;
3 public class MyPainting2 extends JPanel {
4     public void paintComponent(Graphics g) {
5         // Vertical line
6         g.setColor(Color.red);
7         g.drawLine(20, 20, 20, 120);
8         // Horizontal line
9         g.setColor(Color.green);
10        g.drawLine(20, 20, 120, 20);
11        // Diagonal line
12        g.setColor(Color.blue);
13        g.drawLine(20, 20, 120, 120);
14    }
15    public static void main(String[] args) {
16        JFrame.setDefaultLookAndFeelDecorated(true);
17        JFrame frame = new JFrame("Draw Arrow (i.e., 3 lines)");
18        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19        frame.setBackground(Color.white);
20        frame.setSize(250, 250);
21        MyPainting2 panel = new MyPainting2();
22        frame.add(panel);
23        frame.setVisible(true);
24    }
25 }
```



Who calls `paintComponent`?

- We never call it directly
- Java run-time system calls it whenever it needs to paint the panel, e.g., when the user restores a window or moves it
- When a panel's state changes and we want to update its display, we call the method

```
void repaint()
```

which will eventually call `paintComponent`

Event model

- An *event* is any occurrence an application might want to respond to, e.g.,
 - User clicks the mouse on a button
 - User moves the mouse
 - User enters text into a text-field
- Java event model is a method of allowing our program to respond to events
- Java event model is based on classifying events into different types

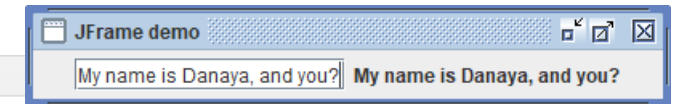
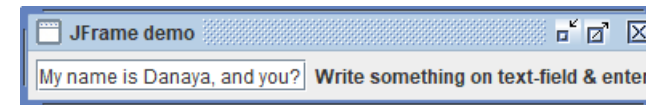
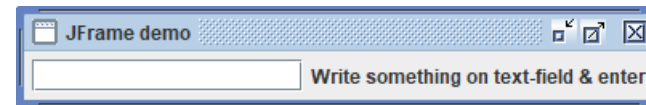
Event types

- We will consider 4 (four) types of events, and explain how to write a panel to respond to each kind
- A panel can also respond to more than one type of event
- Major even types
 - Action events → button click
 - Item events → click check-box
 - Mouse events → mouse click
 - Mouse motion event → mouse moves in a panel

Reading text from text field

- Recall how a panel was written to respond to the event of a user entering text in a text-field, and then press the enter/return key

```
MyPanel4.java x
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class MyPanel4 extends JPanel
4     implements ActionListener { // Declare panel listens
5         // to this type of event
6     JTextField t = new JTextField(15);
7     JLabel l;
8     public MyPanel4() {
9         add(t);
10        t.addActionListener(this); // "Register" with text field
11        l = new JLabel("Write something on text-field & enter");
12        add(l);
13    }
14    public void actionPerformed(ActionEvent e) {
15        // Define required method
16        l.setText(t.getText());
17    }
18 }
```



```
MyPanel4Test.java x
1 import javax.swing.*;
2 public class MyPanel4Test {
3     public static void main(String[] args) {
4         JFrame.setDefaultLookAndFeelDecorated(true);
5         JFrame frame = new JFrame("JFrame demo");
6         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7         MyPanel4 panel = new MyPanel4();
8         frame.setContentPane(panel);
9         frame.pack();
10        frame.setVisible(true);
11    }
12 }
```

Listening to events

- For a program to listen to any of the types of events, it must do these three things
 - Declare itself to listen to that type of event
 - Register with any components that can trigger the event → if the event is a mouse event, it doesn't need to be registered
 - Define method(s) required to listen to that type of event
- The following slides show how to do this for each type of event

Action events

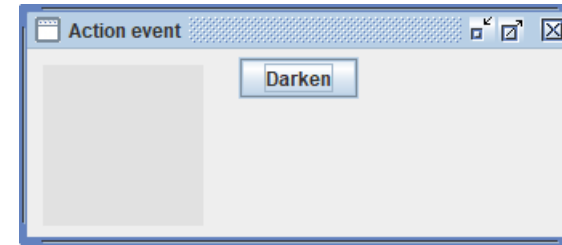
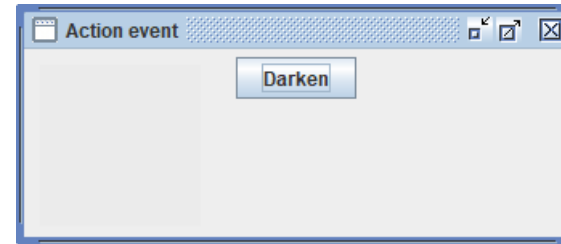
- **Declare panel:** `implements ActionListener`
- **Register component:** `component.addActionListener(this);`
- **Required methods:**

```
public void actionPerformed(ActionEvent a)
```

- **Action events are:**
 - User clicks on button
 - User hits the enter/return key in text-field
- E.g., the following program respond to button click by drawing a rectangle in a darker grey

Action events: Example

```
MyActionEvent.java x
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class MyActionEvent extends JPanel implements ActionListener {
5     JButton darken = new JButton("Darken");
6     int red = 255, green = 255, blue = 255;
7     public MyActionEvent() {
8         add(darken);
9         darken.addActionListener(this);
10    }
11    public void paintComponent(Graphics g) {
12        super.paintComponent(g);
13        g.setColor(new Color(red, green, blue));
14        g.fillRect(10, 10, 100, 100);
15    }
16    public void actionPerformed(ActionEvent e) {
17        red = red - 10;
18        green = green - 10;
19        blue = blue - 10;
20        repaint();
21    }
22    public static void main(String[] args) {
23        JFrame.setDefaultLookAndFeelDecorated(true);
24        JFrame frame = new JFrame("Action event");
25        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26        frame.setSize(350, 150);
27        MyActionEvent panel = new MyActionEvent();
28        frame.add(panel);
29        frame.setVisible(true);
30    }
31 }
```



Item events

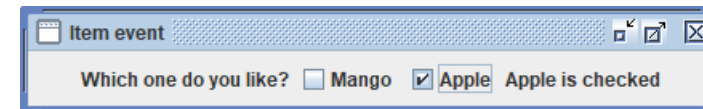
- **Declare panel:** `implements ItemListener`
- **Register component:** `component.addItemListener(this);`
- **Required methods:**

```
public void itemStateChanged(ItemEvent e)
```

- **Item events are:**
 - User clicks on check-box

Item events: Example

```
MyItemEvent.java x
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class MyItemEvent extends JPanel implements ItemListener {
4     JLabel label = new JLabel("Which one do you like?");
5     JCheckBox cb1 = new JCheckBox("Mango");
6     JCheckBox cb2 = new JCheckBox("Apple");
7     JLabel explanation = new JLabel("You haven't check anything");
8     public MyItemEvent() {
9         add(label);
10        add(cb1);
11        add(cb2);
12        add(explanation);
13        cb1.addItemListener(this);
14        cb2.addItemListener(this);
15    }
16    public void itemStateChanged(ItemEvent e) {
17        JCheckBox cb3 = (JCheckBox) e.getItemSelectable();
18        if (cb3.isSelected()) {
19            explanation.setText(cb3.getText() + " is checked");
20        } else {
21            explanation.setText(cb3.getText() + " is unchecked");
22        }
23    }
24    public static void main(String[] args) {
25        JFrame.setDefaultLookAndFeelDecorated(true);
26        JFrame frame = new JFrame("Item event");
27        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28        frame.setSize(350, 150);
29        MyItemEvent panel = new MyItemEvent();
30        frame.add(panel);
31        frame.pack();
32        frame.setVisible(true);
33    }
34 }
```



Mouse motion events

- **Declare panel:** `implements MouseMotionListener`
- **Register component:** no component to register. Just write:

```
addMouseMotionListener(this);
```

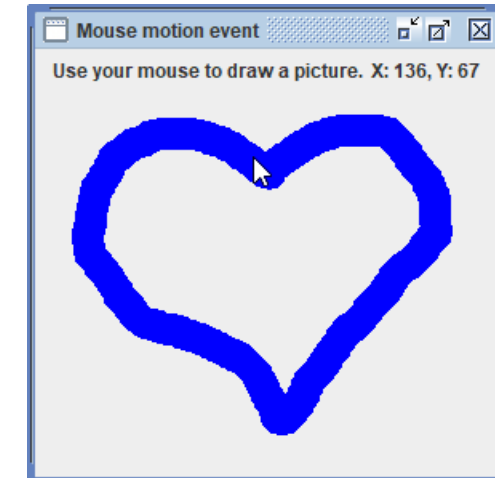
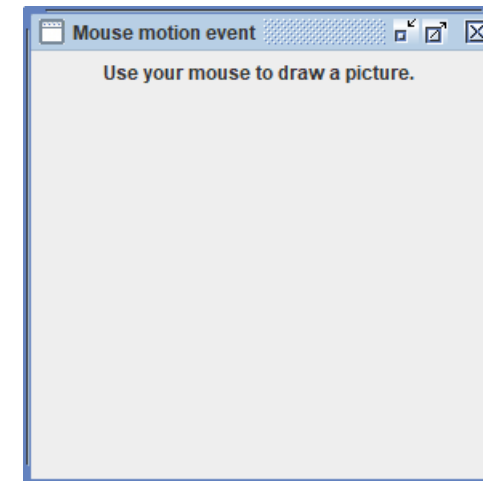
- **Required methods:**

```
public void mouseMoved(MouseEvent e)
```

```
public void mouseDragged(MouseEvent e)
```


Mouse motion events: Example

```
MyMouseMotion.java x
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 public class MyMouseMotion extends JPanel implements MouseMotionListener {
5     JLabel label = new JLabel("Use your mouse to draw a picture.");
6     JLabel xy = new JLabel();
7     public MyMouseMotion() {
8         add(label);
9         add(xy);
10        addMouseListener(this);
11    }
12    public void mouseDragged(MouseEvent e) {
13        xy.setText("X: " + e.getX() + ", Y: " + e.getY());
14        Graphics g = getGraphics();
15        g.setColor(Color.blue);
16        g.fillOval(e.getX(), e.getY(), 20, 20);
17    }
18    public void mouseMoved(MouseEvent e) {
19        xy.setText("X: " + e.getX() + ", Y: " + e.getY());
20    }
21    public static void main(String[] args) {
22        JFrame.setDefaultLookAndFeelDecorated(true);
23        JFrame frame = new JFrame("Mouse motion event");
24        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25        frame.setSize(300, 300);
26        MyMouseMotion panel = new MyMouseMotion();
27        frame.add(panel);
28        frame.setVisible(true);
29    }
30 }
```



Mouse events

- **Declare panel:** `implements MouseListener`
- **Register component:** no component to register. Just write:
`addMouseListener(this);`
- **Required methods:**

```
public void mousePressed(MouseEvent e)
public void mouseReleased(MouseEvent e)
public void mouseEntered(MouseEvent e)
public void mouseExited(MouseEvent e)
public void mouseClicked(MouseEvent e)
```
- **Example:**
 - Darken box when mouse is clicked

Mouse events: Example

```
MyMouseEvent.java x
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class MyMouseEvent extends JPanel implements MouseListener {
5     JLabel label = new JLabel("Click for darkening the rectangle");
6     int red = 255, green = 255, blue = 255;
7     public MyMouseEvent() {
8         add(label);
9         addMouseListener(this);
10    }
11    public void paintComponent(Graphics g) {
12        super.paintComponent(g);
13        g.setColor(new Color(red, green, blue));
14        g.fillRect(10, 30, 100, 100);
15    }
16    @Override
17    public void mousePressed(MouseEvent e) {}
18    @Override
19    public void mouseReleased(MouseEvent e) {}
20    @Override
21    public void mouseEntered(MouseEvent e) {}
22    @Override
23    public void mouseExited(MouseEvent e) {}
24    @Override
25    public void mouseClicked(MouseEvent e) {
26        red = red - 10;
27        green = green - 10;
28        blue = blue - 10;
29        repaint();
30    }
}
```

```
public static void main(String[] args) {
    JFrame.setDefaultLookAndFeelDecorated(true);
    JFrame frame = new JFrame("Mouse event");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(350, 300);
    MyMouseEvent panel = new MyMouseEvent();
    frame.add(panel);
    frame.setVisible(true);
}
```



An event: Getting information about

- We can always get information about a component by using instance methods from the components class, e.g., in `JTextField`: `String getText ()`
- When an event occurs, you can get information from the event object that is passed as argument `e` to the event's method
- The type of information depends upon the type of event

An event: Getting information about (cont'd)

- Action events and item events. Use

```
e.getSource()
```

to find out which text-field or button or check-box was the source of the event

- Mouse events. Use

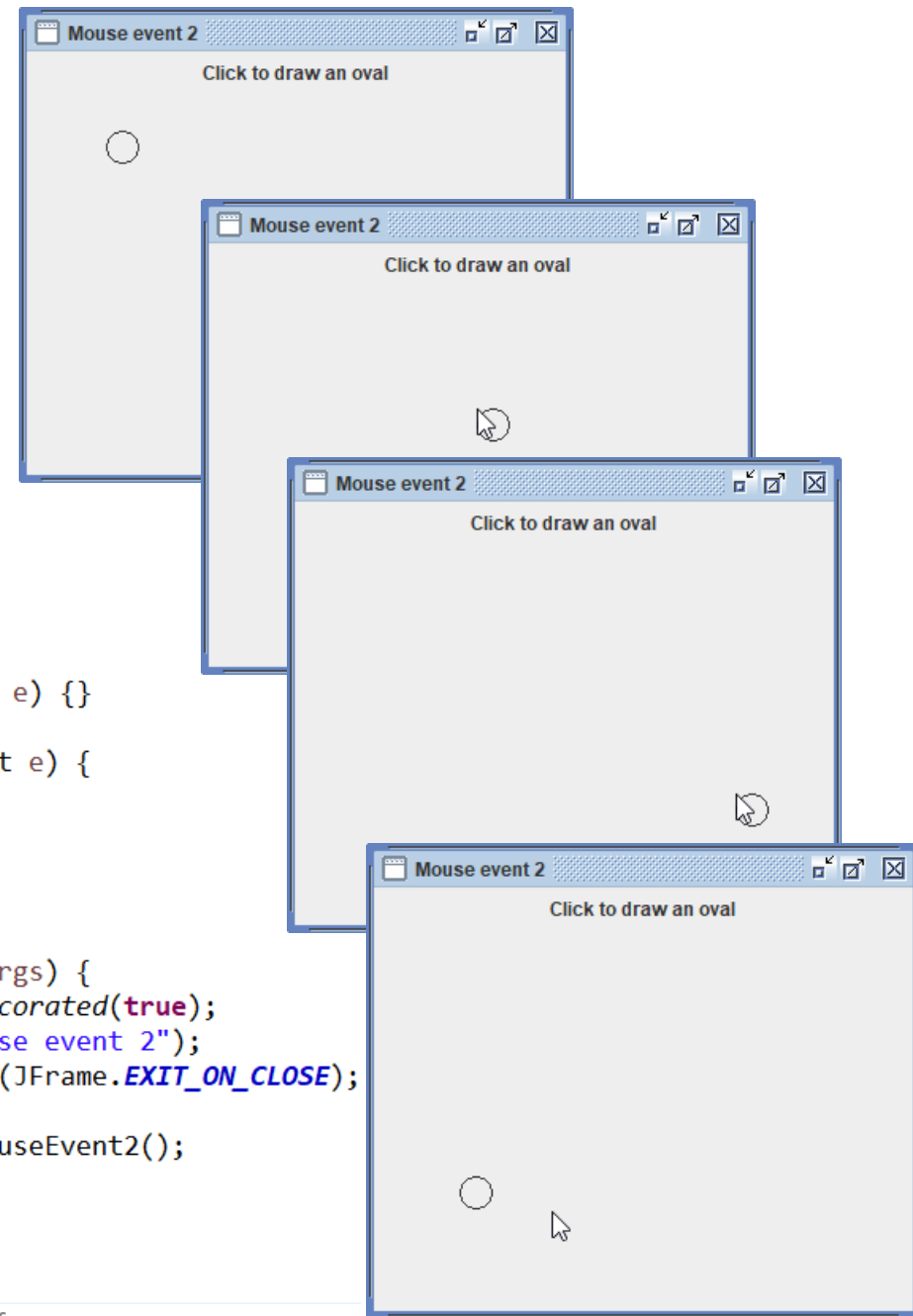
```
e.getPoint()
```

to find the location of the mouse. This returns a `Point` object. Use `.x` and `.y` to find `x` and `y` coordinates.

Mouse event: Other example

- Place a small circle wherever mouse is clicked

```
MyMouseEvent2.java x
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class MyMouseEvent2 extends JPanel implements MouseListener {
5     JLabel label = new JLabel("Click to draw an oval");
6     int x = 50, y = 50;
7     public MyMouseEvent2() {
8         add(label);
9         addMouseListener(this);
10    }
11    public void paintComponent(Graphics g) {
12        super.paintComponent(g);
13        g.drawOval(x, y, 20, 20);
14    }
15    @Override
16    public void mousePressed(MouseEvent e) {}
17    @Override
18    public void mouseReleased(MouseEvent e) {}
19    @Override
20    public void mouseEntered(MouseEvent e) {}
21    @Override
22    public void mouseExited(MouseEvent e) {}
23    @Override
24    public void mouseClicked(MouseEvent e) {
25        Point p = e.getPoint();
26        x = p.x;
27        y = p.y;
28        repaint();
29    }
30    public static void main(String[] args) {
31        JFrame.setDefaultLookAndFeelDecorated(true);
32        JFrame frame = new JFrame("Mouse event 2");
33        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
34        frame.setSize(350, 300);
35        MyMouseEvent2 panel = new MyMouseEvent2();
36        frame.add(panel);
37        frame.setVisible(true);
38    }
39 }
```

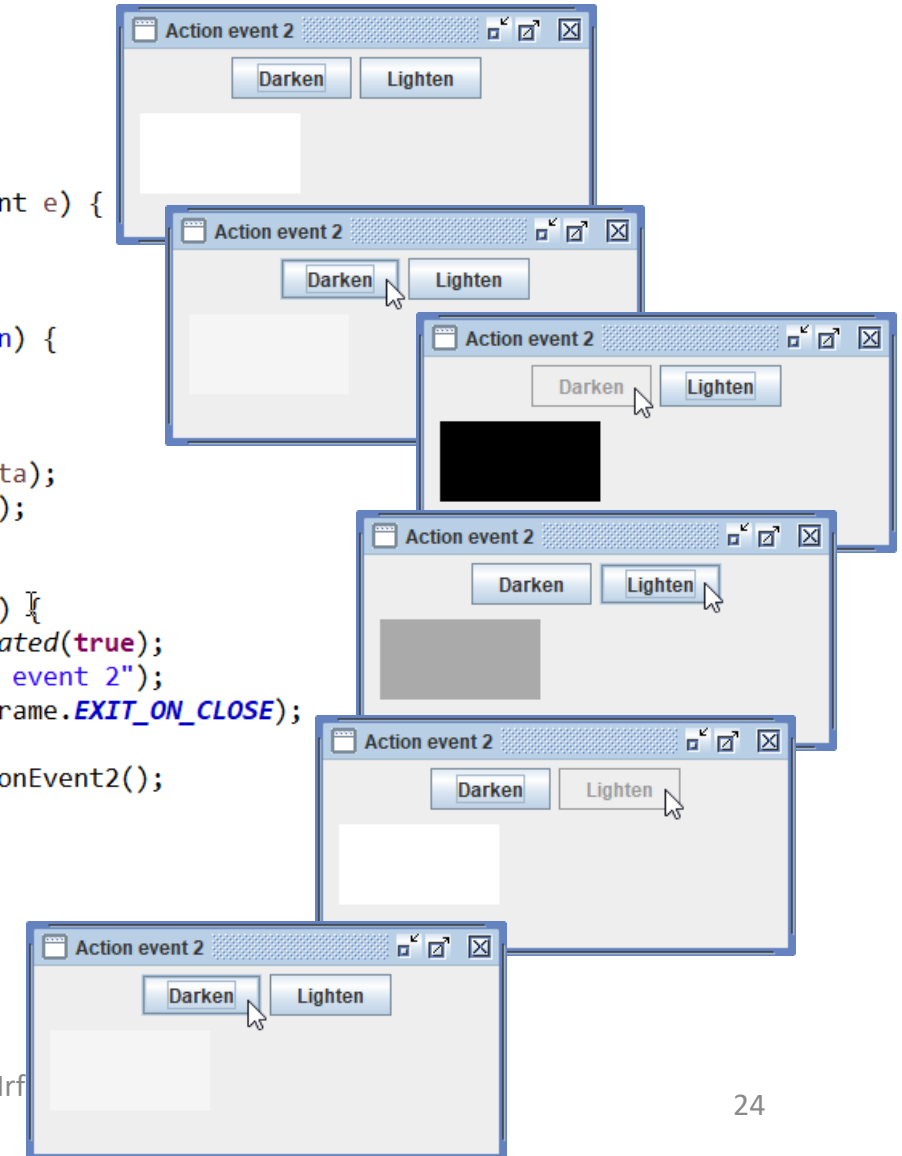


Which component caused the event?

- There can be multiple event-producing components
- Use `getSource()` to figure out which one caused the event
- E.g., the following program can either lighten or darken the rectangle
 - However, if it's totally dark, then it cannot be darkened anymore!
 - Likewise, if it's totally light, then it cannot be lightened anymore!

Which component caused the event? (cont'd)

```
MyActionEvent2.java x
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class MyActionEvent2 extends JPanel implements ActionListener {
5     JButton darken = new JButton("Darken"),
6         lighten = new JButton("Lighten");
7     int red = 255, green = 255, blue = 255;
8     public MyActionEvent2() {
9         add(darken);
10        darken.addActionListener(this);
11        add(lighten);
12        lighten.addActionListener(this);
13    }
14    public void paintComponent(Graphics g) {
15        super.paintComponent(g);
16        g.setColor(new Color(red, green, blue));
17        g.fillRect(10, 40, 100, 50);
18    }
19    int checkBoundary(int x) {
20        if (x < 0) {
21            x = 0;
22            darken.setEnabled(false);
23        } else {
24            darken.setEnabled(true);
25        }
26        if (x > 255) {
27            x = 255;
28            lighten.setEnabled(false);
29        } else {
30            lighten.setEnabled(true);
31        }
32        return x;
33    }
34    public void actionPerformed(ActionEvent e) {
35        int delta = 0;
36        if (e.getSource() == lighten) {
37            delta = 10;
38        } else if (e.getSource() == darken) {
39            delta = -10;
40        }
41        red = checkBoundary(red + delta);
42        green = checkBoundary(green + delta);
43        blue = checkBoundary(blue + delta);
44        repaint();
45    }
46    public static void main(String[] args) {
47        JFrame.setDefaultLookAndFeelDecorated(true);
48        JFrame frame = new JFrame("Action event 2");
49        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50        frame.setSize(300, 150);
51        MyActionEvent2 panel = new MyActionEvent2();
52        frame.add(panel);
53        frame.setVisible(true);
54    }
55 }
```

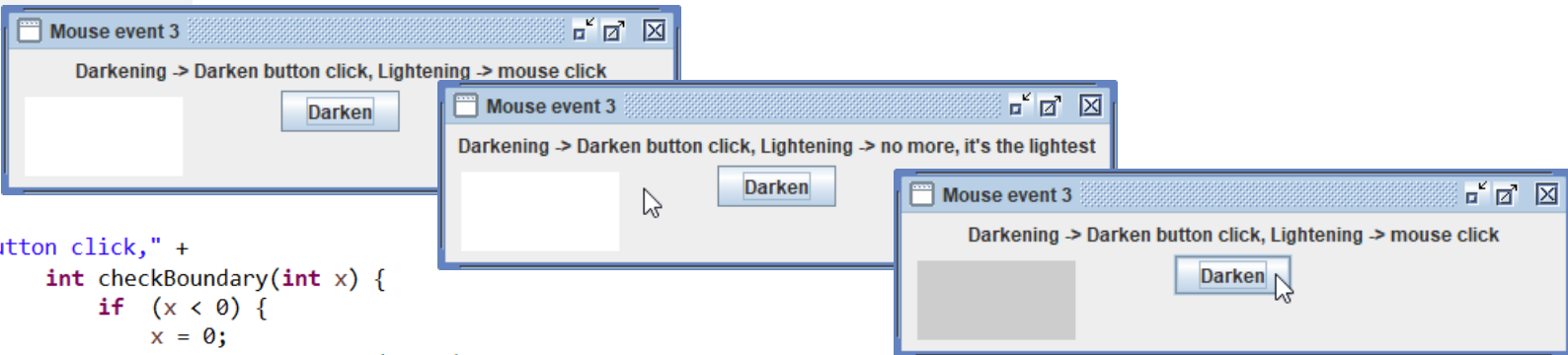


Events: Catching different types of

- A program can catch events of any or all types
 - It just declares that it wants to catch events of those types, registers all components, and defines all required operations
- E.g., the following program responds to button click by darkening box, and to mouse click by lightening it
 - However, if it's totally dark, then it cannot be darkened anymore!
 - Likewise, if it's totally light, then it cannot be lightened anymore!

Events: Catching different types of (cont'd)

```
MyMouseEvent3.java x
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class MyMouseEvent3 extends JPanel implements
5     ActionListener, MouseListener {
6     JButton darken = new JButton("Darken");
7     int red = 255, green = 255, blue = 255;
8     JLabel label = new JLabel("Darkening -> Darken button click," +
9         " Lightning -> mouse click");
10    public MyMouseEvent3() {
11        add(label);
12        add(darken);
13        darken.addActionListener(this);
14        addMouseListener(this);
15    }
16    public void paintComponent(Graphics g) {
17        super.paintComponent(g);
18        g.setColor(new Color(red, green, blue));
19        g.fillRect(10, 30, 100, 50);
20    }
21    int checkBoundary(int x) {
22        if (x < 0) {
23            x = 0;
24            darken.setEnabled(false);
25        } else {
26            darken.setEnabled(true);
27        }
28        if (x > 255) {
29            x = 255;
30        }
31        if (x == 0) {
32            label.setText("Darkening -> no more, it's the darkest" +
33                " Lightning -> mouse click");
34        } else if (x == 255) {
35            label.setText("Darkening -> Darken button click," +
36                " Lightning -> no more, it's the lightest");
37        } else {
38            label.setText("Darkening -> Darken button click," +
39                " Lightning -> mouse click");
40        }
41        return x;
42    }
43    public void actionPerformed(ActionEvent e) {
44        red = checkBoundary(red - 10);
45        green = checkBoundary(green - 10);
46        blue = checkBoundary(blue - 10);
47        repaint();
48    }
49    @Override
50    public void mousePressed(MouseEvent e) {}
51    @Override
52    public void mouseReleased(MouseEvent e) {}
53    @Override
54    public void mouseEntered(MouseEvent e) {}
55    @Override
56    public void mouseExited(MouseEvent e) {}
57    @Override
58    public void mouseClicked(MouseEvent e) {
59        red = checkBoundary(red + 10);
60        green = checkBoundary(green + 10);
61        blue = checkBoundary(blue + 10);
62        repaint();
63    }
64    public static void main(String[] args) {
65        JFrame.setDefaultLookAndFeelDecorated(true);
66        JFrame frame = new JFrame("Mouse event 3");
67        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
68        frame.setSize(430, 120);
69        MyMouseEvent3 panel = new MyMouseEvent3();
70        frame.add(panel);
71        frame.setVisible(true);
72    }
73 }
```



The application window 'Mouse event 3' displays a label and a 'Darken' button. The label text changes based on the current state of the red, green, and blue color values. The 'Darken' button is disabled when the red value reaches 0 and enabled when it reaches 255.

- Initial state: Label: "Darkening -> Darken button click, Lightning -> mouse click". Button: Enabled.
- After clicking 'Darken': Label: "Darkening -> Darken button click, Lightning -> no more, it's the lightest". Button: Disabled.
- After clicking 'Darken' again: Label: "Darkening -> Darken button click, Lightning -> mouse click". Button: Enabled.
- After clicking 'Darken' again: Label: "Darkening -> no more, it's the darkest Lightning -> mouse click". Button: Disabled.

Subakti

Inner classes

- Java supports *inner classes*, i.e., classes defined inside other classes
- Inner classes are often used to defined listeners for events

```
MyActionEvent3.java ×
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class MyActionEvent3 extends JPanel {
5     JButton darken = new JButton("Darken"),
6         lighten = new JButton("Lighten");
7     int red = 255, green = 255, blue = 255;
8     private class DarkenListener implements ActionListener {
9         public void actionPerformed(ActionEvent e) {
10             int delta = 0;
11             if (e.getSource() == lighten) {
12                 delta = 10;
13             } else if (e.getSource() == darken) {
14                 delta = -10;
15             }
16             red = checkBoundary(red + delta);
17             green = checkBoundary(green + delta);
18             blue = checkBoundary(blue + delta);
19             repaint();
20         }
21     }
```

Inner classes (continued)

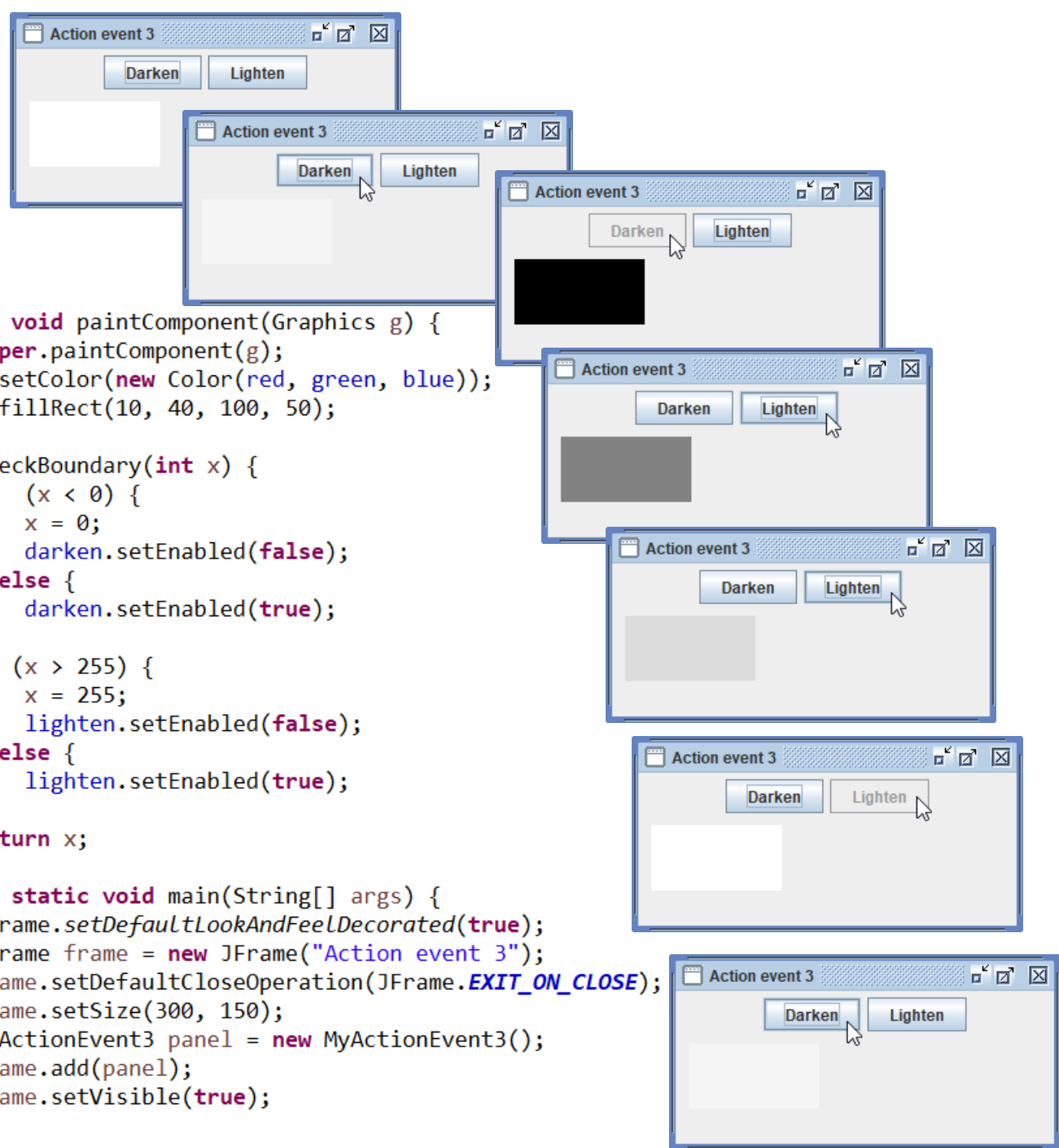
- The inner class can access the instance variables of the outer class (e.g., `darken`, `lighten`, `red`, `green`, `blue`) as well as the methods (e.g., `repaint()`);
- An instance of the inner class can be supplied as the listener for events

```
22 public MyActionEvent3() {  
23     add(darken);  
24     darken.addActionListener(new DarkenListener());  
25     add(lighten);  
26     lighten.addActionListener(new DarkenListener());  
27 }
```

Why use inner classes?

- We can handle each event separately in a self-contained method, e.g.,
 - `DarkenListener.actionPerformed`
 - `LightenListener.actionPerformed`
- Leads to better structure when a class has to implement listeners for a large number of events

Inner classes: Example



```
MyActionEvent3.java x
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class MyActionEvent3 extends JPanel {
5     JButton darken = new JButton("Darken"),
6         lighten = new JButton("Lighten");
7     int red = 255, green = 255, blue = 255;
8     private class DarkenListener implements ActionListener {
9         public void actionPerformed(ActionEvent e) {
10             int delta = 0;
11             if (e.getSource() == lighten) {
12                 delta = 10;
13             } else if (e.getSource() == darken) {
14                 delta = -10;
15             }
16             red = checkBoundary(red + delta);
17             green = checkBoundary(green + delta);
18             blue = checkBoundary(blue + delta);
19             repaint();
20         }
21     }
22     public MyActionEvent3() {
23         add(darken);
24         darken.addActionListener(new DarkenListener());
25         add(lighten);
26         lighten.addActionListener(new DarkenListener());
27     }
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57 }
```

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(new Color(red, green, blue));
    g.fillRect(10, 40, 100, 50);
}
int checkBoundary(int x) {
    if (x < 0) {
        x = 0;
        darken.setEnabled(false);
    } else {
        darken.setEnabled(true);
    }
    if (x > 255) {
        x = 255;
        lighten.setEnabled(false);
    } else {
        lighten.setEnabled(true);
    }
    return x;
}
public static void main(String[] args) {
    JFrame.setDefaultLookAndFeelDecorated(true);
    JFrame frame = new JFrame("Action event 3");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 150);
    MyActionEvent3 panel = new MyActionEvent3();
    frame.add(panel);
    frame.setVisible(true);
}
```