AWT Event Handling

1. Event

1.1 What is an Event

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical use interface components.

This article introduces the foreground event which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list etc.

1.2 What is Event Handling

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. The mechanism has the code which is known as event handler that is executed when an event occurs. Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key words:

- ✓ **Source**: The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object.
- ✓ **Listener**: It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener processes the event and then returns.

When an event is generated, event handling is performed as follows.

(1) The object of concerned event class is created automatically and information about the **source** and the event get populated with in same object.
(2) Event object is forwarded to the method of registered **listener** class.
(3) The method is now get executed and returns.

2. Event Adapters

Generally, event listeners are used for event handling. The event listener represent the “interfaces” responsible to handle events. Java provides us various event listener classes. However, we did not learn about “interface”.

This chapter introduces “event adapters” instead of event listeners. Since the event adapters are used with inheritance, they can be handled more easily than event listeners.

“MouseAdapter” related to mouse is introduced here.

The MouseAdapter is an class for receiving mouse events. All methods of this class are empty. So, only the necessary methods need be overridden. This class is convenience class for creating listener objects. The methods provided by the MouseAdapter are shown below.

Method & Description of MouseAdapter
void mouseClicked(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released on the same place) on a component
void mouseDragged(MouseEvent e) Invoked when a mouse button is pressed on a component and then dragged.
void mouseEntered(MouseEvent e) Invoked when the mouse enters a component.
void mouseExited(MouseEvent e) Invoked when the mouse exits a component.
void mouseMoved(MouseEvent e) Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.
void mousePressed(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void mouseReleased(MouseEvent e) Invoked when a mouse button has been released on a component.

MouseEvent is passed when each method is called. We can extract event information from the MouseEvent object (MouseWheelEvent object).

The main methods we open use for MouseEvent are as follows.

Method & Description of MouseEvent
int getClickCount() Returns the number of mouse clicks associated with this event.
int getModifiersEx() Returns the extended modifier mask for this event.
int getX() Returns the horizontal x position of the event relative to the source component.
int getY() Returns the vertical y position of the event relative to the source component.

3. Inner Class

A relatively simple way to do event processing using MouseAdapter is to inherits MouseAdapter. However, we are already using a program that inherits JFrame to create a window and render graphics. In Java, a class cannot inherit multiple classes. Therefore, we will use “inner class” here. The object of the inner class can be registered using the “addMouseListener” method of Component class which is an ancestor class of JFrame.

Writing a class within another is allowed in Java. The class written within is called the “inner class”, and the class that holds the inner class is called the “outer class”.

The syntax is as follows.

```
class Outer_Class {
    class Inner_Class {
    }
}
```

We can use modifiers for access levels such as private, protected and public. Also inner classes refer to field variables and methods in the same class.

4. Examples

(A) Mouse Information

As a first example, the program “MouseInformation.java” using “addMouseListener” is introduced.

In order to handle mouse events, inner class “MyListener” which inherits MouseAdapter is created (lines 31 to 52). The object of “MyListener” is registered using “addMouseListener” method (lines 16 and 17).

MouseInformation.java

```java
/*
 * Showing Mouse Information
 * by Hitoshi Ogawa on November 1st, 2021
 */

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;

public class MouseInformation extends JFrame {

    public MouseInformation() {
        setSize(500,500);   // setting window size
        MyListener myListener = new MyListener();
        addMouseListener(myListener);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void paint(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(0,  0,  500, 500);
    }

    public static void main(String[] args) {
        new MouseInformation();
    }

    class MyListener extends MouseAdapter {

        public void mouseClicked(MouseEvent e) {
            System.out.println("Clicked at (" + e.getX() + ", " + e.getY() + ")");
        }

        public void mouseEntered(MouseEvent e) {
            System.out.println("Entered at (" + e.getX() + ", " + e.getY() + ")");
        }

        public void mouseExited(MouseEvent e) {
            System.out.println("Exited at (" + e.getX() + ", " + e.getY() + ")");
        }

        public void mousePressed(MouseEvent e) {
            System.out.println("Pressed at (" + e.getX() + ", " + e.getY() + ")");
        }

        public void mouseReleased(MouseEvent e) {
            System.out.println("Released at (" + e.getX() + ", " + e.getY() + ")");
        }
    }
}
```

The window size of “MouseInformation” is 500 × 500. Since the default background color of the window is gray, it is painted white by fillRect method in paint method.

The method is called according to the content of the event that occurred. Event information is given in the MouseEvent variable “e”.

An example of the execution is shown in Figure 1. In this example, the coordinates of the mouse when the event occurred are displayed. The following cane be seen from each line.

line 1: The mouse entered the window from the coordinate (498, 214).
line 2: The button was pressed when the mouse coordinate was (292, 189).
line 3: The button was released when the mouse coordinate was (292, 189).
line 4: A click event occurs when a button is pressed and released at the same coordinate.
line 5: The button was pressed when the mouse coordinate was (292, 189).
line 6: The button was pressed when the mouse coordinate was (280, 258). A click event does not occur because a button is pressed and released at the different coordinate.
line 7: The mouse exited the window from the coordinate (500, 272).

```
Entered at (498, 214)
Pressed at (292, 189)
Released at (292, 189)
Clicked at (292, 189)
Pressed at (292, 189)
Released at (280, 258)
Exited at (500, 272)
```

Figure 1. An example of “MouseInformation” execution

(B) Double Click

“DoubleClick” class is introduced as a program example to count the number of clicks. This program draws a red arc with one click, and draws a blue circle with double click. An example of execution is shown in Figure 2.

When the mouse button is pressed, the number of presses can be obtained with “getClickCount” method of MouseEvent class.

This program draws without the “paint” method of DoubleClick class, so the “getGraphics” method is used to obtain a graphics context (lines 35 and 40). If it is pressed twice, the circle is painted in blue (lines 36 and 37), otherwise the arc is drawn in red (lines 41 and 42). When the mouse button is pressed twice, the red arc is drawn by the first press and the blue circle is drawn by the second press.

The “dispose” method of Graphics class is used to dispose of the graphics context and release any system resources that it is using (lines 38 and 43). This method is recommended for efficient processing. Graphics objects which are provided as arguments to the “paint” method are automatically released by the system when it return.

DoubleClick.java

```java
/*
 * An example of double click
 * by Hitoshi Ogawa on November 1st, 2021
 */

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;

public class DoubleClick extends JFrame {
    int size = 50;

    public DoubleClick() {
        setSize(500,500);   // setting window size
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        MyListener myListener = new MyListener();
        addMouseListener(myListener);
    }

    public void paint(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(0,  0,  500, 500);
    }

    public static void main(String[] args) {
        new DoubleClick();
    }

    class MyListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
            if(e.getClickCount() == 2) {
                Graphics g = getGraphics();
                g.setColor(Color.blue);
                g.fillOval(e.getX() - size / 2, e.getY() - size / 2, size, size);
                g.dispose();
            } else {
                Graphics g = getGraphics();
                g.setColor(Color.red);
                g.drawOval(e.getX() - size / 2, e.getY() - size / 2, size, size);
                g.dispose();
            }
        }
    }
}
```

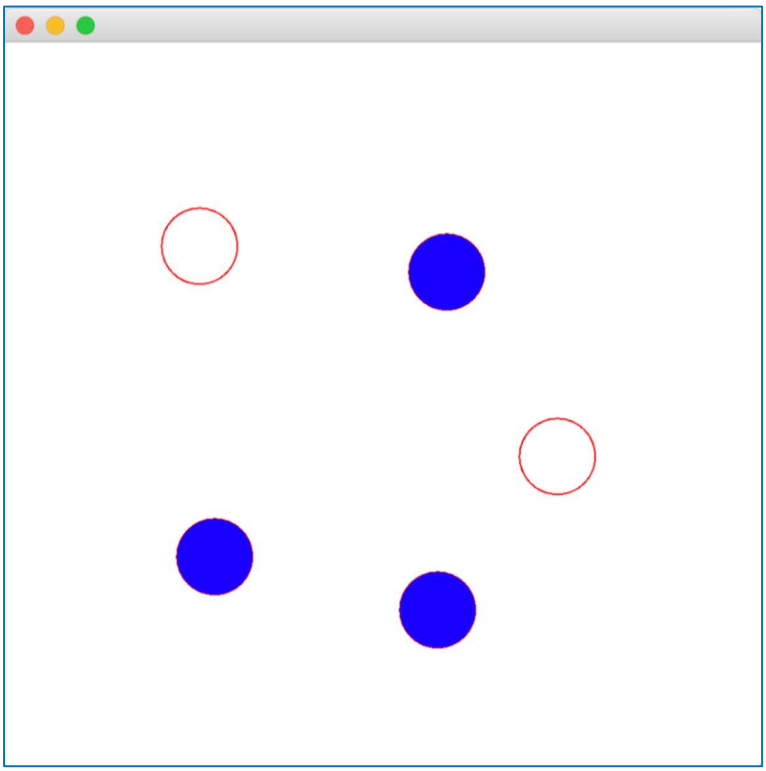

Figure 2. An example of “DoubleClick” execution

(C) Right Click

“RightClick” class is introduced as a program example to recognize right click (In the case of MacBook, tap two fingers on trackpad). The result of RightClick class is shown in Figure 3.

When a mouse button is pressed, information such as which button was pressed, whether there was another button pressed, etc. can be obtained from “getModifiersEx” method of MouseEvent method. The method getModifiresEx returns the extended modifier mask for the event. Extended modifiers are the modifiers that ends with the “_DOWN_MASK” suffix, such as “ALT_DOWN_MASK”, “BUTTON1_DOWN_MASK”, and others. The type of event can be found under the following conditions.

```
(event.getModifiersEx() & MouseEvent.extendedModifierMask)
                                == MouseEvent.extendedModifierMask
```

The variable event is the variable of MouseEvent. The field *extendedModifierMask* indicates one one of the extended modifier masks. Table 1 shows the main extended modifier masks which are provided by InputEvent which is the parent class of MouseEvent.

The character “&” means the bitwise logical operator which compares each binary digit of two integers and gives back 1 if both are 1, otherwise it returns 0. An example is shown below.

Example program

```
int value1 = 6;                          //      0 1 1 0
int value2 = 5;                          //      0 1 0 1
int result = value1 & value2;            //      --------
System.out.println(“result: “ + result);  //      0 1 0 0
```

Output

```
result: 4
```

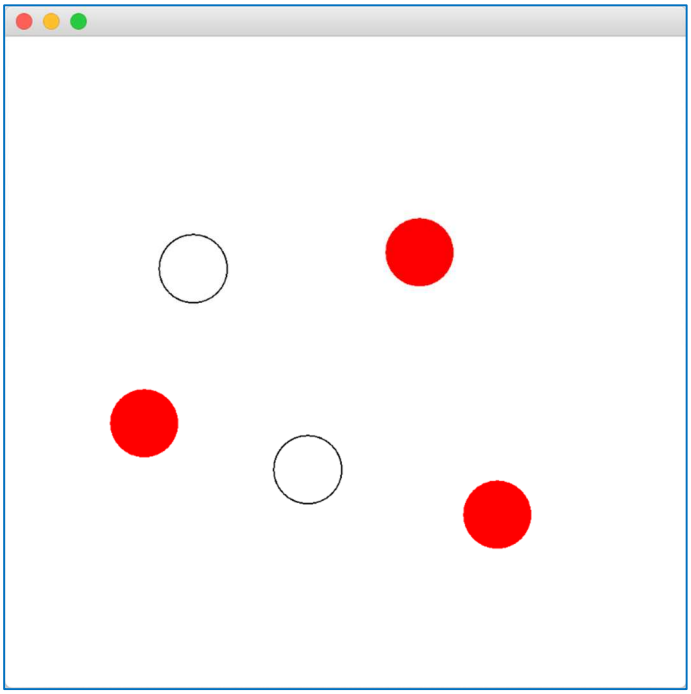

Figure 3. An example of “RightClick” execution.

Table 1. The main extended modifier masks.

Field	Description
ALT_DOWN_MASK	The Alt key extended modifier constant.
BUTTON1_DOWN_MASK	The Mouse Button1 extended modifier constant.
BUTTON2_DOWN_MASK	The Mouse Button2 extended modifier constant.
BUTTON3_DOWN_MASK	The Mouse Button3 extended modifier constant.
CTRL_DOWN_MASK	The Control key extended modifier constant.
SHIFT_DOWN_MASK	The Shift key extended modifier constant.

RightClick.java

```
1  /*
2   * An example of right click
3   * by Hitoshi Ogawa on November 1st, 2021
4   */
5
6  import java.awt.Color;
7  import java.awt.Graphics;
8  import java.awt.event.MouseAdapter;
9  import java.awt.event.MouseEvent;
10 import javax.swing.JFrame;
11
12 public class RightClick extends JFrame {
13     int size = 50;
14
15     public RightClick() {
16         setSize(500,500);   // setting window size
17         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18         setVisible(true);
19         MyListener myListener = new MyListener();
20         addMouseListener(myListener);
21     }
22
23     public void paint(Graphics g) {
24         g.setColor(Color.white);
25         g.fillRect(0,  0,  500, 500);
26     }
27
28     public static void main(String[] args) {
29         new RightClick();
30     }
31
32     class MyListener extends MouseAdapter {
33         public void mousePressed(MouseEvent e) {
34             System.out.println(e.getModifiersEx());
35             if((e.getModifiersEx() & MouseEvent.BUTTON3_DOWN_MASK)
36                     == MouseEvent.BUTTON3_DOWN_MASK) {
37                 Graphics g = getGraphics();
38                 g.setColor(Color.red);
39                 g.fillOval(e.getX() - size / 2, e.getY() - size / 2, size, size);
40                 g.dispose();
41             } else {
42                 Graphics g = getGraphics();
43                 g.setColor(Color.black);
44                 g.drawOval(e.getX() - size / 2, e.getY() - size / 2, size, size);
45                 g.dispose();
46             }
47         }
48     }
49 }
```

When the mouse button is pressed, it is checked whether the right button is pressed (lines 35 and 36). If so, a red circle is drawn. Otherwise, a black arc is drawn.

(D) Mouse Motion

"MouseMotion" class is introduced as a program example using "mouseDragged" method. An example of execution is shown in Figure 4.

This program realized a method of drawing a circle of arbitrary size at arbitrary position using "mouseDragged" method. In order to use "mouseDragged" method, it is necessary to register using "addMouseMotionListener" method of JFrame class (line 21). When pressing the mouse button, the x coordicate and y coordinate of the center of the circle (lines 37 and 38). Every time it is dragged, the size is calculated using the distance between the center coordinate and the mouse (line 42). Refer to 5.3 for details. The paint method is executed with the "repaint" method (line 43).

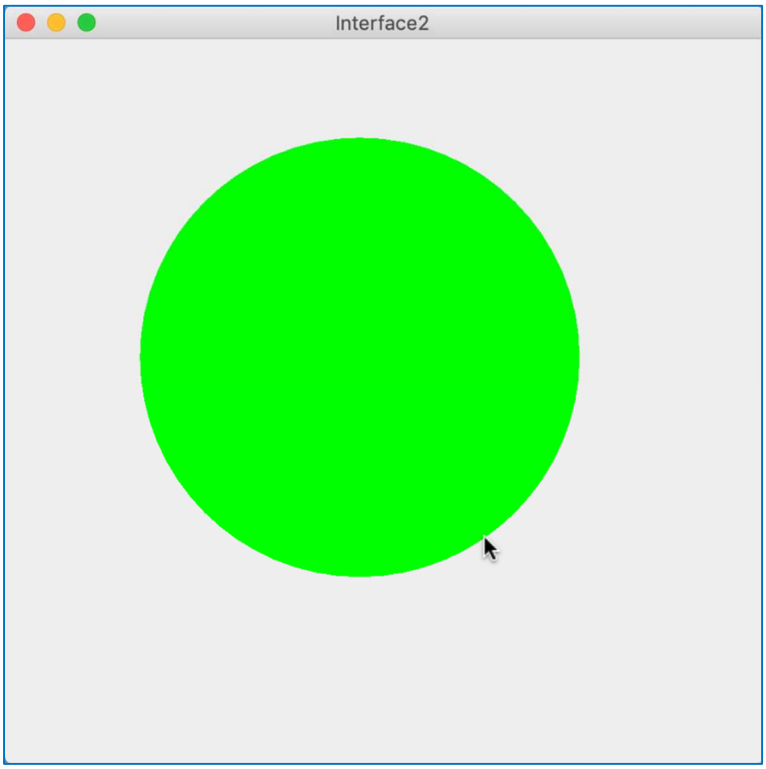

Figure 4. An example of "MouseMotion" execution.

MouseMotion.java

```java
/*
 * An example of mouse motion
 * by Hitoshi Ogawa on November 1st, 2021
 */

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;

public class MouseMotion extends JFrame {
    int size, xCenter, yCenter;

    public MouseMotion() {
        setSize(500,500);   // setting window size
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        MyListener myListener = new MyListener();
        addMouseListener(myListener);
        addMouseMotionListener(myListener);
    }

    public void paint(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(0,  0,  500, 500);
        g.setColor(Color.green);
        g.fillOval(xCenter - size / 2,  yCenter - size / 2, size, size);
    }

    public static void main(String[] args) {
        new MouseMotion();
    }

    class MyListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
            xCenter = e.getX();
            yCenter = e.getY();
        }

        public void mouseDragged(MouseEvent e) {
            size = (int)Math.hypot(xCenter - e.getX(), yCenter - e.getY()) * 2;
            repaint();
        }
    }
}
```

5. Beyond Basic Arithmetic

The "Math" class in the java.lang package provides methods and constants for doing advanced mathematical computation.

The methods in the "Math" class are static, so we call them directly from the class, like this:

Math.*method*;

where *method* is a method provided by Math.

5. 1 Constants

The Math class includes two constants:

- Math.E which is the base of natural logarithms
- Math.PI which is the ratio of the circumference of a circle to its diameter

5. 2 Basic Methods

The following table lists the main of the basic methods of the Math class.

Basic Method & Description of Math
double abs(double d) **float abs(float f)** **int abs(int i)** **long abs(long lng)** Returns the absolute value of the argument.
double min(double arg1, double arg2) **float min(float arg1, float arg2)** **int min(int arg1, int arg2)** **long min(long arg1, long arg2)** Returns the smaller of the two arguments.
double max(double arg1, double arg2) **float max (float arg1, float arg2)** **int max (int arg1, int arg2)** **long max (long arg1, long arg2)** Returns the larger of the two arguments.

5. 3 Exponential and Logarithmic Methods

The following table lists exponential and logarithmic methods of the Math class.

Basic Method & Description of Math
double exp(double d) Returns the base of the natural logarithms, e, to the power of the argument.
double hypot(double x, double y) Returns $sqrt(x^2 + y^2)$.
double log(double d) Returns the natural logarithm of the argument.
double pow(double base, double exponent) Returns the value of the first argument raised to the power of the second argument.
double sqrt(double d) Returns the square root of the argument.

5. 4 Trigonometric Methods

The Math class also provides a collection of trigonometric functions, which are summarized in the following table. The value passed into each of these methods is an angle expressed in radians. The “toRadians” method can be used to convert from degrees to radians.

Basic Method & Description of Math
double sin(double d) Returns the sine of the specified double value.
double cos(double d) Returns the cosine of the specified double value.
double tan(double d) Returns the tangent of the specified double value.
double toDegrees(double d) Converts the argument to degrees.
double toRadians(double d) Converts the argument to radians.

5. 5 Random Numbers

The “random()” method returns a pseude-randomly selected number 0.0 and 1.0. The range includes 0.0 but not 1.0. In other words:

0.0 <= Math.random() < 1.0.

To get a number in a different range, we can perform arithmetic on the value returned by the random method. For example, to generate an integer between 0 and 9, we would write:

int number = (int)(Math.ramdom() * 10);

By multiplying the value by 10, the range of possible values becomes

0 <= number < 10.