There are (at least) two types of mouse events.

1. Defined in interface MouseListener: Clicking, pressing, or releasing a mouse button. Also, entering and exiting a component, like a JPanel.
2. Defined in interface MouseMotionListener: moving a mouse or pressing a mouse button and dragging it in some component, like a JPanel.

The second type of mouse event is more expensive because tracking the cursor movement involves significantly more system overhead —the system must continuously monitor the cursor movement. So, separating the two kinds of events makes it possible to use only the first kind when that is sufficient for an application.

Nevertheless, for this first foray into mouse listening, we choose a simple solution that allows both kinds of mouse event.

**Class MouseInputAdaptor**

Java class javax.swing.event.MouseInputAdaptor has empty procedures (they don’t do anything, but very fast) for these procedures:

|  |  |
| --- | --- |
| mouseClicked(MouseEvent e) | called when the mouse button is clicked (pressed and released) on a component. |
| mouseDragged(MouseEvent e) | called when a mouse button is pressed on a component and then dragged. |
| mouseEntered(MouseEvent e) | called when the mouse enters a component. |
| mouseExited(MouseEvent e) | called when the mouse exits a component. |
| mouseMoved(MouseEvent e) | called when the mouse cursor has been moved onto a component but no buttons have been pushed |
| mousePressed(MouseEvent e) | called when a mouse button has been pressed on a component. |
| mouseReleased(MouseEvent e) | called when a mouse button has been released on a component. |
| mouseWheelMoved(  MouseWheelEvent e) | called when the mouse wheel is rotated. |

We explore doing this for the event of pressing a button.

The GUI, a JFrame, shown to the right has two buttons labeled East and West. Exactly one of them is enabled. When the enabled button is pressed, it becomes disabled and the other one becomes enabled. We show how this is done.

The two buttons are declared like this:

**private** JButton westButton= **new** JButton("west");

**private** JButton eastButton= **new** JButton("east");

To the right, we show a method that is to be called when the enabled button is pressed. It does three things: First, store true in local variable b if the east button is enabled and false otherwise. Second, complement the enabled property of the east button —if it was enabled it become disabled and vice versa. Third, change the enabled property of the west button accordingly.

/\*\* Process a click of a button \*/

**public** **void** buttonClick(ActionEvent e) {

**boolean** b= eastButton.isEnabled();

eastButton.setEnabled(!b);

westButton.setEnabled(b);

}

To have this method called when the east button is pressed, execute this call:

eastButton.addActionListener(e -> buttonClick(e));

The call registers the argument, the anonymous function, as a listener for the east button. So, when the east button is clicked, the anonymous function is called. What does it do? It calls procedure buttonClick, giving it as argument the parameter e of the anonymous function. Procedure buttonClick doesn’t use the parameter, but it could, and we’ll show its use in a later example.

Looking at procedure buttonClick, we can see that parameter e of the anonymous function has type ActionEvent.

Similarly, we register an anonymous function with the west button:

westButton.addActionListener(e -> buttonClick(e));

**Using one anonymous function instead of two**

The code shown above actually creates two anonymous functions, one for the east and one for the west button. If we want only one, we can store the anonymous function in a local variable and then use the variable twice:

ActionListener al= (ActionEvent e) -> buttonClick(e);

westButton.addActionListener(al);

eastButton.addActionListener(al);

You see that the type of the anonymous function is ActionListener. We did not have to put in the type of parameter e; it would have been inferred if we left it out.

Finally, note that we didn’t really need function buttonClick, for we could have written the assignment to al as shown below. The anonymous function does all the work without calling buttonClick. We advise against this when the body of an anonymous function becomes complicated. Certainly don’t do this until you know more about anonymous functions.

ActionListener al= e -> {

**boolean** b= eastButton.isEnabled();

eastButton.setEnabled(!b);

westButton.setEnabled(b);

};