The Event Delegation Model

The event delegation model is a design pattern used in Java for handling events, particularly in graphical user interface (GUI) programming with AWT and Swing. It follows the principle of separating the event handling code from the event source, which is the component that generates the event.

Here's how the event delegation model works:

Event Source

An event source is any object that can generate events. In Java GUI programming, these are typically components such as buttons, text fields, or entire windows. The event source is responsible for maintaining a list of listeners that are interested in processing events when they occur.

Event Listeners

Event listeners are objects that want to be notified when a specific type of event occurs. These objects must implement specific listener interfaces corresponding to the type of event they want to handle (for example, `ActionListener` for button clicks, `MouseListener` for mouse events, etc.). Each listener interface has one or more methods that need to be implemented to provide the code that will run when the event is received.

Event Object

When an event occurs, the event source creates an event object that encapsulates information about the event, such as its type and the state of the source when the event occurred. This object is then passed to the listener.

Event Registration

For a listener to receive an event, it must be registered with the source. This is typically done by calling an add listener method on the event source (for example, `button.addActionListener(...)`).

Event Dispatching

When an event occurs, the event source dispatches the event to all registered listeners. The listeners' methods are invoked, and they can use the event object to determine the specifics of the event.

Advantages of the Event Delegation Model

Separation of Concerns: The model cleanly separates the logic for generating events from the logic for handling them, making the code easier to manage and extend.
Flexibility: Multiple listeners can be registered to handle the same type of event for a single component, providing a flexible way to handle events.
Reusability: Event listener classes can be reused across different components, making it easy to create a consistent behavior for similar events.
Performance: Since only interested listeners handle events, the model can be more efficient than older designs where components themselves handled events.

Event Source (Component)	Event Listener Interface	Event Class	Typical Event Object Methods
Button, MenuItem	ActionListener	ActionEvent	`getActionCommand()`, `getSource()`
Component	MouseListener	MouseEvent	`getX()`, `getY()`, `getClickCount()`
Component	MouseMotionListener	MouseEvent	`getX()`, `getY()`
Component	KeyListener	KeyEvent	`getKeyCode()`, `getKeyChar()`
Window	WindowListener	WindowEvent	`getWindow()`, `getNewState()`
Checkbox, RadioButton	ItemListener	ItemEvent	`getItem()`, `getStateChange()`
Component	FocusListener	FocusEvent	`getOppositeComponent()`, `isTemporary()`
Slider, Spinner	ChangeListener (Swing)	ChangeEvent	`getSource()`
Text Component	DocumentListener (Swing)	DocumentEvent	`getOffset()`, `getLength()`
JList, JComboBox	ListSelectionListener (Swing)	ListSelectionEvent	`getFirstIndex()`, `getLastIndex()`

Scrollbar	AdjustmentListener	AdjustmentEvent	`getAdjustable()`, `getValue()`

Listener Interface	Methods	Description
ActionListener	`actionPerformed(ActionEvent e)`	Invoked when an action occurs, such as a button click.
MouseListener	`mouseClicked(MouseEvent e)`	Invoked when the mouse button has been clicked (pressed and released) on a component.
	`mousePressed(MouseEvent e)`	Invoked when a mouse button has been pressed on a component.
	`mouseReleased(MouseEvent e)`	Invoked when a mouse button has been released on a component.
	`mouseEntered(MouseEvent e)`	Invoked when the mouse enters a component.
	`mouseExited(MouseEvent e)`	Invoked when the mouse exits a component.

MouseMotionListener	`mouseDragged(MouseEvent e)`	Invoked when a mouse button is pressed on a component and then dragged.
	`mouseMoved(MouseEvent e)`	Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.
KeyListener	`keyTyped(KeyEvent e)`	Invoked when a key has been typed (pressed and released).
	`keyPressed(KeyEvent e)`	Invoked when a key has been pressed.
	`keyReleased(KeyEvent e)`	Invoked when a key has been released.
WindowListener	`windowOpened(WindowEvent e)`	Invoked when a window has been opened.
	`windowClosing(WindowEvent e)`	Invoked when a window is in the process of being closed.
	`windowClosed(WindowEvent e)`	Invoked when a window has been closed.
	`windowIconified(WindowEvent e)`	Invoked when a window is iconified.
	`windowDeiconified(WindowEvent e)`	Invoked when a window is de-iconified.

	`windowActivated(WindowEvent e)`	Invoked when a window is activated.
	`windowDeactivated(WindowEvent e)`	Invoked when a window is deactivated.
ItemListener	`itemStateChanged(ItemEvent e)`	Invoked when an item's state changes (e.g., checkbox checked/unchecked).
FocusListener	`focusGained(FocusEvent e)`	Invoked when a component gains the keyboard focus.
	`focusLost(FocusEvent e)`	Invoked when a component loses the keyboard focus.
ChangeListener (Swing)	`stateChanged(ChangeEvent e)`	Invoked when the target of the listener has changed its state.
DocumentListener (Swing)	`insertUpdate(DocumentEvent e)`	Invoked when a document has had a portion inserted.
	`removeUpdate(DocumentEvent e)`	Invoked when a portion of a document has been removed.
	`changedUpdate(DocumentEvent e)`	Invoked when a document's attributes are changed.
ListSelectionListener (Swing)	`valueChanged(ListSelectionEvent e)`	Invoked when the selection value changes in a list selection model.

AdjustmentListener	`adjustmentValueChanged(AdjustmentEvent e)`	Invoked when a component's adjustable value has changed.

JLabel

Method	Return Type	Description
`setText(String text)`	void	Sets the text of the label.
`getText()`	String	Returns the text of the label.
`setIcon(Icon icon)`	void	Sets the icon to be displayed in the label.

`getIcon()`	Icon	Returns the icon displayed in the label.
`setHorizontalAlignment(int alignment)`	void	Sets the alignment of the label's contents along the X axis.
`getHorizontalAlignment()`	int	Returns the horizontal alignment of the label's contents.
`setVerticalAlignment(int alignment)`	void	Sets the alignment of the label's contents along the Y axis.
`getVerticalAlignment()`	int	Returns the vertical alignment of the label's contents.
`setIconTextGap(int gap)`	void	Sets the space between the label's text and its icon.
`getIconTextGap()`	int	Returns the space between the label's text and its icon.
`setFont(Font font)`	void	Sets the font for the label's text.
`getFont()`	Font	Returns the font for the label's text.
`setForeground(Color color)`	void	Sets the color of the text displayed in the label.

`getForeground()`	Color	Returns the color of the text displayed in the label.
`setBackground(Color color)`	void	Sets the background color of the label (if opaque is set to true).
`getBackground()`	Color	Returns the background color of the label.
`setOpaque(boolean isOpaque)`	void	Sets whether the label should be opaque or transparent (true for opaque, false for transparent).
`isOpaque()`	boolean	Returns true if the label is opaque, false if it is transparent.

JFrame

Method	Return Type	Description
`setVisible(boolean visible)`	void	Sets the visibility of the frame. True makes it visible; false hides it.

`setSize(int width, int height)`	void	Sets the size of the frame to the specified width and height in pixels.
`setTitle(String title)`	void	Sets the title of the frame.
`getTitle()`	String	Returns the title of the frame.
`setDefaultCloseOperation(int operation)`	void	Specifies the operation that happens by default when the user initiates a "close" on the frame.
`getDefaultCloseOperation()`	int	Returns the default close operation.
`add(Component comp)`	Component	Adds the specified component to the frame.
`setLayout(LayoutManager manager)`	void	Sets the layout manager for the frame's content pane.
`getContentPane()`	Container	Returns the content pane object for this frame.
`pack()`	void	Causes the frame to be sized to fit the preferred size and layout of its subcomponents.

`setResizable(boolean resizable)`	void	Sets whether the frame can be resized by the user.
`isResizable()`	boolean	Returns whether the frame is resizable by the user.
`setLocationRelativeTo(Component c)`	void	Sets the location of the frame relative to the specified component.
`setLocation(int x, int y)`	void	Moves the frame to the specified location.
`getContentPane().setBackground(Color color)`	void	Sets the background color of the frame's content pane.
`getContentPane().getBackground()`	Color	Returns the background color of the frame's content pane.
`setIconImage(Image image)`	void	Sets the image to be displayed as the icon for the frame.
`getIconImage()`	Image	Returns the image displayed as the icon for the frame.

JButton

Method	Return Type	Description
`setText(String text)`	void	Sets the text displayed by the button.
`getText()`	String	Returns the text displayed by the button.
`setEnabled(boolean enabled)`	void	Enables or disables the button. Disabled buttons cannot respond to user input.
`isEnabled()`	boolean	Returns true if the button is enabled, false otherwise.
`addActionListener(ActionListener l)`	void	Adds an `ActionListener` to the button. The listener's `actionPerformed` method is called when the button is pressed.
`removeActionListener(ActionListener l)`	void	Removes an `ActionListener` from the button.
`setIcon(Icon icon)`	void	Sets the Icon to be displayed by the JButton.

`getIcon()`	Icon	Returns the Icon displayed by the JButton.
`setMnemonic(int mnemonic)`	void	Sets the mnemonic (keyboard shortcut) for the button.
`getMnemonic()`	int	Returns the mnemonic for the button.
`setToolTipText(String text)`	void	Sets the tool tip text displayed when the cursor hovers over the button.
`getToolTipText()`	String	Returns the tool tip text for the button.
`setFocusable(boolean focusable)`	void	Sets whether the button can gain focus.
`isFocusable()`	boolean	Returns true if the button can gain focus, false otherwise.
`setHorizontalAlignment(int alignment)`	void	Sets the horizontal alignment of the icon and text within the button.
`getHorizontalAlignment()`	int	Returns the horizontal alignment of the icon and text within the button.

`setVerticalAlignment(int alignment)`	void	Sets the vertical alignment of the icon and text within the button.
`getVerticalAlignment()`	int	Returns the vertical alignment of the icon and text within the button.
`setBackground(Color bg)`	void	Sets the background color of the button.
`getBackground()`	Color	Returns the background color of the button.
`setForeground(Color fg)`	void	Sets the foreground color of the button (used for the text).
`getForeground()`	Color	Returns the foreground color of the button.

The `ItemEvent` class in Java is used in conjunction with item listeners to provide information about item events, such as when a checkbox is checked or unchecked or when a radio button selection changes. Here are some of the key methods provided by the `ItemEvent` class:

Method	Return Type	Description

`getItem()`	Object	Returns the object that was affected by the event.
`getStateChange()`	int	Returns the type of state change (selected or deselected). This method returns either `ItemEvent.SELECTED` or `ItemEvent.DESELECTED`.
`getItemSelectable()`	ItemSelectable	Returns the `ItemSelectable` object that originated the event.
`getSource()`	Object	Returns the source of the event (the object that fired the event).
`paramString()`	String	Returns a string representing the state of this event. Useful for debugging.