Course Code: SL3001	Course: Software Development and construction
Instructor:	Yasir Arfat.

Lab # 04

SWING EVENT HANDLING:

Event handling in Java is the mechanism that controls the events and determines what should happen when an event occurs.

Types of Events: User interface events (like mouse clicks, key presses), system events, and application-specific events.

Event Handling Mechanism

Event Source: The object that generates the event. Examples include buttons, text fields, or any interactive UI component.

Event Object: The object that encapsulates the event information. Examples include ActionEvent, MouseEvent, KeyEvent.

Event Listener: The interface that listens to events and defines the methods to respond to those events. Examples include ActionListener, MouseListener, KeyListener.

Event Handling Process:

Registering the listener with the source. The listener's method is invoked automatically when the event occurs. Event Listeners in Java Swing

ActionListener:

Purpose: Used to handle action events, typically triggered by buttons, menus, etc.

Key Method: actionPerformed(ActionEvent e).

MouseListener:

Purpose: Used to handle mouse events like clicks, presses, and releases. Key Methods: mouseClicked(MouseEvent e), mousePressed(MouseEvent e), mouseReleased(MouseEvent e)

KeyListener:

Purpose: Used to handle keyboard events. Key Methods: keyPressed(KeyEvent e), keyReleased(KeyEvent e), keyTyped(KeyEvent e).

Event Adapters

Adapters are classes that provide empty implementations for listener interfaces. They are useful when you need to handle only some events.

Examples: MouseAdapter, KeyAdapter

Implementing Event Handling

Steps:

Create the event source (e.g., JButton).

Implement the listener interface in your class or use a lambda expression.

Register the listener with the event source using addXXXListener() method.

Inner Classes and Anonymous Classes for Event Handling

Inner Classes: A class within another class used to handle events for the outer class.

Anonymous Classes: A concise way to implement listener interfaces inline without a separate class definition.

Lambda Expressions for Event Handling

A more compact way of writing event listeners, introduced in Java 8.

Example: button.addActionListener(e -> System.out.println("Button clicked!"));

Best Practices in Event Handling

Separation of Concerns: Keep event handling logic separate from business logic.

Use of Anonymous and Lambda Classes: For simple event handling to reduce code verbosity.

Event Queue: Ensuring the code that updates the UI runs on the Event Dispatch Thread (EDT).

Event	Description
keyTyped(KeyEvent e)	This method is triggered when a key is typed. A key typed event occurs when a key press is followed by a key release, resulting in a character being typed. This is primarily used to detect character input.
keyPressed(KeyEvent e)	This method is triggered when a key is pressed. It's used to detect when a key is physically pressed down. This method is useful for detecting the beginning of a key press event.
keyReleased(KeyEvent e)	This method is triggered when a key is released after being pressed. It's used to detect the end of a key press event.
KeyEvent.VK_A-Z	Represents the keys 'A' through 'Z'. When pressed, it generates a KeyEvent that can be used to detect alphabetic key presses.
KeyEvent.VK_0-9	Represents the keys '0' through '9'. Detects numeric key presses.
KeyEvent.VK_ENTER	Represents the Enter key. Often used to trigger form submissions or actions like button clicks.
KeyEvent.VK_ESCAPE	Represents the Escape key. Commonly used to cancel actions or close dialogs.

KeyEvent.VK_SPACE	Represents the Spacebar key. Can be used to trigger an action, like pressing a button or toggling a checkbox.
KeyEvent.VK_TAB	Represents the Tab key. Used for navigation between fields or components within a UI.
KeyEvent.VK_SHIFT	Represents the Shift key. Typically used in combination with other keys for modified input (e.g., capital letters).
KeyEvent.VK_CONTROL	Represents the Control (Ctrl) key. Commonly used for keyboard shortcuts like Ctrl+C (copy) or Ctrl+V (paste).
KeyEvent.VK_ALT	Represents the Alt key. Used in combination with other keys to access menu shortcuts or other special functions.
KeyEvent.VK_BACK_SPACE	Represents the Backspace key. Used to delete the character before the cursor in text fields.
KeyEvent.VK_DELETE	Represents the Delete key. Used to delete the character at the cursor or selected text.
KeyEvent.VK_UP	Represents the Up Arrow key. Often used to move the cursor or focus up in lists, menus, or other navigable components.
KeyEvent.VK_DOWN	Represents the Down Arrow key. Used to move the cursor or focus down in lists, menus, or other navigable components.
KeyEvent.VK_LEFT	Represents the Left Arrow key. Used to move the cursor or focus left in text fields, lists, or other navigable components.
KeyEvent.VK_RIGHT	Represents the Right Arrow key. Used to move the cursor or focus right in text fields, lists, or other navigable components.
KeyEvent.VK_F1-F12	Represents function keys F1 through F12. Commonly used for specific functions in applications, like opening help (F1) or refreshing (F5).
MouseEvent.MOUSE_CLICKED	Represents a mouse click event. Triggered when a mouse button is clicked (pressed and released) on a component.
MouseEvent.MOUSE_PRESSED	Represents a mouse press event. Triggered when a mouse button is pressed on a component.
MouseEvent.MOUSE_RELEASED	Represents a mouse release event. Triggered when a mouse button is released after being pressed on a component.
MouseEvent.MOUSE_MOVED	Represents a mouse move event. Triggered when the mouse is moved over a component without any buttons pressed.

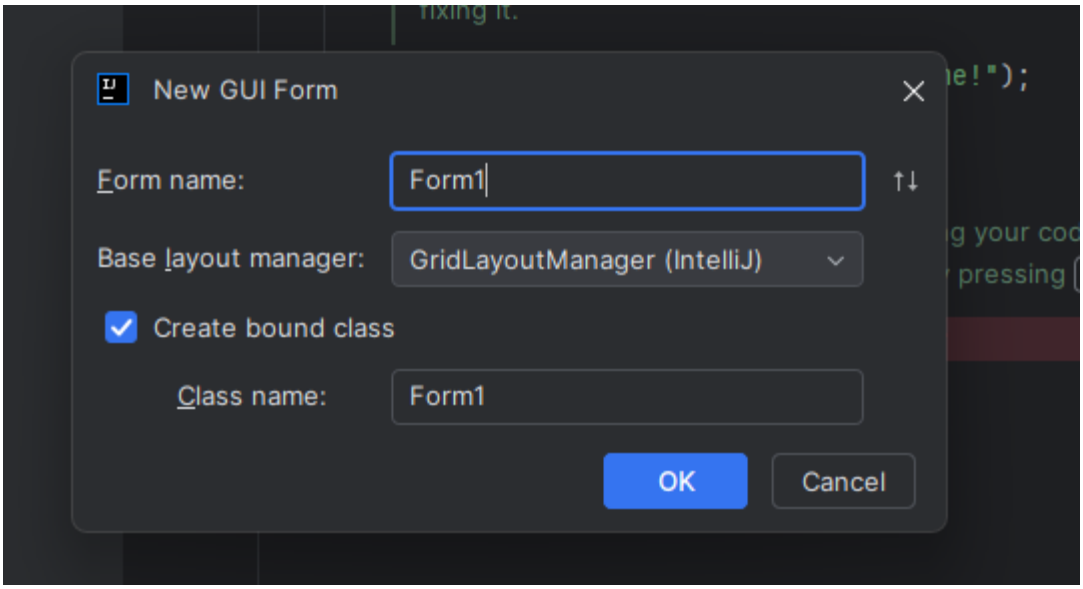
MouseEvent.MOUSE_DRAGGED	Represents a mouse drag event. Triggered when the mouse is moved while a button is pressed (dragging).
FocusEvent.FOCUS_GAINED	Represents a focus gained event. Triggered when a component gains keyboard focus.
FocusEvent.FOCUS_LOST	Represents a focus lost event. Triggered when a component loses keyboard focus.
ActionEvent.ACTION_PERFORMED	Represents an action performed event. Triggered when an action occurs, such as a button click or a menu item selection.
WindowEvent.WINDOW_OPENED	Represents a window opened event. Triggered when a window is opened.
WindowEvent.WINDOW_CLOSING	Represents a window closing event. Triggered when a window is in the process of closing (e.g., when the user clicks the close button).
WindowEvent.WINDOW_CLOSED	Represents a window closed event. Triggered when a window has closed.
WindowEvent.WINDOW_ICONIFIED	Represents a window iconified event. Triggered when a window is minimized to an icon.
WindowEvent.WINDOW_DEICONIFIED	Represents a window deiconified event. Triggered when a window is restored from being minimized.
WindowEvent.WINDOW_ACTIVATED	Represents a window activated event. Triggered when a window becomes the active window.
WindowEvent.WINDOW_DEACTIVATED	Represents a window deactivated event. Triggered when a window is no longer the active window.

Particle Implementation of Event Handling

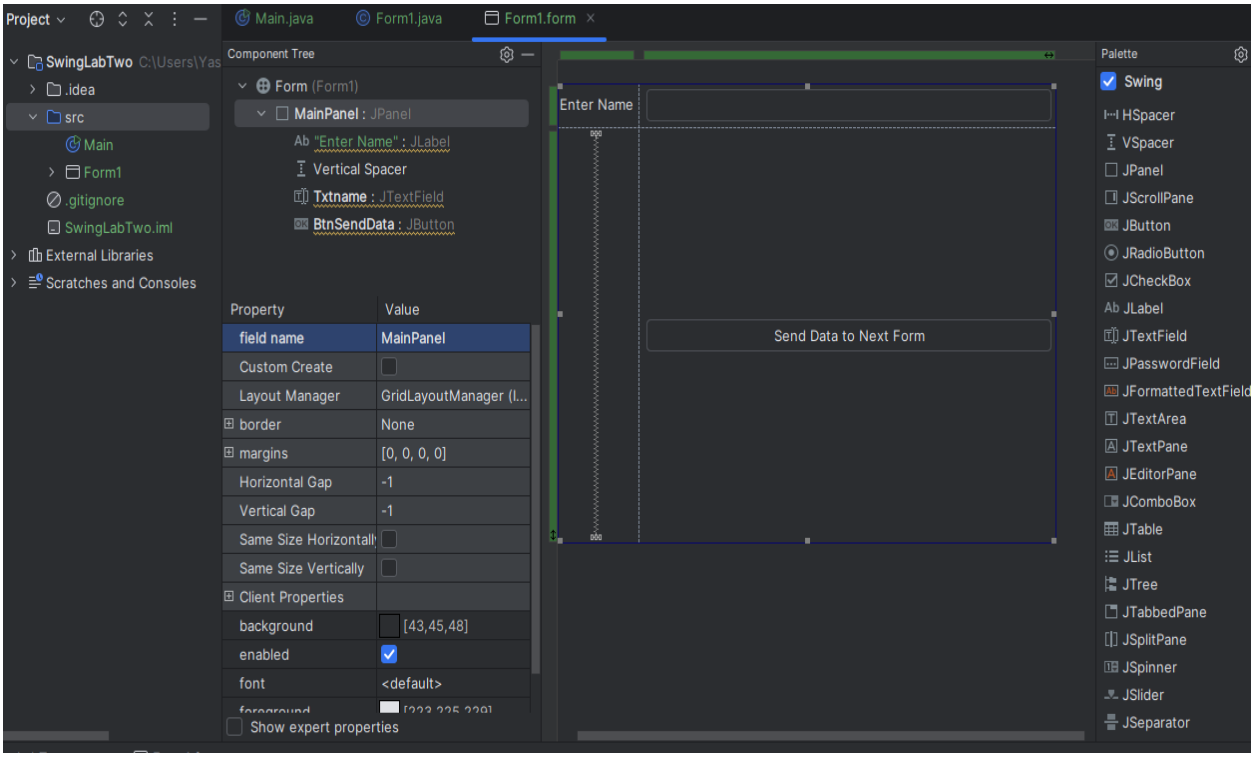
Here will see the implementation of Mouse Event Handling Along with Transferring the data from one tab to Another.

Steps

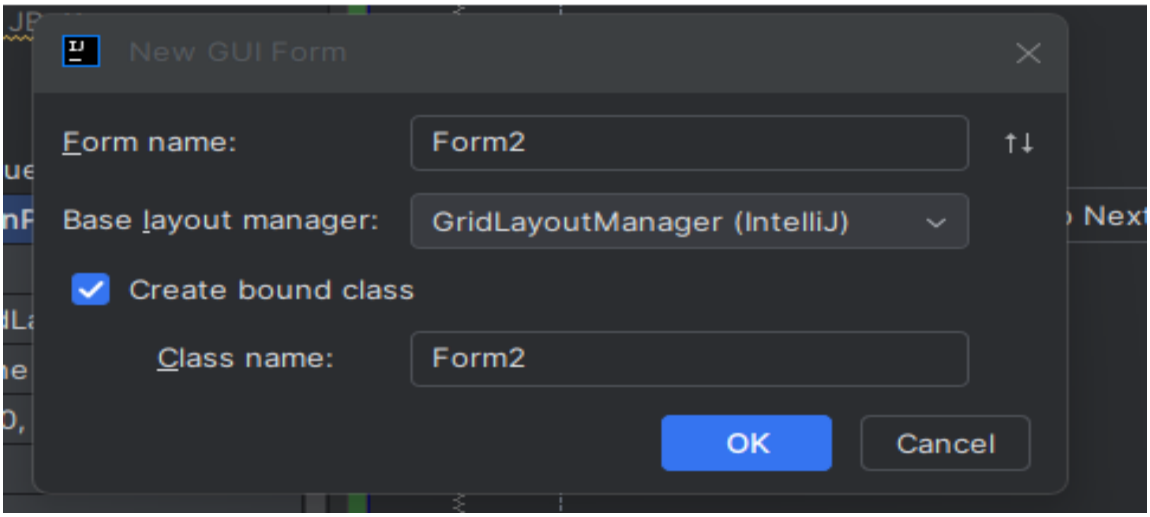
1. Create new Project (e.g SwingLabTwo)
2. Now Add a first form as Form1



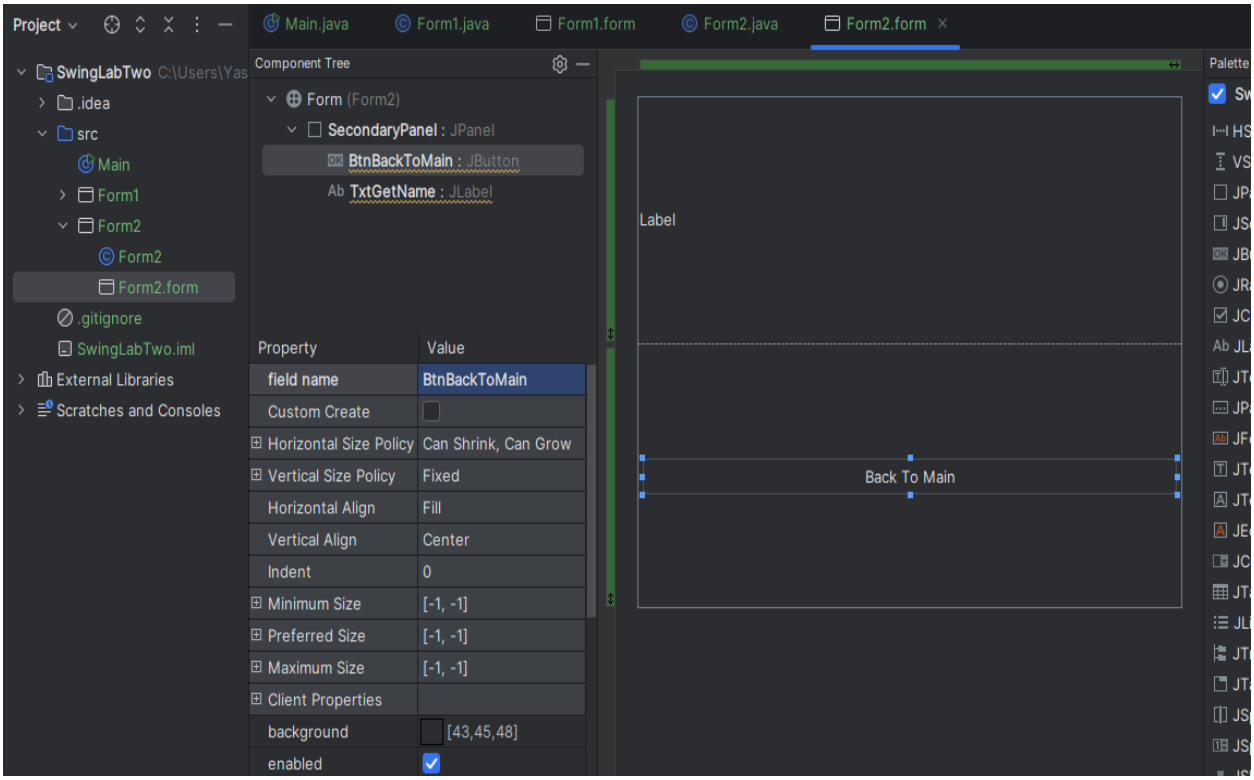
3. Now we will name the Jpanel as MainPanel and Add one JLabel and one JtextField (name it as Txtname) where user will enter his name also add Jbutton (name it as BtnSendData) in your designer



4. Now we will add Another Form as form2



5. In the second Form we will name the JPanel as SecodaryPanel and we will add One JLabel (Name it as TxtGeName) which will be use to Welcome the The user who has enter his name in first form and also add one JButton (Name it as BtnBackToMain) which will be use to back to first form.



6. Now we will create the Action Lister for both buttons

Code For Form1.java

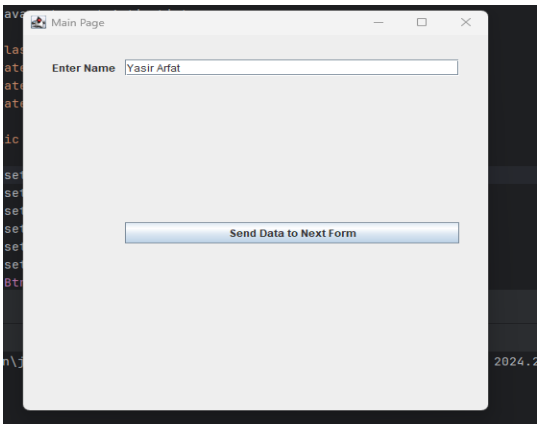
```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Form1 extends  JFrame {
    private JPanel MainPanel;
    private JTextField Txtname;
    private JButton BtnSendData;

    public Form1() {

        setContentPane(MainPanel);
        setTitle("Main Page");
        setSize(500,500);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        BtnSendData.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String name = Txtname.getText();
                new Form2(name);
                dispose();
            }
        });
    }

    public static void main(String[] args) {
        Form1 f = new Form1();
    }
}
```

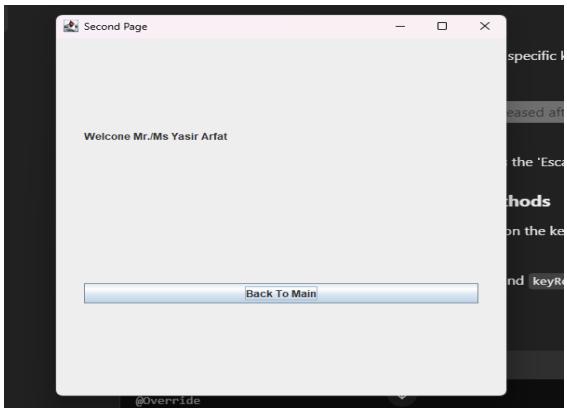
Code For Form2.java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Form2 extends JFrame {
    private JPanel SecondaryPanel;
    private JLabel TxtGetName;
    private JButton BtnBackToMain;

    public Form2(String name) {
        setContentPane(SecondaryPanel);
        setTitle("Second Page");
```

```
        setSize(500,500);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        TxtGetName.setText("Welcone Mr./Ms "+name);
        BtnBackToMain.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Form1 ne = new Form1();
                dispose();

            }
        });
    }


}
```

7. Now you can check the output



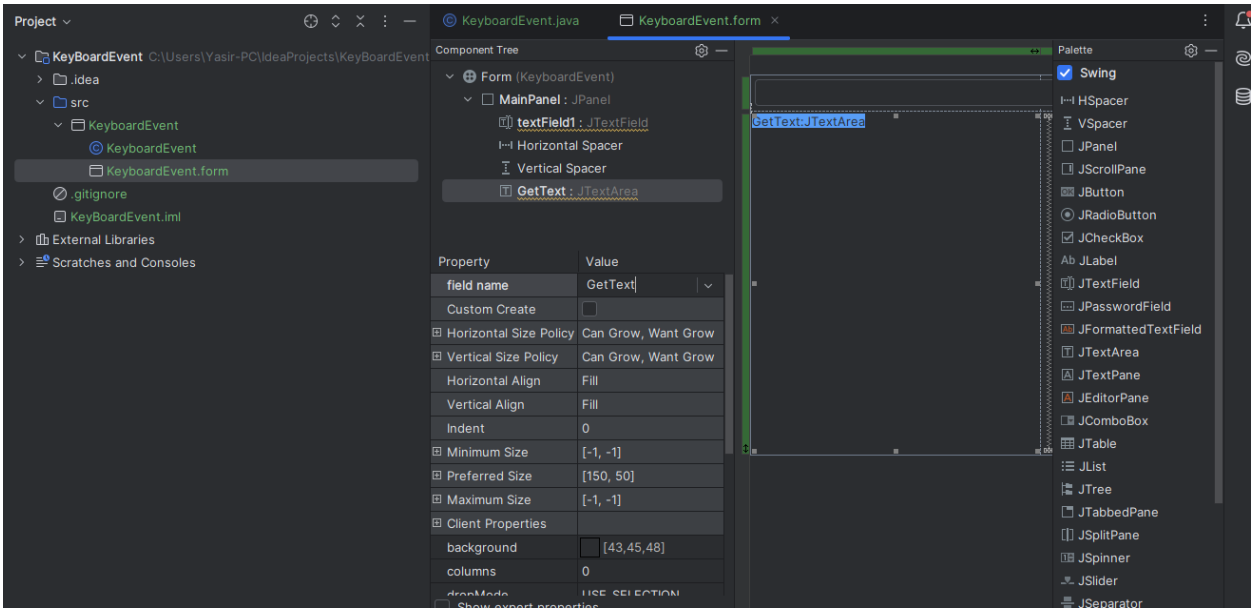


Another Example of Keyboard Event.

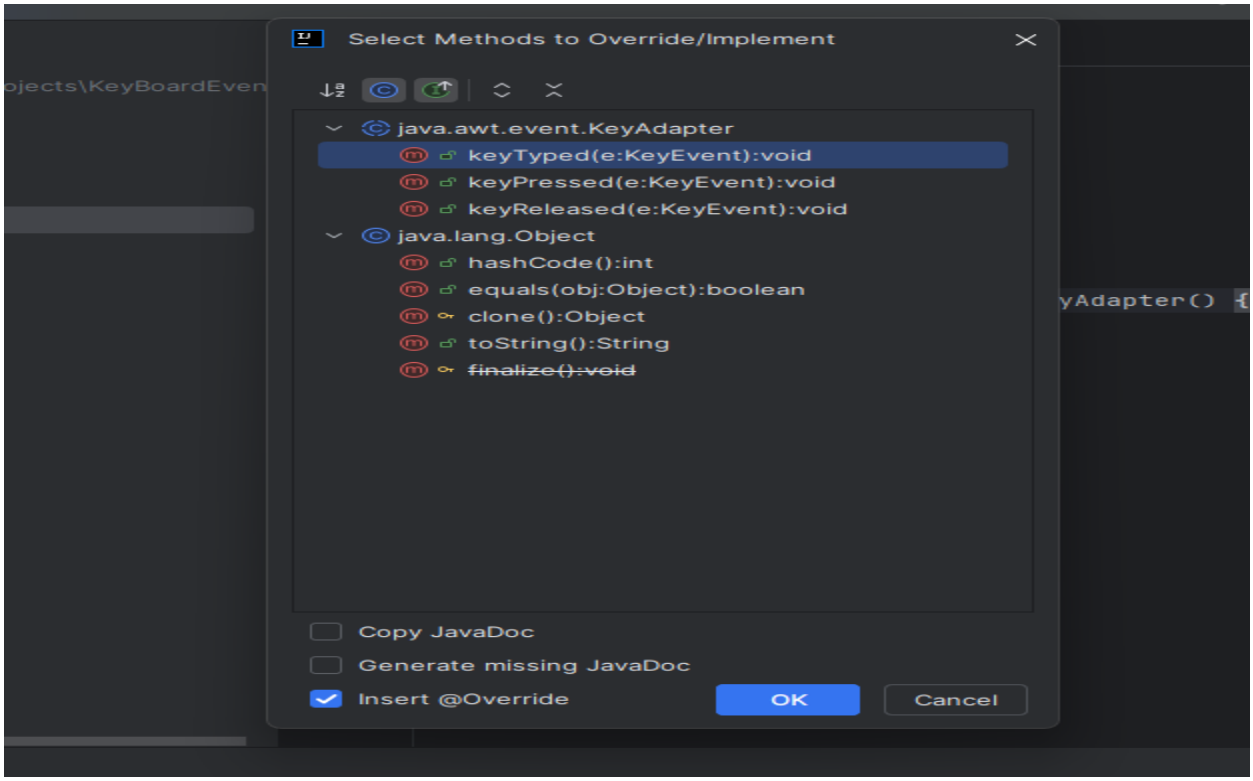
Now we will see one Example of Keyboard Event

Steps

1. Create New Project as KeyboardEvent
2. Now Add a Swing GUI Form
3. Name The JPanel as MainPanel and Add one JTextField (name it as Txt Enter Data) also add one JTextArea (Name it as GetText)



4. Now Right Clik The JTextFiled and Then Click on Create Listner the Select KeyListner after that select your Desired Event



5. Now Add This Code for .java file

```
import javax.swing.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

public class KeyboardEvent extends JFrame {
    private JPanel MainPanel;
    private JTextField textField1;
    private JTextArea GetText;

    public KeyboardEvent() {
        setContentPane(MainPanel);
        setTitle("Keyboard Event");
        setSize(300,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setVisible(true);
        textField1.addKeyListener(new KeyAdapter() {
            @Override
            public void keyTyped(KeyEvent e) {
                super.keyTyped(e);
                GetText.setText(textField1.getText());
            }
        });
    }
```

```
    public static void main(String[] args) {
        KeyboardEvent event = new KeyboardEvent();

    }
}
```

Output