OOP

Lab Manual (Lab 8)

****

Session: Spring 2024

LAB INSTRUCTOR: AYESHA MAJID ALI

To get values from different Java Swing components, you typically need to access the appropriate methods of those components. Here's a general approach for some common Swing components:

1. **Text Fields (JTextField):**

You can use the getText() method to retrieve the text entered into a text field.

String text = textField.getText();

Check Boxes (JCheckBox):

1. **Check Boxes (JCheckBox):**

You can use the isSelected() method to check if the check box is selected or not.

boolean isSelected = checkBox.isSelected();

Radio Buttons (JRadioButton):

1. **Radio Buttons (JRadioButton):**

Radio buttons are often used in groups, so you typically check which one is selected by iterating through the group and checking each button's isSelected() method.

if (radioButton1.isSelected()) {

// radioButton1 is selected

} else if (radioButton2.isSelected()) {

// radioButton2 is selected

}

1. **Combo Boxes (JComboBox):**

You can use the getSelectedItem() method to retrieve the selected item from a combo box.

Object selectedItem = comboBox.getSelectedItem();

List Boxes (JList):

1. **List Boxes (JList):**

If you're using a single-selection list, you can use the getSelectedValue() method to retrieve the selected item. For multi-selection lists, you can use getSelectedValuesList() to retrieve all selected items.

Object selectedValue = list.getSelectedValue();

List<Object> selectedValuesList = list.getSelectedValuesList();

These are just examples of how to retrieve values from common Swing components. Depending on your specific use case and the components you're using, you may need to explore additional methods or event handling mechanisms to capture user input effectively.

**How can we convert the value that we get from text field into some other data type?**

In GUI Swing, you can convert the value obtained from a text field into another data type by first retrieving the text from the text field, then parsing and converting it into the desired data type. Here's a general outline of how you can achieve this in Java:

**Sample Example:**

import javax.swing.\*;

import java.awt.event.\*;

public class TextFieldConversionExample {

public static void main(String[] args) {

JFrame frame = new JFrame("TextField Conversion Example");

JTextField textField = new JTextField(10);

JButton convertButton = new JButton("Convert");

convertButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// Get the text from the text field

String text = textField.getText();

try {

// Convert the text to the desired data type

int intValue = Integer.parseInt(text); // Example: Convert to integer

// Do something with the converted value

System.out.println("Converted value: " + intValue);

} catch (NumberFormatException ex) {

// Handle the case where conversion fails (e.g., invalid input)

System.out.println("Invalid input: " + text);

}

}

});

JPanel panel = new JPanel();

panel.add(textField);

panel.add(convertButton);

frame.add(panel);

frame.setSize(300, 200);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

}

In this example:

* We create a JFrame with a JTextField and a JButton.
* When the button is clicked, we retrieve the text from the text field using textField.getText().
* We then attempt to convert this text into an integer using Integer.parseInt(text).
* If the conversion is successful, we can work with the converted value (intValue in this case). If not, we catch the NumberFormatException which indicates that the input text is not a valid integer.
* You can adapt this code to convert to other data types (e.g., Double.parseDouble(text) for double, Boolean.parseBoolean(text) for boolean, etc.).

**Why are we handling exception in above?**

Handling exceptions, particularly the NumberFormatException, is important for several reasons:

**Input Validation:** It ensures that the user input is valid for the intended operation. In this case, we're expecting the user to input a string representing an integer. If the user enters a string that cannot be parsed as an integer (such as "abc" or "123.45"), the Integer.parseInt() method will throw a NumberFormatException. By catching this exception, we can provide feedback to the user that their input is invalid.

**Sample Example:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class Main {

public static void main(String[] args) {

JFrame frame = new JFrame("Text Field Example");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

JTextField textField = new JTextField(20);

textField.setBounds(50, 100, 100, 30);

frame.add(textField);

JTextField textField2 = new JTextField(20);

textField2.setBounds(50, 150, 100, 30);

frame.add(textField2);

JButton button = new JButton("+");

button.setBounds(50, 200, 100, 30);

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

String value = textField.getText();

String value2 = textField2.getText();

int num1 = Integer.parseInt(value);

int num2 = Integer.parseInt(value2);

int sum = num1 + num2;

JOptionPane.showMessageDialog(frame, "The addition of two numbers is: " + sum);

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(frame, "Please enter valid integers in both fields.", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

});

frame.add(button);

frame.setLayout(null); // setting layout to null so that setBounds work

frame.setSize(300, 300); // setting size manually

frame.setVisible(true);

}

}

TASKS:

TASK 1: **Calculator**

Create a simple calculator with following set of operations.

1. Addition
2. Multiplication
3. Division
4. Subtraction