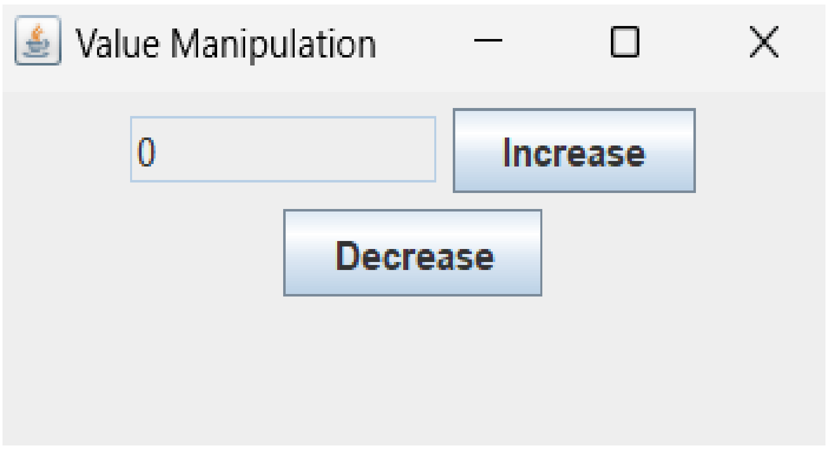
APPLICATIONS AND GAMES GRAPHICAL USER INTERFACE IN JAVA

Button to Increase and Decrease a Value

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

public class ValueManipulationApp extends JFrame {
  private JTextField valueTextField;
  private int currentValue = 0;

  public ValueManipulationApp() {
    setTitle("Value Manipulation");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 150);
    setLayout(new FlowLayout());

    valueTextField = new JTextField(10);
    valueTextField.setEditable(false);
    valueTextField.setText(String.valueOf(currentValue));
    add(valueTextField);

    JButton increaseButton = new JButton("Increase");
    increaseButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
```

```
                currentValue++;
                valueTextField.setText(String.valueOf(currentValue));
            }
        });
        add(increaseButton);

        JButton decreaseButton = new JButton("Decrease");
        decreaseButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                currentValue--;
                valueTextField.setText(String.valueOf(currentValue));
            }
        });
        add(decreaseButton);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new ValueManipulationApp().setVisible(true);
        });
} }
```



Class Structure:

The ValueManipulationApp class represents the main application.

It extends JFrame to create the app window.

Instance Variables:

valueTextField: A JTextField where the current value is displayed.

currentValue: An int variable to store the current numeric value.

Constructor (ValueManipulationApp()**)**:

Initializes the app window (frame).

Sets the window title and default close operation.

Sets the window size to 300x150 pixels.

Uses a FlowLayout for component arrangement.

initUI() **Method**:

Creates the display (valueTextField) at the top of the window.

Sets the display to be non-editable (user cannot directly type into it).

Creates two buttons: “Increase” and “Decrease”.

Adds action listeners to both buttons.

Updates the display text when the buttons are clicked.

ButtonClickListener **Class**:

An inner class that implements the ActionListener interface.

Handles button clicks (action events) for the “Increase” and “Decrease” buttons.

Increments or decrements the currentValue accordingly.

Updates the display text with the new value.

main(String[] args) **Method**:

The entry point of the program.

Creates an instance of ValueManipulationApp and displays the window using SwingUtilities.invokeLater().

Tic Tac Toe Java GUI Code Example

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;


public class TicTacToeGame {

   private static final int BOARD_SIZE = 3;

   private static final int WINNING_LENGTH = 3;

   private static final String EMPTY = "";

   private static final String PLAYER_X = "X";

   private static final String PLAYER_O = "O";


   private JFrame frame;

   private JButton[][] buttons;

   private String currentPlayer;


   public TicTacToeGame() {

      frame = new JFrame("Tic Tac Toe");

      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.setSize(300, 300);

      frame.setLayout(new GridLayout(BOARD_SIZE, BOARD_SIZE));


      buttons = new JButton[BOARD_SIZE][BOARD_SIZE];

      currentPlayer = PLAYER_X;


      initializeBoard();

   }


   private void initializeBoard() {
        for (int row = 0; row < BOARD_SIZE; row++) {
            for (int col = 0; col < BOARD_SIZE; col++) {
                JButton button = new JButton(EMPTY);
                button.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 50));
                button.addActionListener(new ButtonClickListener(row, col));
                buttons[row][col] = button;
                frame.add(button);
            }
        }
    }

    private boolean isWinningMove(int row, int col) {
        String symbol = buttons[row][col].getText();
        return checkRow(row, symbol) || checkColumn(col, symbol) || checkDiagonals(symbol);
    }

    private boolean checkRow(int row, String symbol) {
        for (int col = 0; col < BOARD_SIZE; col++) {
            if (!buttons[row][col].getText().equals(symbol)) {
                return false;
            }
        }
        return true;
    }

    private boolean checkColumn(int col, String symbol) {
        for (int row = 0; row < BOARD_SIZE; row++) {
            if (!buttons[row][col].getText().equals(symbol)) {
                return false;
        }
    }
    return true;
}

private boolean checkDiagonals(String symbol) {
    boolean diagonal1 = true;
    boolean diagonal2 = true;

    for (int i = 0; i < BOARD_SIZE; i++) {
        if (!buttons[i][i].getText().equals(symbol)) {
            diagonal1 = false;
        }
        if (!buttons[i][BOARD_SIZE - 1 - i].getText().equals(symbol)) {
            diagonal2 = false;
        }
    }

    return diagonal1 || diagonal2;
}

private void switchPlayer() {
    currentPlayer = currentPlayer.equals(PLAYER_X) ? PLAYER_O : PLAYER_X;
}

private class ButtonClickListener implements ActionListener {
    private int row;
    private int col;
    public ButtonClickListener(int row, int col) {
        this.row = row;
        this.col = col;
    }

    public void actionPerformed(ActionEvent e) {
        JButton button = (JButton) e.getSource();
        if (button.getText().equals(EMPTY)) {
            button.setText(currentPlayer);
            if (isWinningMove(row, col)) {
                JOptionPane.showMessageDialog(frame, "Player " + currentPlayer + " wins!");
                resetGame();
            } else {
                switchPlayer();
            }
        }
    }
}

private void resetGame() {
    for (int row = 0; row < BOARD_SIZE; row++) {
        for (int col = 0; col < BOARD_SIZE; col++) {
            buttons[row][col].setText(EMPTY);
        }
    }
    currentPlayer = PLAYER_X;
}

public void display() {
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        TicTacToeGame game = new TicTacToeGame();
        game.display();
    }
}
```

Class Structure:

The `TicTacToeGame` class represents the main game.

It initializes the game board, handles button clicks, and checks for winning moves.

Instance Variables:

`frame`: The main window (JFrame) for the game.

`buttons`: A 2D array of JButtons representing the game board.

`currentPlayer`: A string (“X” or “O”) representing the current player.

Constructor (`TicTacToeGame()`):

Initializes the game window (`frame`).

Sets the window title, size, and layout (using `GridLayout`).

Creates the buttons and adds them to the frame.

Initializes the `currentPlayer` to “X”.

`initializeBoard()` Method:

Creates the buttons for the game board.

Sets the font and action listener for each button.

Adds the buttons to the frame.

`isWinningMove(int row, int col)` Method:

Checks if the move at the specified row and column is a winning move.

Calls helper methods (`checkRow`, `checkColumn`, and `checkDiagonals`) to determine if the player has won.

`checkRow(int row, String symbol)` Method:

Checks if all buttons in the specified row have the same symbol (“X” or “O”).

Returns `true` if the row is a winning row.

Other Details:

The game board is a 3x3 grid of buttons.

Players take turns clicking buttons to place their symbol (“X” or “O”).

The game checks for a winning move after each button click.

`checkColumn(int col, String symbol)` Method:

This method checks if all buttons in the specified column have the same symbol (“X” or “O”).

It iterates through each row in the given column and compares the button text with the specified symbol.

If any button in the column does not match the symbol, it returns `false`.

Otherwise, it returns `true`, indicating that the column is a winning column.

`checkDiagonals(String symbol)` Method:

This method checks both diagonals for a winning move.

It initializes two boolean variables (`diagonal1` and `diagonal2`) to `true`.

It iterates through the main diagonal (from top-left to bottom-right) and the secondary diagonal (from top-right to bottom-left).

If any button in either diagonal does not match the specified symbol, the corresponding boolean variable is set to `false`.

The method returns `true` if either diagonal is a winning diagonal (i.e., all buttons have the same symbol).

`switchPlayer()` Method:

This method toggles the current player between “X” and “O”.

If the current player is “X”, it switches to “O”, and vice versa.

`ButtonClickListener` Class:

This is an inner class that implements the `ActionListener` interface.

It handles button clicks (action events) for each button on the game board.

The constructor takes the row and column indices of the clicked button.

In the `actionPerformed` method:

If the clicked button is empty, it sets the button text to the current player's symbol ("X" or "O").

It checks if the move is a winning move using `isWinningMove`.

If the current player wins, it displays a message and resets the game.

Otherwise, it switches the player for the next turn.

resetGame() **Method**:

This method resets the game board to its initial state.

It iterates through all buttons on the board and sets their text to an empty string (clearing any X or O).

It also sets the currentPlayer back to "X" for a fresh start.

display() **Method**:

This method makes the game window visible.

It sets the frame (JFrame) to be visible so that players can interact with the buttons.

main(String[] args) **Method**:

This is the entry point of the program.

It creates an instance of the TicTacToeGame class (the game).

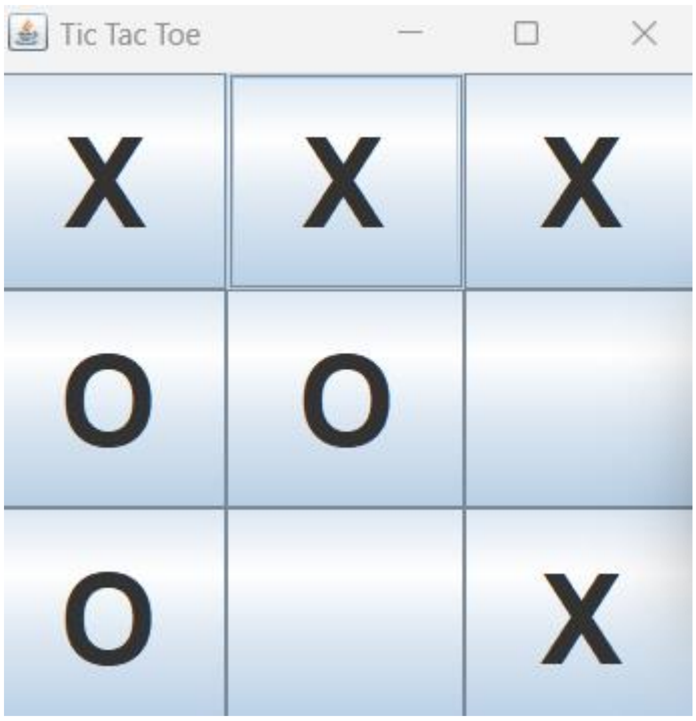
It calls the display() method to show the game window.

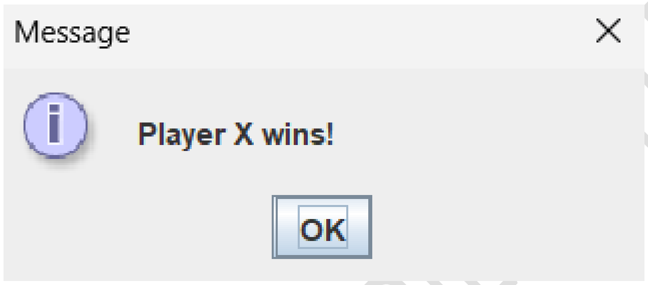
In summary:

The resetGame() method clears the board and resets the player.

The display() method shows the game window.

The main method initializes the game and starts it by displaying the window.

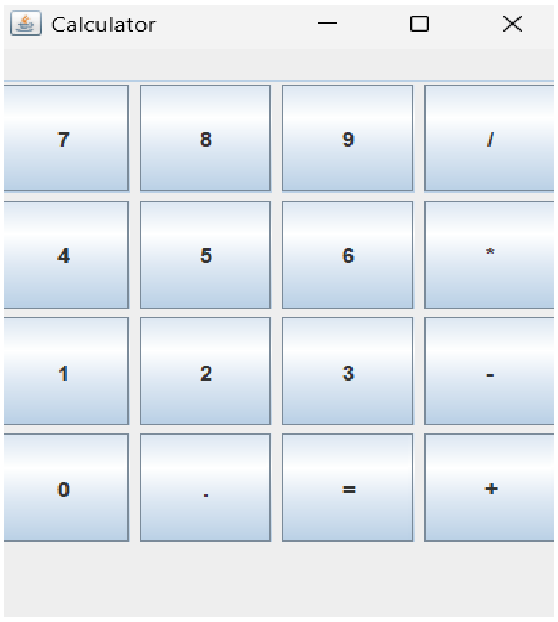




Player X wins!

Calculator GUI Example in Java

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

public class Calculator extends JFrame implements ActionListener {
    private JTextField display;
    private double num1, num2, result;
    private char operator;

    public Calculator() {
        super("Calculator");
        setSize(300, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        display = new JTextField();
        display.setEditable(false);
        add(display, BorderLayout.NORTH);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(5, 4, 5, 5));

        String[] buttonLabels = {
            "7", "8", "9", "/",
            "4", "5", "6", "*",
            "1", "2", "3", "-",
            "0", ".", "=", "+"
    };

    for (String label : buttonLabels) {
        JButton button = new JButton(label);
        button.addActionListener(this);
        buttonPanel.add(button);
    }

    add(buttonPanel, BorderLayout.CENTER);
    setVisible(true);
}

public void actionPerformed(ActionEvent e) {
    String buttonText = ((JButton) e.getSource()).getText();

    if (buttonText.equals("0") || buttonText.equals("1") || buttonText.equals("2") ||
        buttonText.equals("3") || buttonText.equals("4") || buttonText.equals("5") ||
        buttonText.equals("6") || buttonText.equals("7") || buttonText.equals("8") ||
        buttonText.equals("9") || buttonText.equals(".")) {
        display.setText(display.getText() + buttonText);
    } else if (buttonText.equals("+") || buttonText.equals("-") ||
               buttonText.equals("*") || buttonText.equals("/")) {
        num1 = Double.parseDouble(display.getText());
        operator = buttonText.charAt(0);
        display.setText("");
    } else if (buttonText.equals("=")) {
        num2 = Double.parseDouble(display.getText());
        result = calculate();
        display.setText(String.valueOf(result));
        }
    }

    private double calculate() {
        switch (operator) {
            case '+':
                return num1 + num2;
            case '-':
                return num1 - num2;
            case '*':
                return num1 * num2;
            case '/':
                if (num2 != 0) {
                    return num1 / num2;
                } else {
                    JOptionPane.showMessageDialog(this, "Error: Division by zero!");
                    return 0;
                }
            default:
                return 0;
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new Calculator());
    }
}
```



Class Structure:

The Calculator class represents the main calculator application.

It extends JFrame to create the app window.

Instance Variables:

display: A JTextField where the current input and result are shown.

num1, num2: Double variables to store the operands.

result: Double variable to store the calculated result.

operator: Char variable to track the current arithmetic operator.

Constructor (Calculator()**)**:

Initializes the calculator window (frame).

Sets the window title and default close operation.

Sets the window size to 300x400 pixels.

Uses a BorderLayout for component arrangement.

addButton(JPanel panel, String label) **Method**:

Simplifies adding buttons to the panel.

Creates a new JButton with the specified label.

Adds an action listener (handled by this since the class implements ActionListener).

Adds the button to the specified panel.

actionPerformed(ActionEvent e) **Method**:

This method handles button clicks (action events) for all the calculator buttons.

It gets the text of the clicked button using ((JButton) e.getSource()).getText().

If the button text is a digit (0 to 9) or a decimal point (.), it appends it to the display text.

If the button text is an operator (+, -, *, /), it does the following:

Parses the current display text as a double and assigns it to num1.

Sets the operator to the first character of the button text.

Clears the display text.

If the button text is the equals sign (=), it does the following:

Parses the current display text as a double and assigns it to num2.

Calls the calculate() method to perform the actual arithmetic operation.

Displays the result in the text field.

calculate() **Method**:

This method performs the actual arithmetic operations based on the current operator.

It switches on the operator:

For addition (+), it returns the sum of num1 and num2.

For subtraction (-), it returns the difference between num1 and num2.

For multiplication (*), it returns the product of num1 and num2.

For division (/), it checks if num2 is not zero, and if so, returns the quotient of num1 divided by num2. Otherwise, it shows an error message and returns 0.

main(String[] args) **Method**:

The entry point of the program.

Creates an instance of Calculator and displays the window using SwingUtilities.invokeLater().