Report

Electronic Voting Machine (EVM) GUI

Abdul Rehman

133-22-0031

Certificate

It is certified that **Abdul Rehman** a student of **BE-III** has carried out the necessary work of **OOP** as per course of studies prevailed at the Computer System Engineering Department, Sukkur Institute of Business Administration for **FALL-2023.**

Date: _____ Instructor's Signature

Abstract

This project is about creating a simple and easy-to-use **Electronic Voting Machine (EVM)** using **Java** with a graphical user interface. The main idea is to let users cast their votes digitally in a smooth and secure way. To make sure the process is fair, each voter has a unique voter ID, and the system checks that no one votes more than once. The voting screen shows a list of candidates along with their symbols—just like in real elections. Once a voter selects a candidate, their vote is recorded and counted automatically. When the voting ends, the system shows the total number of votes for each candidate and announces the winner clearly. This project not only helps understand important programming concepts like GUI design, input checking, and event handling, but it also gives a glimpse into how digital voting systems can work in real life. It's a good starting point for building larger, more advanced e-voting systems in the future.

Introduction

In today's world, where almost everything is going digital, the need for modern and secure voting systems is more important than ever. Traditional paper-based voting is not only time-consuming but also comes with the risk of errors, fraud, and delayed results. To solve these problems, this project introduces a **simple and user-friendly Electronic Voting Machine (EVM)** developed using **Java** and its **Swing GUI** framework.

The purpose of this EVM is to make the voting process **faster, more reliable, and more secure**. It allows voters to cast their vote using a computer interface by entering a **valid Voter ID** and selecting their preferred candidate from a list. Each candidate is represented by their **name and election symbol**, just like in real-world elections. The system ensures that a person can **only vote once**, and results can be displayed instantly with a single click.

This project serves as a practical example of how **Java programming** can be used to solve real-life problems. It demonstrates the use of **object-oriented concepts**, **event-driven programming**, and **basic data validation**. While it is a small-scale model, it lays the foundation for understanding how electronic voting could be implemented on a larger scale in a secure and efficient way.

Advantages

 User-Friendly Interface
The system uses a graphical interface built with Java Swing, making it easy for users to interact and cast their votes with just a few clicks.

 Secure Voting Mechanism
Each voter must enter a valid Voter ID, and the system checks if the voter has already voted. This prevents duplicate voting.

 Instant Result Generation
The system can instantly display the total number of votes for each candidate and declare the winner without any manual counting.

 Paperless System
Since everything is done digitally, it eliminates the need for printing and handling paper ballots, helping to reduce environmental waste.

 Educational Value
The project provides a practical example for learning Java programming concepts such as classes, objects, GUI components, event handling, and data structures (like HashMaps).

Limitations

1. **Limited Voter IDs**
   The system only works for a small set of hardcoded voter IDs. It doesn't support dynamic voter registration or a large voter database.
2. **No Data Persistence**
   Votes are not saved permanently. If the application is closed or restarted, all votes are lost as there's no database or file storage used.
3. **Single-System Usage**
   The voting machine is not networked, so it can only be used on a single machine. This limits its scalability for real elections.
4. **Lack of Authentication**
   There's no advanced authentication system (like biometric or OTP verification), which would be necessary for a real-world secure election process.
5. **Fixed Candidates**
   The list of candidates is hardcoded and cannot be updated dynamically without modifying the source code.

Methodology

To build the Electronic Voting Machine (EVM), we followed a simple and logical approach that ensured the system was easy to use, reliable, and secure. The entire process was carried out step-by-step, from designing the interface to validating voter input and counting votes accurately.

1. Understanding the Goal

Our first step was to understand what we wanted this voting machine to do. The main idea was to allow registered users to vote for their preferred candidate, make sure that each voter could only vote once, and then show the results clearly — all through a graphical interface.

2. Designing the System

We began by designing the building blocks of the system:

- We created a Candidate class to store each candidate's name, symbol, and vote count.
- A graphical interface (GUI) was designed using **Java Swing**, with buttons for each candidate, a text field to enter voter ID, and a results section.

3. Adding Voter Validation

We used a list of pre-defined voter IDs to make sure that only valid voters could vote. Once a voter used their ID to cast a vote, their status was marked to prevent them from voting again. This helped maintain the integrity of the process.

4. Casting Votes

When a user enters a valid ID and clicks a candidate's button:

- The system checks if the ID is valid and hasn't been used already.
- If everything is okay, the system adds one vote to the selected candidate.
- A message is shown to confirm that the vote has been successfully recorded.

5. Displaying Results

Once all votes are cast, clicking the **"Show Result"** button will:

- Count and display how many votes each candidate received.
- Highlight the candidate with the most votes as the winner.
- Present the results in a neat popup window for users to see.

6. Testing the System

We tested the application with different types of inputs:

- Correct and incorrect voter IDs
- Duplicate voting attempts
- Voting for each candidate
  This helped us catch errors and improve the system before finalizing it.

7. Final Touches

Finally, we polished the interface to make it more user-friendly. Clear messages were added to guide voters through each step, and the layout was kept simple to make voting quick and easy — just like using a real EVM.

Algorithm

The voting system follows a simple, step-by-step algorithm to ensure accurate voting, prevent double voting, and display results clearly. Here's how the logic flows:

Step 1: System Initialization

- Load the list of valid voter IDs (e.g., 11, 22, 33, etc.).
- Initialize all candidates with their names, symbols, and vote count set to zero.

Step 2: Voter ID Input

- User enters their voter ID into a text field.
- The system checks:
  - If the voter ID exists in the list.
  - If the voter has already voted (to prevent repeat voting).

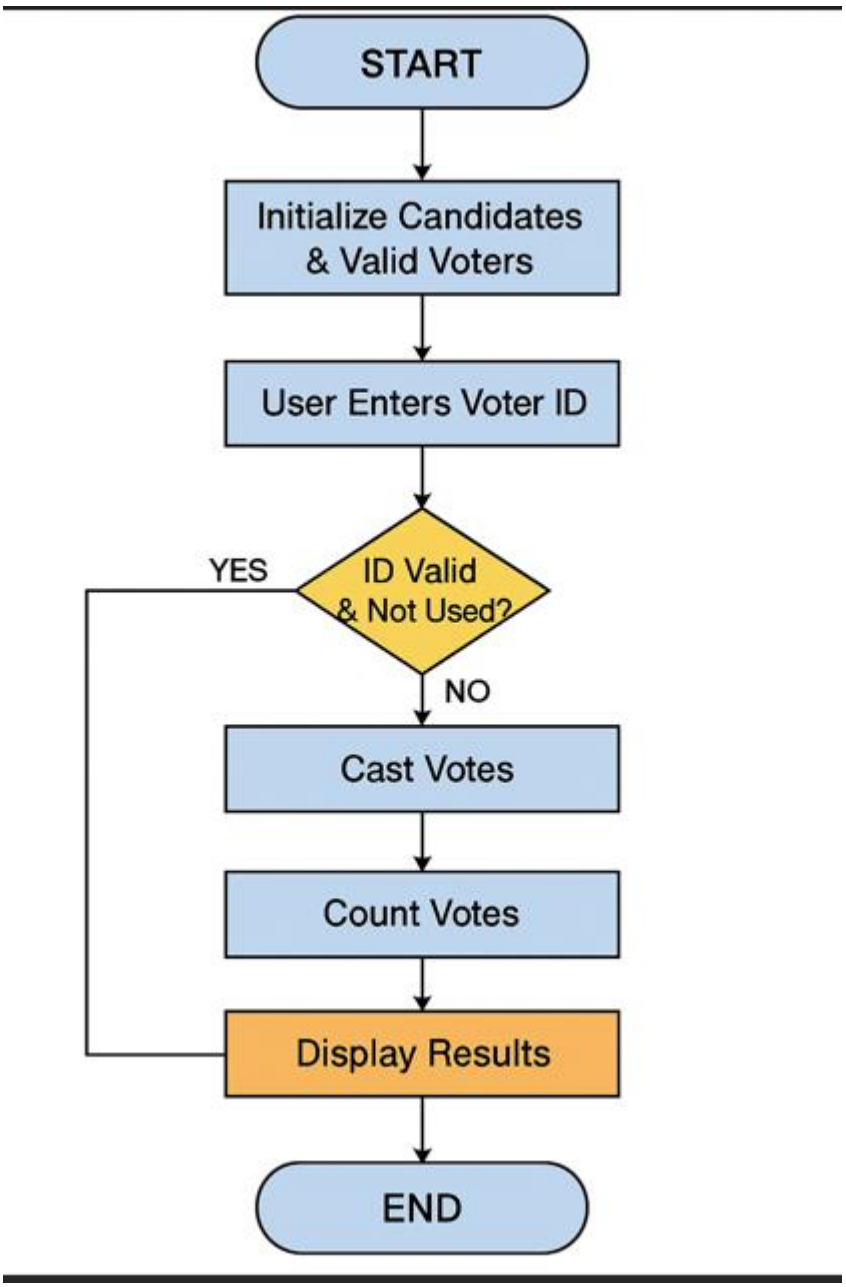
Step 3: Vote Casting

- If the voter is **valid and hasn’t voted yet**:
  - Voter clicks a candidate's button (e.g., “Imran Khan (Bat)”).
  - The system:
    - Adds **+1 vote** to the selected candidate.
    - Marks that voter ID as “used” (so it can’t be used again).
    - Displays confirmation that the vote was successfully recorded.
- If the voter ID is **invalid or already used**, an error message is shown.

Step 4: Result Generation

- When the **"Show Result"** button is clicked:
  - The system:
    - Counts the votes for each candidate.
    - Identifies the candidate with the highest vote count.
  - The result is displayed in a formatted message showing:
    - All candidates and their vote totals.
    - The winner’s name and symbol.

Flow Chart:


START
Initialize Candidates & Valid Voters
User Enters Voter ID
ID Valid & Not Used?
YES
NO
Cast Votes
Count Votes
Display Results
END

Code :

Software Requirements
1: BLUE J

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

class Candidate {
    private String name;
    private String symbol;
    private int votes;

    public Candidate(String name, String symbol) {
        this.name = name;
        this.symbol = symbol;
        this.votes = 0;
    }

    public String getName() {
        return name;
    }

    public String getSymbol() {
        return symbol;
    }

    public int getVotes() {
        return votes;
    }

    public void vote() {
        votes++;
    }
}

class VotingMachineGUI extends JFrame {
    private Map<Integer, Candidate> candidates;
    private JTextArea votedListTextArea;
    private JTextField voterIdTextField;
```

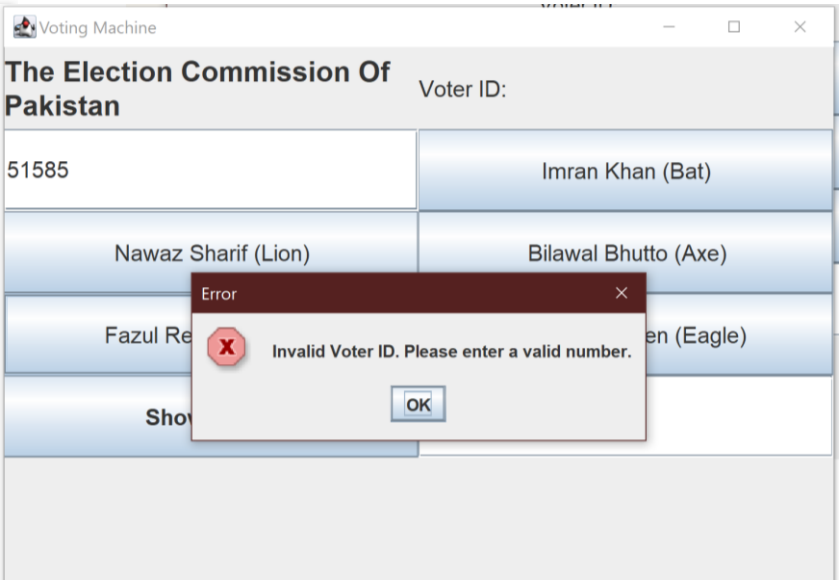
```
    private final int[] validVoterIds = {11, 22, 33, 44, 55, 66, 77, 88, 99, 100};
    private boolean[] votedStatus;

    public VotingMachineGUI() {
        candidates = new HashMap<>();
        initializeCandidates();
        votedStatus = new boolean[validVoterIds.length];
        createGUI();
    }

    private void initializeCandidates() {
        candidates.put(1, new Candidate("Imran Khan", "Bat"));
        candidates.put(2, new Candidate("Nawaz Sharif", "Lion"));
        candidates.put(3, new Candidate("Bilawal Bhutto", "Axe"));
        candidates.put(4, new Candidate("Fazul Rehman", "Book"));
        candidates.put(5, new Candidate("Jahangir Tareen", "Eagle"));
    }

    private void createGUI() {
        setTitle("Voting Machine");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 500);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(8, 1));

        // Adding Election Commission Of Pakistan in Bold
        JLabel electionCommissionLabel = new JLabel("<html><b>The Election Commission
Of Pakistan</b></html>");
        electionCommissionLabel.setHorizontalAlignment(JLabel.CENTER);
        electionCommissionLabel.setFont(new Font("Arial", Font.BOLD, 20));
        panel.add(electionCommissionLabel);

        // Adding JTextField for Voter ID
        JLabel voterIdLabel = new JLabel("Voter ID:");
        voterIdLabel.setFont(new Font("Arial", Font.PLAIN, 16));
        voterIdTextField = new JTextField();
        voterIdTextField.setFont(new Font("Arial", Font.PLAIN, 16));
        panel.add(voterIdLabel);
        panel.add(voterIdTextField);

        for (Map.Entry<Integer, Candidate> entry : candidates.entrySet()) {
            Candidate candidate = entry.getValue();
            JButton button = new JButton(candidate.getName() + " (" + candidate.getSymbol() +
")");
```

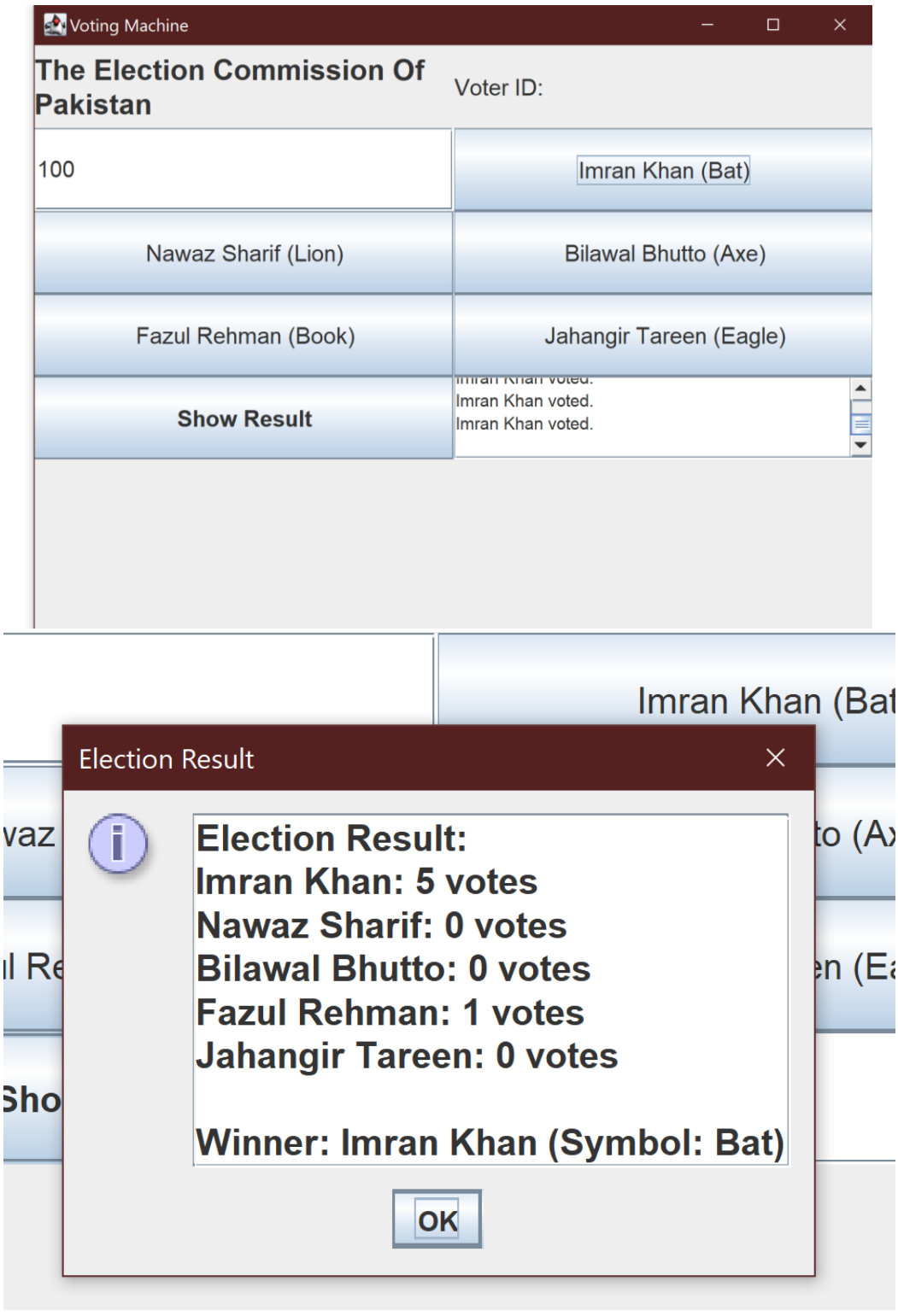
```
            button.addActionListener(new VoteButtonListener(entry.getKey()));
            button.setFont(new Font("Arial", Font.PLAIN, 16));
            panel.add(button);
        }

        JButton resultButton = new JButton("Show Result");
        resultButton.addActionListener(new ResultButtonListener());
        resultButton.setFont(new Font("Arial", Font.BOLD, 16));
        panel.add(resultButton);

        votedListTextArea = new JTextArea();
        votedListTextArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(votedListTextArea);

        panel.add(scrollPane);

        add(panel);
    }

    private class VoteButtonListener implements ActionListener {
        private int candidateId;

        public VoteButtonListener(int candidateId) {
            this.candidateId = candidateId;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            String voterIdStr = voterIdTextField.getText().trim();

            if (voterIdStr.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Please enter Voter ID.", "Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }

            try {
                int voterId = Integer.parseInt(voterIdStr);

                // Check if the entered voterId is valid
                int voterIndex = getVoterIndex(voterId);
                if (voterIndex != -1 && !votedStatus[voterIndex]) {
                    Candidate candidate = candidates.get(candidateId);
                    candidate.vote();
                    votedStatus[voterIndex] = true; // Mark the voter as voted
                    votedListTextArea.append(candidate.getName() + " voted.\n");
```

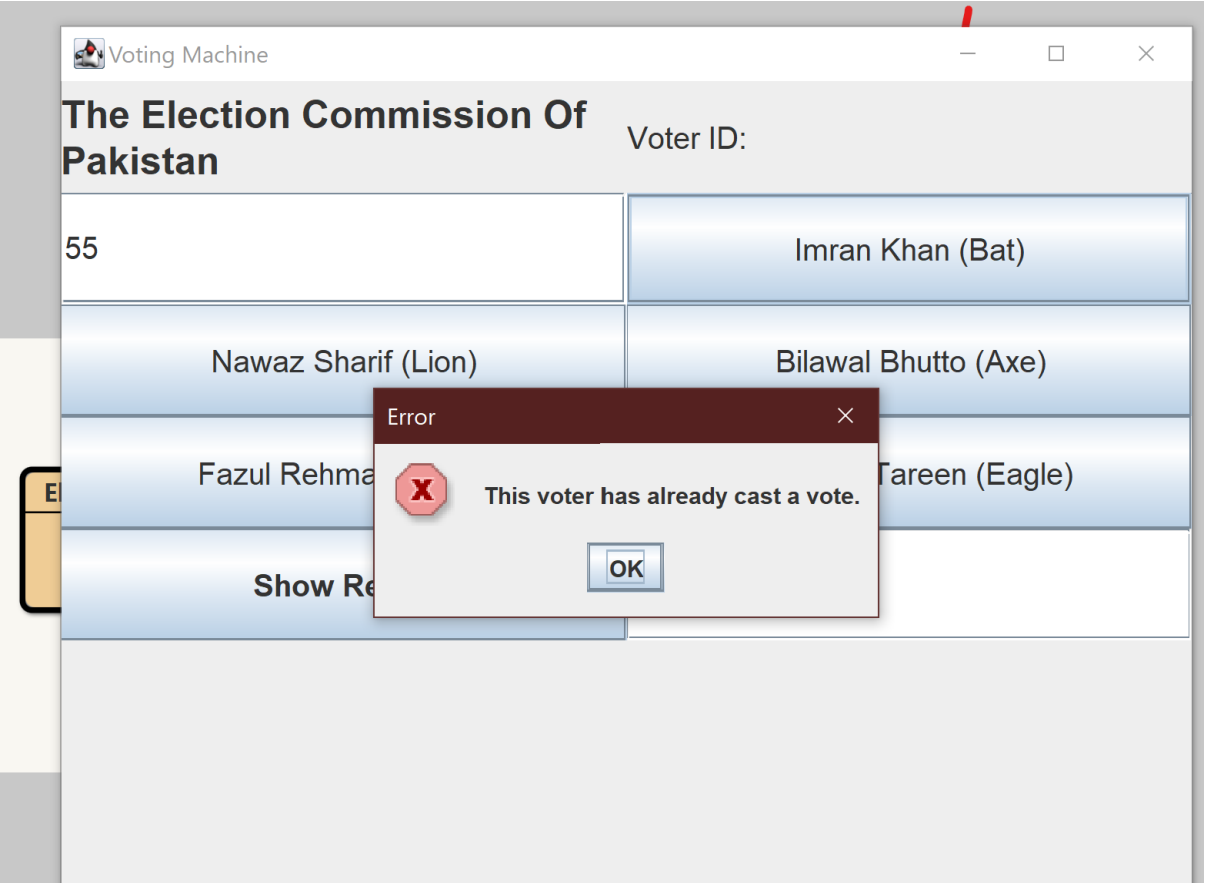
```
            } else if (voterIndex == -1) {
                JOptionPane.showMessageDialog(null, "Invalid Voter ID. Please enter a valid
number.", "Error", JOptionPane.ERROR_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(null, "This voter has already cast a vote.",
"Error", JOptionPane.ERROR_MESSAGE);
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null, "Invalid Voter ID. Please enter a valid
number.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private int getVoterIndex(int voterId) {
        for (int i = 0; i < validVoterIds.length; i++) {
            if (voterId == validVoterIds[i]) {
                return i;
            }
        }
        return -1;
    }
}

private class ResultButtonListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        StringBuilder result = new StringBuilder("Election Result:\n");

        Candidate winner = null;
        for (Candidate candidate : candidates.values()) {
            result.append(candidate.getName()).append(":
").append(candidate.getVotes()).append(" votes\n");
            if (winner == null || candidate.getVotes() > winner.getVotes()) {
                winner = candidate;
            }
        }

        if (winner != null) {
            result.append("\nWinner: ").append(winner.getName()).append(" (Symbol:
").append(winner.getSymbol()).append(")");
        }

        JTextArea resultTextArea = new JTextArea(result.toString());
        resultTextArea.setFont(new Font("Arial", Font.BOLD, 16));
        resultTextArea.setEditable(false);
```

```
            JOptionPane.showMessageDialog(null, new JScrollPane(resultTextArea), "Election
Result", JOptionPane.INFORMATION_MESSAGE);
        }
    }
}

public class ElectionCommisionOFPakistan {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            VotingMachineGUI votingMachineGUI = new VotingMachineGUI();
            votingMachineGUI.setVisible(true);
        });
    }
}
```

Results:




Voting Machine
The Election Commission Of Pakistan
Voter ID:
100
Imran Khan (Bat)
Nawaz Sharif (Lion)
Bilawal Bhutto (Axe)
Fazul Rehman (Book)
Jahangir Tareen (Eagle)
Show Result
Imran Khan voted.
Imran Khan voted.
Election Result
Election Result:
Imran Khan: 5 votes
Nawaz Sharif: 0 votes
Bilawal Bhutto: 0 votes
Fazul Rehman: 1 votes
Jahangir Tareen: 0 votes
Winner: Imran Khan (Symbol: Bat)
OK



Applications

The Electronic Voting Machine (EVM) developed in this project can be applied in various practical scenarios, especially where secure and efficient vote collection is needed. Its simple interface, input validation, and secure vote handling make it ideal for the following areas:

1. **Educational Institutions**
   The system can be used in schools, colleges, and universities for conducting student council elections, mock voting sessions, or classroom-based democratic exercises. It offers a hands-on way for students to understand the voting process.
2. **Corporate and Organizational Voting**
   This voting system can be used in offices, departments, or clubs for internal elections, team leader selections, or collecting opinions during meetings or decision-making sessions.
3. **Community and Local Elections**
   Small-scale elections such as neighborhood committees or non-profit organizational elections can benefit from this system, offering a transparent and digital alternative to paper-based methods.
4. **Surveys and Feedback Collection**
   With slight modifications, this system can also be used for collecting responses in public opinion polls, feedback forms during workshops, or interactive classroom surveys.

5. **Learning and Demonstration Tool**
   The project serves as an excellent example for students and beginner developers to learn Java GUI design, event-driven programming, and secure input handling. It demonstrates how real-world applications can be developed with core programming skills.
6. **Secure Input Systems**
   With further development, this system can be adapted for use in secure data entry scenarios such as quizzes, controlled registration systems, or simple authentication tasks.

Conclusion

This Electronic Voting Machine project is a practical example of how technology can make our lives easier and more secure—especially when it comes to something as important as voting. By using Java and a simple graphical interface, we've created a system that helps people vote easily, while also making sure their votes are counted fairly.

One of the best parts of this system is that it only lets registered voters vote—and just once. This means no fake votes, no repeated votes, and a clear result at the end. With the click of a button, votes are counted, and the winner is shown instantly, which saves both time and effort.

While this project is just a basic model, it opens the door for bigger and better ideas—like adding fingerprint scanners or even making voting possible from home using the internet. It shows how combining programming with real-world problems can lead to smart and helpful solutions.

In short, this project is more than just code—it's a small step toward building a fair, transparent, and modern voting system for everyone.

References

1. **Java Swing Tutorial**
   Learn how to build GUI applications using Swing in Java.
   🔗 https://docs.oracle.com/javase/tutorial/uiswing/

2. **Java HashMap Documentation**
   Official guide on how to use the `HashMap` class in Java, which is used to store candidates.
   🔗 https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
3. **Java Event Handling**
   Guide to understanding `ActionListener` and handling button events in GUI applications.
   🔗 https://www.javatpoint.com/java-event-handling
4. **SwingUtilities.invokeLater() Explanation**
   Why and how Swing applications are run on the Event Dispatch Thread (EDT).
   🔗 https://docs.oracle.com/javase/tutorial/uiswing/concurrency/initial.html
5. **Java JOptionPane for Dialogs**
   Learn how to create alert boxes and input prompts in a GUI.
   🔗 https://www.geeksforgeeks.org/java-swing-joptionpane/
6. **Basic Electronic Voting Concepts**
   Understand how digital voting works, the need for validation, and secure output.
   🔗 https://en.wikipedia.org/wiki/Electronic_voting