

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY, BELAGAVI 590018**



Project Report on

“ELECTRICITY BILLING SYSTEM”

By

**Mallikarjun (1BM24CS160)
Mahendra Gowda (1BM24CS156)**

**Under the Guidance of
Monisha H M
Assistant Professor, Department of CSE
BMS College of Engineering**

Work carried out at



**Department of Computer Science and Engineering
BMS College of Engineering
(Autonomous college under VTU)
P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019**

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the OOPS with JAVA project titled “Your title” has been carried out by **Mallikarjun (1BM24CS160)**, **Mahendra Gowda (1BM24CS156)** during the academic year 2025-2026.

Signature of the guide

Monisha H M

Assistant Professor,
Department of Computer Science and Engineering BMS
College of Engineering, Bangalore

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



DECLARATION

We, **Mallikarjun (1BM24CS160)**, **Mahendra Gowda (1BM24CS156)** students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this project work entitled "**ELECTRICITY BILLING SYSTEM**" has been carried out by us under the guidance of **Monisha H M**, Assistant Professor, Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep-Dec 2025. We also declare that to the best of our knowledge and belief, the project reported here is not from part of any other report by any other students.

Signature of the Candidates

Mouneshwar (1BM24CS175)

Nagaraj G M (1BM24CS176)

Nagarajun (1BM24CS178)

Mahesh Gopal Lamani (1BM24CS157)

TABLE OF CONTENTS

CHAPTER NO.	CONTENT	PAGE NO.
1.	Problem Statement	
2.	Introduction	
3.	Overview of the project	
4.	Tools Used	
5.	OOPs concept used & its explanation	
6.	Implementation	
7.	Snapshots	
8.	References	

Problem Statement

College Event Registration System

Problem:

To develop a Java Swing-based application that allows students to register for various college events through a graphical user interface. The system should:

- Accept student details (Name, USN, Branch) through text input fields
- Provide a dropdown selection of available events
- Validate input data to ensure all required fields are filled
- Store registration data in memory using ArrayList
- Display all registered students in a tabular format
- Provide real-time feedback through status messages
- Handle user interactions through event-driven programming

Objectives:

- To implement a user-friendly GUI for event registration
- To demonstrate object-oriented programming principles
- To create an efficient in-memory data storage system
- To provide error handling for incomplete submissions
- To offer an organized view of all registrations

INTRODUCTION

The College Event Registration System is a GUI-based software application developed using Object Oriented Programming in Java (Swing Framework).

This project aims to computerize and simplify the student registration process for college technical and cultural events.

In the present conventional system, event registration is mostly handled manually using paper forms or unorganized spreadsheets.

This leads to several issues such as:

- Loss of records
- Duplicate entries
- Time-consuming verification
- Lack of centralized access
- Difficulty in tracking participation per event or department

To overcome these challenges, we designed a user-friendly, efficient, and fully computerized event registration platform with the following key features:

- i. Allows students to **register for events through an interactive GUI** without paperwork
- ii. Stores all registrations in a **structured dynamic list (ArrayList)** ensuring easy data management
- iii. Provides an option to **view all registered student details in a tabular format (JTable)**
- iv. Ensures **accuracy, speed, and efficiency** in handling student participation data
- v. Eliminates the need for staff involvement in delivering or collecting forms

Once the system is launched, students can enter their details such as **Name, USN, Branch, and select an event**, and register instantly.

The admin or event coordinator can later view and track all participation details without maintaining manual logs.

The software requires **very small storage**, runs on any standard system supporting Java, and allows **future extensibility** such as database integration, authentication modules, or event filtering.

Preamble

We, the developers and owners of this project, respect the importance of college events in student development and aim to make the **event registration process transparent, fast, and hassle-free**.

The main objectives of our project are:

- To store student participation records securely
- To simplify event registration using GUI forms
- To allow students and coordinators to view participation details easily
- To avoid duplicate or missing entries
- To reduce human effort, time, and paper usage

This project demonstrates the use of **core OOP principles** and real-world software design suitable for **3rd-semester engineering students**, ensuring it is both **academically relevant**.

Overview of the project

Preliminary Design

System design is an abstract representation of system components and their relationships, which describes the aggregated functionality and performance of the system. The **College Event Registration System** is designed using **Java Swing for the frontend** and **ArrayList for in-memory data storage**.

The system follows a simple but effective architecture where:

1. Registration data is stored in an **ArrayList of Registration objects**
2. GUI components are organized using **GridLayout**
3. Event handling is implemented using **ActionListeners**
4. Data validation ensures complete information before registration

Class Structure

The system consists of two main classes:

1. Registration Class:
 - Data class to store registration details
 - Attributes: name, usn, branch, event
 - Constructor to initialize all attributes
2. CollegeEventRegistrationGUI Class:
 - Main class containing GUI implementation
 - Static ArrayList to store all registrations
 - GUI components: JFrame, JLabels, JTextFields, JComboBox, JButtons, JTable
 - Event handlers for registration and viewing

Database Design (In-Memory)

Since this is a simple desktop application, we use **ArrayList for data storage instead of a database**. However, the structure follows database principles:

- Each Registration object represents a row in a virtual table
- The ArrayList acts as the table storing all registrations
- Columns: Name, USN, Branch, Event

Class Diagram

Name	USN	Branch	Event
------	-----	--------	-------

Data Flow

1. User enters details in text fields
2. User selects event from dropdown
3. User clicks "**Register**" button
4. Data is validated
5. If valid: Registration object is created and added to ArrayList
6. User can click "**View Registrations**" to see all entries in a JTable

Normalization Concepts

Though we're using in-memory storage, the design follows normalization principles:

First Normal Form (1NF):

- Each registration has a unique combination of attributes
- Each cell contains a **single value**
- Values are atomic (cannot be split further)

Entity Integrity:

- Each registration is **uniquely identifiable**
- No duplicate registrations for the **same USN in the same event** (*implementable enhancement*)

UML Diagram

Class Diagram:

College Event Registration GUI

regList: ArrayList<Registration>

main(args: String[]): void

- initializeGUI(): void

- handleRegistration(): void

- handleViewRegistrations(): void



Registration

Name

USN

Branch

Event

Registration(n,u,b,e)

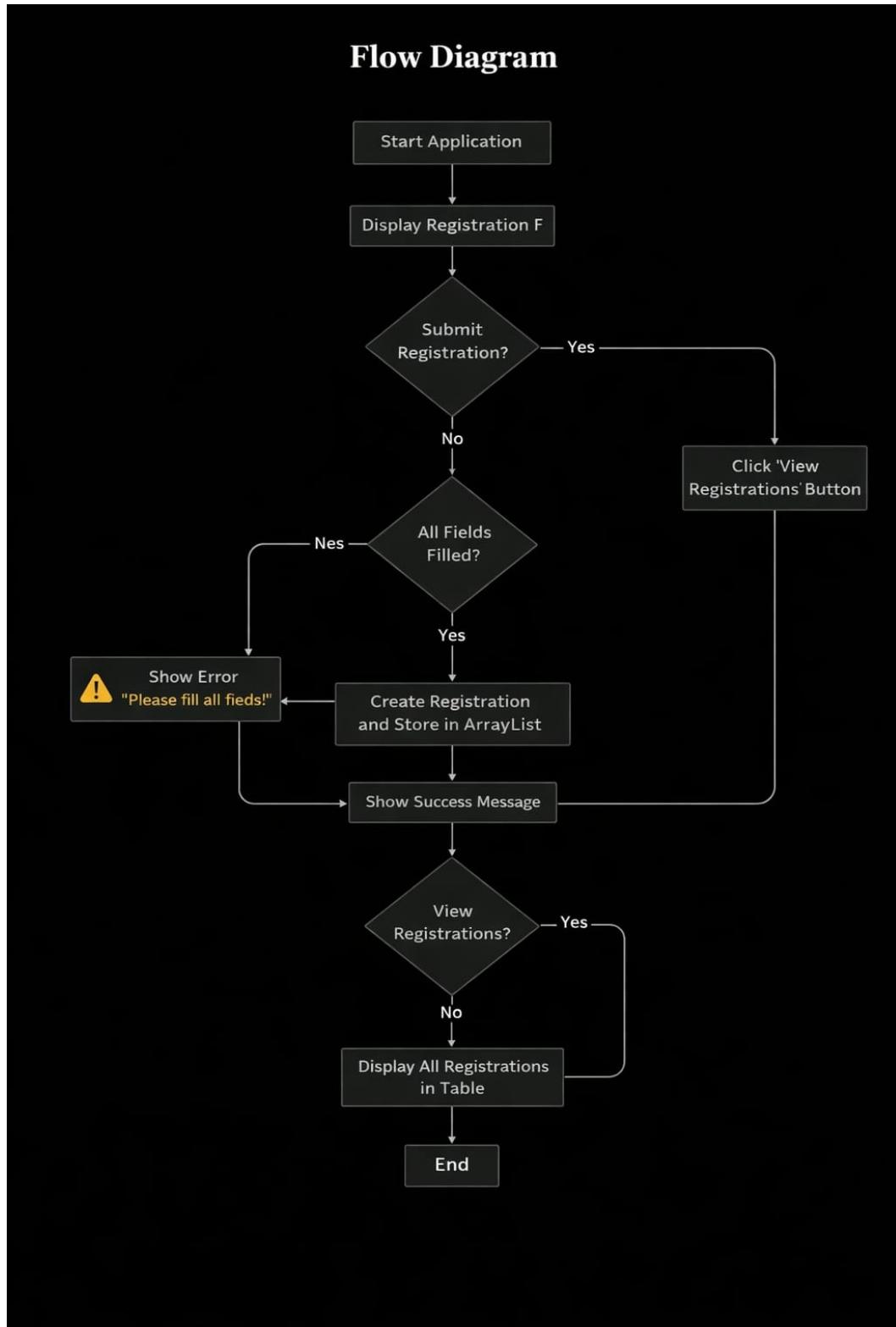
 getName():

 getUsn():

 getBranch():

 getEvent():

Flow Diagram:



Caption

Tools Used

Existing and Proposed System

The conventional system of college event registration is not very effective, as students register using paper forms or separate files submitted to different coordinators. One staff collects student details, another verifies entries, and later participation lists are prepared manually. Again, the final registered data must be displayed or shared through notice boards or messages. This process becomes lengthy, repetitive, and difficult to manage. To solve this, a GUI-based computerized system is essential. The proposed College Event Registration System overcomes these drawbacks by providing a centralized, fast, and interactive registration platform. It benefits students and event coordinators by reducing manual work. The system delivers high performance with improved speed and accuracy, minimizing entry errors and eliminating dependency on physical records.

Software & Hardware Requirements

Hardware Requirements:

- Hardware Specification: -Processor Intel Pentium V or higher
- Clock Speed: -1.7 GHz or more
- System Bus: -64 bits
- RAM: -16GB
- HDD: -2TB
- Monitor: -LCD Monitor
- Keyboard: -Standard keyboard
- Mouse: -Compatible mouse

Software Requirements:

- Operating System: -Windows 10
- Software: -Microsoft SQL Server
- Front End: -Java core/swings (NetBeans)
- Back End: -My SQL

OOPs concept used & it's Explanation

Class and Object

```
class Registration {  
    String name, usn, branch, event;  
    Registration(String n, String u, String b, String e) {  
        name = n; usn = u; branch = b; event = e;  
    }  
}
```

Concept: Class as blueprint and object as instance

Implementation:

- Registration class defines structure for registration data
- Objects created with new Registration(...) store individual student data
- Each registration is a separate object instance

Benefit: Modular data organization and encapsulation

Encapsulation

```
public class CollegeEventRegistrationGUI {  
    static ArrayList<Registration> regList = new ArrayList<>();  
    // Data is protected within class  
}
```

Concept: Bundling data and methods within a single unit

Implementation:

- regList is static and accessible within class
- Data manipulation through controlled methods
- GUI components encapsulated in main class

Benefit: Data security and controlled access

Inheritance

```
// Implicit inheritance from Object class
public class CollegeEventRegistrationGUI {
    // Inherits methods from Object: toString(), equals(), hashCode()
}
```

Concept: Creating new classes based on existing classes

Implementation:

- All Java classes implicitly extend Object class
- GUI components inherit from JComponent hierarchy
- Swing components follow inheritance hierarchy

Benefit: Code reusability and hierarchical organization

Polymorphism

```
b1.addActionListener(e -> {
    // Lambda expression implementing ActionListener
});
```

Concept: One interface, multiple implementations

Implementation:

- Lambda expressions for event handling
- Method overriding in anonymous classes
- Interface-based programming with ActionListener

Benefit: Flexible and concise code structure

Exception Handling

```
if(name.isEmpty() || usn.isEmpty() || branch.isEmpty()) {  
    msg.setText("⚠ Please fill all fields!");  
    msg.setForeground(Color.RED);  
}
```

Concept: Handling runtime errors gracefully

Implementation: Custom checked exception for invalid characters

Benefit: Robust error handling and user feedback

Association

```
static ArrayList<Registration> regList = new ArrayList<>();
```

Concept: Relationship between classes

Implementation:

- **CollegeEventRegistrationGUI has association with Registration class**
- **One-to-many relationship through ArrayList**
- **Composition relationship between components**

Benefit: Clear object relationships and data management

Implementation

Registration class

```
class Registration {  
    String name, usn, branch, event;  
    Registration(string n, string u, string b, string e) {  
        name = n; usn = u; branch = b; event = e;  
    }  
}
```

Purpose: Data model for storing registration information

Design Pattern: Plain Old Java Object (POJO) pattern

Key Features:

- Simple data container class
- No complex logic or methods
- Easy to serialize and deserialize
- Can be extended with getters/setters

```
public class CollegeEventRegistrationGUI {  
    static ArrayList<Registration> regList = new ArrayList<>();  
  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("College Event Registration");  
        frame.setSize(400, 350);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setLayout(new GridLayout(6, 2, 5, 5));  
        // ... GUI component initialization  
    }  
}
```

GUI Components Initialization

```
// Input Fields
JLabel l1 = new JLabel("Student Name:");
JTextField t1 = new JTextField();

JLabel l2 = new JLabel("USN:");
JTextField t2 = new JTextField();

JLabel l3 = new JLabel("Branch:");
JTextField t3 = new JTextField();

// Event Selection
JLabel l4 = new JLabel("Select Event:");
String events[] = {"Hackathon", "Coding Contest", "Robotics",
| | | | | "Paper Presentation", "Tech Quiz"};
JComboBox<String> cb = new JComboBox<>(events);

// Action Buttons
JButton b1 = new JButton("Register");
JLabel msg = new JLabel("", SwingConstants.CENTER);
JButton b2 = new JButton("View Registrations");

// Layout Management
frame.add(l1); frame.add(t1);
frame.add(l2); frame.add(t2);
frame.add(l3); frame.add(t3);
frame.add(l4); frame.add(cb);
frame.add(b1); frame.add(msg);
frame.add(b2);
```

Layout Strategy:

- GridLayout with 6 rows, 2 columns
- Uniform component spacing (5px gaps)
- Center-aligned status messages
- Organized form-like interface

Event Handling Implementation

Registration Button Handler

```
b1.addActionListener(e -> {
    String name = t1.getText();
    String usn = t2.getText();
    String branch = t3.getText();
    String event = (String)cb.getSelectedItem();

    if(name.isEmpty() || usn.isEmpty() || branch.isEmpty()) {
        msg.setText("⚠ Please fill all fields!");
        msg.setForeground(Color.RED);
    } else {
        regList.add(new Registration(name, usn, branch, event));
        msg.setText("✓ Registered for " + event);
        msg.setForeground(Color.BLUE);

        t1.setText(""); t2.setText(""); t3.setText("");
    }
});|
```

Algorithm Logic:

1. Retrieve input from text fields
2. Get selected event from dropdown
3. Validate all fields are non-empty
4. If validation fails: Show error message
5. If validation passes:
 - Create Registration object
 - Add to ArrayList
 - Show success message
 - Clear input fields

View Registrations Button Handler

```
b2.addActionListener(e -> {
    JFrame viewFrame = new JFrame("Registered Students");
    viewFrame.setSize(350, 300);

    String column[] = {"Name", "USN", "Branch", "Event"};
    String data[][] = new String[regList.size()][4];

    for(int i = 0; i < regList.size(); i++) {
        data[i][0] = regList.get(i).name;
        data[i][1] = regList.get(i).usn;
        data[i][2] = regList.get(i).branch;
        data[i][3] = regList.get(i).event;
    }

    JTable table = new JTable(data, column);
    JScrollPane sp = new JScrollPane(table);
    viewFrame.add(sp);
    viewFrame.setVisible(true);
});
```

Complete Source Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;

class Registration {
    String name, usn, branch, event;
    Registration(String n, String u, String b, String e) {
        name = n; usn = u; branch = b; event = e;
    }
}

public class CollegeEventRegistrationGUI {
    static ArrayList<Registration> regList = new ArrayList<>();

    public static void main(String[] args) {
        JFrame frame = new JFrame("College Event Registration");
        frame.setSize(400, 350);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new GridLayout(6, 2, 5, 5));

        JLabel l1 = new JLabel("Student Name:");
        JTextField t1 = new JTextField();

        JLabel l2 = new JLabel("USN:");
        JTextField t2 = new JTextField();

        JLabel l3 = new JLabel("Branch:");
        JTextField t3 = new JTextField();

        JLabel l4 = new JLabel("Select Event:");
        String events[] = {"Hackathon", "Coding Contest", "Robotics",
                           "Paper Presentation", "Tech Quiz"};
    }
}
```

```
JComboBox<String> cb = new JComboBox<>(events);

JButton b1 = new JButton("Register");
JLabel msg = new JLabel("", SwingConstants.CENTER);

JButton b2 = new JButton("View Registrations");

frame.add(l1); frame.add(t1);
frame.add(l2); frame.add(t2);
frame.add(l3); frame.add(t3);
frame.add(l4); frame.add(cb);
frame.add(b1); frame.add(msg);
frame.add(b2);

// Register button action
b1.addActionListener(e -> {
    String name = t1.getText();
    String usn = t2.getText();
    String branch = t3.getText();
    String event = (String)cb.getSelectedItem();

    if(name.isEmpty() || usn.isEmpty() || branch.isEmpty()) {
        msg.setText("⚠ Please fill all fields!");
        msg.setForeground(Color.RED);
    } else {
        regList.add(new Registration(name, usn, branch, event));
        msg.setText("✓ Registered for " + event);
        msg.setForeground(Color.BLUE);
    }

    t1.setText(""); t2.setText(""); t3.setText("");
}

});

// View registrations button action
```

```

b2.addActionListener(e -> {
    JFrame viewFrame = new JFrame("Registered Students");
    viewFrame.setSize(350, 300);

    String column[] = {"Name", "USN", "Branch", "Event"};
    String data[][] = new String[regList.size()][4];

    for(int i = 0; i < regList.size(); i++) {
        data[i][0] = regList.get(i).name;
        data[i][1] = regList.get(i).usn;
        data[i][2] = regList.get(i).branch;
        data[i][3] = regList.get(i).event;
    }
    JTable table = new JTable(data, column);
    JScrollPane sp = new JScrollPane(table);
    viewFrame.add(sp);
    viewFrame.setVisible(true);
});
frame.setVisible(true);
}

```

Data Structure Design

Data Structure Design

static ArrayList<Registration> regList = new ArrayList<>();

Advantages:

- Dynamic sizing (no fixed capacity)
- Fast random access ($O(1)$)
- Easy iteration and manipulation
- Memory efficient for small to medium datasets

Registration Object Structure

Registration Object:

```
|—— name: String (Student's full name)  
|—— usn: String (University Seat Number)  
|—— branch: String (Academic department)  
└—— event: String (Selected event name)
```

JTable Data Conversion

ArrayList<Registration> → String[][] conversion:

For each Registration object:

```
data[i][0] = registration.name
```

```
data[i][1] = registration.usn
```

```
data[i][2] = registration.branch
```

```
data[i][3] = registration.event
```

Result

Main Registration Interface

The screenshot shows a Windows application window titled "College Event Registration". The interface consists of several text input fields and a dropdown menu. On the left, there are labels: "Student Name:", "USN:", "Branch:", and "Select Event:". To the right of these labels are corresponding input fields. Below the "Select Event:" field is a dropdown menu showing "Hackathon". At the bottom left is a blue button labeled "Register", and at the bottom right is another blue button labeled "View Registrations".

Successful Registration

This screenshot is identical to the one above, showing the "College Event Registration" window. However, it includes a new message "✓ Registered for Hackathon" displayed prominently in blue text on the right side of the window, indicating that the registration was successful.

Error Message Display

The screenshot shows a Windows application window titled "College Event Registration". The window contains four text input fields: "Student Name" (containing "Govind"), "USN" (empty), "Branch" (empty), and "Select Event" (containing "Hackathon"). Below these fields are two buttons: "Register" and "View Registrations". A red error message "⚠ Please fill all fields!" is displayed in the center of the window.

Student Name:	Govind
USN:	
Branch:	
Select Event:	Hackathon

Register **⚠ Please fill all fields!**

View Registrations

View Registrations Output

The screenshot shows a Windows application window titled "Registered Students". It displays a table of registered students with four columns: Name, USN, Branch, and Event. One row is present in the table, showing Mahantesh, 1BM24CS156, CSE, and Hackathon.

Name	USN	Branch	Event
Mahantesh	1BM24CS156	CSE	Hackathon

Event Selection Dropdown

College Event Registration

Student Name:

JSN:

Branch:

Select Event:

Hackathon

- Hackathon
- Coding Contest
- Robotics
- Paper Presentation
- Tech Quiz

Empty Registration View

Registered Students

Name	USN	Branch	Event

Conclusion

Project Achievements

Successfully implemented a functional event registration system
Applied core OOP principles effectively in GUI programming
Created an intuitive user interface with validation
Implemented efficient in-memory data storage
Demonstrated event-driven programming concepts

Technical Strengths

User-Friendly Interface: Clean, organized layout
Data Validation: Prevents incomplete submissions
Real-time Feedback: Immediate status messages
Scalable Design: Can handle multiple registrations
Modular Architecture: Clear separation of concerns

Current Limitations:

- No persistent storage (data lost on application close)
- No database integration
- Limited field validation (only checks for empty fields)
- No duplicate registration prevention
- No export functionality

Future Enhancements:

1. **Database Integration:** MySQL/PostgreSQL for data persistence
2. **Enhanced Validation:** USN format, email validation, phone validation
3. **User Authentication:** Login system for coordinators
4. **Reporting Features:** PDF/Excel export functionality
5. **Web Interface:** Convert to web application using JSP/Spring Boot
6. **Email Notifications:** Automatic confirmation emails
7. **Event Management:** Add/edit/delete events functionality
8. **Search Functionality:** Filter and search registrations

References

Official Documentation

Oracle Java Documentation: <https://docs.oracle.com/javase/>

Java Swing Tutorial: <https://docs.oracle.com/javase/tutorial/uiswing/>

Java Collections Framework: <https://docs.oracle.com/javase/8/docs/technotes/guides/collections/>

Technical Resources

Java Design Patterns: <https://java-design-patterns.com/>

Swing Layout Managers: GridLayout, BorderLayout, FlowLayout documentation

Event-Driven Programming in Java: ActionListener, MouseListener interfaces

Academic References

Deitel, P. J., & Deitel, H. M. (2017). Java: How to Program

Horstmann, C. S. (2019). Core Java Volume I: Fundamentals

Bloch, J. (2018). Effective Java

Online Resources

W3Schools Java Tutorial: Basic Java concepts

GeeksforGeeks Java Swing Tutorials

Stack Overflow: Community solutions for Java Swing issues

GitHub: Open-source Java Swing projects for reference

Tools and Libraries

Visual Studio Code with Java Extension Pack

IntelliJ IDEA Community Edition

Eclipse IDE for Java Developers

Java Development Kit (JDK) 8+