ELECTRICITY BILLING SYSTEM

**A MINI-PROJECT REPORT**

**Submitted by**

**SATHISH KUMAR O 230701297**

**BENITO B 230701050**

**In partial fulfillment of the award of the degree**

**of**

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**



**RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

**An Autonomous Institute**

**CHENNAI**

**NOVEMBER 2023**

BONAFIDE CERTIFICATE

Certified that this project “ELECTRICITY BILLING SYSTEM” is the

bonafide work of “SATHISH KUMAR O, BENITO B” who carried out the project work under

my supervision.

|  |  |
| --- | --- |
| SIGNATURE | SIGNATURE |
| MR SARAVANA GOKUL | MRS V JANANEE |
| ASSISTANT PROFESSOR | ASSISTANT PROFESSOR |
| Dept. of Computer Science and Engg, Engg, | Dept. of Computer Science and |
| Rajalakshmi Engineering College | Rajalakshmi Engineering College |
| Chennai | Chennai |

This mini project report is submitted for the viva voce examination to be held on \_\_\_\_\_\_

# TABLEOFCONTENTS

|  |  |
| --- | --- |
| S.NO TITLE  Abstract | PAGE |
| 1 Introduction | 05 |
| 2 Scope of Project | 06 |
| 3 UML Diagrams | 09 |
| 4 Code Implementation | 22 |
| 5 Conclusion | 31 |
| 6 Future Work | 33 |
| 7 Reference | 35 |

# ABSTRACT

The **Electricity Billing System** is a graphical user interface (GUI) application designed to facilitate the calculation, storage, and management of electricity bills for customers. The system is implemented using Java's Swing library and provides an intuitive interface for users to input customer details, calculate electricity bills based on consumption, and manage stored billing data.

This system provides an easy-to-use interface for both residential and commercial billing management, while also ensuring that users can interact with and update the data dynamically.

The application is designed with a clean, modern user interface and offers an efficient way to handle basic electricity billing tasks. The data management features are supported by the JTable component, which dynamically updates when records are added, modified, or removed.

# INTRODUCTION

The **Electricity Billing System** is a desktop application built using Java's Swing library, aimed at automating and simplifying the process of calculating and managing electricity bills for customers. This application provides a user-friendly graphical interface (GUI) that allows users to input customer details, calculate electricity bills based on their consumption, and manage billing records in an organized manner.

In today's world, utilities such as electricity require accurate tracking and billing of consumption, and this system serves as a tool to streamline this process.

Key Features of the Online Voting System in JAVA:

1 **Bill Calculation**: The system applies a tiered billing rate, with different pricing for various consumption ranges.

2 **Record Management**: Users can add new records, update existing ones, or delete records from the results table.

3 Data Validation: Input validation is implemented to ensure that all fields are properly filled and that numeric values are correctly entered for units consumed.

SCOPEOFTHEPROJECT

The **Electricity Billing System** provides a comprehensive yet simple solution for managing electricity consumption and generating bills. The scope of this project encompasses several key functionalities and provides a platform for future enhancements and expansion. Below is a detailed outline of the project’s scope:

**1. Customer Billing Management**

* **Customer Details**: The system allows the user to input customer details such as name, address, contact number, email, and meter number. These details are stored temporarily in the application for the purpose of generating bills.
* **Bill Calculation**: The application uses a tiered pricing model to calculate electricity bills based on the number of units consumed. The system provides calculations for various consumption levels:
  + Up to 100 units: A lower rate per unit.
  + 101-200 units: A higher rate for consumption exceeding 100 units.
  + 201-500 units: A still higher rate for consumption exceeding 200 units.
  + Above 500 units: The highest rate per unit for extensive consumption.
* **Output of Bill Amount**: Once the units are entered, the system calculates and displays the bill amount, making it easy for users to understand how their consumption translates into charges.

**2. Data Storage and Record Management**

* **Table of Results**: The system stores customer billing details in a JTable. This table records key information like customer name, meter number, units consumed, and the bill amount for each entry.
* **Dynamic Updates**: Users can add new records to the table, update existing ones, or delete any rows based on their selections.
* **Persistence Within Session**: All customer data, such as calculated bills and customer information, remains available within the session until the application is closed.

**3. User Interaction and Interface**

* **Input Forms**: The user is prompted to input relevant customer information (such as name, meter number, and units consumed) through text fields in a clear and user-friendly form.
* **Error Handling and Validation**: Input validation ensures that fields such as the number of units are correctly filled and that the input data is valid (numeric for units consumed). The system displays appropriate error messages for incorrect or incomplete data entries.
* **Actionable Buttons**: The system provides buttons like "Calculate Bill", "Clear", "Delete", and "Update" that the user interacts with to perform tasks such as generating the bill, clearing the input fields, or modifying existing records.

**4. Enhanced Usability and Customization**

* **Clear and Informative Interface**: The system is designed with an intuitive user interface, including color-coded labels, neatly organized form fields, and responsive buttons for easy navigation.
* **Visual Feedback**: Users receive immediate visual feedback after performing any operation, such as successful bill calculation or error messages for invalid inputs.

**5. Flexibility for Future Enhancements**

* **Rate Structure Adjustment**: The tiered billing structure can be easily modified or extended to accommodate new pricing models or changes in rate plans.
* **Database Integration**: The current implementation stores data temporarily. Future iterations could integrate a database (e.g., MySQL, SQLite) to persist billing records and customer information across sessions.
* **Multi-User Support**: Future versions could introduce multi-user functionality, where different users can access and modify the system with varying permission levels (e.g., administrator, operator).

**6. Application Performance and Scalability**

* **Efficient Calculation**: The system ensures that bill calculations are performed efficiently, even for larger numbers of customers or higher units of consumption.
* **Scalability for Larger Datasets**: While the current system is designed for small-scale use (individual or small utility billing), it can be scaled to handle a larger volume of customer records by optimizing the data storage method (e.g., integrating databases for large datasets).

**UMLDIAGRAMS**

## 

### The **UML Class Diagram** for the **Electricity Billing System** provides a visual representation of the key components, their attributes, methods, and relationships. Below is a breakdown of each class and how they interact in the system.

**1.ElectricityBillingSystem** **Class**

Purpose:

This is the main class that represents the graphical user interface (GUI) of the electricity billing system. It manages the user input (customer details), calculates the bill based on the units consumed, and displays results in a table.

Attributes:

tfName, tfAddress, tfContact, tfEmail, tfUnits, tfBillAmount, tfMeterNumber: These are instances of JTextField that allow users to input their details like name, address, contact, email, units consumed, meter number, and the calculated bill amount.

**2.DefaultTableModel** **Class**

Purpose:

This class is part of Java Swing and is responsible for managing the data in the table (JTable). It provides methods to manipulate the data in the table, such as adding, updating, and removing rows.

Attributes:

data: A 2D array of Objects that holds the data of customer records (name, meter number, units, bill amount).

columnNames:

An array of column names (Customer Name, Meter Number, Units Consumed, Bill Amount).

Methods:

addRow(rowData: Object[]): Adds a new row of data (customer details and bill) to the table.

setValueAt(value: Object, row: int, col: int): Updates the value at a specific row and column in the table.

**3.JTable** **Class**

Purpose:

This class is responsible for displaying the data in a tabular format. It is linked to the DefaultTableModel and reflects changes made in the model.

Attributes:

rowData: An array of objects representing a row in the table.

columnNames: An array of strings that define the column headers.

Methods:

The table automatically reflects updates and changes made in the model (via DefaultTableModel).

**4.JButton** **Class**

Purpose:

This class is used for creating buttons in the GUI, such as "Calculate Bill", "Clear", "Update", and "Delete".

Attributes:

text: The label displayed on the button.

actionListener: A reference to an ActionListener that listens for button clicks and performs specific actions.

Methods:

addActionListener(listener: ActionListener): Adds an ActionListener to the button to handle user interactions (such as clicking the button).

**5.** **ActionListener** **Interface**

Purpose:

This is an interface used for event handling. It defines the method actionPerformed() to be implemented by any class that needs to respond to user actions (like button clicks).

Methods:

actionPerformed(e: ActionEvent): Called when a button (or other interactive component) is clicked. The specific implementation determines what happens when a button is pressed (e.g., calculating the bill or updating the table).

**PROGRAM**:-

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class ElectricityBillingSystem extends JFrame {

    private JTextField tfName, tfAddress, tfContact, tfEmail, tfUnits, tfBillAmount, tfMeterNumber;

    private JButton btnCalculate, btnClear, btnDelete, btnUpdate;

    private JLabel lblName, lblAddress, lblContact, lblEmail, lblUnits, lblBillAmount, lblMeterNumber, lblTitle;

    private DefaultTableModel tableModel;

    private JTable resultsTable;

    private int selectedRow = -1; // Variable to track the selected row

    public ElectricityBillingSystem() {

        setTitle("Electricity Billing System");

        setLayout(new BorderLayout(10, 10));

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setSize(700, 500);

        // Create the tabbed pane

        JTabbedPane tabbedPane = new JTabbedPane();

        // First tab: Billing System

        JPanel billingPanel = createBillingPanel();

        tabbedPane.addTab("Billing System", billingPanel);

        // Second tab: Results Table

        JPanel resultsPanel = createResultsPanel();

        tabbedPane.addTab("Results", resultsPanel);

        add(tabbedPane, BorderLayout.CENTER);

        setLocationRelativeTo(null);  // Center window

        setVisible(true);

    }

    private JPanel createBillingPanel() {

        JPanel billingPanel = new JPanel();

        billingPanel.setLayout(new BorderLayout());

        // Title label at the top with custom background color

        lblTitle = new JLabel("Electricity Billing System", JLabel.CENTER);

        lblTitle.setFont(new Font("Arial", Font.BOLD, 18));

        lblTitle.setForeground(Color.WHITE);

        lblTitle.setBackground(new Color(0, 102, 204));

        lblTitle.setOpaque(true);

        billingPanel.add(lblTitle, BorderLayout.NORTH);

        // Panel for customer details (center)

        JPanel panelCenter = new JPanel();

        panelCenter.setLayout(new GridLayout(8, 2, 10, 10)); // Added one more row for meter number

        panelCenter.setBackground(new Color(240, 240, 240));  // Light gray background

        billingPanel.add(panelCenter, BorderLayout.CENTER);

        // Creating labels and text fields with consistent styling

        lblName = new JLabel("Customer Name:");

        lblName.setFont(new Font("Arial", Font.PLAIN, 14));

        lblName.setForeground(new Color(0, 102, 204));

        tfName = new JTextField();

        styleTextField(tfName);

        panelCenter.add(lblName);

        panelCenter.add(tfName);

        lblAddress = new JLabel("Customer Address:");

        lblAddress.setFont(new Font("Arial", Font.PLAIN, 14));

        lblAddress.setForeground(new Color(0, 102, 204));

        tfAddress = new JTextField();

        styleTextField(tfAddress);

        panelCenter.add(lblAddress);

        panelCenter.add(tfAddress);

        lblContact = new JLabel("Contact Number:");

        lblContact.setFont(new Font("Arial", Font.PLAIN, 14));

        lblContact.setForeground(new Color(0, 102, 204));

        tfContact = new JTextField();

        styleTextField(tfContact);

        panelCenter.add(lblContact);

        panelCenter.add(tfContact);

        lblEmail = new JLabel("Email:");

        lblEmail.setFont(new Font("Arial", Font.PLAIN, 14));

        lblEmail.setForeground(new Color(0, 102, 204));

        tfEmail = new JTextField();

        styleTextField(tfEmail);

        panelCenter.add(lblEmail);

        panelCenter.add(tfEmail);

        lblMeterNumber = new JLabel("Meter Number:");

        lblMeterNumber.setFont(new Font("Arial", Font.PLAIN, 14));

        lblMeterNumber.setForeground(new Color(0, 102, 204));

        tfMeterNumber = new JTextField();

        styleTextField(tfMeterNumber);

        panelCenter.add(lblMeterNumber);

        panelCenter.add(tfMeterNumber);

        lblUnits = new JLabel("Units Consumed:");

        lblUnits.setFont(new Font("Arial", Font.PLAIN, 14));

        lblUnits.setForeground(new Color(0, 102, 204));

        tfUnits = new JTextField();

        styleTextField(tfUnits);

        panelCenter.add(lblUnits);

        panelCenter.add(tfUnits);

        lblBillAmount = new JLabel("Bill Amount: ");

        lblBillAmount.setFont(new Font("Arial", Font.PLAIN, 14));

        lblBillAmount.setForeground(new Color(0, 102, 204));

        tfBillAmount = new JTextField();

        tfBillAmount.setEditable(false);

        styleTextField(tfBillAmount);

        panelCenter.add(lblBillAmount);

        panelCenter.add(tfBillAmount);

        // Panel for buttons (bottom)

        JPanel panelSouth = new JPanel();

        panelSouth.setLayout(new FlowLayout());

        panelSouth.setBackground(new Color(0, 102, 204));

        billingPanel.add(panelSouth, BorderLayout.SOUTH);

        // Calculate button with custom background and font

        btnCalculate = new JButton("Calculate Bill");

        btnCalculate.setFont(new Font("Arial", Font.BOLD, 14));

        btnCalculate.setBackground(new Color(0, 204, 102));

        btnCalculate.setForeground(Color.WHITE);

        btnCalculate.setBorder(BorderFactory.createLineBorder(Color.BLACK, 1));

        // Clear button with custom background and font

        btnClear = new JButton("Clear");

        btnClear.setFont(new Font("Arial", Font.BOLD, 14));

        btnClear.setBackground(new Color(255, 77, 77));

        btnClear.setForeground(Color.WHITE);

        btnClear.setBorder(BorderFactory.createLineBorder(Color.BLACK, 1));

        panelSouth.add(btnCalculate);

        panelSouth.add(btnClear);

        btnCalculate.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                try {

                    String customerName = tfName.getText();

                    String customerAddress = tfAddress.getText();

                    String contactNumber = tfContact.getText();

                    String email = tfEmail.getText();

                    String meterNumber = tfMeterNumber.getText();

                    int unitsConsumed = Integer.parseInt(tfUnits.getText());

                    if (customerName.isEmpty() || customerAddress.isEmpty() || contactNumber.isEmpty()

                            || email.isEmpty() || meterNumber.isEmpty() || unitsConsumed < 0) {

                        JOptionPane.showMessageDialog(null, "Please enter valid information.");

                        return;

                    }

                    double billAmount = calculateBill(unitsConsumed);

                    tfBillAmount.setText("Rs. " + String.format("%.2f", billAmount));

                    // Add bill details as a new row in the table

                    Object[] rowData = {customerName, meterNumber, unitsConsumed, String.format("Rs. %.2f", billAmount)};

                    tableModel.addRow(rowData);

                } catch (NumberFormatException ex) {

                    JOptionPane.showMessageDialog(null, "Invalid input. Please enter numeric values for units.");

                }

            }

        });

        btnClear.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                tfName.setText("");

                tfAddress.setText("");

                tfContact.setText("");

                tfEmail.setText("");

                tfMeterNumber.setText("");

                tfUnits.setText("");

                tfBillAmount.setText("");

            }

        });

        return billingPanel;

    }

    private JPanel createResultsPanel() {

        JPanel resultsPanel = new JPanel();

        resultsPanel.setLayout(new BorderLayout());

        // JTable to display results in table format

        String[] columnNames = {"Customer Name", "Meter Number", "Units Consumed", "Bill Amount"};

        tableModel = new DefaultTableModel(columnNames, 0);

        resultsTable = new JTable(tableModel);

        JScrollPane scrollPane = new JScrollPane(resultsTable);

        resultsPanel.add(scrollPane, BorderLayout.CENTER);

        // Panel for Delete and Update buttons

        JPanel buttonPanel = new JPanel();

        buttonPanel.setLayout(new FlowLayout());

        resultsPanel.add(buttonPanel, BorderLayout.SOUTH);

        btnDelete = new JButton("Delete Selected Row");

        btnUpdate = new JButton("Update Selected Row");

        buttonPanel.add(btnDelete);

        buttonPanel.add(btnUpdate);

        // Delete button action

        btnDelete.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                int row = resultsTable.getSelectedRow();

                if (row != -1) {

                    tableModel.removeRow(row);

                } else {

                    JOptionPane.showMessageDialog(null, "Please select a row to delete.");

                }

            }

        });

        // Update button action

        btnUpdate.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                int row = resultsTable.getSelectedRow();

                if (row != -1) {

                    // Retrieve data from selected row

                    String customerName = (String) tableModel.getValueAt(row, 0);

                    String meterNumber = (String) tableModel.getValueAt(row, 1);

                    int unitsConsumed = (int) tableModel.getValueAt(row, 2);

                    String billAmount = (String) tableModel.getValueAt(row, 3);

                    // Set the data in the form fields

                    tfName.setText(customerName);

                    tfMeterNumber.setText(meterNumber);

                    tfUnits.setText(String.valueOf(unitsConsumed));

                    tfBillAmount.setText(billAmount);

                    // Update the row with the new data when user calculates a new bill

                    btnCalculate.addActionListener(new ActionListener() {

                        @Override

                        public void actionPerformed(ActionEvent e) {

                            try {

                                String newCustomerName = tfName.getText();

                                String newMeterNumber = tfMeterNumber.getText();

                                int newUnitsConsumed = Integer.parseInt(tfUnits.getText());

                                if (newCustomerName.isEmpty() || newMeterNumber.isEmpty() || newUnitsConsumed < 0) {

                                    JOptionPane.showMessageDialog(null, "Please enter valid information.");

                                    return;

                                }

                                double newBillAmount = calculateBill(newUnitsConsumed);

                                tfBillAmount.setText("Rs. " + String.format("%.2f", newBillAmount));

                                // Update the row with new values

                                tableModel.setValueAt(newCustomerName, row, 0);

                                tableModel.setValueAt(newMeterNumber, row, 1);

                                tableModel.setValueAt(newUnitsConsumed, row, 2);

                                tableModel.setValueAt(String.format("Rs. %.2f", newBillAmount), row, 3);

                            } catch (NumberFormatException ex) {

                                JOptionPane.showMessageDialog(null, "Invalid input. Please enter numeric values for units.");

                            }

                        }

                    });

                } else {

                    JOptionPane.showMessageDialog(null, "Please select a row to update.");

                }

            }

        });

        return resultsPanel;

    }

    private double calculateBill(int units) {

        double billAmount = 0;

        if (units <= 100) {

            billAmount = units \* 5.0;

        } else if (units <= 200) {

            billAmount = 100 \* 5.0 + (units - 100) \* 7.0;

        } else if (units <= 500) {

            billAmount = 100 \* 5.0 + 100 \* 7.0 + (units - 200) \* 10.0;

        } else {

            billAmount = 100 \* 5.0 + 100 \* 7.0 + 300 \* 10.0 + (units - 500) \* 15.0;

        }

        return billAmount;

    }

    // Method to style text fields with consistent look

    private void styleTextField(JTextField textField) {

        textField.setFont(new Font("Arial", Font.PLAIN, 14));

        textField.setBackground(Color.WHITE);

        textField.setForeground(Color.BLACK);

        textField.setBorder(BorderFactory.createLineBorder(new Color(0, 102, 204), 2));

    }

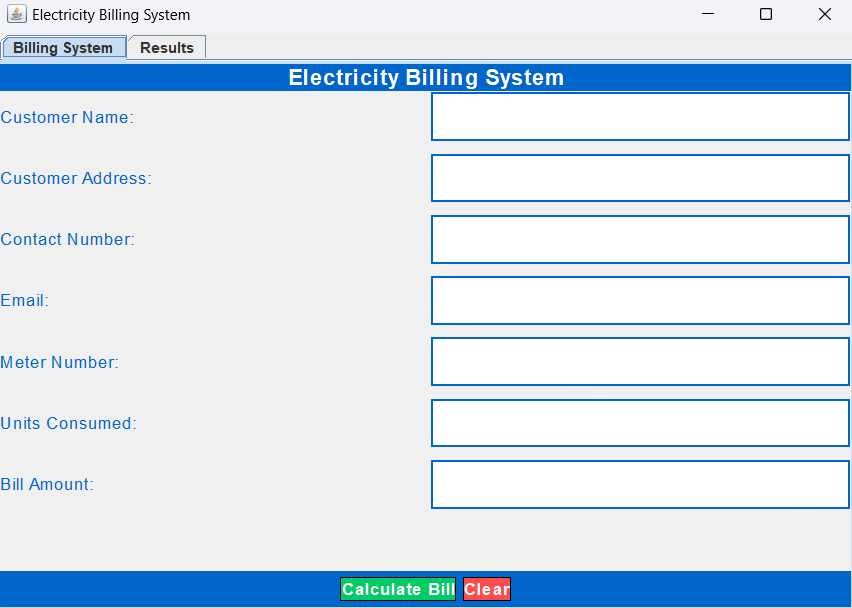
    public static void main(String[] args) {

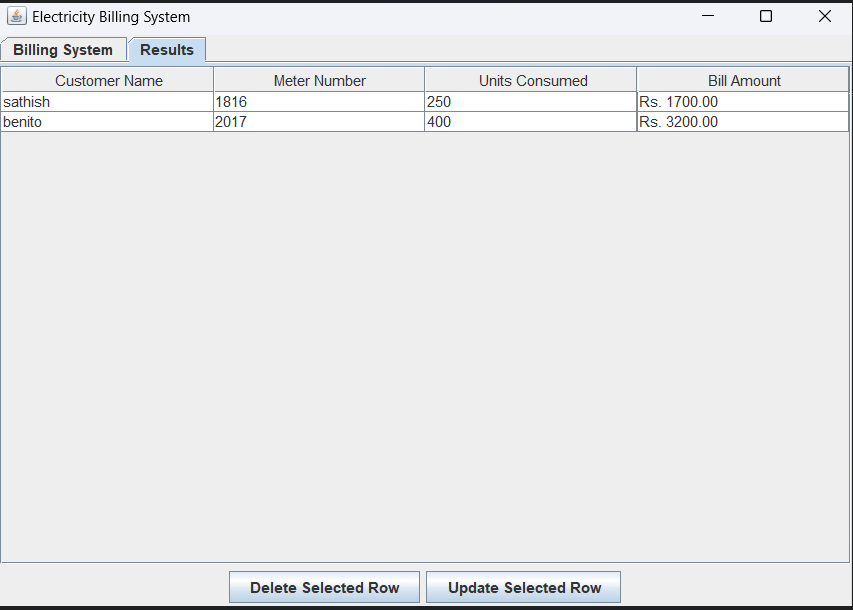
        new ElectricityBillingSystem();

    }

}

OUTPUT:





### **CONCLUSION**

### **Conclusion for the Electricity Billing System**

### The **Electricity Billing System** implemented in Java with a Swing-based graphical user interface (GUI) serves as a practical application for automating the process of calculating electricity bills based on customer consumption. This system simplifies the manual process of bill calculation by providing a user-friendly interface where users can input customer information, enter the units consumed, and instantly calculate the corresponding bill.