ELECTRICITY BILLING SYSTEM

A PROJECT REPORT

Submitted by

SOWMIYAA V S (2303811710422154)

in partial fulfillment of requirements for the award of the course

CGB1201 - JAVA PROGRAMMING

In

COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

NOVEMBER- 2024

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on **"ELECTRICITY BILLING SYSTEM"** is the bonafide work of **SOWMIYAA V S (2303811710422154)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

CGB1201-JAVA PROGRAMMING Dr.A.DELPHIN CAROLINA RANI, M.E.,Ph.D., HEAD OF THE DEPARTMENT PROFESSOR [illegible]	CGB1201-JAVA PROGRAMMING Mr.MALARMANNAN A, M.E., [illegible] ASSISTANTP ROFESSOR [illegible]
SIGNATURE	**SIGNATURE**
Dr.A.Delphin Carolina Rani, M.E.,Ph.D.,	Mr. A. Malarmannan, M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
PROFESSOR	ASSISTANT PROFESSOR
Department of CSE	Department of CSE
K.Ramakrishnan College of Technology (Autonomous)	K.Ramakrishnan College of Technology (Autonomous)
Samayapuram–621112.	Samayapuram–621112.

Submitted for the viva-voce examination held on …06.12.2024………….

CGB1201-JAVA PROGRAMMING Mr. M. SARAVANAN, M.E., INTERNAL EXAMINER ASSISTANT PROFESSOR [illegible]	CGB1201-JAVA PROGRAMMING Mr.R. KARTHIK, M.E., EXTERNAL EXAMINER ASSISTANT PROFESSOR 8138-SCE, TRICHY. [illegible] 06/12/2024
INTERNAL EXAMINER	EXTERNAL EXAMINER

DECLARATION

I declare that the project report on **“ELECTRICITY BILLING SYSTEM”** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **“ANNA UNIVERSITY CHENNAI”** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1201 - JAVA PROGRAMMING.**

.

Signature

[illegible]

SOWMIYAA V.S

Place: Samayapuram

Date:06.12.24

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. A. DELPHIN CAROLINA RANI, M.E.,Ph.D.,** Head of the department, **COMPUTER SCIENCE AND ENGINEERING** for providing her encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **MR. A. MALARMANNAN, M.E.,** Department of **COMPUTER SCIENCE AND ENGINEERING,** for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To serve the society by offering top-notch technical education on par with global standards

MISSION OF THE INSTITUTION

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of the industry and society.
- Be an institute with world class research facilities
- Be an institute nurturing talent and enhancing the competency of students to transform them as all-round personality respecting moral and ethical values

VISION OF DEPARTMENT

To be a center of eminence in creating competent software professionals with research and innovative skills.

MISSION OF DEPARTMENT

M1: Industry Specific: To nurture students in working with various hardware and software platforms inclined with the best practices of industry.

M2: Research: To prepare students for research-oriented activities.

M3: Society: To empower students with the required skills to solve complex technological problems of society.

PROGRAM EDUCATIONAL OBJECTIVES

1. PEO1: Domain Knowledge

To produce graduates who have strong foundation of knowledge and skills in the field of Computer Science and Engineering.

2. PEO2: Employability Skills and Research

To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

3. PEO3: Ethics and Values

To develop leadership skills and ethically collaborate with society to tackle real-world challenges.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

This Electricity Billing System program is a Java-based application with a graphical user interface (GUI) built using AWT. Key features includes, Customer Management: Add customers with names and meter numbers. Meter Reading Updates: Update meter readings to calculate consumed electricity units.Rate Configuration: Adjust the rate per unit of electricity.Bill Generation: Generate and display electricity bills with details of consumption and costs.User-Friendly GUI: Interactive interface for admin operations, including buttons for adding customers, updating readings, changing rates, and generating bills. Backend Logic: A robust backend with classes for customer details and admin functionality using HashMap for storage.This application simplifies electricity billing with a clear, modular structure.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD JAVA APPLICATIONS FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The Electricity Billing System is a Java-based application that features a user-friendly graphical interface built with AWT. It allows for efficient customer management, including adding customer names and meter numbers, and updating meter readings for accurate consumption tracking. The system enables rate configuration and bill generation based on usage, displaying detailed consumption and cost data. With an intuitive GUI for admin operations and backend logic using HashMap for data storage, this application simplifies electricity billing tasks. Its modular structure ensures clear and easy management of all functionalities.	**PO1 -3** **PO2 -3** **PO3 -3** **PO4 -3** **PO5 -3** **PO6 -3** **PO7 -3** **PO8 -3** **PO9 -3** **PO10 -3** **PO11-3** **PO12 -3**	**PSO1 -3** **PSO2 -3** **PSO3 -3**

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	viii
1	**INTRODUCTION**	
	1.1 Objective	1
	1.2 Overview	1
	1.3 Java Programming concepts	2
2	**PROJECT METHODOLOGY**	
	2.1 Proposed Work	4
	2.2 Block Diagram	6
3	**MODULE DESCRIPTION**	
	3.1 Customer module	7
	3.2 Adminpanel module	7
	3.3 Billing calculation module	8
	3.4 GUI module	9
	3.5 Event handling module	9
4	**CONCLUSION & FUTURE SCOPE**	
	4.1 Conclusion	10
	4.2 Future Scope	10
	APPENDIX A (SOURCE CODE)	13
	APPENDIX B (SCREENSHOTS)	21
	REFERENCES	23

CHAPTER 1
INTRODUCTION

1.1 Objective

The main objective of this Java program is to create an (Electricity Billing System **)** that allows an admin to manage customer data, update meter readings, set electricity rates, and generate bills for customers. This system provides a simple graphical user interface (GUI) for interacting with these functionalities.

1.2 Overview

The program is composed of multiple components:

1. **Customer Class**: Represents a customer with attributes like name, meter number, and meter readings. It provides methods to calculate consumed units and generate the bill based on the readings and rate per unit.
2. **AdminPanel Class**: Acts as the controller and manages all the customer records and functionalities such as adding a customer, updating meter readings, and updating the rate per unit.
3. **ElectricityBillingApp Class**: A GUI-based application (extends `Frame`) that provides a user interface for the admin to interact with the system. It includes options like adding customers, updating meter readings, setting rates, and generating bills. The interaction happens via buttons and text fields.

The **GUI** allows:

- Adding new customers.
- Updating meter readings for a specific customer.
- Updating the rate per unit of electricity.
- Generating a bill for a customer based on their meter readings and rate per unit.

1.3 Java Programming Concepts

The code demonstrates various core Java programming concepts, including:

1. Object-Oriented Programming (OOP):

- **Classes and Objects:**
    - Customer class represents a customer, with attributes like name, meter-Number, previousReading, and currentReading. Methods like getCon-sumedUnits() and displayBill() define behaviors.
    - AdminPanel class manages a collection of customers and provides methods to perform operations like adding a customer, updating meter readings, and generating bills.
    - The ElectricityBillingApp class serves as the main program with a GUI to interact with the admin.
- **Encapsulation**:
    - Private fields (e.g., name, meterNumber, ratePerUnit) in the Customer and AdminPanel classes encapsulate the internal state. Access is provid-ed via getter and setter methods.

2. Event Handling:

- **ActionEvent**: The ElectricityBillingApp implements ActionListener to re-spond to user actions (button clicks). When a button is clicked, the correspond-ing action is performed, such as adding a customer or generating a bill.
- **ActionCommand**: Used to distinguish between different buttons (e.g., "Add Customer", "Update Rate") in the actionPerformed() method.

3. Collections Framework:

- **HashMap**: The AdminPanel class uses a HashMap<Integer, Customer> to store and manage customers. The meterNumber is used as the key, and Cus-tomer objects are the values.

4. GUI Components:

- **AWT (Abstract Window Toolkit)**:
  - The program uses various AWT components like Frame, TextField, TextArea, Button, and Label to create the graphical user interface (GUI).
  - **Layout Management**: The FlowLayout is used for placing components in a simple layout.
  - **Event Handling**: GUI components are connected to action listeners to perform specific tasks when the user interacts with them.

5. Exception Handling:

- Basic error handling is implemented when the rate is set to a negative value. Although the program doesn't use try-catch blocks explicitly, it performs checks (e.g., if (newRate < 0)) to ensure the rate is valid.

6. Inheritance and Polymorphism:

- **Inheritance**: The ElectricityBillingApp class inherits from Frame to create a windowed interface.
- **Polymorphism**: The actionPerformed() method demonstrates polymorphism as the same method can handle multiple button actions and perform different operations based on the button clicked.

7. Data Persistence:

- While this program does not save customer data persistently (e.g., to a file or database), it manages data in memory using the HashMap. In a more complex version, data could be saved to allow for persistence across sessions.

CHAPTER 2
PROJECT METHODOLOGY

2.1 Proposed Work

- **Data Persistence**: Store customer and billing data permanently (via files or database).
- **Rate Management**: Implement tiered or time-based rates.
- **Enhanced UI/UX**: Improve the user interface, input validation, and user feedback.

1. Persistence of Customer Data:

- **Current Limitation**: The program does not store customer data permanently. Once the program is closed, all customer data is lost.
- **Proposed Work**: Implement a data storage mechanism to save customer details and billing history persistently. This could be achieved by:
  - **File-based Storage**: Store customer data in a text file or CSV file that the program can read/write during startup and shutdown.
  - **Database Integration**: Use a database like SQLite or MySQL to store customer information and bill records. This would allow the system to handle larger datasets, perform advanced queries, and provide better scalability.

2. Advanced Rate Management:

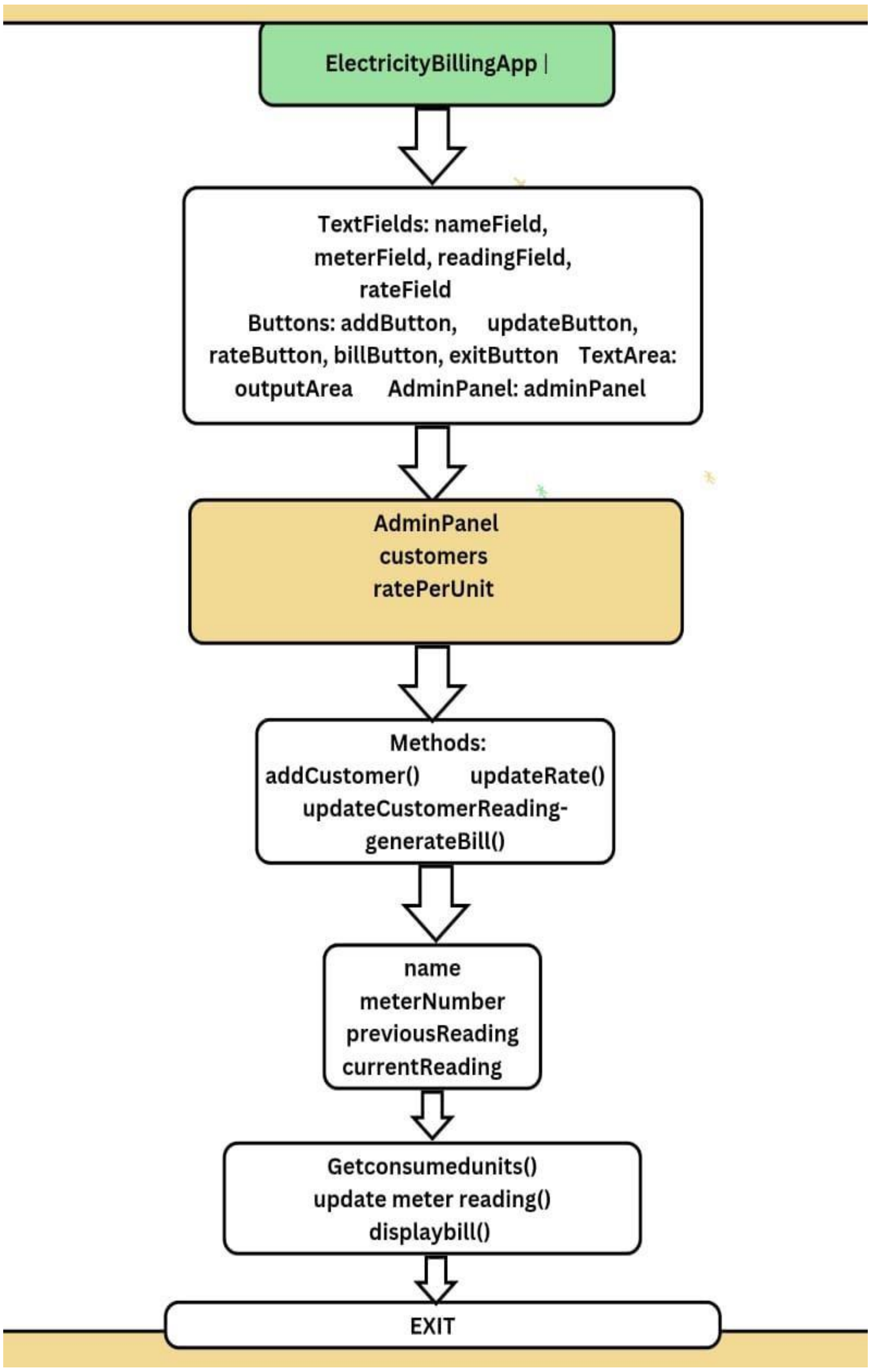
- **Current Limitation**: The program currently uses a single, fixed rate per unit for all customers.
- **Proposed Work**: Implement flexible rate management, allowing the admin to define different rates based on:
  - **Time-of-day** (e.g., peak and off-peak rates).
  - **Customer type** (e.g., residential, commercial, or industrial customers).
  - The admin panel could offer a more advanced interface.

3. Enhanced User Interface (UI) and User Experience (UX):

- **Current Limitation**: The GUI is functional but quite basic with limited user interactivity and responsiveness.

- **Proposed Work**:
    - **Use of a Modern GUI Framework**: Move from AWT to a more sophisticated framework like **Swing** or **JavaFX**. These frameworks offer better layouts, controls, and a more polished user experience.
    - **Form Validation**: Add validation for user inputs (e.g., ensure that readings are numeric and within a valid range, that rate per unit is positive, etc.).
    - **Clearer Feedback**: Improve user feedback by providing more intuitive messages and error handling (e.g., highlighting fields with invalid input).

2.2 Block Diagram



CHAPTER 3
MODULE DESCRIPTION

3.1 CUSTOMER MODULE

Purpose: To represent individual customer data, manage meter readings, and calculate the bill**.**

Methods:

- getConsumedUnits(): Calculates the electricity consumed by the customer based on the difference between the current and previous meter readings.
- updateMeterReading(double newReading): Updates the meter readings for the customer, setting the previous reading to the current one.
- displayBill(double ratePerUnit): Calculates the total bill by multiplying consumed units by the rate per unit, and then displays a detailed bill with customer information.

3.2 ADMINPANEL MODULE

Purpose: Acts as the controller to manage customer records, update rates, and process bill generation.

Methods:

- addCustomer(String name, int meterNumber): Adds a new customer to the system with the specified name and meter number.
- updateRate(double newRate): Updates the rate per unit of electricity.
- updateCustomerReading(int meterNumber, double newReading): Updates the meter reading for a specific customer identified by their meter number.
- generateBill(int meterNumber): Generates and displays a bill for the specified customer.

3.3 BILLING CALCULATION MODULE

Purpose: Calculates the electricity bill based on consumed units and the rate per unit.

Methods:

- getConsumedUnits(): Calculates the total consumption based on the difference between the current and previous meter readings.
- displayBill(double ratePerUnit): Generates a detailed bill for the customer, displaying information such as the customer's name, meter number, previous and current readings, consumed units, rate per unit, and the total bill amount

3.4 GUI MODULE

Purpose: Provides the graphical user interface for the electricity billing system, allowing the admin to interact with the system.

Key Components:

- **Text Fields**: nameField, meterField, readingField, and rateField for entering customer information, meter readings, and rate values.
- **Buttons**: addButton, updateButton, rateButton, billButton, and exitButton to trigger various actions (e.g., adding customers, updating readings, generating bills, etc.).
- **Text Area**: outputArea to display the results or status messages (e.g., confir-mation of added customers, updated readings, generated bills).

3.5 EVENT HANDLING MODULE

Purpose: Handles user interactions with the GUI and triggers corresponding actions in the AdminPanel.

Key Features:

- Each button (e.g., "Add Customer", "Update Meter Reading", "Update Rate", "Generate Bill", "Exit") is associated with an action event that is handled in the actionPerformed() method.
- Depending on the user's action (button click), the corresponding operation (add customer, update reading, generate bill) is executed, and the output is displayed in the outputArea text area.

CHAPTER 4
CONCLUSION & FUTURE SCOPE

4.1CONCLUSION

The ElectricityBillingApp is a Java-based application designed to streamline the process of managing electricity billing for customers. It is composed of three main classes: the Customer class, which holds the details of each customer, including their name, meter number, and electricity readings, and provides methods to calculate electricity consumption and display the bill; the AdminPanel class, which allows the administrator to manage customers, update meter readings, adjust the rate per unit, and generate bills; and the ElectricityBillingApp class, which serves as the user interface for the admin to perform all tasks. The application's graphical user interface includes text fields for entering customer information and buttons to trigger actions like adding customers, updating readings, adjusting rates, and generating bills. Through this setup, the program offers a straightforward solution for administrators to efficiently manage customer data and billing, while also providing an easy-to-use interface. The system is functional for basic electricity billing and can be expanded with additional features, such as data persistence or more sophisticated customer management.

4.2 FUTURE SCOPE

While the current version of the **Electricity Billing System** is functional, there are several areas where the system can be expanded and improved:

1. **Data Persistence**:
   - The current version of the system stores customer data only in memory (in a HashMap). Implementing data persistence via a database or file system (e.g., saving to a **JSON**, **CSV**, or **SQL database**) will ensure

that customer information is retained even after the application is closed.

2. **Error Handling and Validation**:
   - The system lacks robust error handling and input validation. It can be improved by adding validation for input fields to ensure that the user cannot enter invalid data (e.g., negative numbers for readings or rates). Proper error messages and safeguards can prevent system crashes.
3. **Security and Authentication**:
   - Implementing authentication mechanisms (e.g., login functionality for admins) will secure the system and prevent unauthorized access. This can be achieved through username and password authentication, ensuring that only authorized personnel can modify customer data.
4. **Notification System**:
   - Adding a notification system would be beneficial. The system could automatically send reminders to customers about bill payments, usage alerts, or upcoming rate changes through **email** or **SMS.**

5. **Advanced Billing Features:**
   - The billing module can be expanded to support more complex features like tiered pricing, time-of-use rates, or tax calculations. This would make the system suitable for larger utilities with more complicated pricing structures.
6. **GUI Enhancements**:
   - While the current GUI provides basic functionality, it can be made more sophisticated by adding features like dropdown menus for selecting existing customers, charts/graphs to visualize usage trends, and a more modern and intuitive interface with improved navigation.
7. **Multi-user Support**:
   - The system can be extended to support multiple users with different roles (e.g., customer service, admin, billing staff). This would require

role-based access control to ensure that users only access the features they are authorized for.

8. **Mobile Application**:
    - A mobile version of the system could be developed to allow customers to check their bills, meter readings, and usage from their smartphones. This would improve accessibility and enhance customer engagement.
9. **Integration with Smart Meters**:
    - For future scalability, the system could be integrated with **smart meters** to automatically collect and update meter readings. This would eliminate the need for manual readings and allow for real-time billing

APPENDIX A
(SOURCE CODE)

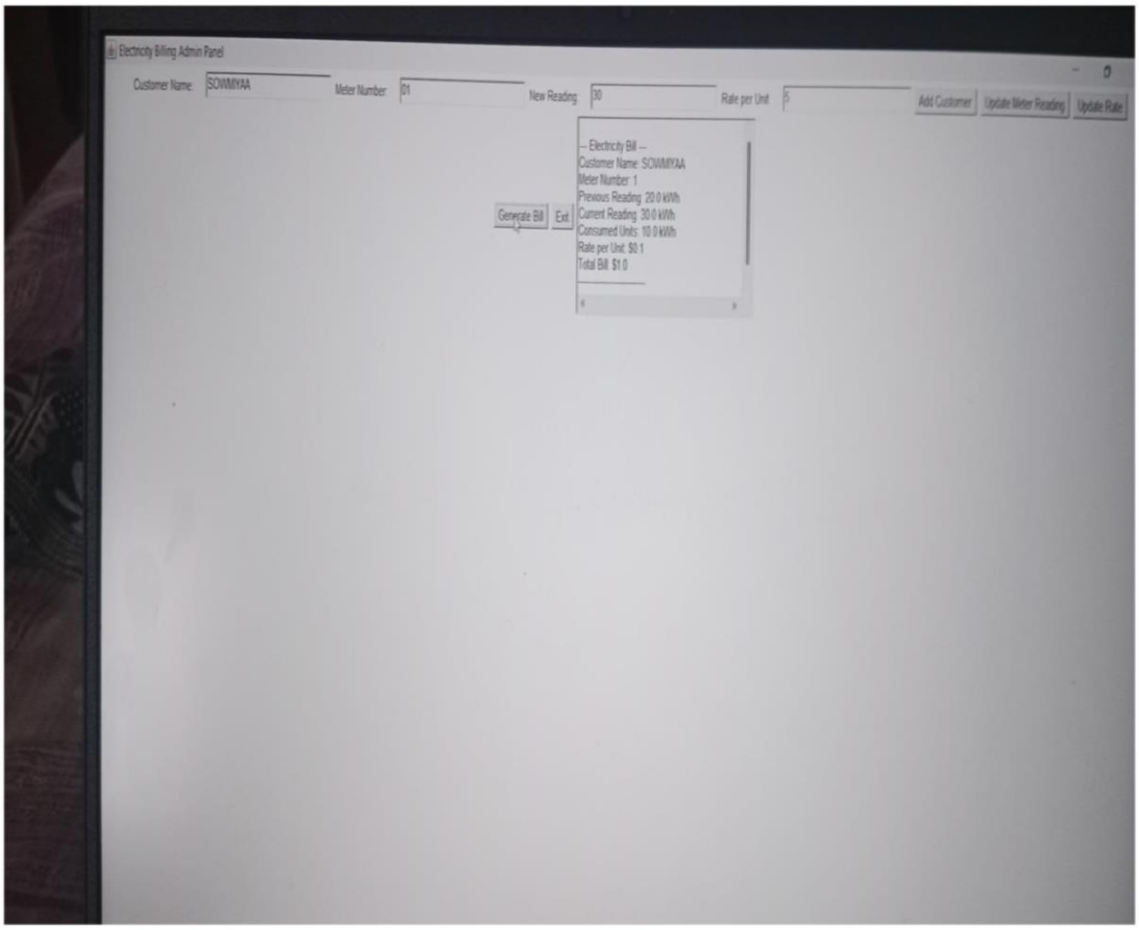
```
import java.awt.*;
import java.awt.event.*;
import java.util.HashMap;

class Customer {
    String name;
    int meterNumber;
    double previousReading;
    double currentReading;

    public Customer(String name, int meterNumber) {
        this.name = name;
        this.meterNumber = meterNumber;
        this.previousReading = 0;
        this.currentReading = 0;
    }

    public double getConsumedUnits() {
        if (currentReading < previousReading) {
            return currentReading;
        }
        return currentReading - previousReading;
    }

    public void updateMeterReading(double newReading) {
        this.previousReading = this.currentReading;
        this.currentReading = newReading;
    }

    public String displayBill(double ratePerUnit) {
        double consumedUnits = getConsumedUnits();
        double totalBill = consumedUnits * ratePerUnit;
        StringBuilder billDetails = new StringBuilder();
        billDetails.append("\n--- Electricity Bill ---\n");
        billDetails.append("Customer Name: ").append(name).append("\n");
        billDetails.append("Meter Number: ").append(meterNumber).append("\n");
        billDetails.append("Previous Reading: ").append(previousReading).append("
kWh\n");
        billDetails.append("Current Reading: ").append(currentReading).append("
kWh\n");
        billDetails.append("Consumed Units: ").append(consumedUnits).append("
kWh\n");
        billDetails.append("Rate per Unit: $").append(ratePerUnit).append("\n");
        billDetails.append("Total Bill: $").append(totalBill).append("\n");
        billDetails.append("------------------------ \n");
        return billDetails.toString();
    }
}

class AdminPanel {
    private HashMap<Integer, Customer> customers;
    private double ratePerUnit;

    public AdminPanel() {
        customers = new HashMap<>();
        ratePerUnit = 0.10; // Default rate
    }

    public void addCustomer(String name, int meterNumber) {
        Customer customer = new Customer(name, meterNumber);
        customers.put(meterNumber, customer);
    }

    public void updateRate(double newRate) {
        if (newRate < 0) {
            System.out.println("Invalid rate. Rate per unit cannot be negative.");
        } else {
            ratePerUnit = newRate;
        }
    }

    public void updateCustomerReading(int meterNumber, double newReading) {
        if (customers.containsKey(meterNumber)) {
            customers.get(meterNumber).updateMeterReading(newReading);
        }
    }

    public void generateBill(int meterNumber) {
        if (customers.containsKey(meterNumber)) {
            customers.get(meterNumber).displayBill(ratePerUnit);
        }
    }

    // Getter for ratePerUnit
    public double getRatePerUnit() {
        return ratePerUnit;
    }

    // Getter for customer
    public Customer getCustomer(int meterNumber) {
        return customers.get(meterNumber);
    }
}

public class ElectricityBillingApp extends Frame implements ActionListener {
    // Components
    private TextField nameField, meterField, readingField, rateField;
    private Button addButton, updateButton, rateButton, billButton, exitButton;
    private TextArea outputArea;
    private AdminPanel adminPanel;

    // Constructor to set up the GUI
    public ElectricityBillingApp() {
        adminPanel = new AdminPanel();

        // Frame settings
        setTitle("Electricity Billing Admin Panel");
        setSize(400, 400);
        setLayout(new FlowLayout());

        // Initialize components
        nameField = new TextField(20);
        meterField = new TextField(20);
```

```
readingField = new TextField(20);
rateField = new TextField(20);
addButton = new Button("Add Customer");
updateButton = new Button("Update Meter Reading");
rateButton = new Button("Update Rate");
billButton = new Button("Generate Bill");
exitButton = new Button("Exit");
outputArea = new TextArea(10, 30);

// Add components to frame
add(new Label("Customer Name:"));
add(nameField);
add(new Label("Meter Number:"));
add(meterField);
add(new Label("New Reading:"));
add(readingField);
add(new Label("Rate per Unit:"));
add(rateField);
add(addButton);
add(updateButton);
add(rateButton);
add(billButton);
add(exitButton);
add(outputArea);

// Add action listeners
addButton.addActionListener(this);
updateButton.addActionListener(this);
rateButton.addActionListener(this);
```

```
        billButton.addActionListener(this);
        exitButton.addActionListener(this);

        // Window closing behavior
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        });

        // Make frame visible
        setVisible(true);
    }

    // Action event handling for button clicks
    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        int meterNumber;
        String name;
        double newReading;
        double newRate;

        switch (command) {
            case "Add Customer":
                name = nameField.getText();
                meterNumber = Integer.parseInt(meterField.getText());
                adminPanel.addCustomer(name, meterNumber);
                outputArea.setText("Customer added: " + name + " with Meter Number: "
+ meterNumber);
                break;
            case "Update Meter Reading":
                meterNumber = Integer.parseInt(meterField.getText());
                newReading = Double.parseDouble(readingField.getText());
                adminPanel.updateCustomerReading(meterNumber, newReading);
                outputArea.setText("Meter reading updated for meter number: " +
meterNumber);
                break;
            case "Update Rate":
                newRate = Double.parseDouble(rateField.getText());
                adminPanel.updateRate(newRate);
                outputArea.setText("New rate per unit set to: $" + newRate);
                break;
            case "Generate Bill":
                meterNumber = Integer.parseInt(meterField.getText());
                Customer customer = adminPanel.getCustomer(meterNumber);
                if (customer != null) {

outputArea.setText(customer.displayBill(adminPanel.getRatePerUnit()));
                } else {
                    outputArea.setText("Customer not found.");
                }
                break;
            case "Exit":
                System.exit(0);
                break;
        }
    }
    // Main method to start the program
    public static void main(String[] args) {
        new ElectricityBillingApp();
    }
}
```

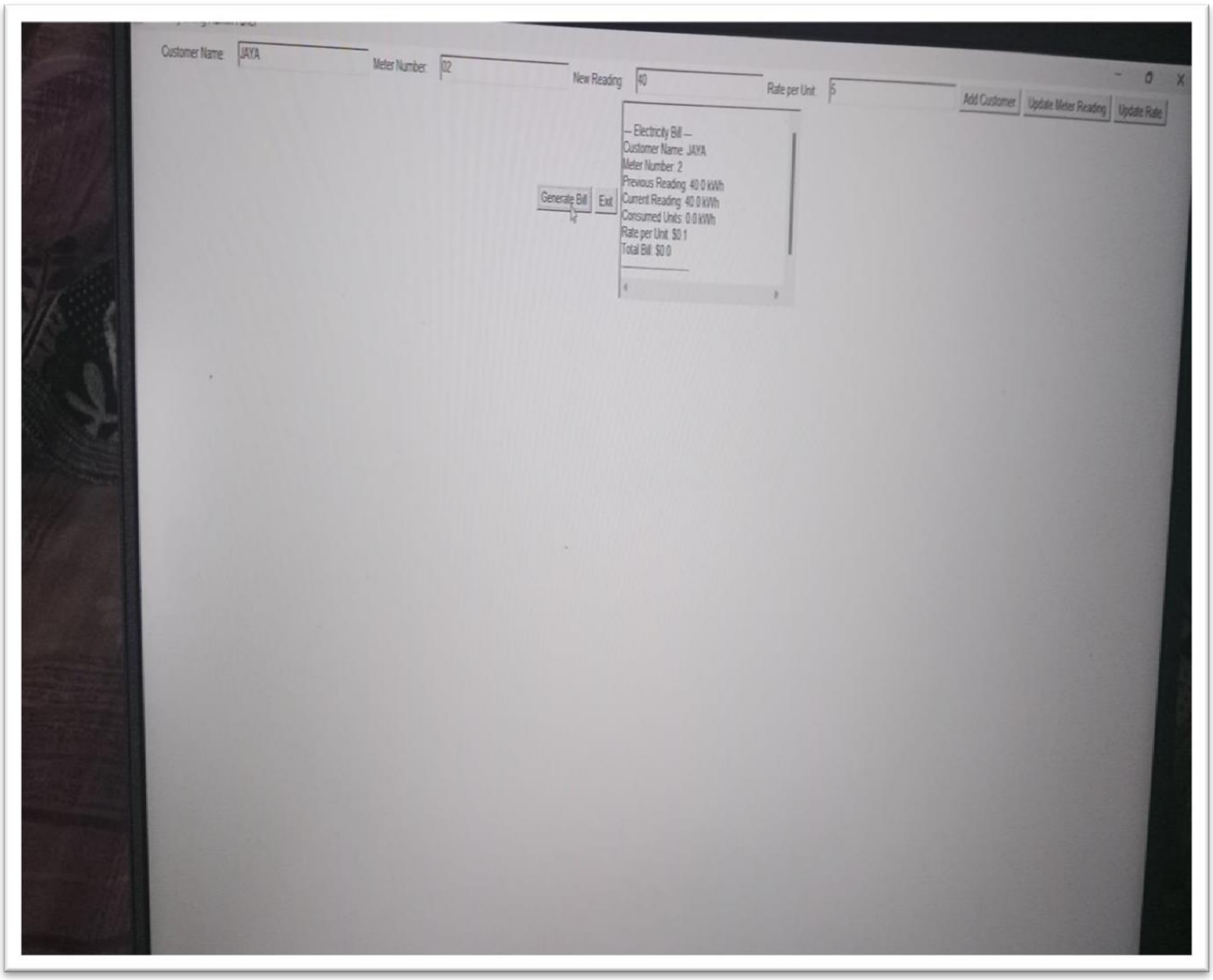
APPENDIX B

(SCREENSHOTS)



Customer Name: SOWMIYAA Meter Number: 01 New Reading: 30 Rate per Unit: 5

```
--- Electricity Bill ---
Customer Name: SOWMIYAA
Meter Number: 1
Previous Reading: 20.0 kWh
Current Reading: 30.0 kWh
Consumed Units: 10.0 kWh
Rate per Unit: $0.1
Total Bill: $1.0
```



Customer Name: JAYA

Meter Number: 02

New Reading: 40

Rate per Unit: 5

```
--- Electricity Bill ---
Customer Name: JAYA
Meter Number: 2
Previous Reading: 40.0 kWh
Current Reading: 40.0 kWh
Consumed Units: 0.0 kWh
Rate per Unit: $0.1
Total Bill: $0.0
```

REFERENCES

1. Core Java Volume I–Fundamentals by Cay S. Horstmann, 11th Edition (2018)
2. Beginning Java by Ivor Horton, 6th Edition (2014).