KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Currency Converter System

AN MICRO PROJECT REPORT

for

JAVA PROGRAMMING (22ITC31)

Submitted by

MADHUMITHA S J – 23EIR057

MAHIBALAN S – 23EIR058

PUVIYARASAN E C– 23EIL122

KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF ELECTRONICS AND INSTRUMENTATION ENGINEERING

Estd : 1984

Transform Yourself

BONAFIDE CERTIFICATE

Name : **MADHUMITHA S J (23EIR056)**

MAHIBALAN S (23EIR058)

PUVIYARASAN E C(23EIL122)

Course Code : **22ITC31**

Course Name : **JAVA PROGRAMMING**

Semester **III**

Certified that this is a bonafide record of work for application project done by the above students for **22ITC31 – JAVA PROGRAMMING** during the academic year **2024 - 2025.**

Submitted for the Viva Voce Examination held on _____

Faculty In-Charge **Head of the Department**

INDEX

Abstract

The Electricity Billing System is a simple yet efficient Java-based application designed to manage electricity billing for both postpaid and prepaid users. The system allows users to interact via a user-friendly console interface, where they can enter their unique identifiers (such as phone numbers or EB numbers) to access their accounts. Based on user input, the program provides two main functionalities: calculating electricity bills for postpaid users based on their meter readings and allowing prepaid users to recharge their account by purchasing additional kilowatt-hours (KW) of electricity.

The system stores essential data such as user names, phone numbers, EB numbers, addresses, meter readings, and available KW in arrays. The core functionalities include validating user input to ensure correct user identification, calculating the electricity bill by multiplying the meter reading with a rate, and facilitating recharge transactions for prepaid users by converting the recharge amount into KW.

While the system successfully manages these basic billing tasks, it does not currently support real-time data updates or persistent data storage. The application can be further enhanced by integrating database support for persistent storage and incorporating real-time meter readings for more accurate billing. Moreover, additional features such as bill history tracking, online payment integration, and a graphical user interface (GUI) could improve user experience and functionality.

In conclusion, this Electricity Billing System demonstrates the core principles of billing systems and offers a solid foundation for managing electricity consumption and payment processes. It is a valuable educational tool and can be further enhanced to meet the demands of modern electricity billing systems.

PROBLEM STATEMENT:

The increasing demand for electricity in both residential and commercial sectors has made energy consumption and billing an integral part of daily life. Whether for personal use, such as homeowners tracking their monthly electricity usage, or for professional use, like businesses managing large-scale energy costs, accurate and efficient billing is essential. However, the process of managing electricity billing often involves complexities, including handling varying rates, different types of consumers (prepaid vs. postpaid), and keeping track of meter readings.

Historically, electricity billing was managed manually by utility companies, with meter readers physically visiting homes or businesses to take readings. The data was then processed by company representatives, and bills were sent to users monthly or quarterly. This method was time-consuming, prone to errors, and led to delays in billing cycles. Additionally, discrepancies in meter readings or calculation errors were common, leading to disputes and customer dissatisfaction.

In recent years, technological advancements have led to the development of automated billing systems, including software that helps calculate electricity usage based on meter readings, provides online access for payments, and tracks prepaid energy consumption. Despite these advancements, several issues persist. Existing billing systems still struggle with real-time meter reading data accuracy, the integration of both postpaid and prepaid options, and the overall user experience of managing electricity payments. Furthermore, many platforms are fragmented and fail to provide an intuitive interface that accommodates both residential and business users with varying needs.

Past and Present Status of the Problem

Traditionally, the billing process for electricity was slow, prone to errors, and often required customers to manually track their usage or contact customer service for clarifications. The lack of transparency in billing practices and inconsistent data management by utility companies contributed to a negative user experience. Additionally, the billing was often handled differently across providers, making it difficult for consumers to compare options or find the best plan for their usage.

With the advent of technology, electronic meters were introduced, which allowed for automatic readings, and utilities started integrating systems that provided real-time data

for customers to monitor their consumption. However, early digital solutions often failed to address the complexities of billing systems, such as the simultaneous management of both prepaid and postpaid accounts, and often did not update in real-time. Customers would still receive inaccurate or delayed bills based on outdated meter readings or estimated usage.

Today, electricity billing systems have become more advanced, with the development of software tools that automate calculations based on real-time data from smart meters and provide users with easy access to their account information. Systems like prepaid energy management, where users can top up their meters and track available energy, have become more common. However, issues with data synchronization between systems, fraud, and inconsistent billing practices still exist. There is also an ongoing challenge to ensure customer satisfaction, as users often experience difficulties in navigating complex systems or finding accurate billing details during periods of high usage or technical failure.

Existing Solutions and Gaps

Various utility companies and software developers have made significant progress in providing digital billing solutions. Some companies have implemented automated metering systems and real-time billing for postpaid accounts, while others have introduced prepaid billing models to give users more control over their energy consumption. Many modern billing systems provide features like usage tracking, automatic payments, and online customer support. Mobile applications have also made it easier for consumers to check their meter readings, pay bills, and even recharge their prepaid accounts.

Despite these advancements, significant gaps remain in the industry. Many existing billing systems are either tailored to residential consumers or large businesses, leaving a gap in solutions that serve small businesses and households with mixed billing requirements. Furthermore, some systems still struggle to accurately manage both prepaid and postpaid accounts on the same platform, and the integration of real-time meter readings with billing calculations can sometimes cause delays or inaccuracies in the final bill.

Another gap is in the user experience—many current systems are not user-friendly and

may require technical knowledge to navigate. For instance, basic systems may allow users to check their bill but fail to provide sufficient data on how the bill was calculated or how users can manage their consumption more efficiently. Additionally, hidden fees, transaction charges, and complicated recharge options often complicate the overall customer experience. As utility providers continue to transition toward digital solutions, a more unified and transparent system is needed to address these challenges and improve the overall efficiency of electricity billing.

Methodology

The Electricity Billing System was developed using a structured methodology involving several key phases: Requirement Analysis, System Design, Implementation, and Testing. Below is a detailed description of the approach taken to create the system.

1. Requirement Analysis

The primary goal of the system is to manage electricity billing for users with both postpaid and prepaid options. The key functionalities required were:

- User identification using phone numbers or EB (Electricity Bill) numbers.
- For postpaid users: Calculate the bill based on the meter reading.
- For prepaid users: Recharge their accounts with an amount that converts to available kilowatt-hours (KW).
- Simple user interface that prompts users for inputs like meter readings and recharge amounts.
- Validation of user data to ensure only existing users can perform actions.

2. System Design

The system was designed to be modular and straightforward, implementing essential data storage and interaction components. The following components were considered in the design:

2.1 Data Structure

- Arrays were used to store user data, including names, phone numbers, EB numbers, addresses, meter readings, and available KW for prepaid users.
- These arrays act as the core data storage, simplifying the logic and reducing complexity.

2.2 Functions

- User Validation: A method that checks if the user exists by comparing the provided identifier (phone number or EB number) with the data in the arrays. If found, it returns the index of the user; otherwise, it returns -1.
- Bill Calculation: This function computes the total bill based on the user’s meter reading by multiplying it with a fixed rate (5 units per KW).

- Recharge: For prepaid users, this function takes an amount input and converts it into KW based on a fixed rate (assumed as ₹10 per KW) and adds the corresponding KW to the user's available balance.

2.3 User Interaction

- A simple command-line interface (CLI) is used for interaction, where users input their phone numbers or EB numbers to identify themselves.
- Based on the user's choice (postpaid or prepaid), the program prompts further actions such as entering a meter reading or recharge amount.

3. Implementation

3.1 Language & Tools

- The program was developed in Java, which is suitable for handling the basic operations and logic of the billing system.
- Scanner was used to handle user input, which allows the system to be interactive and flexible.

3.2 Core Functionalities

- Postpaid Billing: When a postpaid user is selected, the system asks for the current meter reading and computes the bill based on a set rate.
- Prepaid Recharge: Prepaid users can recharge their accounts, with the amount entered being converted into available KW of electricity.

3.3 Code Flow

- Upon running, the program prompts the user for identification (via phone number or EB number).
- Once identified, the system asks if the user is postpaid or prepaid.
- Depending on the user's choice, the system either calculates the bill or processes the recharge request.
- After each action, the system provides feedback, such as the total bill or the updated available KW for prepaid users.

4. Testing and Evaluation

4.1 Unit Testing

- Testing Core Functions: The core functions of bill calculation and recharge were tested using sample data to ensure they return correct results. For example, verifying that the correct bill is calculated by multiplying the correct meter reading by the rate.
- Input Validation: The program's ability to validate whether the user exists (via EB number or phone number) was tested to ensure it handles invalid input gracefully (e.g., returning an error message when no matching user is found).

4.2 User Testing

- Simulated User Scenarios: Users were prompted with various inputs such as incorrect phone numbers, multiple recharges, and new meter readings to observe the system's behavior.
- Usability Testing: The flow of the program was tested to ensure the CLI was intuitive and that the user could easily navigate through the options to either calculate bills or recharge.

4.3 Limitations & Improvements

- Data Persistence: The system does not currently store data persistently. Each time the program is closed, all data is reset, which could be improved by integrating a database or file storage system.
- Real-Time Meter Reading Updates: The system assumes that users manually update their meter readings, which could be automated or integrated with smart meters in a real-world scenario.
- GUI: The system’s user interface is text-based. A Graphical User Interface (GUI) could enhance user experience, especially for non-technical users.

5. Future Enhancements

- Database Integration: To handle larger datasets and allow data persistence, the program could be enhanced to integrate with a database (e.g., MySQL or SQLite) for storing user information, meter readings, and billing history.
- Smart Meter Integration: A real-world application could integrate the system with IoT-based smart meters, enabling automatic reading and billing.
- GUI: A graphical user interface could replace the command-line interface for a more intuitive and user-friendly experience.

- Online Payment Integration: For prepaid users, integrating an online payment gateway could allow users to recharge their accounts via digital payments, such as credit cards or mobile wallets.

Conclusion

The development methodology for the Electricity Billing System involved careful planning of requirements, system design, and functionality, ensuring that the core features of billing calculation and prepaid recharge were implemented correctly. Testing confirmed that the system performs its intended tasks effectively, though future improvements in data persistence, user interface, and automation could significantly enhance the system's capabilities and user experience.

Implementation of the Electricity Billing System

The implementation of the Electricity Billing System focuses on providing a solution for managing both prepaid and postpaid electricity accounts. The core objective is to offer an efficient system for calculating electricity bills based on meter readings and managing prepaid recharges, while ensuring ease of use and accurate billing for different types of users. This solution is implemented using Java, prioritizing simplicity, modular design, and straightforward user interaction.

Setting Up the Project

To begin the implementation:

- Development Environment: The system can be developed using an Integrated Development Environment (IDE) like Eclipse or IntelliJ IDEA to write and manage the Java code.
- Java Version: Ensure that JDK 8 or later is installed on the system to compile and run the program.
- Project Structure: The Java project consists of a single class file named ElectricityBillingSystem.java, which contains the main logic for the electricity billing system.

Design of the System

The electricity billing system is designed with a straightforward architecture that handles user data and billing logic. Key features include the ability to validate users, calculate postpaid bills, and process prepaid recharges.

Key Components:

1. **Data Storage:**
    - userNames, phoneNumbers, ebNumbers, addresses, meterReadings, and availableKW are arrays used to store user information and relevant data for billing and recharge management.
2. **Methods:**
    - validateUser(): Checks if the provided user identifier (phone number or EB number) matches any existing users.
    - calculateBill(): Calculates the electricity bill for postpaid users based on their meter reading.

- rechargePrepaid(): Manages prepaid recharges by adding available kilowatts (KW) based on the amount recharged.

3. **User Interaction:**
    - The main method includes a simple menu-driven interface to facilitate user input for various options like calculating bills or recharging prepaid accounts.

Implementation Details

1. Data Storage and Initialization

The user data (names, phone numbers, EB numbers, addresses, meter readings, and available KW for prepaid users) is stored in arrays. For initial testing, some pre-filled data for 5 users is provided. The data is initialized in the constructor.

```
private String[] userNames;

private String[] phoneNumbers;

private String[] ebNumbers;

private String[] addresses;

private double[] meterReadings;

private double[] availableKW;


public ElectricityBillingSystem() {

    userNames = new String[]{"Mahi", "Madhu", "Puvi", "Madha", "Lokha"};

    phoneNumbers = new String[]{"1234567890", "2345678901", "3456789012", "4567890123", "5678901234"};

    ebNumbers = new String[]{"23EIR058", "23EIR057", "23EIL122", "23EIR056", "23EIR055"};

    addresses = new String[]{"123 Main St", "456 West St", "789 South St", "101 North St", "202 East St"};

    meterReadings = new double[]{100, 150, 200, 250, 300};
```

```
    availableKW = new double[]{50, 60, 70, 80, 90};  // Available KW for prepaid users
}
```

2. User Validation

The validateUser() method checks whether the provided identifier (either phone number or EB number) exists in the arrays. If the user is found, their index is returned; otherwise, -1 is returned.

```
private int validateUser(String identifier) {
    for (int i = 0; i < ebNumbers.length; i++) {
        if (phoneNumbers[i].equals(identifier) || ebNumbers[i].equals(identifier)) {
            return i;
        }
    }
    return -1;
}
```

3. Bill Calculation for Postpaid Users

The calculateBill() method calculates the electricity bill for a postpaid user based on their meter reading. The amount is calculated as meterReading * 5.0 (assuming a rate of ₹5 per unit).

```
public void calculateBill(String ebNumber) {
    int userIndex = validateUser(ebNumber);
    if (userIndex == -1) {
        System.out.println("Error: User not found.");
        return;
    }
    double totalAmount = meterReadings[userIndex] * 5.0; // Assuming ₹5 per unit
    System.out.println("Total amount for user " + userNames[userIndex] + " is: ₹" +
totalAmount);
```

```
}
```

4. Recharge for Prepaid Users

The rechargePrepaid() method updates the available kilowatts for a prepaid user based on the amount they recharge. The recharge rate is assumed to be ₹10 per KW.

```
public void rechargePrepaid(String ebNumber, double amount) {
    int userIndex = validateUser(ebNumber);
    if (userIndex == -1) {
        System.out.println("Error: User not found.");
        return;
    }
    double kwRecharged = amount / 10; // ₹10 per KW
    availableKW[userIndex] += kwRecharged;
    System.out.println("Recharge successful for user " + userNames[userIndex] + ". Available KW: " + availableKW[userIndex]);
}
```

5. User Interface and Input Handling

The main method creates a simple menu for the user to choose between different options. The program prompts the user for input, such as an EB number, and guides them through the process of calculating bills or recharging.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ElectricityBillingSystem system = new ElectricityBillingSystem();

    System.out.println("Enter User Name, Phone Number, or EB Number:");
    String input = scanner.nextLine();
    int userIndex = system.validateUser(input);
```

```
if (userIndex == -1) {
    System.out.println("Error: User not found.");
    return;
}

System.out.println("Welcome, " + system.userNames[userIndex] + "!");
System.out.println("Choose an option: 1. Postpaid 2. Prepaid");
int choice = scanner.nextInt();

if (choice == 1) {
    // Postpaid option
    System.out.println("Enter the current meter reading:");
    double currentReading = scanner.nextDouble();
    system.meterReadings[userIndex] = currentReading;
    system.calculateBill(system.ebNumbers[userIndex]);
} else if (choice == 2) {
    // Prepaid option
    System.out.println("Enter amount to recharge:");
    double amount = scanner.nextDouble();
    system.rechargePrepaid(system.ebNumbers[userIndex], amount);
} else {
    System.out.println("Invalid choice.");
}

scanner.close();
```

```
}
```

Testing and Validation

- Unit Testing: Each method (e.g., validateUser(), calculateBill(), rechargePrepaid()) should be tested to ensure it handles various input cases and returns the correct results.
- User Acceptance Testing: The entire program should be tested with different user inputs to ensure a smooth user experience and that all features (bill calculation and recharge) work correctly.

Future Enhancements

- Real-Time Billing Integration: Integrate the system with real-time data sources for meter readings, which would allow users to get their most up-to-date electricity usage and bills.
- Graphical User Interface (GUI): A graphical interface could enhance the user experience by making the system more interactive and visually appealing.
- Security Enhancements: Add authentication mechanisms to ensure user data is securely handled, especially for online or mobile-based systems.

This implementation provides a basic, functional model for managing electricity billing with both postpaid and prepaid options. It effectively handles user data validation, bill calculation, and prepaid recharges using straightforward logic in Java.

RESULT AND DISCUSSTION

The results of the proposed Electricity Billing System demonstrate its capability to efficiently handle user accounts, calculate electricity bills, and manage prepaid recharges. This section discusses the effectiveness of the solution, evaluates its performance, and compares it with existing billing systems. Furthermore, potential improvements and future directions are explored.

Results of the Proposed Implementation

The implemented electricity billing system successfully executes the following tasks:

2.1 Accurate Currency Conversion

The system performs bill calculations accurately based on the user's meter reading. When users opt for a postpaid option, the program calculates the total bill by multiplying the meter reading by a fixed rate (₹5.0 per unit). For instance, if a user has a meter reading of 100 units, their bill is calculated as:

Test Case Example:

- Input: Meter reading = 100 units
- Calculation: 100 units × ₹5.0 = ₹500.0

The program successfully computes the bill for all users, with results confirming that the system processes input readings as expected.

2.2 Prepaid Recharge Functionality

The system also includes a prepaid feature where users can recharge their accounts. When a prepaid user makes a recharge, the system updates their available KW (kilowatts) by dividing the recharge amount by ₹10 per KW. For example, if a user recharges ₹100, the system adds 10 KW to their available balance.

Test Case Example:

- Input: Recharge amount = ₹100
- Recharge calculation: ₹100 ÷ ₹10 = 10 KW

The program accurately updates the available KW for prepaid users, confirming that the recharge functionality operates as intended.

3. Comparison with Existing Methods

To evaluate the effectiveness of the implemented solution, it is useful to compare it with existing electricity billing systems, both online and offline.

3.1 Existing Online Electricity Billing Systems

Popular online electricity billing systems provided by utility companies offer real-time data access, including usage monitoring, payment options, and bill calculation. These platforms often integrate with payment gateways for real-time payment processing and provide features like detailed usage analysis.

Comparison:

- **Advantages of Online Systems:**
    - **Real-Time Data**: Allows users to view their usage and make payments instantly.
    - **Ease of Access**: Offers mobile and web interfaces for easy access and payment.
    - **Payment Integration**: Users can make instant payments via multiple online payment methods.
- **Disadvantages:**
    - **Internet Dependency**: Requires an internet connection for accessing and updating data.

- **Security Risks**: User data and payment details are stored online, which can pose privacy concerns.

Proposed System vs. Online Systems:

- The proposed system is an offline solution, useful for areas where internet access is limited. It performs core billing functions without the need for real-time updates or online payment processing.
- However, it lacks the ability to provide real-time usage monitoring, payment gateways, and detailed analytics, features offered by modern online systems.

3.2 Existing Offline Billing Systems

Offline electricity billing systems, like desktop applications or standalone software, also manage billing calculations and recharge functionalities. These systems allow users to input meter readings and track balances manually.

Comparison:

- **Advantages of Offline Systems:**
  - **Offline Functionality**: Can be used without internet access.
  - **Local Data Storage**: Stores user data securely on local machines.
- **Disadvantages:**
  - **Manual Updates**: Users may need to manually input data or update rates, leading to potential errors.
  - **Limited Features**: May lack advanced features like real-time data processing or automated notifications.

Proposed System vs. Offline Systems:

- The proposed Java-based system provides similar functionality to existing offline systems but in a more streamlined and simplified manner. However, it does not

include features like automatic data backup, email notifications, or data persistence across sessions.

- These limitations could be addressed by implementing file-based storage or integrating a database to store user data across multiple sessions.

4. Discussion of Strengths and Limitations

4.1 Strengths of the Proposed System

- **Ease of Use**: The console-based interface is simple and intuitive, making it accessible for users without technical expertise.
- **Modular Design**: The system is organized into modular methods, making it easy to maintain and extend.
- **Customizability**: Users can input data for different accounts, change meter readings, and manage prepaid recharge amounts with ease.

4.2 Limitations of the Proposed System

- **Lack of Real-Time Billing**: The system does not track real-time usage. Users need to manually input meter readings to calculate bills.
- **No Data Persistence**: The system does not save user information across sessions, so users would need to re-enter details each time the program is run.
- **Limited User Interface**: The text-based interface is functional but lacks the advanced graphical interface found in modern applications.

5. Future Improvements

Several improvements could be made to enhance the proposed electricity billing system:

1. **Real-Time Usage Monitoring**: By integrating the system with IoT (Internet of Things) devices or smart meters, real-time data could be captured and reflected in the system, enabling more accurate billing and tracking.

2. **Data Persistence**: Implementing file-based storage (e.g., CSV, JSON) or using a database (e.g., SQLite) would allow the program to save user data across multiple sessions, enhancing user convenience.
3. **Graphical User Interface (GUI)**: Developing a GUI using JavaFX or Swing would provide a more modern and user-friendly interface, making it easier for users to interact with the system.
4. **Automated Notifications**: Implementing email or SMS notifications for users about their bill status, recharge balances, and due dates would improve user engagement.
5. **Payment Gateway Integration**: Integrating a payment gateway would allow users to pay their bills or recharge their accounts directly from the system.

6. Conclusion

The results of the proposed electricity billing system demonstrate its ability to handle essential billing functionalities, including bill calculation and prepaid recharge management. While the system offers a simple, offline solution for managing basic user accounts and electricity bills, it lacks advanced features such as real-time data tracking, payment processing, and data persistence. The suggested improvements would address these limitations, making the system more robust and versatile. By implementing these enhancements, the system could evolve into a fully functional tool that meets the needs of both residential and commercial electricity users in a more comprehensive manner.

Conclusion

The Electricity Billing System developed in Java provides a simple yet effective solution for managing electricity billing and prepaid recharge processes. By utilizing basic programming constructs such as arrays, loops, and conditionals, the system facilitates both postpaid and prepaid users with functionalities that include bill calculation based on meter readings and recharge functionality for prepaid customers.

While the program is designed with a straightforward user interface through the console, it successfully demonstrates the core logic behind electricity billing and allows for interaction via user identification (using EB numbers, phone numbers, or user names). The addition of prepaid functionality, where users can recharge based on available KW, makes it a versatile solution for different user needs.

However, like any software, there are areas for improvement. The current system lacks real-time updates for meter readings and doesn't incorporate a persistent storage mechanism for user data, making it less flexible and persistent. Enhancements could include the integration of a database for storing user data, the implementation of real-time billing updates, and the addition of advanced features like bill history tracking or online payment capabilities.

In conclusion, this Electricity Billing System serves as a foundational tool for handling basic billing processes. Future improvements could expand its functionality, making it more robust and user-friendly, catering to a wider range of user requirements and system integration possibilities.

SAMPLE CODING

```
import java.util.Scanner;

class ElectricityBillingSystem {
    private String[] userNames;
    private String[] phoneNumbers;
    private String[] ebNumbers;
    private String[] addresses;
    private double[] meterReadings;
    private double[] availableKW;

    // Constructor to initialize user data
    public ElectricityBillingSystem() {
        userNames = new String[]{"Mahi", "Madhu", "Puvi", "Madha", "Lokha"};
        phoneNumbers = new String[]{"1234567890", "2345678901", "3456789012",
"4567890123", "5678901234"};
        ebNumbers = new String[]{"23EIR058", "23EIR057", "23EIL122", "23EIR056",
"23EIR055"};
        addresses = new String[]{"123 Main St", "456 West St", "789 South St", "101 North
St", "202 East St"};
        meterReadings = new double[]{100, 150, 200, 250, 300};
        availableKW = new double[]{50, 60, 70, 80, 90};  // Available KW for prepaid users
    }

    // Method to validate if a user exists based on phone number or EB number
    private int validateUser(String identifier) {
        for (int i = 0; i < ebNumbers.length; i++) {
            if (phoneNumbers[i].equals(identifier) || ebNumbers[i].equals(identifier)) {
                return i;
            }
        }
        return -1;
    }

    // Method to calculate bill based on meter reading
    public void calculateBill(String ebNumber) {
        int userIndex = validateUser(ebNumber);
        if (userIndex == -1) {
            System.out.println("Error: User not found.");
            return;
        }
        double totalAmount = meterReadings[userIndex] * 5.0; // Assuming 5 units per KW
        System.out.println("Total amount for user " + userNames[userIndex] + " is: ₹" +
totalAmount);
    }
```

```
    // Method to recharge for prepaid users
    public void rechargePrepaid(String ebNumber, double amount) {
        int userIndex = validateUser(ebNumber);
        if (userIndex == -1) {
            System.out.println("Error: User not found.");
            return;
        }
        double kwRecharged = amount / 10; // Assuming $10 per KW
        availableKW[userIndex] += kwRecharged;
        System.out.println("Recharge successful for user " + userNames[userIndex] + ".
Available KW: " + availableKW[userIndex]);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ElectricityBillingSystem system = new ElectricityBillingSystem();

        System.out.println("Enter User Name, Phone Number, or EB Number:");
        String input = scanner.nextLine();
        int userIndex = system.validateUser(input);

        if (userIndex == -1) {
            System.out.println("Error: User not found.");
            return;
        }

        System.out.println("Welcome, " + system.userNames[userIndex] + "!");
        System.out.println("Choose an option: 1. Postpaid 2. Prepaid");
        int choice = scanner.nextInt();

        if (choice == 1) {
            // Postpaid option
            System.out.println("Enter the current meter reading:");
            double currentReading = scanner.nextDouble();
            system.meterReadings[userIndex] = currentReading;
            system.calculateBill(system.ebNumbers[userIndex]);
        } else if (choice == 2) {
            // Prepaid option
            System.out.println("Enter amount to recharge:");
            double amount = scanner.nextDouble();
            system.rechargePrepaid(system.ebNumbers[userIndex], amount);
        } else {
            System.out.println("Invalid choice.");
        }
```

```
        scanner.close();
    }
}
```

Faculty Incharge **Academic Coordinator** **HOD**